POSITION-ADAPTIVE DIRECTION FINDING FOR MULTI-PLATFORM RF EMITTER LOCALIZATION USING EXTREMUM SEEKING CONTROL

Dissertation

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Doctor of Philosophy in Electrical Engineering

By

Huthaifa Ahmad Al Issa

Dayton, Ohio

August, 2012
POSITION-ADAPTIVE DIRECTION FINDING FOR MULTI-PLATFORM RF EMITTER LOCALIZATION USING EXTREMUM SEEKING CONTROL

Name: Al Issa, Huthaifa Ahmad

APPROVED BY:

__________________________________________  _______________________________________
Raúl Ordóñez, Ph.D.                             Robert Penno, Ph.D.
Advisor Committee Chairman                     Committee Member
Professor, Department of Electrical             Professor, Department of Electrical
and Computer Engineering                        and Computer Engineering

__________________________________________  _______________________________________
John Loomis, Ph.D.                              Robert Gorton, Ph.D.
Committee Member                                Committee Member
Professor, Department of Electrical             Professor, Department of
and Computer Engineering                        Mathematics

__________________________________________  _______________________________________
John G. Weber, Ph.D.                            Tony E. Saliba, Ph.D.
Associate Dean                                  Dean, School of Engineering
School of Engineering                           & Wilke Distinguished Professor
ABSTRACT

POSITION-ADAPTIVE DIRECTION FINDING FOR MULTI-PLATFORM RF
EMITTER LOCALIZATION USING EXTREMUM SEEKING CONTROL

Name: Al Issa, Huthaifa Ahmad
University of Dayton
Advisor: Dr. Raúl Ordóñez

In recent years there has been growing interest in Ad-hoc and Wireless Sensor Networks (WSNs) for a variety of indoor applications. Localization information in these networks is an enabling technology and in some applications it is the parameter of primary importance. WSNs are being used in a variety of ways – from reconnaissance and detection in military to biomedical applications and a wide variety of commercial endeavors. In recent years, position-based services have become more important. Thus, recent developments in communications and RF technology have enabled system concept formulations and designs for low-cost radar systems using state-of-the-art software radio modules, which are capable of local processing and wireless communication, a reality. Such nodes are called as sensor nodes. Each sensor node is capable of only a limited amount of processing.

This research focused on the modeling and implementation of distributed, mobile radar sensor networks. In particular, we worked on the problem of Position-Adaptive Direction Finding (PADF), to determine the location of a non-collaborative transmitter, possibly hidden within a structure, by using a team of cooperative intelligent sensor networks. Our
purpose is to further develop and refine position-adaptive RF sensing techniques based on the measurement and estimation of RF scattering metrics. Topics planned for this entrepreneurial research project are focused on the investigation, analysis/simulation, and development of real time multi-model (i.e., complex multipath) environments scattering decompositions for PADF geometries. PADF is based on the formulation and investigation of path-loss based RF scattering metrics (i.e., estimation of distributed Path Loss Exponent, or PLE) that are measured and estimated across multiple platforms in order to enable the robotic/intelligent position-adaptation (or self-adjustment) of the location of each platform.

We provide a summary of recent experimental results in localization of a non-cooperative sensor node using static and mobile sensor networks. In this study we used IRIS wireless sensor nodes. In order to localize the transmitter, we used the Received Signal Strength Indicator (RSSI) data to approximate distance from the transmitter to the revolving receivers. We provided an algorithm for on-line estimation of the PLE that is used in modeling the distance based on RSSI measurements. The emitter position estimation is calculated based on surrounding sensors RSSI values using Least-Square Estimation (LSE). The PADF has been tested on a number of different configurations in the laboratory via the design and implementation of four IRIS wireless sensor nodes as receivers and one hidden sensor as a transmitter during the localization phase. The robustness of detecting the transmitter’s position is initiated by getting the RSSI data through experiments and then data manipulation in MATLAB will determine the robustness of each node and ultimately that of each configuration. The parameters that are used in the functions are the median values of RSSI and rms values. From the result it is determined which configurations possess high robustness. High values obtained from the robustness function indicate high robustness, while low values indicate lower robustness.
Finally, we present the experimental performance analysis on the application aspect. We apply Extremum Seeking Control (ESC) schemes by using the swarm seeking problem, where the goal is to design a control law for each individual sensor that can minimize the error metric by adapting the sensor positions in real-time, thereby minimizing the unknown estimation error. As a result we achieved source seeking and collision avoidance of the entire group of the sensor positions.
To my father

To my mother

To all my brothers and sisters

To Dr. Bader Almadi

To all those who have helped me along the way with their love and support

For without their support I would never have come this far.
ACKNOWLEDGMENTS

I give a sincere gratitude to the people who do science with pure, unselfish and honest passion as they are the people who made me grow and appreciate the world in the way I see. They helped in ways ranging from personal to technical, and my gratitude to each cannot be understated. I would like to express my gratitude to all those who gave me the possibility to complete my Ph.D study. I would like to thank my family, friends, advisors, and committee members for all they have given me over the last few years.

This dissertation would not have been possible without the support of many people over the last few years. Over the last few years, I have had the privilege to work with a number of people who have made my time at the University of Dayton enjoyable and rewarding. I’d like to thank all of them. Without them this dissertation would not be possible. First, I am deeply grateful to my advisor, Professor Raúl Ordóñez. Prof. Ordóñez is an outstanding control scientist with broad knowledge, sharp intuition, and grand vision. Prof. Ordóñez is also a great mentor. He is very patient and gives me lots of freedom to explore the field by myself. Under his guidance, I was able to learn the fundamental lessons of being a researcher: finding valuable problems, investigating innovative ideas and presenting meaningful results. His inspiration and warm personality have won my highest respect and trust. I am grateful for his continued support and guidance throughout the whole dissertation process. I am very grateful to had his trust in my ability, and I have often benefited from his insight and advice.
I am extremely grateful to my dissertation committee members, Professor Robert Penno, Professor John Loomis, Professor Robert Gorton for their providing valuable suggestions, guidance, comments and encouragements, time and effort that they invested in criticizing my dissertation, and for the interest they expressed for my research.

Many thanks to my colleagues at the University of Dayton for making this Ph.D such an enjoyable experience. My special thanks goes to Professor Don Moon for his help and support. I am thankful to the staff of University of Dayton: Marilyn Knisley for helping me with all administrative issues, support and encouragement during all my studies.

My gratitude also goes to Dr. Mohammad Eyadat from the California State University for his advice and encouragement.

I give special thanks to Dr. Mohammad Almajali. Directorate of Strategic Planning/JAF. International Cooperation Branch. For his support and his help.

I would like to thank senate: Tala Almadi for pushing me to get my Ph.D and for his advice.

I wish to deeply thank Dr. Ali Rawabdeh for his help and advice.

I would like to thank Dr. Alan Jennings for his insight in \LaTeX. His help was greatly appreciated.

I am fortunate to be among many good fellow students and friends. I give great thanks to my friends for being there for me when I needed them the most. I would like to thank the following friends for many helpful discussions and for friendships: Shamsul Anwar Shamsudin, Mosa Abdesalam, Osama Abolaeha, Tanner Adams.

Finally, my gratitude goes to my family for their love, support and encouragement during all my studies. Most of all, thank you dad, mum, sisters, and brothers for supporting and encouraging me whenever fear or discouragement were looming over me. Without
them, I would not have reached this point. I thank God for looking over me during these past years and for granting me the ability to complete this degree.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter/Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>iii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td></td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td></td>
<td>xxi</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td></td>
<td>xxiii</td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Motivation and Significance</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>History and Literature Overview</td>
<td>7</td>
</tr>
<tr>
<td>1.3</td>
<td>Contributions</td>
<td>19</td>
</tr>
<tr>
<td>1.4</td>
<td>Outline</td>
<td>22</td>
</tr>
<tr>
<td>1.5</td>
<td>Wireless Sensor Networks</td>
<td>24</td>
</tr>
<tr>
<td>1.5.1</td>
<td>Hardware Specifications</td>
<td>27</td>
</tr>
<tr>
<td>1.5.2</td>
<td>Sensor Node Protocol</td>
<td>29</td>
</tr>
<tr>
<td>1.5.3</td>
<td>Wireless Sensor Networks Applications</td>
<td>33</td>
</tr>
<tr>
<td>1.6</td>
<td>What is Localization?</td>
<td>35</td>
</tr>
<tr>
<td>1.6.1</td>
<td>Localization Principle</td>
<td>38</td>
</tr>
<tr>
<td>1.6.2</td>
<td>Range-Free Vs Range-Based</td>
<td>38</td>
</tr>
<tr>
<td>1.6.3</td>
<td>Localization Techniques</td>
<td>39</td>
</tr>
</tbody>
</table>
II. POSITION-ADAPTIVE DIRECTION FINDING (PADF) ............... 44
   2.1 Problem Statement ........................................... 45
   2.2 Position-Adaptive Direction Finding Concepts ................. 46
   2.3 Hardware and Software Components .......................... 50
      2.3.1 IRIS Wireless Sensor Nodes ............................... 51
      2.3.2 Tiny Operating System .................................... 52
      2.3.3 IEEE 802.14.2 Technology ................................. 55

III. MODELING AND LOCALIZATION METHODOLOGY ....................... 59
   3.1 Position Estimation Based Signal Strength ..................... 59
      3.1.1 Emitter Localization Estimation ......................... 62
      3.1.2 RSSI Model on IRIS Sensor Nodes ...................... 63
   3.2 Sensitivity and Robustness Issues ............................ 66
      3.2.1 Experimental Procedure ................................. 67
   3.3 Error Metric .................................................. 91

IV. EXPERIMENTAL AND LAB SETUP ..................................... 96
   4.1 Case One .................................................... 97
      4.1.1 Case One: Experiments Without Obstruction .............. 97
      4.1.2 Case One: Experiments With Obstruction .............. 97
   4.2 Case Two ................................................... 99
      4.2.1 Case Two: Experiments Without Obstruction ............ 99
      4.2.2 Case Two: Experiments With Obstruction .............. 101

V. EXPERIMENTAL RESULTS ............................................. 105
   5.1 At IDCAST .................................................. 105
      5.1.1 Case One ................................................ 110
      5.1.2 Case Two ............................................... 113
   5.2 Experiments at AFRL MAV LAB .................................. 115

VI. SOURCE SEEKING BASED EXTREMUM SEEKING CONTROL ............ 122
   6.1 Why Extremum Seeking Control ................................. 122
      6.1.1 History and Literature Overview ....................... 124
      6.1.2 Applications of Extremum Seeking Control ............ 126
   6.2 Swarm Source Seeking ........................................ 128
   6.3 Problem Statement ............................................ 130
   6.4 Simulation Results Without Obstruction for Two Different Cases 133
      6.4.1 Case One ................................................. 133
      6.4.2 Case Two ................................................. 139
6.5 Simulation Results With Obstruction for Two Different Cases . . . . . . . 139
  6.5.1 Case One . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
  6.5.2 Case Two . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147
6.6 Monte Carlo Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . 150

VII. CONCLUSIONS AND FUTURE WORK . . . . . . . . . . . . . . . . . . . . . 160

7.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 160
7.2 Future Work and Open Problems . . . . . . . . . . . . . . . . . . . . . . 163

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 166

APPENDICES:

A. GRAPHICAL USER INTERFACE (GUI) CODE BY MATLAB . . . . . . . . 192
  A.1 How To Run The Code . . . . . . . . . . . . . . . . . . . . . . . . . . . . 192
  A.2 Padf.Gui.m . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 193
  A.3 Process Data Func Code . . . . . . . . . . . . . . . . . . . . . . . . . . 203
  A.4 Read_BaseStation Code . . . . . . . . . . . . . . . . . . . . . . . . . . 205
  A.5 Try_Serial_Reading Code . . . . . . . . . . . . . . . . . . . . . . . . . . 206
  A.6 ABC values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206

B. TINY OPERATING SYSTEM . . . . . . . . . . . . . . . . . . . . . . . . . . . 207
  B.1 IRIS Mote Program: About This Software . . . . . . . . . . . . . . . . 207
    B.1.1 How_To_Make_Nc_Code . . . . . . . . . . . . . . . . . . . . . . . 207
    B.1.2 ReadME_BEFORE_Making . . . . . . . . . . . . . . . . . . . . . . 207
    B.1.3 RssiDemoMessages Code . . . . . . . . . . . . . . . . . . . . . . . 208
  B.2 ForwardingMote . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 208
    B.2.1 ForwardToBaseAppC Code . . . . . . . . . . . . . . . . . . . . . . 208
    B.2.2 ForwardToBaseC Code . . . . . . . . . . . . . . . . . . . . . . . . 209
  B.3 Build Iris Code . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
    B.3.1 App . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
  B.4 InterceptBase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 372
    B.4.1 BaseStationC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 372
    B.4.2 BaseStationP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 373
  B.5 RssiBase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 376
    B.5.1 RssiBaseAppC Code . . . . . . . . . . . . . . . . . . . . . . . . . . 376
    B.5.2 RssiBaseC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 377
  B.6 SendingMote . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 378
    B.6.1 SendingMoteAppC . . . . . . . . . . . . . . . . . . . . . . . . . . . 378
    B.6.2 SendingMoteC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 378
C. ROBUSTNESS CODE ......................................................... 380
  C.1 Robustness Code ..................................................... 380
    C.1.1 Compute Median and Rms From Experimental Data .......... 380
    C.1.2 Huthaifa_2048 Code ........................................... 380
    C.1.3 Robustness For Configuration I and II Code ............... 382

D. EXTREMUM SEEKING CONTROL ......................................... 386
  D.1 Extremum Seeking Control Code ................................... 386
    D.1.1 Without Obstruction Code .................................... 386
    D.1.2 Extremumswarming_No2_Derivative Code ..................... 388
    D.1.3 Min2d Code ................................................... 389
    D.1.4 Metric_Potential Code ....................................... 390
    D.1.5 Localization_Error Code .................................... 391
    D.1.6 Compute Metric Code ........................................ 392
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of a wireless sensor network.</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Sensor network communication structure.</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Basic architecture of a wireless sensor node [115].</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Sensor node protocol stack</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>Position-adaptive radar concepts for single or multiple mini-UAV helicopters.</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>(a) Notional PADF initial state. (b) Notional PADF final state.</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>Basic functional block diagram model for PADF development.</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>Modified IRIS sensor node for improved omni-directional RSSI measurements.</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>IEEE 802.15.4 and ZigBee stack architectures.</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>Estimated position of emitter lies in the intersections of the corresponding circles graph.</td>
<td>60</td>
</tr>
<tr>
<td>11</td>
<td>Model of a sensor network used to localize a non-cooperative emitter.</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>Receiving sensors nodes in configuration I surrounding the transmitting node.</td>
<td>68</td>
</tr>
<tr>
<td>13</td>
<td>Median and rms plot for configuration I: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).</td>
<td>70</td>
</tr>
</tbody>
</table>
14 Robustness values for each mote in configuration I: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4). .......................................................... 72

15 Receiving sensors nodes in configuration I surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale). .......................................................... 73

16 Robustness for configuration I. .............................................. 74

17 Receiving sensors nodes in configuration II surrounding the transmitting node. .............................................................. 74

18 Median and rms plots for configuration II: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4). .......................................................... 76

19 Robustness values for each mote in configuration II: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4). .......................................................... 77

20 Receiving sensors nodes in configuration II surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale). .......................................................... 78

21 Robustness for configuration II. .............................................. 79

22 Robustness for configurations I and II in the same axis. .......... 79

23 Receiving sensors nodes in configuration III surrounding the transmitting node. .............................................................. 80

24 Median and rms plots for configuration III: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4). .......................................................... 82

25 Robustness values for each mote in configuration III: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4). .................................................. 83

26 Receiving sensors nodes in configuration III surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale). .......................................................... 84

27 Robustness for configuration III. .............................................. 85
Receiving sensors nodes in configuration IV surrounding the transmitting node.

Median and rms plots for configuration IV: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).

Robustness values for each mote in configuration IV: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).

Receiving sensors nodes in configuration IV surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale).

Robustness for configuration IV.

Robustness for configurations III and IV in the same axis.

Closed-loop to obtain true position-adaptive estimator.

Metric and error for configurations I and II.

Metric and error for configurations III and IV.

Discrete position-adaptive configurations without obstruction for case one at IDCAST.

Discrete position-adaptive configurations with obstruction for case one at IDCAST.

An intersection of circles with radii equal to RSS measured at individual nodes for configuration 1.

Discrete position-adaptive configurations without obstruction for case two at IDCAST.

Discrete position-adaptive configurations with obstruction for case two at IDCAST.

Selected pictures of experimental apparatus developed for PADF investigations at IDCAST [7].
MATLAB GUI for PADF calculation [99], showing sample output depicting a PLE-based RF scattering error metric (blue curve) versus localization error (red curve) ............... 108

Collection of pictures that illustrate the experimental apparatus developed for PADF experiments at IDCAST. ................. 109

Metric (unitless) and error (meters) for multiple configurations (corresponds to case one configurations 1-5 from Figure 37). ............... 111

Metric (unitless) and error (meters) for multiple configurations (case one corresponds to configurations 1-5 from Figure 38). ............... 112

Metric (unitless) and error (meters) for multiple configurations (corresponds to case two configurations 1-6 from Figure 40). ............... 114

Metric (unitless) and error (meters) for multiple configurations (corresponds to case two configurations 1-6 from Figure 41). ............... 115

Functional block diagram model for PADF development with optional steps for improving robustness and accuracy ............... 116

MODE1 experimental PADF concept using MAVs with wireless sensor nodes. ......................... 117

MODE2 experimental PADF concept using MAVs with wireless sensor nodes. ......................... 118

Metric and error for circular trajectory. ......................... 120

Metric and error for 10-point zigzag. ......................... 121

General block diagram for extremum seeking control. ......................... 123

Source Seeking. ......................... 128

ESC closed-loop with potential function that can attain true position-adaptive sensor network reconfiguration autonomously. ......................... 133

Trajectory of the swarm sensors “without obstruction” using ESC for case one example one. ......................... 134

xvii
Potential functions “without obstruction” for case one example one.

Error metric “without obstruction” for case one example one that is minimized by ESC.

Localization error “without obstruction” for case one example one that is minimized by ESC.

Trajectory of the swarm sensors “without obstruction” using ESC for case one example two.

Potential functions “without obstruction” for case two example two.

Error metric “without obstruction” for case one example two that is minimized by ESC.

Localization error “without obstruction” for case one example two that is minimized by ESC.

Trajectory of the swarm sensors “without obstruction” using ESC for case one example two.

Potential functions “without obstruction” for case two example two.

Error metric for case one example two “without obstruction” that is minimized by ESC.

Localization error “without obstruction” for case one example two that is minimized by ESC.

Trajectory of the swarm sensors “with obstruction” using ESC for case one example one.

Potential functions “with obstruction” for case one example one.

Error metric “with obstruction” for case one example one that is minimized by ESC.

Localization error “with obstruction” for case one example one that is minimized by ESC.
73 Trajectory of the swarm sensors “with obstruction” using ESC for case one example two. ........................................ 145
74 Potential functions “with obstruction” for case one example two. ........ 145
75 Error metric “with obstruction” for case one example two that is minimized by ESC. ........................................ 146
76 Localization error “with obstruction” for case one example two that is minimized by ESC. .............................. 146
77 Trajectory of the swarm sensors “with obstruction” using ESC for case two example one. .................................. 148
78 Potential functions “with obstruction” for case two example one. ........ 148
79 Error metric “with obstruction” for case two example one is minimized by ESC. ........................................ 149
80 Localization error “with obstruction” for case two example one that is minimized by ESC. .............................. 149
81 Trajectory of the swarm sensors “with obstruction” using ESC for case two example two. .................................. 150
82 Potential functions “with obstruction” for case two example two. ....... 151
83 Error metric “with obstruction” for case two example two that is minimized by ESC. ........................................ 151
84 Localization error “with obstruction” for case two example two that is minimized by ESC. .............................. 152
85 Case one Monte Carlo study: (a) mean error metric; (b) standard deviation error metric; (c) median error metric. ........................................ 154
86 Case two Monte Carlo study: (a) mean error metric; (b) standard deviation error metric; (c) median error metric. ........................................ 155
87 Case one Monte Carlo study: (a) mean localization error; (b) standard deviation localization error; (c) median localization error. ........................................ 156
Case two Monte Carlo study: (a) mean localization error; (b) standard deviation localization error; (c) median localization error.

Contour plot for error metric, where the sign indicates the position of minimum metric value.

Contour plot for localization error, where the sign indicates the position of minimum error value.

Sample TREX 600 RC helicopter platform recently acquired by IDCAST to support PADF research via integration of onboard IRIS sensor motes.

Integration and demonstration of IRIS sensor nodes onto multiple RC helicopters.
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration I.</td>
<td>69</td>
</tr>
<tr>
<td>2</td>
<td>Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration I.</td>
<td>71</td>
</tr>
<tr>
<td>3</td>
<td>Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration II.</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration II.</td>
<td>78</td>
</tr>
<tr>
<td>5</td>
<td>Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration III.</td>
<td>81</td>
</tr>
<tr>
<td>6</td>
<td>Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration III.</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration IV.</td>
<td>86</td>
</tr>
<tr>
<td>8</td>
<td>Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration IV.</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>Position of sensors and emitter configurations for case one as shown in Figures 37 and 38.</td>
<td>97</td>
</tr>
<tr>
<td>10</td>
<td>Position of sensors and emitter configurations for case two as shown in Figures 40 and 41.</td>
<td>104</td>
</tr>
<tr>
<td>11</td>
<td>IRIS platform for multiple configurations with error and error metric (case one corresponds to configurations 1-5 from Figure 37).</td>
<td>110</td>
</tr>
<tr>
<td>12</td>
<td>IRIS platform for multiple configurations with error and error metric (case one corresponds to configurations 1-5 from Figure 38).</td>
<td>112</td>
</tr>
<tr>
<td>13</td>
<td>IRIS platform for multiple configurations with error and error metric (case two corresponds to configurations 1-6 from Figure 40).</td>
<td>113</td>
</tr>
<tr>
<td>14</td>
<td>IRIS platform for multiple configurations with error and error metric (case two corresponds to configurations 1-6 from Figure 41).</td>
<td>114</td>
</tr>
<tr>
<td>15</td>
<td>Monte Carlo study for 500 simulations, where ( \hat{e}(0) ) means error metric for initial simulation, and ( \hat{e}(50) ) means error metric for final simulations.</td>
<td>152</td>
</tr>
<tr>
<td>16</td>
<td>Monte Carlo study for 500 simulations, where ( e(0) ) means localization error for initial simulation, and ( e(50) ) means localization error for final simulations.</td>
<td>153</td>
</tr>
</tbody>
</table>
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSNs</td>
<td>Wireless Sensor Networks</td>
</tr>
<tr>
<td>PADF</td>
<td>Position-Adaptive Direction Finding</td>
</tr>
<tr>
<td>LSE</td>
<td>Least-Square Estimation</td>
</tr>
<tr>
<td>ESC</td>
<td>Extremum Seeking Control</td>
</tr>
<tr>
<td>FCC</td>
<td>Federal Communications Commissions</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>MS</td>
<td>Mobile Station</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RSSI</td>
<td>Received Signal Strength Indicator</td>
</tr>
<tr>
<td>ToF</td>
<td>Time of Arrival</td>
</tr>
<tr>
<td>DoA</td>
<td>Direction of Arrival</td>
</tr>
<tr>
<td>ISM</td>
<td>Industrial, Scientific and Medical</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>AFRL</td>
<td>Air Force Research Laboratory</td>
</tr>
<tr>
<td>IDCAST</td>
<td>Integration Lab at Institute for Development Commercialization of Advanced Sensor Technology</td>
</tr>
<tr>
<td>RSSI</td>
<td>Received Signal Strength Indicator</td>
</tr>
<tr>
<td>UAVs</td>
<td>Unmanned Aerial Vehicles</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MEMS</td>
<td>Microelectromechanical Systems</td>
</tr>
<tr>
<td>LoS</td>
<td>Line-of-Sight</td>
</tr>
<tr>
<td>LS</td>
<td>Least-Squares</td>
</tr>
<tr>
<td>DF</td>
<td>Direction-Finding</td>
</tr>
<tr>
<td>ToF</td>
<td>Time of Flight</td>
</tr>
<tr>
<td>MAVs</td>
<td>Micro-Air Vehicles</td>
</tr>
<tr>
<td>DMPC</td>
<td>Decentralized Model Predictive Control</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>MAVSeN</td>
<td>Micro-Aerial Vehicle/Wireless Sensor Network</td>
</tr>
<tr>
<td>SUAV</td>
<td>Small Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>PLE</td>
<td>Path Loss Exponent</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>CCA</td>
<td>Clear Channel Assessment</td>
</tr>
<tr>
<td>SFD</td>
<td>Start Frame Delimiter</td>
</tr>
<tr>
<td>LLC</td>
<td>Logical Link Control</td>
</tr>
<tr>
<td>MAC</td>
<td>Medium Access Control</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>OLSR</td>
<td>Optimized Link State Routing</td>
</tr>
<tr>
<td>AODV</td>
<td>Ad-hoc On-demand Distance Vector</td>
</tr>
<tr>
<td>TCP</td>
<td>Transport Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>Universal Data Protocol</td>
</tr>
<tr>
<td>SMP</td>
<td>Sensor Management Protocol</td>
</tr>
<tr>
<td>TADAP</td>
<td>Task Assignment and Data Advertisement Protocol</td>
</tr>
<tr>
<td>SQDDP</td>
<td>Sensor Query and Data Dissemination Protocol</td>
</tr>
</tbody>
</table>

xxiv
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESN</td>
<td>Emergency Sensor Networks</td>
</tr>
<tr>
<td>TDoA</td>
<td>Time Difference of Arrival</td>
</tr>
<tr>
<td>AoA</td>
<td>Angle of Arrival</td>
</tr>
<tr>
<td>WPANs</td>
<td>Wireless Personal Area Networks</td>
</tr>
<tr>
<td>CSMA/CA</td>
<td>Carrier Sense Multiple Access with Collision Avoidance</td>
</tr>
<tr>
<td>UTRC</td>
<td>United Technologies Research Center</td>
</tr>
<tr>
<td>SPSA</td>
<td>Simultaneous Perturbation Stochastic Approximation</td>
</tr>
<tr>
<td>ABS</td>
<td>Antilock Braking System</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1 Motivation and Significance

Wireless Sensor Networks (WSNs) are a significant technology attracting considerable research interest. Recent advances in wireless communications and electronics have enabled the development of low-cost, low-power and multi-functional sensors that are small in size and communicate over short distances. Cheap, smart sensors, networked through wireless links and deployed in large numbers, provide unprecedented opportunities for monitoring and controlling homes, cities, and the environment. In addition, networked sensors have a broad spectrum of applications in the defense area, generating new capabilities for reconnaissance and surveillance as well as other tactical applications [1]. Wireless communication has enjoyed explosive growth over the past decade. As demands for increased capacity and quality grow, improved methods for harnessing the multi-path wireless channel must be developed. One area driving research is public demand for improved safety using wireless communications. Position location technologies have traditionally been of interest to the military and intelligence communities. In addition to the traditional applications of position location technologies, two new commercial applications have spurred research in this area: the use of position location technology for vehicle MAYDAY services
and position location for mobile phones dialing 911 under the Enhanced-911 standard. The development of position location technologies for mobile phones has been prompted by the Federal Communications Commission’s (FCC) requirement that wireless service providers, such as cellular, specialized mobile radio, and the newly allocated personal communications services, be able to determine the location of mobile phone users dialing 911.

WSNs are often being used in different environments to perform various monitoring tasks such as search, rescue, disaster relief, target tracking and a number of tasks in smart environments. In many such tasks, node localization is inherently one of the system parameters. Node localization is required to report the origin of events, assist group querying of sensors and to route and to answer questions on the network coverage. So, one of the fundamental challenges in WSNs is node localization. Localization is a key aspect of such networks, since the knowledge of a sensor’s location is critical in order to process information originating from this sensor, to actuate responses to the environment, etc. Localization in WSNs is becoming more important, because many applications need to locate the source of incoming measurement as precisely as possible.

For a large number of applications in home automation, the service system requires precisely sensing a user’s location by certain sensors. Moreover, the system sometimes requires recognizing the time and the weather for making decisions. On the other hand, the users always hope to be served correctly and suitably by the service system in the house. For satisfying the user’s demands, one of the key success factors is to accurately estimate the user’s location. It is considered a challenge to automatically serve a mobile user in the house [2]. Indoor localization cannot be carried out effectively by the well-known Global Positioning System (GPS), which is subject to be blockaded in the urban
and indoor environments [3, 4, 5, 6]. Thus, in recent years, WSNs are popularly used to locate a mobile object in the indoor environment.

In recent years, location estimation in WSNs has raised a lot of interest from researchers. The idea of having sensing data without any information of the location does not make sense. Existing techniques, such as GPS, are usually inappropriate in large scale networks, due to the increase in the cost and size of the nodes. For that reason a variety of location estimation techniques has been proposed for WSNs. The success of outdoor positioning and applications based on the GPS provides an incentive to the research and development of indoor positioning systems. It has been developed over several decades and it relies on a constellation of satellites. Ranges to several satellites are used in multilateration procedure to infer the position of the receiver. This localization scheme requires hardware that is both expensive and consumes significant power. As a result, indoor positioning systems require alternative means to detect the Mobile Station (MS) location without relying on the direct signal from GPS satellites. Infrared, Radio Frequency (RF), and ultra sound signals are major technologies used for indoor positioning systems. Unlike outdoor areas, the indoor environment imposes different challenges on location discovery due to the dense multi-path effect and building material dependent propagation effect. Thus, an in-depth understanding of indoor radio propagation for positioning is crucial for efficient design and deployment. Several localization techniques have been proposed in the literature. The GPS provides global coverage but specialized equipment is needed, which is energy inefficient and mostly ineligible for WSNs, except for the case of limited number of sensors with extended battery capabilities. Furthermore, GPS is not capable of operating indoors, because of the large attenuation introduced by buildings’ walls and ceilings; therefore it cannot comprise a ubiquitous localization method.
The position of the sensor nodes is very important for several reasons:

- Measurements without a location where they were gathered are generally useless.
- Full covered sensor networks enable energy aware geographic routing.
- Self-configuration and self-organization are key mechanisms for robustness and can easily be supported by position information.
- In many applications the position itself is the information of interest.

Localization is important to make the sensor data valuable when the position of the mote is in doubt. In some applications, location information is critical for the interpretation of sensor data. We can use location information of each node to: display the topology of the network, compare the readings from different sensors and analyze the distribution of these readings according to their locations, resolve the specific positions of some nodes if their readings seem special or unreasonable, and to observe the collaborations of each mote in the network.

Indoor localization in the literature is based on various techniques, ranging from simple Received Signal Strength Indicator (RSSI) to the more demanding Time of Arrival (ToA) or Direction of Arrival (DoA) of the incoming signals. Indoor localization is one of the newly emerging technologies having potential for numerous applications in the commercial and public safety fields. RF localization has gained prominence because of its potential for supporting various position based applications. RF localization based on RSSI uses the strength of a received signal from a target by listening to infer the range, which is subsequently used for position estimation. RF localization deals with position computation of a wireless device. Research in this field has gained prominence because of its potential
applications like E-911, navigation, WSNs, asset tracking, patient monitoring, and many more. RSSI-based estimations are based on the well-known radio propagation path-loss model. This technique has become the most inexpensive and simple because RSSI or RF signals can be measured during normal transmissions between nodes.

Location determination systems use different technologies and techniques to find the position of a stationary or moving object. However, due to the large number of wireless communications and their availability in most areas, many of the location estimation systems make use of RF based sensor technology. With this in mind, the employment of IEEE 802.11 infrastructure for location determination recently attracted much attention. The increasing popularity of these networks and their availability in most office and home environments are advantageous. These networks work in Industrial, Scientific and Medical (ISM) frequency band that is a hassle free frequency band and is used by a variety of devices. An important advantage of using IEEE 802.11 technology for location determination is that the resulted system can work on top of the existing Wireless Local Area Network (WLAN) infrastructure and the need to customize hardware is eliminated. Every location determination system that uses IEEE 802.11 network infrastructure has two major characteristics. First, it maps the RSSI to locations and stores them in a database; and second it searches the database to find the best match. At the present time, a number of methods have been suggested and deployed in existing systems. However, the accuracy of these systems are affected by a variety of parameters, most importantly, the RF propagation environment. Since these systems rely heavily on RF propagation, a thorough evaluation of the factors that affect RF propagation can be useful.

Position estimation refers to a process used to obtain location information of a MS with respect to a set of reference positions within a pre-defined space. A system deployed
to determine or estimate the location of an entity is called a position location system or positioning system. A wireless indoor positioning system refers to a wireless network infrastructure that provides indoor location information to any requesting end user. Position-Adaptive radar concepts have been formulated and investigated at the Air Force Research Laboratory (AFRL) within the past few years. Adopting a position-adaptive approach to the design of distributed radar systems shows potential for the development of future radar systems that function under new and challenging environments that contain large clutter discrete and require co-functionality within multi-signal RF environments. This research focuses on the modeling and implementation of distributed, mobile radar sensor networks. In particular, we worked on the problem of Position-Adaptive Direction Finding (PADF), to determine the location of a non-collaborative transmitter, possibly hidden within a structure, by using a team of cooperative intelligent sensor networks. Our purpose is to further develop and refine position-adaptive RF sensing techniques based on the measurement and estimation of RF scattering metrics. Topics planned for this entrepreneurial research project are focused on the investigation, analysis/simulation, and development of real-time multi-model (i.e., complex multipath) environments scattering decompositions for PADF geometries.

This new PADF research approach incorporates RF scattering metrics into multiplatform objective functions and, based on the measurements at RF Systems Integration Lab at Institute for Development Commercialization of Advanced Sensor Technology (IDCAST) [7], shows potential for sensing and localization performance gains in embedded environments. As discussed above, this research will make a difference to the Air Force. This approach to RF sensing provides the Air Force with low-cost approaches to designing
processing-intensive intelligent multi-Unmanned Aerial Vehicles (UAVs) systems that provide new RF sensing capabilities in challenging environments.

1.2 History and Literature Overview

In recent years there has been growing interest in ad-hoc and WSNs for a variety of applications. The development of microelectromechanical systems (MEMS) technology as well as the advancement in digital electronics and wireless communications has made it possible to design small size, low-cost energy efficient sensor nodes that could be deployed in different environments for a variety of applications [8].

The relatively new WSNs are currently subjected to a deep development in order to exploit the benefits of these simple, cheap, and robust networks. In the last two decades, many applications of wireless communications have become very successful. The starting point was over a decades ago when, according to their characteristics, a wide range of application possibilities, beyond the military domain, appeared. For example, nowadays cellular phones play a very important role in our daily lives. Mainly, commercial domains such industrial controls, automotive security, surveillance controls are the current research topics within the WSNs. All these applications have appeared under the commercial name of ZigBee, based on the IEEE 802.15.4 standard [9]. The current standard, agrees in terms of the scalability and low-cost requirements needed for the developing of the previously mentioned applications. There is extensive research in the development of new algorithms for data aggregation [10], ad-hoc routing [11, 12, 13], and distributed signal processing in the context of WSNs [14, 15]. As the algorithms and protocols for WSNs are developed, they must be supported by a low-power, efficient and flexible hardware platform.
In [1], sensor networks are defined as those networks that use multiple sensors to collect and process information on certain entities of interests. Under this definition, some network systems with history of several decades can be regarded as some kinds of sensor networks. For example, the radar networks for air traffic control or military purposes; the national power grid with many sensors at different positions of the network, etc. These systems were developed and deployed much earlier than the term sensor networks, which became popular in recent years [1]. Akyildiz and Sankarasubramaniam et al. [8] presents a global overview of the sensor networks. It describes the protocol stack as being divided in a physical, data link, network, transport and application layers; and gives characteristics and issues of each of them. A description of ad-hoc localization system is given in [16].

Localization in WSNs has recently become an active area of research. Much work has been done on localization algorithms in WSNs; however, most of them are vulnerable to location attacks, node malfunction and excessive environment noise since they do not examine the rationality of the information they get before using them in the location estimation. First studies to use the capabilities of sensor networks for localization have been done by the group of Estrin and Heidemann [17] and Savvides et al. [18]. Nevertheless, they only concentrated on an inherent localization problem in sensor networks. Numerous studies have been performed since then for a civil use [19]. Researchers have pointed out the influence of noise on the localization process [20], and the importance of various system parameters on the accuracy and efficiency of the localization process [21], but there is no consensus of a single best algorithm for localization in sensor networks. This indeed depends on the environment and the specifications of the used motes. The use of mobile nodes has also been studied [22], should it be used for beacons or for nodes. A very interesting algorithm that simulates mobility has been developed [23], and presents good
results, even if it needs a large amount of nodes. Indeed, localization is still a very active and largely open field of research [8]. Review of wireless network localization techniques can be found in [24, 25, 26]. The focus of these references is on localization techniques in cellular network and WLAN environments and on the signal processing aspect of localization techniques. Sensor networks vary significantly from traditional cellular networks and WLAN, in that sensor nodes are assumed to be small, inexpensive, cooperative and deployed in large quantity. These features of sensor networks present unique challenges and opportunities for WSNs localization. In [27] described some general signal processing tools that are useful in cooperative WSNs localization algorithms.

Hightower and Borriello In [28], presents a survey and taxonomy of location systems for mobile computing applications. Comparing different techniques and their performance can be very difficult because many different applications solves many different problems, and each technique can differ from others in many ways. The physical media used for the location, the context where the position is needed, power requirements, infrastructure and resolution of results will all change from application to application. Hightower and Borriello attempts to provide the means to make an easier comparison of implementations and applications. Another survey is presented in [29], where Guolin et al. provide a survey of positioning designs for wireless communications technology, Guolin et al. look at localization both in cellular technology and in wireless networks like WSNs. The survey provides a categorization of the different technologies ranging from GPS to sensor network techniques. Practical results concerning two key issues in the deployment of localization service in WSNs are presented in [30], where Anlauf and Sunbul discusses both the selection of appropriate sensors for acquiring the data needed to preform localization and the
actual localization algorithm. The authors look at the use of motes or motes-like technologies and points out that for most applications there will not be any map of the monitored area. Motes are tiny devices equipped with a sensor, an onboard computer and wireless communication technology. Motes with both long and short distance antennas were tested to determine the ranges of the transmissions [31].

In order to defend against location attacks, some secure localization schemes have been proposed recently, among which are location verification [32], distance verification [33, 34], distance-bounding [35], RSSI measurements [36, 37, 38, 39] uses hidden and mobile base stations to localize and verify location estimate. Location estimation in WSNs has raised a lot of interest from researchers [40, 41, 42]. Existing techniques, such as GPS, are, usually, inappropriate in large scale networks, due to the increase in the cost and size of the nodes. For that reason a variety of location estimation techniques has been proposed for wireless sensor networks [40]. Localization algorithms play an important role in enhancing the utility of data collected, by enabling sensors to determine the location from which each data packet is obtained [43, 44, 45, 24].

In [46] a technique that uses range measurements between pairs of nodes and the known coordinates of the anchor nodes in wireless ad-hoc networks, to estimate the position of every node is proposed. The method will first establish the position of the nodes close to one of the anchor nodes, and then use this information to estimate positions of nodes further away from the anchor nodes. Another similar distributed algorithm for localization in WSNs is described in [47]. This algorithm is based on the iterative propagation of information through the network, and measurements, with limited accuracy, of the distances between pairs of nodes used to derive position estimates. [47] mention the standard methods for acquiring these measurements, but the technique described is independent of the
method used for distance measurement. The authors also mention that different techniques offer different tradeoffs between accuracy, complexity cost, etc. Errors in the measurements can come from multi-path interference, Line-of-Sight (LoS) obstructions and more.

In [48] the DoA estimation [49] is reviewed, and the authors also considers a Least-Squares (LS) method applied to the DoA bearing crossings to perform the source localization. In [50] localization based on the use of a single mobile beacon is presented. The technique uses RSSI to estimate the distances between nodes and the beacon. This gives the method an accuracy of a few meters. The beacon will broadcast its own position and any node receiving this messages can infer that it is positioned in a certain area with a certain probability.

The RSSI is measured for every received beacon message and will place a constraint on the possible position of a node. This technique can be implemented both centralized and distributed, and scales well, both in terms of number of nodes and density of nodes. The trajectory of the mobile beacon will affect the results and must be considered. A localization technique that uses the signal strength from two static beacons is presented in [51]. Interference and fading in the signals are handled by using the average over several measurements within a certain time-frame. The two beacons are placed in two non-opposite corners of a rectangle that encloses all devices. Problems occurring both with static and mobile nodes are considered and the best performance in both scenarios is obtained in a fast fading environment. The minimum error is in both cases where the variance in error as a function of the size of the time-frame is said to be very small. Beacons are also used in [52], where an efficient localization algorithm using four mobile beacons is described. The rectangle, formed by the four beacon nodes, will have the sensor node in the center, and localization is performed by moving the beacon nodes towards the node. The
node can then estimate its own position by position information from the four beacons. In this method the beacon nodes will perform the distance measurements in several iterations and provide the node with this information. In [53], an approximate maximum likelihood strategy and the received strength of signals from nodes with well-known positions is used to estimate the position of the receiving node. Beacons are used as the nodes with known positions. A parameter called normalized collinearity is described. This new parameter is used as a metric for the nodes to evaluate how the different beacon nodes are aligned relative to the nodes. Based on the metric, the best aligned beacon nodes can be chosen by the nodes. Ecolocation in [54], makes use of distance constraints based on RSSI and develop sequences to determine the relative position of transmitter with respect to known reference stations.

Many localization techniques have been developed in literature based on electromagnetic signal properties which enable in inferring the location information. RF techniques have been alternatively suggested in order to locate wireless units. RF position location systems are classified into two broad categories: Direction-Finding (DF) and Range-Based (RB) systems [55]. DF systems utilize antenna arrays and DoA estimation techniques in order to locate the MS, and are mainly used in areas with limited clutter [56, 57, 58, 59, 60, 61, 62]. On the other hand, RB systems measure the distance between the MS and a number of Base Stations (BS), and then the MS’s position arises as the intersection point of the corresponding curves. The range of the MS is calculated using either ToA or RSSI measurements [57, 59, 63, 64].

RADAR is a user tracking system [65]. It uses empirical as well as mathematical models to determine the signal strength. The RF based RADAR [65] is a Range-Based indoor localization system that measures RSSI at all positions in the entire building and records
the RSSI into a database during the calibration phase. In the localization phase, the location and orientation of a user are determined by finding the best match of a set of RSS measurements in the database. Calamari uses RSSI and acoustic Time of Flight (ToF) to estimate the distance [66]. It uses an audible frequency to reduce the complexity of the system. SpotON is a RSSI based ad-hoc localization system for WSNs [67]. It can be used for relative and absolute position determination. In this system, all the nodes need to be calibrated before being used. The paper “RSS-based location estimation with unknown path-loss model” [68] dynamically estimates the distance-power gradient; parameter of the radio propagation path-loss model. It adapts automatically to the environment, thus eliminating the need for extensive channel measurements. This leads to a more accurate conversion of RSSI to distance.

The estimation of link distance on unknown indoor environments has been a topic of research in recent years since it is a foundation for many different applications. As a result, many studies have explored new algorithms that exploit the RSSI collected in ad-hoc measurement campaigns to calibrate the localization system. In particular, authors in [69] evaluate the performance of their localization system based on fingerprinting technique, deploying 20 anchors in about 1700m². They show that their system can reach the 50th and 80th percentile of the localization error with 0.9 meters and 1.6 meters, respectively. MoteTrack [69] is a decentralized location tracking system based on RF. The location of each blind node is computed using a RSS signature from the anchor nodes to a database of signatures. This database is stored at the anchor nodes themselves. MoteTrack focuses on a robust, decentralized implementation [69] as the most localization systems need a base station for data processing. The majority of the nodes take over only a limited role in the localization process to support the case that one or more base stations are down.
Cricket [70] is decentralized and uses RF and ultrasound to determine the location of a blind node. The Cricket [70] location support system uses ultrasound instead for indoor localization. Anchor nodes broadcast beacon messages, together with the RF message, it will transmit a ultrasonic pulse. A noteworthy survey is [71] by K. Langendoen. This survey describes three algorithms: ad-hoc positioning [72], N-hop multilateration [18] and robust positioning [47]. These algorithms are fully distributed algorithms; they require no central processing node and are designed towards multi-hop localization.

A number of indoor location tracking systems have been proposed in the literature, based on RF signals, ultrasound, infrared and some combinations of these. RF based positioning system are widely studied [65, 69]. Given a model of radio signal propagation in a building or other environment, RSSI can be used to estimate the distance between the transmitter and receiver [65, 73]. The existence of radio connectivity and the attenuation of radio signal with distance are attractive properties that could potentially be exploited to estimate the positions of small wireless devices featuring low-power radios. RSSI, a standard feature in most radios, has attracted a lot of attention in the recent literature for obvious reasons. RSSI eliminates the need for additional hardware in small wireless devices, and exhibits favourable properties with respect to power consumption, size and cost.

As RSSI values vary because of mobility and obstructions, the RSSI variation can be used in the investigation of localization and mobility [65, 74]. Much of the research experiment results in this area assume RSSI as a parameter in their localization algorithms and confirm that the variation in RSSI values can be used for localization and mobility [37, 75, 76]. Reference [77] describes a system where the signal strength of a mobiles transmitter is measured on a statistical basis by a set of BSs. From a priori information of the corresponding contours, the most probable location of the mobile is determined. Using
elementary geometric considerations, and a LS estimate to smooth measurement errors, [78] develops a trilateration method based on RF travel time measurements between the vehicle and fixed sensors located at the edges of a square. This method was further pursued by [79] for channel allocation in cellular networks. Refined trilateration techniques for road environments, using time delays between reception of a mobiles transmission at different nodes as input data, are investigated by [80]. Signal strength measurements are used in [81] to assign a mobile to a certain base station coverage zone. A channel allocation algorithm is introduced which uses this information as a basic ingredient.

Mobile sensor networks and their application in sensing, localization, and control have gained significant interest with the development of sensor networks and modern control algorithms,[82, 83, 85, 86, 87] . Cooperative control algorithms that maximize the probability of detection are given in [83] [87]. Control and coordination for groups of autonomous vehicles performing distributed sensing was presented in [85]. With the development of Micro-Air Vehicles (MAVs), cooperative control and sensing gained attention for applications such as military, weather forecast, chemical sensing, etc. A problem of cooperative path planning for a fleet of UAVs in uncertain environments was presented in [82]. Similarly, a robust decentralized model predictive control for a team of aerial vehicles is given in [86] and for situations in urban terrain or urban battlefield in [88].

Li and Cassandras give a tutorial-style overview of sensor networks from a systems and control theory perspective, providing a comprehensive background [83] on sensor networks. Complementing this overview, they later presented a distributed coverage control scheme for cooperating mobile sensor networks [87]. They developed a gradient based algorithm requiring local information at each sensor and maximizing the joint detection probabilities of random events. Cortes et al. presented control and coordination algorithms
for groups of autonomous vehicle networks performing distributed sensing tasks where each vehicle plays the role of a mobile tunable sensor [85]. Bellingham et al. address the problem of cooperative path planning for a fleet of UAVs in uncertain or adverse environments, by modeling for the probability of UAV loss [82]. Similarly, Richards and How implemented a robust Decentralized Model Predictive Control (DMPC) for a team of cooperating UAVs [86]. Using this DMPC each vehicle plans only for its own actions, but still allowing the UAVs to communicate relevant plan data to ensure those decisions are consistent across the team. In a simple case like collision avoidance, DMPC guaranteed constraint satisfaction and offered significant computation improvement, compared to an equivalent centralized algorithm, for only a small degradation in performance, such as UAV flight time. Chandler et al. researched the development of cooperative rendezvous and cooperative target classification agents in a hierarchical distributed control system for unmanned aerospace vehicles [82]. For cooperative target classification he developed templates, followed optimal trajectories, and assigned adjacent vehicles to view at complementary aspect angles; hence, he combined these to maximize the probability of correct target classification over various aspect angles. Singh and Fuller developed a receding-horizon optimal control scheme for autonomous trajectory generation and flight control of an UAV in an urban terrain [88]. Because environments may be dynamic, or the vehicles need to change dynamics mid-flight due to sensor or actuator failure, they proposed a Model Predictive Control (MPC) scheme that navigates a vehicle with nonlinear dynamics through a vector of known way-points to a goal, and manages constraints for missions that will require vehicles with increased autonomy in dangerous situations and with tight maneuvering and operational capability e.g., missions in urban environments. Continuance and improvements of research in these various areas are the motivation behind the creation of
the Louisiana Tech University Micro-Aerial Vehicle/Wireless Sensor Network (MAVSeN) Laboratory.

Recent developments in communications and RF technology have enabled system concept formulations and designs for low-cost radar systems using state-of-the-art software radio modules. One of the major benefits of using these RF communications products is the potential for generating frequency-agile waveforms that are re-programmable in real-time and potentially adapt to a scattering environment. In addition, recent simulation results [89] indicate that this type of system enables the development and implementation of multi-function RF systems that yield good performance within embedded shared-spectrum environments. Previous work in this area position-adaptive radar [90, 91] has addressed topics such as short-range monostatic position-adaptive radar design for mini-UAV helicopter platforms as well as electromagnetic phenomenology and associated signal processing for position-adaptive radar.

Mitra formulates and investigates a concept of position-adaptive radar systems in [91, 92, 93, 94, 95, 96]. Adopting a position-adaptive approach to the design of distributed radar systems shows potential for the development of future radar systems that function under new and challenging environments that contain large clutter discrete and require co-functionality within multi-signal RF environments [93]. Sean Young, Mitra and Ordóñez et al. present the designing position adaptive radar systems for multi-platform of Small Unmanned Aerial Vehicle (SUAV) [97]. Specifically, they investigate notional control geometries and trajectories for multi-platform SUAV applications by integrating additional electromagnetic scattering-based metrics within more generic overall objective functions for multi-SUAV controls systems. They show that the formulation of these new categories of objective functions lead to realizations of multi-platform SUAV trajectories that position
adaptively converge to a set of RF leakage points. They show that an embedded scatterer (i.e., a metal cylinder) can be imaged by applying radar processing techniques derived for sparse apertures [97].

Mitra and Ordóñez et al. present several approaches to analysis and design of sensor agnostic networks based on assuming canonically structured architectures that are comprised of low-cost wireless sensor node technologies. They provide a logical development that motivates the potential adaptation of distributed low-cost sensor networks that leverage state-of-the-art wireless technologies and are specifically designed with pre-determined hooks, or facets, in place that allow for quick and efficient sensor swaps between cost-low RF sensors, EO sensors, and Chem/Bio sensors [98]. Ordóñez, et al. provide a result on a novel multi-platform RF emitter localization technique denoted as position-adaptive RF direction finding [99]. These results indicate that this position-adaptive approach shows potential for accurate emitter localization in challenging embedded multi-path environments (i.e., urban environments)[99]. As part of a general introductory discussion on PADF techniques, Gates, Al Issa, and Ordóñez, et al. [100, 101] provide a summary of the recent results on PADF, and give a discussion on the underlying and enabling concepts that provide potential enhancements in RF localization accuracy in challenging environments. Also, they provide an outline of recent results that incorporate sample approaches to real-time multi-platform data pruning as part of a discussion on potential approaches to refining a basic PADF technique in order to integrate and perform distributed self-sensitivity and self-consistency analysis as part of a PADF technique with distributed robotic/intelligent features [100, 101]. Some of these position-adaptive radar designs are based on state-of-the-art ultra-wideband impulse radar technology. An integral part of this “smart sensor”
concept is to interface the onboard radar system with the onboard autonomous control system so that the mini-UAV helicopter platform has the capability to “positionally-adapt” to characteristics of a “leakage signal” that is measured by the radar.

1.3 Contributions

WSNs are being used in a variety of ways – from reconnaissance and detection in the military to biomedical applications and a wide variety of commercial endeavors. In recent years, position based services have increased. Thus, recent developments in communications and RF technology have enabled system concept formulations and designs for low-cost radar systems using state-of-the-art software radio modules. WSNs are thus popularly used to locate a mobile object in an indoor environment. Some physical features are widely discussed to solve indoor localization in WSNs.

In WSNs, the distance between sensor nodes can be estimated by measuring RSSI. The wireless RSSI based localization techniques have attracted significant research interest for their simplicity. The RSSI based localization techniques can be divided into two categories: the distance estimation based and the RSSI profiling based techniques. The Path Loss Exponent (PLE) is a key parameter in the distance estimation based localization algorithms, where distance is estimated from the RSSI.

The main goal of this dissertation is to study and focus on a multi-platform RF emitter localization technique denoted as Position-Adaptive RF Direction Finding (PADF), with the specific goal of addressing the need and potential for developing real-time multi-platform control metrics based on RF-scattering. These control metrics should enable the development and advancement of RF sensing capabilities in complex multi-path environments. In particular, we worked on the problem of PADF, to determine the location of a
non-collaborative transmitter, possibly hidden within a structure, by using a team of cooperative intelligent sensor networks. We provide a summary of recent experimental results in localization of a non-cooperative sensor node using static and mobile sensor networks. We designed and integrated commercial state-of-the-art wireless sensor node/mote technologies onto PADF-platforms. In this study we used IRIS wireless sensor nodes. In order to localize the transmitter, we use the RSSI data to approximate distance from the transmitter to the revolving receivers. We have experimentally demonstrated that RSSI can effectively be used for localization of hidden emitters in WSNs environment. Localization accuracy is improved as the number of sensor nodes and their spatial distribution/spread increases. We provide an algorithm for on-line estimation of the PLE that is used in modeling the distance based on RSSI measurements. The emitter position estimation is calculated based on surrounding sensor’s RSSI values using LSE. We developed a set of experiments that read sensor data, convert the data into approximate distance values based on approximated PLE values, and estimate the position of the emitter using the LSE method. The PADF has been tested on a number of different configurations in the laboratory via the design and implementation of four stationary sensors as receivers and one hidden sensor as a transmitter during the localization phase. Several experiments were performed to test optimal RSSI values. Among these experiments some were tested with obstruction and some without obstruction. We placed the emitter inside of a partially sealed enclosure that was covered on three sides with aluminum foil to shield the radio signal. This creates a more challenging environment to detect the hidden emitter, similar to situations where there is an RF leakage point or leakage areas. We have developed a Graphical User Interface (GUI) using MATLAB that allows for easy implementation of the PADF concepts.
In this dissertation we first provide an overview of the basics of cooperative WSNs, hardware specifications, sensor node protocol and WSNs applications in Section 1.5. Furthermore, we give a brief introduction of the localization principle, range-free and range-based and localization techniques in Section 1.6.

In Chapters 2 and 3, we propose a novel sensor-node based approach to emitter localization that is PADF. We demonstrate that this approach shows potential for accurate emitter localization in challenging embedded multi-path environments (i.e., urban environments). PADF is based on the formulation and investigation of path-loss based RF scattering metrics (i.e., estimation of distributed PLE) that is measured and estimated across multiple platforms in order to enable the robotic/intelligent position-adaptation (or self-adjustment) of the location of each platform.

The dissertation is organized into seven chapters. Chapter 2, Section 2 introduces the concept of PADF, and hardware modules using cooperative WSN technologies. We present in Chapter 3 the position estimation based signal strength methodology, emitter localization estimation, and RSSI model on IRIS sensor nodes. Then, we introduce the experimental procedure for sensitivity and robustness issues. Then we explain the PADF error metric we proposed. Moreover, the PADF has been tested on a number of different configurations in the laboratory via the design and implementation of four IRIS wireless sensor nodes as receivers and one hidden sensor as a transmitter during the localization phase. The robustness of detecting the transmitter’s position is initiated by getting the RSSI data through experiments and then data manipulation in MATLAB will determine the robustness of each node and ultimately that of each configuration. The parameters that are used in the functions are the median values of RSSI and rms values. From the result it is determined which
configurations possess high robustness. High values obtained from the robustness function indicate high robustness, while low values indicate lower robustness.

In Chapter 4, we present the details that were involved to set up the experimental testbed, and the lab setup.

Chapter 5 contains the results from the aforementioned experiments, and illustrates these approaches to MAV/sensor platform integration and indoor flight test simulation geometries presently under investigation within the AFRL MAV Lab facility at the AFRL Air Vehicles Directorate.

In Chapter 6, we present the experimental performance analysis on the application aspect. We apply Extremum Seeking Control (ESC) schemes by using the swarm seeking problem, where the goal is to design a control law for each individual sensor that can minimize the error metric by adapting the sensor positions in real-time, thereby minimizing the unknown estimation error. As a result we achieved source seeking and collision avoidance of the entire group of the sensor positions.

Chapter 7 presents the conclusions for the dissertation and closes out the dissertation by imparting some future work thoughts.

1.4 Outline

PADF configuration concepts are developed for purposes of investigating potential refinements in consistency, sensitivity, and robustness via the design and implementation of four stationary sensors as receivers and one hidden sensor as a transmitter during the localization phase. We first review the most relevant WSNs and localization algorithms. Then, we survey the existing body of literature on these fields. Furthermore, the results on PADF are present. The PADF tests are conducted with an emitter inside a partially metal-sealed
enclosure where varied hardware simulations are performed across several sets of discrete position-adaptive sensor node configurations/geometries. We provide a summary of recent experimental results in localization of a non-cooperative sensor node using static and mobile sensor networks. The PADF has been tested on a number of different configurations in the laboratory via the design and implementation of four stationary sensors as receivers and one hidden sensor as a transmitter during the localization phase. The resulting analysis and simulation has been applied towards the development of intelligent and cooperative real-time networked PADF processing mechanisms with features that enable localization performance enhancements such as multi-platform self-data-pruning capabilities that integrate on-going and recent results in the area of consistency and sensitivity analysis of experimental PADF solutions. We used a combination of experimental data and MATLAB simulations to gain better insight into the difficult problems of quality of estimation, data reliability, and sensitivity. Then we performed analysis of these results grounded on solid nonlinear analysis tools and our observations. The end result of our analysis was the development of a theoretically sound control algorithm that is able to adapt in real-time the positions of the sensor network in order to provide the best possible estimates.

A brief review of WSNs and hardware specifications, is presented in Section 1.5. The localization principles and localization techniques are presented in Section 1.6. The remainder of this dissertation is organized as follows: in Chapter 2, a novel PADF is proposed. We have introduced the concept of PADF applied to localization of a hidden, non-cooperative emitter where static and mobile nodes cooperate in the localization mission. We introduce as well the hardware and software modules using cooperative WSNs technologies. Then, in Chapter 3, we present the position estimation method based on signal
strength using LSE and the robustness and the sensitivity issues. Furthermore, in Chapter 4, we discuss the experimental and lab setup. In this chapter, we study two different cases, where each case has different configurations. Chapter 5 contains the results from the aforementioned experiments, and illustrates these approaches to MAV/sensor platform integration and indoor flight test simulation geometries presently under investigation within the AFRL MAV Lab facility at the AFRL Air Vehicles Directorate. In Chapter 6, we present the experimental performance analysis on the application aspect. We design a control system using extremum seeking methods that can minimize the error metric by adapting the sensor positions in real-time, thereby minimizing the unknown estimation error, by using the swarm seeking problem, where we design a control law for each individual sensor to achieve source seeking and collision avoidance of the entire group of the sensor positions.

Chapter 7, concludes the dissertation and the future work. Following this chapter are an appendices.

1.5 Wireless Sensor Networks

Mobile communications and wireless networking technology has seen a thriving development in recent years. There has been growing interest in ad-hoc and WSNs for a variety of indoor applications. A WSN [8, 102, 103, 104, 105, 106, 107, 108] is a network of many small sensing and communicating devices called sensor nodes or motes, which can sense the environment and communicate the information gathered from the monitored field (e.g., an area or volume) through wireless links [109]. Typically, a WSN contains one node, the base station, that connects the network to a more capable computer as illustrated in Figure 1, and probably to a network of general purpose computers through it. Sensors attached to these nodes allow them to sense various phenomena within the environment. The typical
The purpose of a sensor network is to collect data via sensing interfaces and propagate those data to the central computer, allowing easy monitoring of an environment [110].

Sensor networks consist of a large number of independent autonomous intelligent sensor nodes communicating in a cooperative manner over a wireless channel to achieve a common goal. This goal varies between simple data collection to complex measurements [111]. Each node has a CPU, a power supply and a radio transceiver for communication. Interconnection between nodes is achieved via the transceiver. WSN is a kind of ad-hoc network densely populated with small, low cost, resource constrained, wireless-communication enabled and immobile sensor nodes. Each node, or sensor, will have a very limited functionality; this will normally consist of some type of short range wireless

Figure 1: Example of a wireless sensor network.
communication capabilities and some form of sensing, or actuator, equipment. The sensing or monitoring of, for example, temperature, humidity, etc., constitutes one of the two main tasks of each sensor. The other main task is packet forwarding using the equipped wireless technology. Whichever way data is transmitted, the network must provide a way of transporting information from different sensors to wherever this information is needed.

A WSN is composed of a number of low-cost, tiny sensor nodes that are capable of sensing, data processing, short range wireless communication, and even actuation [112, 113]. The data collected is routed to special nodes, called sink nodes, also called BS, in a multi-hop basis, as shown in Figure 2. Then, typically, the sink node sends data to the user the Internet or satellite, through a gateway, though, depending on the distance between the user and the network, a gateway might not be needed (local monitoring). Due to the small size of the wireless sensor nodes, they can be deployed very easily. For example, a large number of small sensors can be spread by aircrafts to a battlefield or a wild forest. The architecture of a typical sensor network is shown in Figure 2, where the sensor nodes can send their data to the sink node, and it is possible to remotely operate the sensor network through the Internet. The wireless link can allow the sensor nodes to communicate and forward information. In some systems, for example the Crossbow [223], the battery life can be more than one year. Moreover, some systems like Crossbow provide programming interface and customizable sensing units depending on the specific applications.

Sensor nodes have many limitations regarding computational power, but actually their only task is to report their sensing values to a single base station or sink node which is, in most cases, connected to a PC or notebook with a lot of computational resources. This way we can put the intelligence into the software application. Large amounts of cheap motes
construct the network by collaborating with each other, and the data sensed by each mote congregate to the base station. Indeed, a formal definition of WSN is given in [8].

1.5.1 Hardware Specifications

Typically, a sensor node consists of four basic components: sensing unit, processing unit, a radio transceiver (transmitter and receiver), and a power (usually, a battery) unit. Figure 3 shows the basic architecture of a sensor node. The following is a brief explanation on what each of these components does. The analog signals that are measured by the sensors are digitized via an ADC and in turn fed into the processing unit. The processing unit and its associated storage manage the procedures that make the sensor node carry out its assigned sensing and collaboration tasks. The radio transceiver connects the node with the network and serves as the communication medium of the node. The power unit is the most important component of the sensor mote because it implicitly determines the lifetime
of the entire network. Due to size limitations AA batteries or quartz cells are used as the primary sources of power.

Figure 3: Basic architecture of a wireless sensor node [115].

Recent advancements allow for the current generation of sensor nodes to become even smaller and cheaper [8]. Consequently, nodes have reduced memory and processing capacities. Battery is, typically, limited. Moreover, due to a short transmission range (caused by restrained transmission power), nodes can only communicate with its local neighbors. The transmission power for each mote is often low to avoid interfacing with each other. Thus the range of communication or range of connectivity is limited to some extent. In a large WSN, the readings from each mote may have to arrive at the base station via several steps.
In sensor network, finding and maintaining a high efficient multi-hop routing algorithm is very important to guarantee the high reliability and low energy consuming [115].

1.5.2 Sensor Node Protocol

In this section, we specify the task of each layer of the stack and the most common protocols coupled with each layer as shown in Figure 4 [8].

![Figure 4: Sensor node protocol stack](image-url)
1. Physical Layer: The physical layer is responsible for carrier frequency generation, frequency selection, signal detection, modulation and data encryption. Techniques such as Ultra Wideband, Impulse Radio and Pulse Position modulation have been used to reduce complexity and energy requirements, whilst improving reliability and reducing path loss effects and shadowing. The physical layer, commonly referred to as Layer 1, interfaces the protocol stack to the network communications hardware to transmit and receive messages one bit or symbol at a time. Individual physical layer functions include Clear Channel Assessment (CCA), data frame synchronization, and encryption. The CCA mechanism senses the physical channel medium to determine if another transmission is currently in progress in order to prevent communications collisions. Data frame synchronization occurs by appending a preamble and Start Frame Delimiter (SFD) sequence of bits to align the radio timing among the network radios. Finally, once the radios are synchronized, the physical layer receives analog symbols from the medium and converts them to digital bits for further processing in higher protocol layers. WSN IEEE 802.15.4 standard radio platforms also offer additional physical layer functions to provide encryption and auto-acknowledgement messages.

2. Data Link Layer: The data link layer, commonly referred to as the link layer or Layer 2, interfaces with both the network and the physical layer. The link layer is comprised of two separate functions: the Logical Link Control (LLC) module and the Medium Access Control (MAC) layer. The data link layer is responsible for medium access, error control, multiplexing of data streams and data frame detection. It ensures reliable point to point and point to multi-hop connections in the network. Due
to the network constraints conventional MAC protocols are not suited to sensor networks. The LLC provides the standard interface between the link and network layer by encapsulating the message segment from the network layer with additional header information that ensures proper sequencing on the distant end. The LLC also assembles and disassembles MAC frames which include data, control information, Cyclic Redundancy Check (CRC) calculation, source address, destination address, and the intermediate network device addresses. For WSNs seeking to conserve energy, the MAC layer offers significant opportunities to reduce energy and prolong network lifetime since it controls the radio. The MAC layer provides a fair contention mechanism to share access to the medium, data authenticity/privacy security options, and a reliable delivery system using a frame exchange protocol. The frame exchange protocol supports successful delivery by employing acknowledgement frames and data integrity CRC checks.

3. Network Layer: The network layer, commonly referred to as Layer 3, provides the end-to-end routing protocol to connect the sending and receiving stations. Unlike the single-link responsibility of the link layer, the network layer protocol determines the entire delivery sequence of links a packet will traverse across the network on its way to the destination. The network layer is responsible for routing information through the sensor network i.e., finding the most efficient path for the packet to travel on its way to a destination. Based upon the network source or destination address of a packet, the network layer uses algorithmic strategies to optimize the packet routing across various links traversing a network. Wireless ad hoc networks can proactively send control messages to maintain an accurate routing table or reactively determine a route on demand. WSNs can use either method, but proactive route discovery
techniques like Optimized Link State Routing (OLSR) require additional energy for periodic control messages and memory table allocation for routes that may never be utilized [116]. The Ad-hoc On-demand Distance Vector (AODV) algorithm [117] enables dynamic, self-starting, multi-hop routing between communicating nodes. Since the protocol establishes routes upon request, nodes do not need to maintain routes to inactive nodes.

4. Transport Layer: The transport layer, or Layer 4, offers the ability to regulate traffic flow through the network to the distant end and provide data delivery reliability measures. The transport layer divides large, upper layer application data into sequential segments for delivery. The Transport Layer is needed when the sensor network intends to be accessed through the internet. However, no scheme has been devised to fully address this issue. Upon receipt of these segments, the destination’s transport layer reorders and reassembles them into data packages for forwarding up to the application layer. Depending on the application requirements, the transport layer can either send out message segments without any reliability mechanisms, or the transport layer can provide flow control, high-level packet error checking, data acknowledgement, and network congestion control. The two standard transport protocol options are Transport Control Protocol (TCP) and Universal Data Protocol (UDP). TCP offers the reliable delivery mechanisms, and UDP maintains simplicity for applications that do not require the control overhead. Unlike the wired transport layer, the wireless transport layer cannot make the assumption that unacknowledged packets are indicative of a congested network. Standard TCP mechanisms which significantly reduce the rate of flow due to perceived congestion will actually cause the system harm in wireless systems.
5. **Application Layer**: The applications layer is responsible for presenting all required information to the application and propagating requests from the application layer down to the lower layers. Some preliminary protocols in this area include Sensor Management Protocol (SMP), Task Assignment and Data Advertisement Protocol (TADAP), Sensor Query and Data Dissemination Protocol (SQDDP). Network applications provide the purpose for the entire protocol stack. The application layer offers network services directly to the user for electronic mail, file transfers, virtual terminal, and file servers. Network applications use application layer protocols to define the format and order of message exchanges between processes operating on separate networked computers [118, 119, 120].

### 1.5.3 Wireless Sensor Networks Applications

WSNs are quickly gaining popularity due to the fact that they are potentially low cost solutions to a variety of real-world challenges [8]. Advancement in wireless sensing technologies has encouraged in developing a model for indoor object-tracking application using low power IEEE 802.15.4 compliant radios. Stimulus used in this model is RSSI. This application is not only limited to object tracking in an indoor environment. Concierge services enable users to become aware of nearest supporting facilities. For example, smart home applications such as multimedia appliances that forward multimedia stream to the nearest video screen can be achieved with a home positioning system [121]. In the field of robotics, a robot can navigate by itself using the assistance of indoor positioning system [122]. Granularity of location information is most important in location determination applications. For example, to locate a book in a library requires coarse grained granularity whereas locating a user requires coarse grained granularity.
Sensor nodes are deployed in areas of interest to cooperatively monitor physical or environmental conditions, such as sound, vibration, temperature, pressure, motion, electromagnetic disturbance, etc. There are several applications for WSNs, which involve monitoring and tracking. WSNs have shown a wide range popularity for both military applications [8, 123, 124, 125, 126, 127, 128] and civil applications, including industrial process monitoring and control [129, 130, 131, 132, 133], structure health monitoring [134, 135, 136, 137, 138], environmental monitoring that can be embedded into a hospital building to track and monitor patients and all medical resources, disaster relief [139], home automation [140], habitat and environment monitoring [141, 142, 143, 144, 145, 146], health-care applications [167, 148, 149, 150, 151, 152], home automation [153, 154, 155, 156, 157], vehicle networks and intelligent transportation systems [158, 159, 160, 161, 162] sensor networks have been used for vehicle traffic monitoring and control for some time, biomedical applications [163, 164, 165, 166, 167], environmental sensing applications [168] that can be used for animal tracking, forest surveillance, flood detection, and weather forecasting, and a variety of commercial endeavors [43], and infrastructure security [1].

Some of the most important application areas of sensor networks include hurricane areas, volcano areas, forest fire areas, etc. and these networks are usually known as Emergency Sensor Networks (ESN) [169]. In most of the applications related to ESN the data collected will become useless unless the location information can be discerned from it. The above mentioned applications of WSNs are just a few examples. WSN technology is said to be one of the most important technologies in this century, and a lot of applications of using WSNs were proposed and some demo systems were developed in the works mentioned above. Similar to many technological developments. Today, WSNs have become a key technology for different types of smart environments. An intense research effort is
currently underway to enable the application of WSNs for a wide range of industrial problems. Wireless networks are of particular importance when a large number of sensor nodes have to be deployed, and/or in hazardous situations [170]. WSNs can also be employed in hazardous environment or disasters zones, that may present danger to people or simply be unreachable by other means. A self calibrating sensor network could be placed out covering a large area, and with very little user input quickly start collecting data, transporting and presenting this data to a user at a safe distance.

1.6 What is Localization?

In recent years, location estimation in WSNs has raised a lot of interest from researchers, and many efforts have been made in developing algorithms and methodologies for building efficient localization mechanisms for indoor and outdoor environments. The idea of having sensing data without any information of the location does not make sense. Estimating the node localization is an extremely significant task in this field. Node localization is considered one of the complicated issues in WSNs. Localization is considered the fundamental service that is analogous to various operations such as cluster creation, routing, communication, coverage of network, etc. With the aid of cooperation [171], localization can be achieved with the help of sensor nodes itself without any involvement of humans. The critical issues in WSN operation is to determine the substantial locus of the sensor nodes as the position information will be deployed to find the locus at which the sensor reading originates as well as in any energy aware geographic routing.

Localization means to determine location of nodes in a network. With the support of some infrastructure, a node can determine its location in the network by extracting information received from the infrastructure; also, by making a node send signals periodically,
the infrastructure can calculate the location of the node. Localization can usually be described as the process of dynamically determining the position, or location, of someone or something, relative to someone or something else. For sensor networks this simply means locating a sensor node in a network. Localization refers to the ability to determine the position (relative or absolute) of a sensor node, with an acceptable accuracy. Sensor network localization algorithms estimate the locations of sensors with initially unknown location information by using knowledge of the absolute positions of a few sensors and inter-sensor measurements such as distance and bearing measurements. Sensors with known location information are called anchors and their locations can be obtained by using a GPS, or by installing anchors at points with known coordinates. In applications requiring a global coordinate system, these anchors will determine the location of the sensor network in the global coordinate system [172]. Most WSN applications require knowing or measuring locations of thousands of sensors accurately. For example, sensing data without knowing the sensor location is often meaningless [173]. Locations of sensor nodes are fundamental to providing location stamps, locating and tracking objects, forming clusters, facilitating routing, etc. However, a priori knowledge of locations is unavailable in large scale and ad-hoc deployments, and a pure GPS [174, 175] solution is viable only with costly GPS receivers and good satellite coverage. GPS techniques are usually inappropriate in large scale networks, due to the increase in the cost and size of the nodes. For that reason a variety of location estimation techniques have been proposed for WSNs [40]. These techniques have raised considerable interest from researchers [176, 177]. GPS is a typical localization system. In a general scenario, only a few anchor nodes are aware of their positions either through manual configuration or equipped with GPS receivers, and the others (called unknown nodes) have to estimate their positions by making use of the positions of anchors.
Though GPS is a popular location estimation system, it does not work indoors because it uses signals from satellites. This system requires LoS to some satellites, consumes additional energy and is too expensive to get integrated on hundreds of energy constrained sensor nodes. Using sensor networks instead of GPS makes indoor localization possible. Thus localization algorithms [44, 178, 202, 45] play a significant role in the domain of WSNs. The localization issue is important for WSNs since many applications such as environment monitoring, vehicle tracking and mapping depend on knowing the locations of sensor nodes where there is an uncertainty about some positioning. If the sensor network is used for monitoring the temperature in a building, it is likely that we can know the exact position of each node. On the contrary, if the sensor network is used for monitoring the temperature in a remote forest, nodes may be deployed from an airplane and the precise location of most sensors may be unknown. An effective localization algorithm can then use all the available information from the motes to compute all the positions. Localization in mobile indoor WSNs can be used for tracking objects or people. In some applications, location information is critical for the interpretation of sensor data. We can use location information of each node to display the topology of the network, compare the readings from different sensors, and analyze the distribution of these readings according to their locations, resolving the specific positions of some nodes if their readings seem special or unreasonable, and observe the collaborations of each mote in the network.

Since most applications depend on a successful localization, i.e., to compute their positions in some fixed coordinate system, it is of great importance to design efficient localization algorithms. In large scale ad-hoc networks, node localization can assist in routing [180, 181, 182]. In the smart kindergarten [183] node localization can be used to monitor the progress of the children by tracking their interaction with toys and also with each other.
It can also be used in hospital environments to keep track of equipment, patients, doctors and nurses [180].

### 1.6.1 Localization Principle

Localization in sensor networks is to identify sensor node’s position. For any WSN, the accuracy of its localization technique is highly desired. To estimate a node position, most recently proposed localization algorithms for wireless ad-hoc networks engage each node and is able to estimate the distance to all nodes within its radio range, and an appropriate set of location aware nodes called references or anchors are established within the network. There were many algorithms on localization which have been discussed in the past to provide localization information for every node. The existing algorithms for localization can be broadly classified into two basic categories: range-based technique, and range-free technique.

#### 1.6.2 Range-Free Vs Range-Based

Range-free approaches normally rely on proximity, near-far information or less accurate distance estimation to infer the locations of unknown nodes [184, 185, 72, 186, 187], and range-based approaches require accurate distance or angle measurements to locate the unknown nodes [65, 188, 189]. Both approaches must rely on the positions of anchor nodes and some measured/estimated parameters, and the localization accuracy depends on the accuracy of reference positions and relative parameters. The range-based approach uses absolute distance estimate or angle estimate, meaning that a node in a network can measure the distances from itself to the beacons. In contrast, range-free approach means that it is impossible for a node to measure the direct distances from itself to beacons. Only through
connectivity and proximity, a node can estimate its regions or areas where it stays. The range-based approach is precise while the range-free method is often inaccurate.

1.6.3 Localization Techniques

[184] gives an overview of the various localization techniques. There are many different techniques through which a position can be found relative to other known positions. These can be split up mainly into the categories of time based, signal strength based, and proximity to a reference point based techniques.

In range-based mechanisms, such as Cricket [190], Radar [65], APS [188] Pin-Point [191], TPS [192], etc, the location of a sensor node can be determined with the help of the distance. The distance between the receiver node and a reference point is determined by the time of flight of the communication signal. There are different ranging approaches to measure distance between two nodes. These approaches are ToA (time taken by the radio signal to propagate from one node to another), Time Difference of Arrival (TDoA) (time interval between the reception of a radio signal and an ultrasound emitted by a beacon), Angle of Arrival (AoA) (an estimate of the relative angles between nodes), and RSSI (an index of received signal power).

The range-free localization is being considered as a cost-effective alternative to range-based methods because of hardware limitation of deployment of WSN devices. Irregularity in transmission propagation as well as stringent restriction on the cost of hardware has rendered localization a challenge. The range-free localization is more capable of achieving higher localization accuracy without introducing any extra hardware in comparison to range-based technique of localization which depends on RSSI to calculate absolute point-to-point distance. The range-free localization technique deploys information related to
network topology as well as connectivity status for evaluating location. Low cost, no extra
hardware, little communication traffic as well as flexible precision in position estimation are
some of the advantageous features of range-free methods. Therefore the range-free tech-
nique is considered to be the most effective solution for the localization issues in WSNs
[193].

In [65, 72, 46], the RSSI, which is the easiest to obtain for the current sensors, is
utilized. Range-based techniques are highly accurate, but they are equipped with highly
expensive hardware and require a lot of computation. It increases the cost of the network
and is inefficient in terms of computations.

In range-free techniques, the position of the sensor node is identified on the basis infor-
mation transmitted by nearby anchor nodes or neighboring nodes, based on hop or on trian-
gulation basis. The various range-free techniques are APIT [194], chord selection approach
[195], three dimensional multilateration approach [196], SerLOC [197], centroid scheme
[198], etc. Many more techniques are discussed in [172, 199, 200, 170]. The range-free
techniques have an error in accuracy up to 10% of the communication range of individual
node [195], but these techniques are much cheaper than the range-based techniques.

Among them, RSSI based localization techniques have received considerable research
interest [201, 65, 51, 202, 203, 204, 205, 206]. Although RSSI based localization tech-
niques can only provide a coarse-grained location estimate, they are attractive because of
their simplicity. RSSI has become a standard feature in most wireless devices. RSSI is a
generic metric to measure the energy integral of a radio signal. As RSSI values vary be-
cause of mobility and obstructions, the RSSI variation can be used in the investigation of
localization and mobility [65].
In many ways, RSS is considered as an appealing modality for range estimation in WSNs, mostly because RSSI information can be obtained at almost no additional cost with each radio message sent and received [207, 208]. The major challenge is that radio signal strength is so unpredictable [209, 210, 211, 212, 213, 214], where reflecting and attenuating caused by objects in the environment can have much larger effects on RSS than distance, making it difficult to infer distance from RSS without a detailed model of the physical environment [207, 208, 215]. RSSI measures the power of the signal at the receiver and based on the known transmit power, the effective propagation loss can be calculated. Next, by using theoretical and empirical models we can translate this loss into a distance estimate. This method has been used mainly for RF signals. RSSI is a relatively cheap solution without any extra devices, as all sensor nodes are likely to have radios. The performance, however, is not as good as other ranging techniques due to the multi-path propagation of radio signals. In [207], the authors characterize the limits of a variety of approaches to indoor localization using signal strengths from 802.11 routers. They also suggest that adding additional hardware or altering the model of the environment is the only alternative to improve the localization performance.

Much of the research experiment results in this area assume RSSI as a parameter in their localization algorithms and confirm that the variation in RSSI values can be used for localization and mobility [75, 76]. RSSI is one of the few ranging techniques that does not require additional hardware. Many radio chips can measure the RSSI of a message. By applying a radio signal propagation model, one can calculate the distance from the RSSI.

According to path-loss models, the RSSI varies as a function of distance. This fact helps to determine the distance of an object transmitting a radio wave to a receiver from the signal strength that the receiver measures. Three receivers can measure the RSSI of
an object and calculate their distance to the transmitter. Therefore RSSI based localization techniques require no additional hardware, and are unlikely to significantly impact local power consumption, sensor size and thus cost. RSSI based localization techniques can be further divided into distance-estimation based [201, 51, 202, 203, 204] and RSSI-profiling based techniques [65, 205, 216, 217]. The RSSI method works on the principle that the energy of a signal decreases as it move farther away from its source. Therefore, if one knows the relationship between the decrease in energy and the distance traveled (i.e., a Path-Loss Equation (PLE)), the distance between the receiver and transmitter can be calculated from the signal strength observed at the receiver. One simple form of the PLE is:

\[ P(d) \ (dBm) = P_0 \ (dBm) - 10\alpha \log \left( \frac{d}{d_0} \right), \]  

(1)

where \( \alpha \) is a PLE, and \( P_0 \) power at distance at some distance \( d_0 \)(dBm).

Existing distance-estimation based techniques rely on a log-normal radio propagation model [218] to estimate inter-sensor distances from RSSI measurements. The PLE is a key parameter in the log-normal model. An accurate knowledge of the PLE is required in order to obtain an accurate estimate of the inter-sensor distance from the corresponding RSSI measurement. Existing techniques either consider the PLE is known a priori by assuming the WSN’s environment is free space, or obtain the PLE through extensive channel measurement and modeling by measuring both RSSI and distances in the same environment of WSNs prior to system deployment. However, the PLE is environment dependent. Even in the same environment, the channel characteristics may change considerably over a long period of time due to seasonal changes and weather changes [218]. Therefore it is an over-simplification to assume a free space environment. On the other hand priori channel measurement and modeling may not be possible for monitoring and surveillance
applications in hostile or inaccessible environments. More detailed information about how
to compute the RSSI or PLE equations will be discussed in chapter 3.
CHAPTER II

POSITION-ADAPTIVE DIRECTION FINDING (PADF)

Position-Adaptive radar concepts have been formulated [91, 92, 93, 94, 95, 96, 97] and investigated at the AFRL within the past few years. Adopting a position-adaptive approach to the design of distributed radar systems shows potential for the development of future radar systems. These radar systems will function under new and challenging environments that contain large clutter discretes and require co-functionality within multi-signal RF environments.

This research introduces a PADF method using mobile sensor networks, and presents the latest experimental results in the localization of a non-cooperative sensor node using static and mobile sensor networks [219]. PADF is based on the formulation and investigation of path-loss (i.e., PLE) based metrics that are measured and estimated across multiple platforms in order to robotically/intelligently adapt the location of each platform [99]. This approach shows potential for accurate emitter localization in challenging embedded multi-path environments (i.e., urban environments) [93]. Initial experimental test results (with ground-based distributed RF sensor nodes) conducted within the AFRL and RF Systems Integration Lab at IDCAST [7] indicate correlated trends between multi-platform RF-based control metrics formulated for this investigation and localization accuracy. These tests are conducted with an emitter inside a partially metal-sealed enclosure where variational
hardware simulations are performed across several sets of discrete position-adaptive sensor node configurations/geometries.

A sensor swarm adapts its position based on the sensed values, which in turn depend on the environment medium, and converges towards leakage points in order to detect a hidden RF source. By using the relationship between RSS values and the associated distance between sender and receiver, the transmitter position can be approximated. We developed a set of experiments that read sensor data, converted the data into approximate distance values based on approximated PLE values, and estimated the position of the emitter using the Least-Square Estimation (LSE) method. The algorithm is based on the LSE method described in detail in [220].

2.1 Problem Statement

This research focuses on the modeling and implementation of distributed, mobile radar sensor networks. In particular, we worked on the problem of PADF, to determine the location of a non-collaborative transmitter, possibly hidden within a structure, by using a team of cooperative intelligent sensor networks. Our purpose is to further develop and refine position-adaptive RF sensing techniques based on the measurement and estimation of RF scattering metrics. Topics planned for this entrepreneurial research project are focused on the investigation, analysis/simulation, and development of real-time multi-model (i.e., complex multipath) environments scattering decompositions for PADF geometries. The objective of this task is to conduct basic research activities and experiments in the PADF area, with the specific goal of addressing the need and potential for developing real-time
multi-platform control metrics based on RF-scattering. These control metrics should enable the development and advancement of RF sensing capabilities in complex multi-path environments.

### 2.2 Position-Adaptive Direction Finding Concepts

Position-adaptive radar concepts [92] are based on the development of “smart” radar sensors for SUAV platforms in an effort to detect and exploit “leakage signals” that, for example, propagate between two buildings or “leak through” penetrable surfaces such as walls or layers of the ground. These concepts are developed such that either a passive receiver or a monostatic radar receiver measures and processes leakage signals and then “self adapts” in position to establish LoS between a mini-UAV platform and a obscuration channel that propagates the leakage signal. This allows a mini-UAV platform to process signals in real-time while gathering intelligence information and locating objects-of-interest that may be embedded within a obscuration channel.

In Figure 5, one mini-UAV or several mini-UAVs surround a building-type structure to position-adaptively isolate leakage points, i.e., converge to the neighborhood of windows and doors based on real-time analysis of on-board radar returns. Under this concept, once this set of mini-UAVs has position-adaptively converged to the neighborhood of a set leakage points, the leakage signals are transmitted by three UAVs at three different frequencies and received by the forth UAV. This receive UAV processes signals over the entire bandwidth of the transmit tones from the three transmitting UAVs to analyze and characterize objects-of-interest that may be embedded within a building-type structure and can potentially map the internal structure of a building.
A short-range monostatic position-adaptive radar design for mini-UAV helicopter platforms that is compatible with the concept illustrated in Figure 5 is outlined in [90]. The radar portion of the design is based on state-of-the-art ultra-wideband impulse radar technology. An integral part of this “smart sensor” concept is to interface the onboard radar system with the onboard autonomous control system so that the mini-UAV helicopter platform has the capability to “positionally-adapt” to characteristics of a “leakage signal” that is measured by the radar.

The concept of PADF was introduced in [219, 93, 221, 222]. Figure 6 illustrates two states of a notional PADF geometry using three UAVs that are integrated with sensor motes programmed to function as RF receivers [219, 93].

Dimensions for the notional urban structure in Figure 6 are $10m \times 10m$ for each of the six $(2 \times 3)$ cells, $3m$ for the width of each of the channels, and $0.5m$ for each of the...
Figure 6: (a) Notional PADF initial state. (b) Notional PADF final state.
openings in the cells. A 2.4 GHz [114, 223] source is located in the center (5 m from each wall) of the front-center cell of this urban structure. UAV-1 is used as the (0, 0) location reference for these sample 2-D calculations. For both of the sample geometries in Figure 6, UAV-1 is assumed to be located 10 m in front of the center of the urban structure.

Each of the three notional UAVs are associated with a set of basic sensor mote radio parameters for receiver sensitivity and transmit power:

\[ Rx(\text{min}) \approx -100(dBm). \]  
\[ TX(\text{max}) \approx 20(dBm). \]

A theoretical signal analysis for this geometry is provided in [219] for purposes of formulating and illustrating this technique. As shown in Figure 7, the formulation and investigation of PLE based metrics that are measured across multiple platforms are intrinsic to this technique. For purposes of preliminary illustration of this concept, a series of approximate sample computations is considered in [219] that correspond to a notional initial state multi-platform geometry, as depicted in Figure 6a, and a notional final state multi-platform geometry, as depicted in Figure 6b. The following PLE based multi-platform control metric for initial state, denoted as \( PLE_{PADF0} \) was adopted in [219]:

\[
PLE_{PADF0} = \min \left( \frac{P_{r1}(dB) - P_{r2}(dB)}{10 \log_{10}(\frac{d_2}{d_1})}, \frac{P_{r1}(dB) - P_{r3}(dB)}{10 \log_{10}(\frac{d_3}{d_1})} \right),
\]

where \( P_{r1}, P_{r2}, \) and \( P_{r3} \), are based on a theoretical model for RSSI values at sensors 1, 2 and 3, respectively. Similarly, \( d_1, d_2, \) and \( d_3 \) represent theoretical distance estimates from
each sensor to an RF emitter and are also derived from a theoretical model developed for RSSI measurements at each mote.

Alternative multi-platform control metrics can be defined/investigated that estimate PLE values within the closed-loop of Figure 7 via a series of cooperative transmissions between the UAV platforms.

2.3 Hardware and Software Components

RF direction finding is performed using mobile sensors that cooperate in sensing missions, adapt position in real-time, and localize an unknown hidden RF source based on optimal detection algorithms. For instance, a possible implementation of this scenario would
use MAVs equipped with light-weight RF sensors. In this study we use IRIS wireless sen-
sor nodes [114, 223]. In order to localize the transmitter, we use RSSI as an approximation
of distance from the transmitter to the mobile receivers.

During the initial phases of this investigation, a series of basic sensor mote tests and
distributed software control/measurement functions were developed for the purpose of per-
forming PADF experiments and hardware simulations using IRIS networked wireless sen-
sor motes [114, 223]. In order to obtain more reliable RSSI measurements a portion of
this experiment design phase was allocated to customizing the aperture plane on the sensor
motes, as shown in Figure 8.

The following hardware and software modules were used in the PADF experiment with
wireless sensor nodes: MEMSIC IRIS wireless sensor nodes, MATLAB, and TinyOS [224]
programs. A brief description of these components along with a synopsis of how they are
integrated in the system is defined in the following sections.

2.3.1 IRIS Wireless Sensor Nodes

The IRIS 2.4 GHz mote by MEMSIC Technology, shown in Figure 8, is a module
used for enabling low-power WSNs. The IRIS Mote features several new capabilities
that enhance the overall functionality of MEMSIC’s wireless sensor networking products
[114, 223]. It has a 2.4 to 2.48 GHz globally compatible ISM band; a 250 kbps high data
rate radio (outdoor LoS tests yielded ranges as far as 500 meters between nodes without
amplification); an IEEE 802.15.4 compliant RF transceiver; and a direct sequence spread
spectrum radio (resistant to RF interference/provides inherent data security).

The IRIS supported by MoteWorks WSN platform for reliable, ad-hoc mesh network-
ing, and plug and play with MEMSIC’s sensor boards, data acquisition boards, gateways,
MoteWorks enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks. MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-the-air-programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and configuration [114, 223].

The IRIS platform consists of the low-power microcontroller Atmega 1281 which runs MoteWorks from its internal flash memory, the FLASH memory AT45DB041 and the radio transceiver AT86RF230 from Atmel. ATmega1281 is a low-power 8-bit microcontroller. It is part of Atmels well known AVR series. The microcontroller runs with up to 16 MIPS throughput at 16 MHz and there is 128-K Byte flash program memory and 8-K Byte SRAM included.

As shown in Figure 8, we added a ground antenna plane that shields the rest of the node electronics from interfering with RSSI measurements. This creates more omnidirectional measurements, which is very important for integration with ground vehicles and MAVs.

These nodes are used as a basis for the localization of the RF transmitter. We used four nodes in the development stage to test the functionality of PADF. Their primary function is not only to detect the RF source (which is also an IRIS mote), but also transmit the RSSI values between neighboring nodes and the transmitter. These RSSI values will be instrumental in determining the position of the transmitter using the LSE.

2.3.2 Tiny Operating System

The WSNs architecture we used here includes both a hardware platform and an operating system designed specifically to address the needs of WSNs. TinyOS has been designed
to meet the requirements of sensor networks. TinyOS is an operating system for WSNs. Its innovative architecture was motivated by the specialized hardware characteristics it is running on. Sensor networks are potentially constructed of thousands of tiny nodes, which communicate among each other through radio channels, infrared or various techniques. TinyOS [224] is a component based operating system designed to run in resource constrained wireless devices. It provides highly efficient communication primitives and fine-grained concurrency mechanisms to application and protocol developers. A key concept in TinyOS is the use of event based programming in conjunction with a highly efficient component model. TinyOS enables system-wide optimization by providing a tight coupling between hardware and software, as well as flexible mechanisms for building application specific modules. There are four broad requirements that motivate the design of TinyOS. First of all, motes have very limited resources. Because of their small size, it is hard to include powerful hardware (8-K Bytes of program memory, 512 Bytes of RAM is typical). Low power consumption is important, most of them are powered by battery or ambient
energy, so efficient energy management is essential. A node has to handle a lot of events like sampling sensors, local data processing, routing data, etc. Many of these events require real-time response. That is why reactive concurrency needs an effective concurrency management.

Flexibility is a also a challenge. There are many different hardware platforms which are developed very fast. It would be an asset if the operating system could be adapted in an accurate way. The extreme constraints of these devices makes the use of legacy systems impractical. TinyOS has been designed to run on a generalized architecture where a single CPU is shared between application and protocol processing [225]. In the past years, TinyOS has become a de facto standard in academic research on WSNs.

The current version of TinyOS has already been ported to over a dozen platforms and numerous sensor boards. Over 500 research groups and companies are using TinyOS on the Berkeley/Crossbow motes for simulation and development of various algorithms and protocols. Multiple groups actively contribute code to the sourceforge [226] site and working together to establish standard, interoperable network services. TinyOS started as a collaboration between the University of Berkeley, California, in co-operation with Intel Research and Crossbow Technology, and has since grown to be an international consortium, the TinyOS Alliance. TinyOS has several important features such as component-based architecture, a simple event-based concurrency model and split-phase operations that influence the development phases and techniques when writing application code. TinyOS is strictly tied to the NesC “Network embedded system C” programming language having native support for TinyOSs features. NesC is a dialect of the C programming language optimized for the memory limitations of sensor networks. Its supplementary tools are mainly in the form of Java and shell script front-ends. The software components are connected with each
other by using interfaces. TinyOS’s component library includes network protocols, distributed services, sensor drivers, and data acquisition tools.

TinyOS’s event based communication is encapsulated in a component model that allows state-machine based components to be efficiently composed together into application-specific configurations. The TinyOS communication system is based on the active messages communication paradigm taken from high-performance computing environments. When data is received from the network, message-specific handlers are automatically invoked. TinyOS includes a special-purpose memory model which allows a fixed set of message buffers to be pre-allocated and shared amongst several applications.

Finally, we can conclude that sensor networks are opening up many new areas of study and facilitate many new applications. The underlying hardware for those sensor nodes is developed very fast. The big challenge is to shrink the physical dimensions while maintaining or even increasing the performance of those nodes. TinyOS is an open-source operating system designed to address the special difficulties in such an operating area. Its component based architecture enables rapid innovations, while reducing the code size to a minimum. The event-driven execution model allows a fin-grained power management and follows the concurrent nature of sensor networks. TinyOS is not a finished system. It is developed by a wide community and continues to evolve.

2.3.3 IEEE 802.14.2 Technology

The IEEE 802.15.4 wireless technology is a short-range communication system intended to provide applications with relaxed throughput and latency requirements in Wireless Personal Area Networks (WPANs). The key features of the IEEE 802.15.4 wireless
technology are low complexity, low cost, low power consumption, low data rate transmis-
sions, to be supported by cheap (either fixed or moving) devices.

- The IEEE 802.15.4 [228] protocol has been adopted as a communication standard
  for low data rate, low power consumption and low cost WSNs. This protocol is quite
  flexible for a wide range of applications if appropriate tuning of its parameters is
  carried out. Most of the mote platforms use this standard for communication be-
  tween the motes. Rene, IRIS, TelosB, SunSPOT and LOTUS are all IEEE 802.15.4
  compliant motes. This standard intends to define the fundamentals of network layers
  used mainly in WPANs where the point is to establish a ubiquitous communica-
  tion cost-effectively. Important features include real-time suitability by reservation
  of guaranteed time slots, collision avoidance through Carrier Sense Multiple Access
  with Collision Avoidance (CSMA/CA) and integrated support for secure communi-
  cations. Devices also include power management functions such as link quality and
  energy detection [229]. This standard defines MAC and the physical layer protocol
  for low-power devices. Figure 9 shows the various layers of the ZigBee wireless
  technology architecture the relationship of the IEEE 802.15.4 standard to the ZigBee
  Alliance MAC layer protocol model. These layers facilitate the features that make
  ZigBee very attractive: low cost, very low power consumption, reliable data trans-
  fer, and easy implementation. Using the IEEE 802.15.4 specifications, the Alliance
  focuses on the design issues related to the network, security and applications layers.

- The ZigBee technology is based on the IEEE 802.15.4 standard and guarantees theo-
  retically a transmission data rate equal to 250 kpbs in a wireless communication link.
  ZigBee [230] is a standard for a suite of high level communication for low power
  and low data rate radio communications. ZigBee is initiated and maintained by the
Figure 9: IEEE 802.15.4 and ZigBee stack architectures.
ZigBee Alliance - a large consortium of industry players. The ZigBee Alliance added the network layer and the network framework in the application layer based on the IEEE 802.15.4 MAC and physical layer. The ZigBee Alliance was formed because its members felt that existing standard technologies were not applicable to ultra-low power application scenarios. ZigBee networks can be configured to operate in a variety of different ways to suit the application and environment. Supported topologies include peer to peer (ad-hoc) and star configuration. The ZigBee communication is effective for the short range information exchange. The key feature is low power and no infrastructure required. The typical application areas of ZigBee include: smart energy monitoring; health care monitoring; remote control; building automation and home automation. However, most past studies on localization systems carried out the performance evaluation on systems based on 802.11 for WLANs, and there has been insufficient investigation of using ZigBee or IEEE 802.15.4.
CHAPTER III

MODELING AND LOCALIZATION METHODOLOGY

3.1 Position Estimation Based Signal Strength

In the following, a position estimation method based on signal strength using LSE is introduced.

Consider two sensor nodes at locations $T_i = (x_i, y_i)$, $T_j = (x_j, y_j)$, and an emitter in the $x$-$y$ plane located at $C = (x, y)$. It follows that the distances between the emitter $C$ and the sensors are given by:

\[
\begin{align*}
    r_i &= \sqrt{(x - x_i)^2 + (y - y_i)^2} \\
    r_j &= \sqrt{(x - x_j)^2 + (y - y_j)^2}
\end{align*}
\]

(5)

We assume the sensors provide enough information (i.e., that two or more sensors are used) such that, using RSSI measurements, $r_i$ and $r_j$ can be calculated. The problem is to determine the location of the emitter $C$, or equivalently, to solve two equations for two unknowns in Equation (5). The solution to the system of equations is the intersections of the corresponding circles or radii $r_i$ and $r_j$, as shown in Figure 10. However, three receiving nodes are required to precisely determine the position of the emitter.
In the case of $N$ surrounding sensors, the problem is to determine the location of the emitter $C$ or to solve the following system of equations

$$r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2},$$  \hspace{1cm} (6)$$

where $i = 1, 2, \cdots N$.

Knowing that sensor measurements are noisy and that the system of Equation (6) has larger number of equations than unknowns, we use LS to determine the location estimate $(\hat{x}, \hat{y})$.

The PLE is a key parameter in the distance estimation based localization algorithms, where distance is estimated from the RSS. The PLE depends on the medium of RF signal propagation and is not known a priori.
Consider the model given in Figure 11, consisting of a network of sensors $S_i$ ($i=1,2,3,4$) and a non-cooperative emitter $S_0$. Distance between nodes $i$ and $j$ is denoted as $d_{ij}$. The signal strength received from node $i$ at node $j$ is denoted as $P_{ij}$. Note that in Figure 10 this notation is simplified by letting $d_{0j} = d_j$

Figure 11: Model of a sensor network used to localize a non-cooperative emitter.

**Assumption 1**. A symmetric wireless signal propagation model is considered, i.e., $P_{ij} = P_{ji}$ for $i,j \in \{0, 1, 2, 3, 4\}$.

**Assumption 2**. Location of sensor nodes $S_i$ ($i=1,2,3,4$) is known. Location of emitter $S_0$ is unknown.
The RSS model is given by [231]

\[ P_{ij} (dBm) = P_0 (dBm) - 10\alpha \log (d_{ij}/d_0), \]  

(7)

where \( \hat{\alpha} \) is a PLE, \( P_{ij}(dBm) \) is power received at node \( j \) from node \( i \) in dBm and \( P_0(dBm) \) is a reference power received at some distance \( d_0 \). Note that the calibration method requires measurements of \( P_0 \) and \( d_0 \).

### 3.1.1 Emitter Localization Estimation

The PLE estimation is given by the following optimization problem, where it is required to find a minimum of the quadratic cost function \( f (\hat{\alpha}) \)

\[ f (\hat{\alpha}) = \sum_{i,j=1}^{4} \left( P_{ij} (dBm) - P_0 (dBm) + 10\hat{\alpha} \log (d_{ij}/d_0) \right)^2. \]  

(8)

The optimal value of the \( \hat{\alpha} \) is given by

\[ \frac{\partial f}{\partial \hat{\alpha}} = 2 \sum_{i,j=1}^{4} 10 \log (d_{ij}/d_0) \left( P_{ij} (dBm) - P_0 (dBm) + 10\hat{\alpha} \log (d_{ij}/d_0) \right) = 0. \]  

(9)

Therefore, the solution is given by

\[ \hat{\alpha} = \frac{\sum_{i,j=1}^{4} \log (d_{ij}/d_0) \left( P_0 (dBm) - P_{ij} (dBm) \right)}{10 \sum_{i,j=1}^{4} \left( \log (d_{ij}/d_0) \right)^2}. \]  

(10)

This PLE estimate is a maximum likelihood estimator value. Note also that calculation of PLE given by Equation (10) requires prior calibration between sensors \( S_i \) (\( i=1,2,3,4 \)). In the case that there are more than four nodes, the Equation (10) still applies requiring just larger number of sensor nodes to be considered.
3.1.2 RSSI Model on IRIS Sensor Nodes

If distances $r_i$ and $r_j$ in Equation (5) are not known, they can be calculated based on RSSI data. The RSSI model that is used is given by

$$P_{0i} = -10\hat{\alpha} \log(d_i) - K,$$

(11)

where $\hat{\alpha}$ is the estimated PLE, $d$ is the distance between transmitter and receiver, and $K$ is the conversion constant of RSSI to $dBm$ (in the case of IRIS motes, $K = -91$). Therefore, the distance estimate is given by

$$\hat{d}_i = 10^{- \frac{P_{0i} + K}{10\alpha}}.$$

(12)

The algorithm attempts to guess whether the hidden transmitter is behind an obstruction. To detect an obstruction, the algorithm first computes the average distance between motes,

$$\bar{T} = \frac{1}{N} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \| T_i - T_j \|.$$

(13)

If $\hat{d}_i > \gamma \bar{T}$, then we modify the distance estimate to be

$$\hat{d}_i = \gamma 10^{- \frac{P_{0i} + K}{10\alpha}}.$$

(14)

The value of $\gamma$ is found experimentally. It is the average observed attenuation of RSSI due to obstructions. In this dissertation, we select $\gamma = 0.75$.

Let $\hat{C} = (\hat{x}, \hat{y})$ be the estimated position of the emitter. The matrix of receiver geometry (known $x - y$ coordinates of receivers) is
\[ A = \begin{bmatrix}
    h_{x(2,1)} & h_{y(2,1)} \\
    \vdots & \vdots \\
    h_{x(N,1)} & h_{y(N,1)} \\
    \vdots & \vdots \\
    h_{x(N,N-1)} & h_{y(N,N-1)}
\end{bmatrix}, \quad (15) \]

where \( N \) is the number of receiver, and \( h_{x(i,j)} = 2(x_j-x_i), h_{y(i,j)} = 2(y_j-y_i) \).

The matrix of RSSI measurements is

\[ B = \begin{bmatrix}
    b_{2,1} \\
    \vdots \\
    b_{N-1} \\
    \vdots \\
    b_{N,N-1}
\end{bmatrix}, \quad (16) \]

where \( b_{i,j} = r_i^2 - r_j^2 + x_j^2 - x_i^2 + y_j^2 - y_i^2 \).

Specifically, the following set is formulated the for purpose of localization of non-cooperative emitters:

\[ PL_{i,j} = \text{median}\left(10^{\frac{\text{RSSI}_{i,j}}{10}}\right), \quad (17) \]

\[ PL_{dB_{i,j}} = 10\log_{10}\left(PL_{i,j}\right), \quad (18) \]

\[ PLE_{i,j} = \log_{10}\left(10^{\frac{PL_{dB_{i,j}}}{10}}\right), \quad (19) \]

where \( i, j \) denote cooperative transmissions between sensor \( i \) and sensor \( j \).

Then, the following modified LS approach can be adopted to produce the position estimate, where the distance terms in the \( A \)-matrix are weighted by the inverse of vector cross...
PLE terms that are estimated from a set of cooperative RSSI measurements between four platforms.

\[ CPLE = \begin{bmatrix} PLE_{1,2}, PLE_{1,3}, PLE_{1,4}, PLE_{2,3}, PLE_{2,4}, PLE_{3,4} \end{bmatrix}, \]  
\[ (20) \]

\[ MPL E = \text{mean}(CPLE), \]  
\[ (21) \]

\[ W = \begin{bmatrix} MPL E, MPL E, MPL E, MPL E, MPL E, MPL E \\ PLE_{1,2}, PLE_{1,3}, PLE_{1,4}, PLE_{2,3}, PLE_{2,4}, PLE_{3,4} \end{bmatrix}. \]  
\[ (22) \]

As illustrated by the following derivation, this set of cooperative inverse PLE weights has the relative effect of de-emphasizing measurements that are affected by non-line-of-sight propagation losses due multi-path and materials attenuation. Hence, in the four receiver case, Equations (15) and (16) becomes

\[ A = \begin{bmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) \\ 2(x_4 - x_1) & 2(y_4 - y_1) \\ 2(x_3 - x_2) & 2(y_3 - y_2) \\ 2(x_4 - x_2) & 2(y_4 - y_2) \\ 2(x_4 - x_3) & 2(y_4 - y_3) \end{bmatrix}, \]  
\[ (23) \]

\[ B = \begin{bmatrix} x_2^2 - x_1^2 - y_2^2 - y_1^2 - d_1^2 - d_2^2 \\ x_3^2 - x_1^2 - y_3^2 - y_1^2 - d_1^2 - d_3^2 \\ x_4^2 - x_1^2 - y_4^2 - y_1^2 - d_1^2 - d_4^2 \\ x_3^2 - x_2^2 - y_3^2 - y_2^2 - d_2^2 - d_3^2 \\ x_4^2 - x_2^2 - y_4^2 - y_2^2 - d_2^2 - d_4^2 \\ x_4^2 - x_3^2 - y_4^2 - y_3^2 - d_3^2 - d_4^2 \end{bmatrix}. \]  
\[ (24) \]

Let \( \cdot \) denote element-wise multiplication, and define

\[ A_\times W = A_\times \cdot W, \]  
\[ (25) \]
where \( A_x \) is the first column of \( A \), and

\[
A_{W,y} = A_y \cdot W;
\]  

(26)

where \( A_y \) is the second column of \( A \). Hence, the weighted matrix \( A_W \) is defined as \( A_W = [A_{W,x} \ A_{W,y}] \).

Then, the estimated transmitter location is calculated based on

\[
A_W C = B,
\]  

(27)

and is given by [83]

\[
\hat{C} = (A_W^T A_W)^{-1} A_W^T B.
\]  

(28)

### 3.2 Sensitivity and Robustness Issues

Robustness is of crucial importance in control system designs because real engineering systems are vulnerable to external disturbances and measurement noise. There are always differences between mathematical models used for the design and the actual system. Typically, a control engineer is required to design a controller which will stabilize a plant if it is not stable originally, and satisfy certain performance levels in the presence of disturbance signals, noise interference, un-modeled plant dynamics and plant parameter variations. Those design objectives are best realized via feedback control mechanism, though, which brings in the issues of high cost (the use of sensors), system complexity, and concerns on stability. In classical single-input single-output control, robustness is achieved by ensuring adequate gain and phase margins.
In order to further investigate PADF concepts in an indoor IDCAST lab environment, many PADF configuration concepts were developed. These configurations are based on potential refinements in consistency, sensitivity, and robustness via the design and implementation of four stationary sensors and one hidden transmitter during the localization phase. These results indicate the potential for developing intelligent PADF systems. The development of these systems may demonstrate a degree of robustness and localization accuracy by some analytical data processes. Then, a robustness figure data manipulation in MATLAB determines of each node and ultimately that of each configuration. The parameters that are used to compute the robustness figure are median values of RSSI and the rms values. From the result it is determined which configurations possess high robustness, (in the sense that it can be experimentally verified that they are not sensitive to small changes in sensor position). High values obtained from the robustness function indicate high robustness, while low values indicate lower robustness.

3.2.1 Experimental Procedure

The data is gathered by running the experiment nine times for each configuration. Each configuration has one mobile sensor node. Each configuration involves different position of the motes 1, 2, 3, and 4, to detect the hidden transmitter which is assigned at (0, 2.5) meters (used throughout the dissertation). Here, the transmitter is designated as mote 0.

Example one:

In configuration I in Figure 12, mote 1 is at (0, 0); mote 2 is at (-1.5, 1.5); mote 3 is at (0, 5); and mote 4 is at (1.5, 3).
Figure 12: Receiving sensors nodes in configuration I surrounding the transmitting node.

Thus we will have the dBm data between (0 to 1), (0 to 2), (0 to 3), and (0 to 4). For each data set, a plot of their robustness based on the median and rms values of the dBm is generated. The median is the middle of the $P_{Oi}$ data, after they are sorted in ascending order. For mote $i$, the median and rms values are calculated as such

\[
\begin{align*}
\text{median}_e^i &= P_{0i \frac{M+1}{2}}, \text{ if } M : \text{Odd} \\
\text{median}_e^i &= P_{0i \frac{M}{2}}, \text{ if } M : \text{Even},
\end{align*}
\]

(29)

\[
rms_e^i = \sqrt{\frac{1}{M} \sum_{l=1}^{M} P_{0i}^2},
\]

(30)

where $e$ is the experiment number, $M$ is the length of data in that experiment, and $P_{0i}$ is the RSSI in dBm.
The median data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in dBm for configuration I are shown in Table 1.

<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Median (0 to 1)</th>
<th>Median (0 to 2)</th>
<th>Median (0 to 3)</th>
<th>Median (0 to 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-51</td>
<td>-46</td>
<td>-62</td>
<td>-60</td>
</tr>
<tr>
<td>2</td>
<td>-51</td>
<td>-48</td>
<td>-63</td>
<td>-59</td>
</tr>
<tr>
<td>3</td>
<td>-51</td>
<td>-46</td>
<td>-62</td>
<td>-59</td>
</tr>
<tr>
<td>4</td>
<td>-51</td>
<td>-49</td>
<td>-64</td>
<td>-60</td>
</tr>
<tr>
<td>5</td>
<td>-51</td>
<td>-47</td>
<td>-64</td>
<td>-58.5</td>
</tr>
<tr>
<td>6</td>
<td>-51</td>
<td>-49</td>
<td>-64</td>
<td>-59</td>
</tr>
<tr>
<td>7</td>
<td>-51</td>
<td>-47</td>
<td>-62</td>
<td>-60</td>
</tr>
<tr>
<td>8</td>
<td>-51</td>
<td>-49</td>
<td>-63</td>
<td>-58</td>
</tr>
<tr>
<td>9</td>
<td>-51</td>
<td>-52</td>
<td>-64</td>
<td>-58</td>
</tr>
</tbody>
</table>

Table 1: Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration I.

The rms data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in dBm for configuration I are shown in Table 2.

Next, these values are plotted and the distribution indicates the robustness or sensitivity of the data. Some of the plots are shown below in Figure 13.

We observe that the median and rms values can overlap one another. This indicates the difference between a robust system and a sensitive system. We propose to evaluate robustness using

\[ R^i_e = \frac{rms_e^i + |median_e^i|}{(rms_e^i)^2}, \]  

where \(i=1, 2, 3,\) and 4.
Figure 13: Median and rms plot for configuration I: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
This formula uses median and rms values because they both represent two different kinds of averages calculated from the RSSI data. A higher median value (corresponding to a stronger signal) indicates that the data is more trustworthy. Similarly, a smaller value for rms means the data is more trustworthy. Figure 14, shows the results of robustness for each mote.

The experiment is repeated nine times. In each case the location of mote 2 is moved 5 cm to the north, south, east, west, northeast, northwest, southwest, and southeast. However, the locations of all other motes (1, 3, 4) are unchanged. The aim is to see which position is more robust by analyzing the sensitivity of the data after changing the position 5 cm in the directions stated above. Figure 15 shows graph (not to scale) with nine repetitions.

The next step is to combine these results so that we get a robustness reading for the whole configuration I. To do this, we sum up the robustness values from each respective experiment from each mote as denoted by
Figure 14: Robustness values for each mote in configuration I: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
Figure 15: Receiving sensors nodes in configuration I surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale).

\[ R_e = \sum_{i=1}^{4} R_{ei} \]  \hspace{1cm} (32)

where \( i \) is the mote number and \( R_{ei} \) is the robustness of that mote.

The results are shown below in Figure 16.

We then repeated these steps for another configuration, say configuration II, shown in Figure 17, where mote 1 is at (0, 0); mote 2 is at (-1.5, 3); mote 3 is at (0, 5); and mote 4 is at (1.5, 3).
Figure 16: Robustness for configuration I.

Figure 17: Receiving sensors nodes in configuration II surrounding the transmitting node.
We applied the robustness formula for each experiment and plotted the individual robustness results. As before, we combined these robustness values so that an overall robustness value for configuration II can be calculated. The results are shown in the following Figures [18 - 21].

The median data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in $dBm$ for configuration II are shown in Table 3.

<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Median (0 to 1)</th>
<th>Median (0 to 2)</th>
<th>Median (0 to 3)</th>
<th>Median (0 to 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-51</td>
<td>-62</td>
<td>-64</td>
<td>-55</td>
</tr>
<tr>
<td>2</td>
<td>-51</td>
<td>-69</td>
<td>-64</td>
<td>-55</td>
</tr>
<tr>
<td>3</td>
<td>-51</td>
<td>-61</td>
<td>-64</td>
<td>-55</td>
</tr>
<tr>
<td>4</td>
<td>-50</td>
<td>-69</td>
<td>-61</td>
<td>-55</td>
</tr>
<tr>
<td>5</td>
<td>-50</td>
<td>-70</td>
<td>-61</td>
<td>-55</td>
</tr>
<tr>
<td>6</td>
<td>-50</td>
<td>-53</td>
<td>-61</td>
<td>-55</td>
</tr>
<tr>
<td>7</td>
<td>-49</td>
<td>-60</td>
<td>-61</td>
<td>-55</td>
</tr>
<tr>
<td>8</td>
<td>-49</td>
<td>-63</td>
<td>-61</td>
<td>-54</td>
</tr>
<tr>
<td>9</td>
<td>-49</td>
<td>-60</td>
<td>-61</td>
<td>-55</td>
</tr>
</tbody>
</table>

Table 3: Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration II.

The rms data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in $dBm$ for configuration II are shown in Table 4.

Moreover, Figure 22 shows the two robustness curves plotted in the same axis for configurations I and II. That also highlights what we have seen earlier [84].
Figure 18: Median and rms plots for configuration II: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
Figure 19: Robustness values for each mote in configuration II: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Rms 0 to 1</th>
<th>Rms 0 to 2</th>
<th>Rms 0 to 3</th>
<th>Rms 0 to 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51.5447</td>
<td>63.0793</td>
<td>65.3756</td>
<td>54.9865</td>
</tr>
<tr>
<td>2</td>
<td>51.5499</td>
<td>68.7873</td>
<td>65.4295</td>
<td>54.9856</td>
</tr>
<tr>
<td>3</td>
<td>52.2440</td>
<td>61.8355</td>
<td>63.2072</td>
<td>54.9535</td>
</tr>
<tr>
<td>4</td>
<td>51.4922</td>
<td>68.9230</td>
<td>61.0114</td>
<td>56.4752</td>
</tr>
<tr>
<td>5</td>
<td>50.6998</td>
<td>70.4066</td>
<td>61.1498</td>
<td>56.9874</td>
</tr>
<tr>
<td>6</td>
<td>51.3384</td>
<td>55.3407</td>
<td>60.9756</td>
<td>58.3577</td>
</tr>
<tr>
<td>7</td>
<td>49.5458</td>
<td>61.1896</td>
<td>60.9991</td>
<td>56.3018</td>
</tr>
<tr>
<td>8</td>
<td>49.0316</td>
<td>63.9805</td>
<td>61.2216</td>
<td>55.6403</td>
</tr>
<tr>
<td>9</td>
<td>49.7152</td>
<td>59.5341</td>
<td>61.7830</td>
<td>57.2794</td>
</tr>
</tbody>
</table>

Table 4: Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration II.

Figure 20: Receiving sensors nodes in configuration II surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale).
Figure 21: Robustness for configuration II.

Figure 22: Robustness for configurations I and II in the same axis.
Example Two:

A second study for this scenario will consider two new different configurations. In configuration III as shown in Figure 23. Mote 1 is located at (0, 0); mote 2 is at (-1.5, 1.5); mote 3 is at (0, 5); and mote 4 is at (1.5, 1.5).

Figure 23: Receiving sensors nodes in configuration III surrounding the transmitting node.

Just as before, we will have the $dBm$ data between (0 to 1), (0 to 2), (0 to 3), and (0 to 4). For each data set, a plot of their robustness based on the median and rms values of the $dBm$ is generated.

For mote $i$, the median and rms values are calculated using Equations 29 and 30. Then, these values are plotted and the distribution indicates the robustness or sensitivity of the data. Some of the plots are shown below in Figure 24.
The median data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in $dBm$ for configuration III are shown in Table 5.

<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Median (0 to 1)</th>
<th>Median (0 to 2)</th>
<th>Median (0 to 3)</th>
<th>Median (0 to 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-51</td>
<td>-46</td>
<td>-62</td>
<td>-44</td>
</tr>
<tr>
<td>2</td>
<td>-51</td>
<td>-47</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>3</td>
<td>-51</td>
<td>-46</td>
<td>-63</td>
<td>-44</td>
</tr>
<tr>
<td>4</td>
<td>-51</td>
<td>-49</td>
<td>-64</td>
<td>-44</td>
</tr>
<tr>
<td>5</td>
<td>-51</td>
<td>-47</td>
<td>-64</td>
<td>-44</td>
</tr>
<tr>
<td>6</td>
<td>-51</td>
<td>-49</td>
<td>-64</td>
<td>-43</td>
</tr>
<tr>
<td>7</td>
<td>-51</td>
<td>-49</td>
<td>-64</td>
<td>-44</td>
</tr>
<tr>
<td>8</td>
<td>-51</td>
<td>-46</td>
<td>-64</td>
<td>-44</td>
</tr>
<tr>
<td>9</td>
<td>-51</td>
<td>-52</td>
<td>-64</td>
<td>-44</td>
</tr>
</tbody>
</table>

Table 5: Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration III.

The rms data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in $dBm$ for configuration III are shown in Table 6.

So, by using Equation 31, the robustness of each mote can be evaluated. Figure 25 shows the results of robustness for each mote in configuration III.

As before, the experiment is repeated nine times. In each case the location of mote 2 is moved 5 cm to the north, south, east, west, northeast, northwest, southwest, and southeast. Nonetheless, the locations of all other motes (1, 3, 4) are fixed. The aim is to see which position is more robust by analyzing the sensitivity of the data after changing the position 5 cm in the directions stated above. Figure 26 shows graph (not to scale) with nine repetitions.

Next, a robustness value for the whole configuration III is obtained by Equation 32. The results are shown below in Figure 27.
Figure 24: Median and rms plots for configuration III: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
Figure 25: Robustness values for each mote in configuration III: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Rms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0 to 1)</td>
</tr>
<tr>
<td>1</td>
<td>53.2684</td>
</tr>
<tr>
<td>2</td>
<td>56.1309</td>
</tr>
<tr>
<td>3</td>
<td>54.4150</td>
</tr>
<tr>
<td>4</td>
<td>54.0550</td>
</tr>
<tr>
<td>5</td>
<td>51.2332</td>
</tr>
<tr>
<td>6</td>
<td>51.6010</td>
</tr>
<tr>
<td>7</td>
<td>52.1349</td>
</tr>
<tr>
<td>8</td>
<td>54.3950</td>
</tr>
<tr>
<td>9</td>
<td>55.5227</td>
</tr>
</tbody>
</table>

Table 6: Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration III.

Figure 26: Receiving sensors nodes in configuration III surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale).
Now consider configuration IV, shown in Figure 28, where mote 1 is located at (0, 0); mote 2 is at (-1.5, 3); mote 3 is at (0, 5); and mote 4 is at (1.5, 1.5).

Next, we applied the robustness formula for each experiment and plotted the individual robustness results. These robustness values are combined so that an overall robustness value for configuration IV can be calculated. The results are shown in the following Figures [29 - 32].

The median data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in $dBm$ for configuration IV are shown in Table 7.

The rms data values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) in $dBm$ for configuration IV are shown in Table 8.
Figure 28: Receiving sensors nodes in configuration IV surrounding the transmitting node.

<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Median (0 to 1)</th>
<th>Median (0 to 2)</th>
<th>Median (0 to 3)</th>
<th>Median (0 to 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-49</td>
<td>-62</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>2</td>
<td>-49</td>
<td>-66</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>3</td>
<td>-49</td>
<td>-59</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>4</td>
<td>-49</td>
<td>-65</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>5</td>
<td>-49</td>
<td>-67</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>6</td>
<td>-49</td>
<td>-55</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>7</td>
<td>-49</td>
<td>-67</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>8</td>
<td>-49</td>
<td>-67</td>
<td>-61</td>
<td>-43</td>
</tr>
<tr>
<td>9</td>
<td>-49</td>
<td>-58</td>
<td>-61</td>
<td>-43</td>
</tr>
</tbody>
</table>

Table 7: Median values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration IV.
Figure 29: Median and rms plots for configuration IV: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
Figure 30: Robustness values for each mote in configuration IV: (a)(0 to 1); (b)(0 to 2); (c)(0 to 3); (d)(0 to 4).
<table>
<thead>
<tr>
<th>Experiment Number</th>
<th>Rms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0 to 1)</td>
</tr>
<tr>
<td>1</td>
<td>50.1423</td>
</tr>
<tr>
<td>2</td>
<td>49.8982</td>
</tr>
<tr>
<td>3</td>
<td>51.0099</td>
</tr>
<tr>
<td>4</td>
<td>49.6118</td>
</tr>
<tr>
<td>5</td>
<td>49.1497</td>
</tr>
<tr>
<td>6</td>
<td>49.0483</td>
</tr>
<tr>
<td>7</td>
<td>49.2932</td>
</tr>
<tr>
<td>8</td>
<td>49.9885</td>
</tr>
<tr>
<td>9</td>
<td>49.5125</td>
</tr>
</tbody>
</table>

Table 8: Rms values between (0 to 1), (0 to 2), (0 to 3), and (0 to 4) for configuration IV.

Figure 31: Receiving sensors nodes in configuration IV surrounding the transmitting node for improving robustness and accuracy by running mote 2 nine times (not to scale).
As well as, Figure 33 shows the two robustness curves plotted in the same axis for configurations III and IV. That also highlights what we have seen earlier.

From configurations III and IV, we see that for all experiments except one (number nine), the robustness of III is larger than that of IV as expected. Thus, this proves our hypothesis on the relationship between robustness and sensitivity of the mote in any configuration.

Furthermore, we repeated this procedure on many other configurations. From these, we observed that the less sensitive configurations had higher robustness values while the more sensitive configurations had lower robustness values. As will be shown later, a higher robustness value for a given experiment is correlated with a smaller estimation error.
3.3 Error Metric

The localization error $e$ is defined as the difference between an actual position and its estimated position, $e = \| C - \hat{C} \|$. This error is not available from the experiment. However, we propose an error metric $\hat{e}$ that provides an estimate of the localization error. This estimate is not expected, nor required, to approximate the actual error $e$. Instead, all that is required is that the slope of $\hat{e}$ (with respect to time) be of the same sign and similar magnitude as that of $e$, and that the values attained by $\hat{e}$ be consistent with those of $e$.

That is, if for two configurations one has two error values, say $e_1 < e_2$, we expect to have $\hat{e}_1 < \hat{e}_2$ as well. Similarly, if for the two configurations we have $e_1 \approx e_2$, we expect to have $\hat{e}_1 \approx \hat{e}_2$. 
This problem can be thought of as a function approximation problem. That is, we want to build a function

\[ \hat{e} = \mathcal{F}(\Theta), \]  

that approximates \( e \) in some sense, where \( \Theta \) is a vector of inputs. Possible ways to do this include the use of a neural network or other universal approximators, but this would require the use of off-line training. Additionally, it is not clear how to choose the inputs to form the vector \( \Theta \). Instead, here we take an ad-hoc approach, where we pick the inputs and the function \( \mathcal{F} \) manually, based on experience with the experiment. We recognize this is not easy to generalize, but it does allow for a possible way forward towards attaining closed-loop control as in Figure 56. Figure 56 given later in chapter 6.

In [231], an alternative multi-platform error metric is defined to estimate PLE values within the closed-loop multi-platform control structure depicted in Figure 7, via a series of cooperative transmissions between a combination of mobile sensor platforms.

The procedure of the multi-platform error metric structure is shown in Figure 34. The sensors detect the signal from the hidden emitter. Then the system estimates the location of the emitter based on RSSI measurements using LSE. The signals \( P_{i,j}, CPL E, T_i \) are used to produce \( \hat{e} \).

Based on difference between distance estimates \( D \) and a sensor positions, we define \( \hat{D} \) as

\[ \hat{D}_i = \| T_i - \hat{C} \|, \]  

(34)
where $T_i$ is the sensor position, $i = 1, \ldots, N$.

From here we compute

$$
\begin{align*}
D_{\text{max}} &= \max(|D - \hat{D}|), \\
D_{\text{min}} &= \min(|D - \hat{D}|).
\end{align*}
$$

(35)

Next, the error metric we propose is given by:

$$
\hat{e} = \left(3 \sqrt[3]{\frac{D_{\text{max}}}{10 + D_{\text{min}}} + \frac{2}{\sqrt{|\text{MPLE}|}} + \frac{1}{\text{std}(G)}} \right) \frac{1}{R_e},
$$

(36)

where, for $k = 1, 2, \ldots, 6$,
\[ CPLE_k = [PLE_1, ..., PLE_6], \]

\[ G = A_{Wk} \cdot CPLE_k, \quad (37) \]

\[ A_{Wk} = \frac{\sqrt{A_{k1}^2 + A_{k2}^2}}{2}. \]

The resulting error metric in (36) is investigated as a multi-platform RF scattering based objective function for purposes of multi-platform position adaptation and, in turn, multi-platform localization of embedded emitters.

The application of such a metric is useful in determining the robustness of a configurations. The overall robustness of configurations I and II in Figures 12 and 17 was using metric and error. We see in Figure 35 that for all experiments except one (number nine), the robustness of I is larger than that of II.

This is true as well for configurations III and IV in Figures 23 and 28. We see in Figure 36 that for all experiments except one (number nine), the robustness of III is larger than that of IV.

As can be seen, the error metric serves the original proposed objective: while its value is not a good approximation of the localization error itself, the metric’s slope is similar to the error’s shape, and the metric’s values are also consistent with the error values.
Figure 35: Metric and error for configurations I and II.

Figure 36: Metric and error for configurations III and IV.
CHAPTER IV

EXPERIMENTAL AND LAB SETUP

During the initial phases of this investigation, a series of basic sensor mote tests and distributed software control/measurement functions were developed for purposes of performing PADF experiments and hardware simulations using IRIS networked wireless sensor motes [114, 223]. We present a set of experiments using mobile sensor networks to localize a non-cooperative sensor node based on measured RSSI at surrounding sensor network. Related results are given in [99, 50].

We use on-line estimation of the PLE to model the distance, based on measured RSS. A detailed analysis on PLE estimation and modeling is given in [231], which is then used in distance calculations based on RSS measurements. The experiments that were developed read sensor data, convert the data into approximate distance values based on approximated PLE values, and estimate the position of the emitter using the LSE method [220].

Using four IRIS nodes as receivers and one IRIS node as a transmitter, several experiments were performed to test optimal RSSI values. Also performed were several experiments to test the proper configuration that gives the best hidden emitter estimation accuracy. We grouped these experiments into two different cases, case one and case two, where each case tested different configurations. Among these experiments, some were tested with obstruction and some without obstruction.
4.1 Case One

4.1.1 Case One: Experiments Without Obstruction

After designing and implementing a PADF experimental apparatus with distributed wireless sensor motes, a number of discrete configurations were synthesized for purposes of investigating PADF concepts, as outlined in previous sections. This set of experimental configurations is shown in Figure 37.

The location of the sensors and emitter configurations in case one, are shown below in Table 9, relative to the \(x\) and \(y\) coordinate plane.

<table>
<thead>
<tr>
<th>Configuration Number</th>
<th>Emitter</th>
<th>Sensor 1</th>
<th></th>
<th>Sensor 2</th>
<th>Sensor 3</th>
<th>Sensor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x)</td>
<td>(y)</td>
<td>(x)  (y)</td>
<td>(x)  (y)</td>
<td>(x)  (y)</td>
<td>(x)  (y)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>-2.5</td>
<td>2.5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9: Position of sensors and emitter configurations for case one as shown in Figures 37 and 38.

4.1.2 Case One: Experiments With Obstruction

Using the initial setup from the previous experiments, the same tests were performed. We placed the emitter inside of a partially sealed enclosure that was covered on three sides with aluminum foil to shield the radio signal. This creates a more challenging environment to detect the hidden emitter, similar to situations where there is an RF leakage point or leakage area. We measured the response of the IRIS nodes and corresponding RSSI values.
Figure 37: Discrete position-adaptive configurations without obstruction for case one at IDCAST.
It is expected that the degradation of the RF signal passing through the enclosure would degrade the accuracy of the LSE position of the transmitter. This set of experimental configurations is shown in Figure 38. For example, the diagram for configuration 1 is a top view perspective of a simulated initial, or initial state, platform geometry with all four sensor motes spaced 2.5 meters apart in both the $x$ and $y$ dimensions (all the receiving nodes surround the transmitter equilaterally). Similarly, the diagram for configuration 3 is a top view perspective of a simulated sensor geometry after two iterative position readjustments of the platform locations. In this configuration, three sensor motes are depicted to adapt in position to locations in front of the RF leakage point that is associated with the (aluminum foil covered) enclosure that, in turn, contains the embedded RF emitter in the center. In this case (i.e., configuration 3), the three sensor motes in front of the enclosure are located 1 meter apart.

Each receiver has a corresponding “disc area” that acts as the sensing radius for the node. The LSE uses the intersections of these disc areas to determine the position of the transmitter located at the red X in the center. Figure 39 shows a MATLAB visualization of the intersecting disc areas and the actual location of the transmitter for experimental configuration 1 as shown in Figure 37.

4.2 Case Two

4.2.1 Case Two: Experiments Without Obstruction

Figure 40 shows the experimental setup for case two. The tests that were conducted for case one were also performed for case two. However, different configurations were used to test optimal RSSI values. Also performed were several experiments to test the proper configuration that gives the best hidden emitter estimation accuracy.
Figure 38: Discrete position-adaptive configurations with obstruction for case one at IDCAST.
Figure 39: An intersection of circles with radii equal to RSS measured at individual nodes for configuration 1.

The location of the sensors and emitter configurations in case two are shown below in Table 10, relative to the $x$ and $y$ coordinate plane.

### 4.2.2 Case Two: Experiments With Obstruction

Using the configurations shown in Figure 40, the same experiments were conducted, but this time the emitter was placed inside of a partially sealed enclosure. This set of experimental configurations are shown in Figure 41.

Chapter 5 provides sets of experimental data analysis using this PADF technique, in two different experimental environments.
Figure 40: Discrete position-adaptive configurations without obstruction for case two at IDCAST.
Figure 41: Discrete position-adaptive configurations with obstruction for case two at IDCAST.
<table>
<thead>
<tr>
<th>Configuration Number</th>
<th>Emitter</th>
<th>Sensor 1</th>
<th>Sensor 2</th>
<th>Sensor 3</th>
<th>Sensor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x$</td>
<td>$y$</td>
<td>$x$</td>
<td>$y$</td>
<td>$x$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.5</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
<td>-2.5</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
<td>-2.5</td>
</tr>
</tbody>
</table>

Table 10: Position of sensors and emitter configurations for case two as shown in Figures 40 and 41.
CHAPTER V

EXPERIMENTAL RESULTS

5.1 At IDCAST

We ran the tests with and without obstructions in the sensing field. The accuracy of the estimation was determined by comparing the estimated and actual positions of the emitter. Figure 42 is a collection of pictures that illustrate the experimental apparatus developed for PADF experiments at IDCAST [7]. The partially sealed enclosure with the embedded non-cooperative emitter is shown in the two pictures on the left and also in the picture on the lower right. The enclosure is covered with aluminum foil on all sides, except the front, in order to simulate an embedded RF environment with one leakage point. The base station for this WSN is shown on the upper right.

We have developed a Graphical User Interface (GUI) using MATLAB that allows for easy implementation of the PADF concepts. The GUI inputs the cartesian coordinates of known nodes, along with the hidden emitter, and gives a prediction or estimation as to where the hidden node could be. Based on the location of the actual hidden emitter compared to estimated hidden emitter, we determine an absolute error, which shows the accuracy of the localization technique. For a proof of concept, we know the location of the hidden emitter. Four $x$ and $y$ positions are used to estimate the transmitter’s position.
Figure 42: Selected pictures of experimental apparatus developed for PADF investigations at IDCAST [7].
On the other side of the loop, the IRIS nodes themselves gather the RSS from emitter to receiver and forward that information to the base station wirelessly, which is then sent to the same MATLAB station via USB. The GUI merges the position data of the nodes as well as its corresponding RSSI measurements, and performs the LSE in order to estimate the location of the hidden emitter.

The blue curve in Figure 43 depicts the resulting PADF error metric $\hat{e}$ that we proposed in chapter 3 as a multi-platform RF-scattering-based objective function for purposes of multi-platform position adaptation and, in turn, multi-platform localization of embedded emitters. Rough correlation between the basic trends in new RF objective function (blue curve) and localization error (red curve) indicates potential for further development of this type of technique for RF measurement and sensing applications. There exists a distinctly observable correlation between the overall trend or shape of the red curve and blue curve for this particular data set. This type of correlated trend between the two curves implies that the PADF error metric $\hat{e}$ developed during this investigation shows promise for further development and investigation as a multi-platform objective function for purposes of furthering the advancement of PADF systems.

The basic trends and potential candidate approaches to further analysis/development of PADF techniques that are indicated via analysis of this IDCAST data set as shown in Figure 43.

Moreover, Figure 44 shows a different collection of pictures that illustrate the experimental apparatus developed for PADF experiments at IDCAST.
Figure 43: MATLAB GUI for PADF calculation [99], showing sample output depicting a PLE-based RF scattering error metric (blue curve) versus localization error (red curve).
Figure 44: Collection of pictures that illustrate the experimental apparatus developed for PADF experiments at IDCAST.
5.1.1 Case One

Without Obstruction

After running five different configurations, the first configuration had the lowest error. This is an expected result since LSE is based on the intersection of the RSSI circles from all of the receiving nodes. According to Table 11, the first two configurations have the lowest localization error.

<table>
<thead>
<tr>
<th>Test Cofig.</th>
<th>Actual (m)</th>
<th>Error (m)</th>
<th>Error Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2.5</td>
<td>0.2231</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.5</td>
<td>0.8182</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.5</td>
<td>0.8343</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2.5</td>
<td>1.4954</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2.5</td>
<td>1.6901</td>
</tr>
</tbody>
</table>

Table 11: IRIS platform for multiple configurations with error and error metric (case one corresponds to configurations 1-5 from Figure 37).

We can observe from Figure 45 that there is a high value of cross-correlation. To see this, let

\[ Z = \frac{|e - \hat{e}|}{\hat{e}}. \]  \hspace{1cm} (38)

For Figure 45, we obtain \( mean(Z) = 0.9315 \), and \( std(Z) = 0.0305 \), which denote high correlations. Similarly, for the experiment with obstruction in Figure 46, we obtain \( mean(Z) = 0.8971 \), and \( std(Z) = 0.0296 \).
These experiments were carried out with sensor nodes placed around the partially sealed enclosure. In this case, configurations where nodes were grouped in front of the leakage point allowed for more accurate emitter localization. The foil on the three sides shields the RF signal and attenuates the RSSI values. The only opening was the front face of the enclosure, causing the RSSI values to be larger in the front of the enclosure. Thus, the position estimation is more accurate. These results are shown in Table 12.
Table 12: IRIS platform for multiple configurations with error and error metric (case one corresponds to configurations 1-5 from Figure 38).

<table>
<thead>
<tr>
<th>Test Config.</th>
<th>$x$</th>
<th>$y$</th>
<th>Error (m)</th>
<th>Error Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2.5</td>
<td>0.7965</td>
<td>15.0135</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.5</td>
<td>1.3937</td>
<td>13.5018</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.5</td>
<td>1.2931</td>
<td>11.9108</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2.5</td>
<td>1.9275</td>
<td>15.8372</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2.5</td>
<td>1.9959</td>
<td>15.5860</td>
</tr>
</tbody>
</table>

Figure 46: Metric (unitless) and error (meters) for multiple configurations (case one corresponds to configurations 1-5 from Figure 38).
5.1.2 Case Two

Without Obstruction

After running six different configurations for case two, configuration 4 had the lowest error. Table 13, shown below, is the error and error metric results for case two.

<table>
<thead>
<tr>
<th>Test Config.</th>
<th>x (m)</th>
<th>y (m)</th>
<th>Error (m)</th>
<th>Error Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2.5</td>
<td>0.9051</td>
<td>10.8269</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.5</td>
<td>1.1319</td>
<td>11.4425</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.5</td>
<td>0.7636</td>
<td>10.1856</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2.5</td>
<td>0.2122</td>
<td>12.3202</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2.5</td>
<td>0.5622</td>
<td>10.9500</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2.5</td>
<td>0.2879</td>
<td>10.4371</td>
</tr>
</tbody>
</table>

Table 13: IRIS platform for multiple configurations with error and error metric (case two corresponds to configurations 1-6 from Figure 40).

By using Equation 38, we can observe a high value of cross-correlation, similar to case one. For Figure 47, we obtain $\text{mean}(Z) = 0.9411$, and $\text{std}(Z) = 0.0324$, which denote high correlations as well. Similarly, for the experiment with obstruction in Figure 48, we obtain $\text{mean}(Z) = 0.9139$, and $\text{std}(Z) = 0.0273$.

With Obstruction

These experiments were carried out with sensor nodes placed around the partially sealed enclosure. In this case, configurations where nodes were grouped in front of the leakage point allowed for more accurate emitter localization, similar to case one. The results for case two are shown in Table 14.
Figure 47: Metric (unitless) and error (meters) for multiple configurations (corresponds to case two configurations 1-6 from Figure 40).

Table 14: IRIS platform for multiple configurations with error and error metric (case two corresponds to configurations 1-6 from Figure 41).
5.2 Experiments at AFRL MAV LAB

As previously mentioned, integrating the networked PADF mote sensors from the discretized PADF experiment design at IDCAST onto a combination of continuously adaptable mobile platforms can enable the investigation of a number of significant refinements to the present baseline PADF technique as outlined in the previous sections. These potential refinements are described and modeled within the dotted circle block of the modified functional PADF block diagram in Figure 49.

In order to further investigate PADF concepts in a indoor MAV Lab environment, two scenarios were experimentally evaluated, as shown in Figures 50 and 51. In these scenarios (labeled MODE1 and MODE2), three receivers are stationary and one is mobile.
In MODE1, the mobile sensor follows a circular trajectory, and in MODE2 it follows a zig-zag trajectory.

As depicted in Figure 50, in MODE1 a MAV that is integrated with a custom sensor mote for networked cooperative PADF is flown in a circular trajectory around an experimental region-of-interest. The box partially wrapped with aluminum foil and then covered with black cloth to avoid inducing ambiguities in the distributed camera-based optical navigation system. An embedded emitter (i.e., a non-cooperative sensor mote) is placed in the embedded closure in the vicinity of a RF leakage point that is cut out from the front of the enclosure. The basic functionality of MODE1 is the detection and demarcation of a contoured sub-interval as indicated by the contour endpoint symbols, C1 and C2, in the MODE1 diagram in Figure 50. The MODE2 configuration is shown on Figure 51.
Figure 50: MODE1 experimental PADF concept using MAVs with wireless sensor nodes.
Figure 51: MODE2 experimental PADF concept using MAVs with wireless sensor nodes.
The contour limits, C1 and C2, in MODE1 are computed by applying thresholds on the RSSI data collected by the MAV during the circular trajectory encompassed by MODE1. Computation of the C1 and C2 sub-contour limits, in turn, enable the placement of two RC-controlled ground vehicles in the vicinity of points C1 and C2 during the beginning phases of MODE2 data collections. In addition, the MAV helicopter platform is placed at a slight standoff distance from the center of the detected contour interval for the RF leakage point in front of the enclosure.

During the dynamic data collection phases of the MODE2 portion of these MAV lab experiments, a fourth PADF sensor mote platform (i.e., another RC-controlled ground vehicle) is operated in either a localized zig-zag pattern or a localized circular pattern. For the data analysis in this paper, data from moving the forth PADF sensor mote platform along a 10-point localized circular pattern is presented.

Figures 52 and 53 illustrate some desirable properties of potential error metric $\hat{e}$ developed for this investigation in the sense that the PLE RF-scattering-based multi-platform control metric computation (blue curve) displays the same general trends as the localization error (red curve). One of the most significant interpretations that can be derived via observation and analysis of these plots is the occurrence of isolated regions of relative stability associated with the distance.
Figure 52: Metric and error for circular trajectory.
Figure 53: Metric and error for 10-point zigzag.
CHAPTER VI

SOURCE SEEKING BASED EXTREMUM SEEKING CONTROL

6.1 Why Extremum Seeking Control

Traditional control system design deals with the problem of stabilization of a system about a known reference trajectory or set point while attaining certain design criteria. This general stabilization problem includes the so-called “tracking” and “regulation” problem. The reference is often easily determined. However, in some occasions it can be very difficult to find a suitable reference value. For instance, the fuel consumption of a car depends on the ignition angle. It is necessary to change the ignition angle as the condition of the road and the load of the car change to maintain optimal efficiency.

Tracking a varying maximum or minimum (extremum, optimum) of a performance (output, cost) function is called Extremum Seeking Control (ESC) [232]. It has two layers of meaning: first, we need to seek an extremum of the output function; secondly, we need to be able to control (stabilize) the system and drive the performance output to that extremum [233]. In the traditional optimal control problem, the performance function generally is an integral function of the state and control, that is, the extremum of the performance function is a trajectory. Therefore, the calculus of variation method is involved in the design, and
the analytical form of the performance function is needed to obtain the necessary conditions of the optimal control. However, in ESC, the performance function is generally a function of the state and not known or poorly known in the design. We can only design an extremum seeking controller based on the measurements of the performance output or its derivatives if available [234]. The goal of an ESC is to operate at a set-point that optimizes the performance function. A general block diagram of ESC can be found on Figure 54.

![General block diagram for extremum seeking control.](image)

ESC is related to optimization; many of the ideas used in ESC have been transferred from numerical optimization. Developments in computers and optimization have led to a renewed interest in ESC. Many investigations of ESC systems assume that the system is static, which can be justified if the time between the changes in the optimal reference is sufficiently long. Then, the extremum seeking is reduced to an optimization problem in some sense.

ESC is applicable in situations where there is a nonlinearity in the control problem, and the nonlinearity has a local minimum or a maximum. The nonlinearity may be in the plant, as a physical nonlinearity, possibly manifesting itself through an equilibrium, or it may be
in the control objective, added to the system through a cost functional of an optimization problem. Hence, one can use ESC both for tuning a set point to achieve an optimal value of the output, or for tuning parameters of a feedback law [235]. ESC is of great practical interest, since even small improvements in the performance can lead to large savings in raw material and energy consumption. There are commercial extremum seeking controllers.

6.1.1 History and Literature Overview

The emergence of ESC dates as far back as the 1922 paper of Leblanc [236], and it was popular in 1950s and 1960s [237, 238, 239, 240, 241, 242], which have been pursued largely by Russian researchers and much before the theoretical breakthroughs in adaptive linear control of the 1980’s. A historical development of ESC is stated in [235] and then it reviews the existing literature especially for the last decades. Several books in the 1950s-60s have been exclusively or partly dedicated to extremum seeking control, such, Tsien [243] (1954), Feldbaum [244] (1959), Krasovskii [245] (1964), Wilde [246], and Chinaev [247] (1969).

In the traditional application of ESC, the optimal parameters were assumed to vary rather slowly. Then, system dynamics were typically neglected and the algorithm was analyzed and designed using traditional “static optimization” techniques [248, 249]. Many investigations of ESC systems assume that the dynamic system is static, which can be justified if the time between the changes in the optimal reference is sufficiently long. ESC of static systems is in essence a problem of numerical optimization [232], which can be approached by the continuous (analog) implementations of some optimization methods, the so called “analog optimization” approach (such as sinusoidal perturbation and sliding
mode-based analog optimization). ESC witnessed a resurgence of interest after Krstić’s publication of stability studies on perturbation-based ESC in [250] and [251].

The first rigorous proof [250] of local stability of perturbation based ESC scheme uses averaging analysis and singular perturbation, where a high-pass filter and slow perturbation signal are employed to derive the gradient information. Reference [235] presents a systematic description of the perturbation-based ESC and its applications. New progress in semiglobal stability appears in [252]. Gradient estimation-based extremum seeking control is studied in [253]. ESC based on sliding mode are studied in [255, 256, 257], where time delay, excessive oscillation and performance improvement issues are addressed. Extremum seeking via continuous time nonderivative optimizers is proposed in [259]. An ESC problem is proposed and solved in [258] for a class of nonlinear systems with unknown parameters, where an explicit structure information for the performance function to be maximized is required.

Numerical optimization-based ESC methods were used successfully. Now, the requirement for gradient measurements is not continuous, and it may have time to collect enough output measurements to estimate the gradient. Moreover, there are non-gradient numerical optimization algorithms that can be implemented as well. ESC via triangular search as in Zhang [265] was employed to attenuate combustor thermoacoustic oscillations and minimize diffuser losses at United Technologies Research Center (UTRC). Nonlinear programming was successfully used in extremum seeking control by defining a readout map as a steady state output function [254]. Simultaneous Perturbation Stochastic Approximation (SPSA) recursive algorithm is used in the ESC design [259]. More systematic studies by Zhang and Ordóñez on numerical optimization-based ESC first appeared in [260], and then
in [261, 262, 263, 264], where numerical optimization algorithm and state regulation are combined to design the ESC scheme.

6.1.2 Applications of Extremum Seeking Control

An extremum controller having no special trial steps or oscillations can be designed if some a priori knowledge about the plant and its disturbance exist [266]. That is, the extremum characteristic of the plant is unimodal and can be approximated by second or third-order polynomials, the distribution of the disturbances is close to uniform and the value of the optimum changes slowly. A self tuning concept is applied to the extremum control problem in [267]. ESC of Wiener type systems is consider in [268], the linear subsystem is described by a discrete-time systems with delay and Gaussian distributed white noise. The nonlinearity is described as a quadratic function with a unique minimum. Therefore, the author solves the minimum of the output and the purpose of the control is to keep the output as close as possible to the minimum. ESC based on probing strategy can be found in [269], the stability and performance issues are performed further in [270]. Discrete-time extremum seeking algorithms for SISO nonlinear systems are proposed in [271], where the reference-to-output equilibrium map is approximated via a quadratic polynomial. The extremum can be explicitly solved by the polynomial parameters, then three extremum seeking algorithms are proposed based on the way how they estimate the polynomial parameters, which are least square estimation, parabola approximation and ellipse approximation. A similar study can be found in [272].
Among the many applications of ESC overviewed in [248] and [232] are combustion processes, grinding processes, solar cell and radio telescope antenna adjustment to maximize the received signal, and blade adjustment in water turbines and wind mills to maximize the generated power. Recent applications include Antilock Braking System (ABS) design [235], exercise machine [273], optimizing bioreactor [274] formation control [275] limit cycle minimization [276], axial-flow compressor [277], combustion instability [278], fermentation processes [279], engine optimization [280], wind turbine [281], flow separation control in diffusers [282], tubular reactor [283], electromechanical valve actuator [284], thermoacoustic cooler [285], blending processes [286], plasma control [287], PID tuning [288], maximum power point tracking [289] and source seeking without position measurements [290].

Control of a single or a group of autonomous agent and sensor networks is one of the most active research area recently. Source seeking or (source localization), i.e., the design of control algorithms for autonomous agents to seek a source with unknown spatial distribution is of great interest, where ESC can be naturally used in the design. Even though it is an application having great theoretical interest, it also does have a significant impact on engineering applications: for instance, in the problem of the developing vehicles with more autonomy, such as the situation where no GPS information is available, or to reduce cost due to position sensors. Some of the direct applications of source seeking can be found in contaminant plume control, autonomous odor sensing or toxic leakage localization. In the application of source seeking, the task of the vehicle is to find a source that emits a signal that decays as a function of distance away from the source, where the signal field is unknown and only the measurement of the signal at the current agent location is available [233].
A basic diagram of source seeking in Figure 55, where the control goal is for the agent to seek an extremum of an unknown signal field based on the measurement of the signal only [233].

![Figure 55: Source Seeking.](image)

The application of ESC in group source seeking (namely, swarm seeking) has emerged recently as well. Swarm seeking is approached via a leader-follower format in [291], and ESC is used to guide the leader vehicle to seek the source; then a passivity framework is used to design the followers’ coordination laws.

### 6.2 Swarm Source Seeking

The problem of coordination and control of autonomous vehicles (agents) has been receiving an extraordinary amount of attention during the past decade due to its critical importance for military, as well as civilian applications. Research on vehicle autonomy will open up the door to a myriad of applications. For instance, the use of robots for navigating and inspecting dangerous areas could be made cheap, reliable and wide-spread. Other similar uses include search over wide areas and automated geographical and topological surveying. Moreover, providing vehicles and robots with autonomy would enormously
facilitate planetary exploration, where automated construction of structures and surveying of areas would help reduce the amount of the much more costly and risky alternative, that of direct human intervention. Possible applications could range from autonomous robot assembly to UAVs scout and counter insurgency. Compared to individuals, swarms, flocks, and schools can have remarkable group-level characteristics, which may allow them to perform complicated task efficiently [233].

In recent years it has become an active area of research to study the biological world and apply the derived principles to the design of engineering. The topic of distributed coordination and control of multiple autonomous agents has gained lots of attention [292, 293, 294]. Cooperative agents can often be used to perform tasks that are too difficult for a single one to perform. Instead of the traditional trajectory tracking problem, people began to study coordinated tracking [295, 296] The swarm source seeking problem is to find a coordinated control scheme for a group of agents to make them achieve and maintain some given geometrical formation. At the same time, the agents need to seek a source of a scalar signal, or track a moving target [234]. Thus, there is a trade-off between maintaining formation and arriving at the final goal [291, 298, 297]. Motivated by the work in [299], swarm tracking is achieved via artificial potentials and ESC. Possible approaches for formation control include leader-following and the virtual structure approaches [300, 301]. In both approaches, one agent, which could be real or virtual, is designated as leader to perform the tracking task without considering the followers, and the remaining agents only have to stay from the leader within a desired offset, without considering the target. However, these approaches are centralized and therefore not robust – if the leader fails, the task would also fail. In that case, decentralized formation control is preferred as discussed in [302], where feedback control laws are used to keep the formation during tracking process.
The potential functions have been first widely used for robot navigation and control including multi-agent coordination [300, 303, 304, 305]. A potential function is created to contain the scalar signal to be tracked, as well as the interaction rules for the group of agents. By minimizing the potential function, one is able to achieve source seeking, formation control, and collision avoidance. Extremum seeking techniques are used to design the controller for each agent.

6.3 Problem Statement

The problem statement is mainly based on the work by Yao, Ordóñez and Gazi in [299]. Consider a multi-agent system (i.e., a swarm) consisting of \( N \) individuals in an \( n \)-dimensional Euclidean space. We assume synchronous motion and no time delays. Let \( x \in \mathbb{R}^n \) denote a vector \( n \)-dimensional Euclidean space, and \( x^i \in \mathbb{R}^n \) denote the position vector of individual agent \( i \), whose motion is governed by the following velocity actuated point-mass kinematic model,

\[
\dot{x}^i = u^i, \tag{39}
\]

where \( u^i \in \mathbb{R}^n \) is the control input for the \( i^{th} \) sensor. Our purpose is to design a control law for each individual sensor using extremum seeking methods that can minimize the error metric by adapting the sensor positions in real-time, thereby minimizing the unknown estimation error. As a result, we can achieve source seeking and collision avoidance.

In this chapter we consider a potential function composed of several parts. First, the inter-connection component puts a constraint on the sensor, based on its neighbors’ positions, in order to maintain a group structure. This part includes functions of the relative distance between each pair of neighbors. In addition, a tracking component containing the
error metric to be tracked is added in order to direct the group’s behavior for source seeking. This tracking component could be an artificial potential function given the knowledge of target position, or the concentration of a chemical source, or an RF source, as in our problem [233].

The specific form of potential function is defined according to the desired geometric formation. The choice of a potential function is important because different potentials might result in different performance even with the same control algorithm.

First, consider a potential of the form of the error metric we proposed in chapter three to estimate the localization error for a given sensor configuration, given by:

\[ J_{\text{metric}}(x^1, x^2, \ldots, x^N, x_t) = \left(3\sqrt{\frac{D_{\text{max}}}{10 + D_{\text{min}}} + \frac{2}{\text{MPLE}}} + \frac{1}{\text{std}(G)}\right), \]  

(40)

Second, consider the potential function for “sensor to sensor” interaction, given by

\[ J_{\text{aa}}(x^1, x^2, \ldots, x^N) = \sum_{i=1}^{N-1} \sum_{j=1,j\neq i}^{N} \frac{1}{\epsilon + (\|x^i - x^j\|)}, \]  

(41)

where \( \frac{1}{\epsilon + (\|x^i - x^j\|)} \) is the potential between the \( i \)th and the \( j \)th sensor agents, whose purpose is to provide a repulsive force between agents. Such potentials can be obtained if one knows the relative distance between sensors, or by measuring a possible field distribution generated by the sensor.
Now, we can put the swarm tracking problem into the framework of ESC design. Let

\[ y = J(x^1, x^2, \ldots, x^N, x_t) \]

\[ = K_{at} J_{\text{metric}}(x^1, x^2, \ldots, x^N, x_t) + K_{aa} J_{\text{aa}}(x^1, x^2, \ldots, x^N) \]

\[ = K_{at} (3\sqrt{D_{\text{max}}/10 + D_{\text{min}}} + 2/\sqrt{|\text{MLE}|} + 1/\text{std}(G)) + K_{aa} \sum_{i=1}^{N-1} \sum_{j=1, j\neq i}^{N} 1/\epsilon + (\|x^i - x^j\|), \]

be the performance function of the system (39) for \( i = 1, \ldots, N \), where \( K_{at}, K_{aa} \) are the weights of the potential components, which is important in balancing the priority in source seeking and inter-agent avoidance. Our objective is to reach the sensor network configuration that minimizes the localization error, while avoiding collisions. A block diagram can be found in Figure 56.

In particular, successful implementation of the detection system initiates the integration of a sensor position control system in the feedback loop so as to make the process autonomous using ESC, as shown in Figure 56. This figure, as we can see, describes the ESC closed-loop with the potential function (42) that was used in this simulation and implementation.

Now, we will study two cases where each one considers two situations, which is with and without obstruction. In the first case, one moving mote, used as a mobile sensor, is used together with three fixed motes. The second case considers four moving motes together as mobile sensors. Furthermore, another simulation implemented the Monte Carlo method of testing to determine the mean, standard deviation and median of the localization error and the error metric.
Figure 56: ESC closed-loop with potential function that can attain true position-adaptive sensor network reconfiguration autonomously.

6.4 Simulation Results Without Obstruction for Two Different Cases

6.4.1 Case One

In this case, we considered three sensor agents fixed and one moving as a mobile sensor, where the target emitter is fixed at (0, 2.5) (used throughout these cases).

Example One

In this example, we considered three sensor agents fixed, sensor agent number 1 assigned at (0, 0), sensor agent 2 at (-1, 0), sensor agent 3 at (0, -1), and sensor agent number 4 is considered as a mobile sensor, where the initial position at (1, 0).
The results for this example experiments are shown below in Figures [57 - 60]. The entire movements of the sensors can be found in Figure 57, where source seeking and collision avoidance are successfully achieved. The initial position of sensor agent number 4 is shown with a green circle, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as seen in Figure 58. The error metric is minimized as well and can be seen in Figure 59, and the localization error is minimized and can be seen in Figure 60.

For simulation purposes, we consider a two dimensional case where $n = 2$ and four sensors $N = 4$, $K_{at} = 1$, $K_{aa} = 0.1$, $\epsilon = 0.1$. The simulation time interval is $[0, 50]$ time units.

Figure 57: Trajectory of the swarm sensors “without obstruction” using ESC for case one example one.
Figure 58: Potential functions “without obstruction” for case one example one.

Figure 59: Error metric “without obstruction” for case one example one that is minimized by ESC.
Example Two

In this example, we considered three sensor agents fixed, sensor agent number 1 assigned at (0, 0), sensor agent 2 at (-1, 0), sensor agent 3 at (0, -1), and sensor agent number 4 is considered as a mobile sensor, where the initial position is chosen randomly.

The results for this example experiments are shown below in Figures [61 - 64]. The entire movements of the sensors can be found in Figure 61, where source seeking and collision avoidance are successfully achieved. The initial position of sensor agent number 4 is shown with a green circle, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as seen in Figure 62. The error metric is minimized as well and can be seen in Figure 63, and the localization error is minimized and can be seen in Figure 64.
Figure 61: Trajectory of the swarm sensors “without obstruction” using ESC for case one example two.

Figure 62: Potential functions “without obstruction” for case two example two.
Figure 63: Error metric “without obstruction” for case one example two that is minimized by ESC.

Figure 64: Localization error “without obstruction” for case one example two that is minimized by ESC.
6.4.2 Case Two

For case two, we considered all four sensor agents moving together as mobile sensors, and the target emitter fixed.

Example One

In this example, we consider all four sensor agents moving together as mobile sensors, where the initial positions of the sensor agents mobile sensors are chosen randomly.

The results for this example experiments are shown below in Figures [65 - 68]. The entire movements of the sensors can be found in Figure 65, where source seeking and collision avoidance are successfully achieved. The initial position of sensor agent number 4 is shown with a green circle, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as seen in Figure 66. The error metric is minimized as well and can be seen in Figure 67, and the localization error is minimized and can be seen in Figure 68.

6.5 Simulation Results With Obstruction for Two Different Cases

6.5.1 Case One

In this case, we considered three sensor agents fixed and one moving as a mobile sensor, where the target emitter is fixed at (0, 2.5) (used throughout these cases).

Example One

In this example, we considered three sensor agents fixed, sensor agent number 1 assigned at (0, 0), sensor agent 2 at (-2.5, 2.5), sensor agent 3 at (0, 5), and sensor agent number 4 is consider as a mobile sensor, where the initial position at (2.5, 2.5).
Figure 65: Trajectory of the swarm sensors “without obstruction” using ESC for case one example two.

Figure 66: Potential functions “without obstruction” for case two example two.
Figure 67: Error metric for case one example two “without obstruction” that is minimized by ESC.

Figure 68: Localization error “without obstruction” for case one example two that is minimized by ESC.
The results for this example experiments are shown below in Figures [69 - 72]. The entire movements of the sensors can be found in Figure 69, where source seeking and collision avoidance are successfully achieved. The initial position of sensor agent number 4 is shown with a green circle, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as seen in Figure 70. The error metric is minimized as well and can be seen in Figure 71, and the localization error is minimized and can be seen in Figure 72.

For simulation purposes, we consider a two dimensional case where \( n = 2 \) and four sensors \( N = 4, K_{at} = 1, K_{aa} = 0.1, \epsilon = 0.1 \). The simulation time interval is \([0, 50]\) time units.

![Swarm Trajectories](image)

Figure 69: Trajectory of the swarm sensors “with obstruction” using ESC for case one example one.
Figure 70: Potential functions “with obstruction” for case one example one.

Figure 71: Error metric “with obstruction” for case one example one that is minimized by ESC.
Example Two

In this example, we considered three sensor agents fixed, sensor agent number 1 assigned at (0, 0), sensor agent 2 at (-2.5, 2.5), sensor agent 3 at (0, 5), and sensor agent number 4 is considered as a mobile sensor, where the initial position is chosen randomly.

The results for this example experiment are shown below in Figures [73 - 76]. The entire movements of the sensors can be found in Figure 73, where source seeking and collision avoidance are successfully achieved. The initial position of sensor agent number 4 is shown with a green circle, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as seen in Figure 74. The error metric is minimized as well and can be seen in Figure 75. The localization error is minimized and can be seen in Figure 76.
Figure 73: Trajectory of the swarm sensors “with obstruction” using ESC for case one example two.

Figure 74: Potential functions “with obstruction” for case one example two.
Figure 75: Error metric “with obstruction” for case one example two that is minimized by ESC.

Figure 76: Localization error “with obstruction” for case one example two that is minimized by ESC.
6.5.2 Case Two

For case two, we considered all four sensor agents moving together as mobile sensors, and the target emitter fixed.

Example One

In this example, we considered all four sensor agents moving together as mobile sensors, where the initial position of sensor agent number 1 are chosen at (0, 0), sensor agent 2 at (-2.5, 2.5), sensor agent 3 at (0, 5), and sensor agent 4 at (2.5, 2.5).

The results for this example experiment are shown below in Figures [77 - 80]. The entire movement of the sensors can be found in Figure 77, where source seeking and collision avoidance are successfully achieved. The sensor agents’ positions are shown with green circles, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as seen in Figure 78. The error metric is minimized as well and can be seen in Figure 79. The localization error is minimized and can be seen in Figure 80.

Example Two

In this example, we consider all four sensor agents moving together as mobile sensors, where the initial positions of the sensor agents mobile sensors are chosen randomly.

The results for this example experiment are shown below in Figures [81 - 84]. The entire movements of the sensors can be found in Figure 81, where source seeking and collision avoidance are successfully achieved. The sensor agents’ positions are shown with green circles, and the agents’ final positions are shown with yellow square. The hidden transmitter is marked with a green asterisk. The performance function is minimized as
Figure 77: Trajectory of the swarm sensors “with obstruction” using ESC for case two example one.

Figure 78: Potential functions “with obstruction” for case two example one.
Figure 79: Error metric “with obstruction” for case two example one is minimized by ESC.

Figure 80: Localization error “with obstruction” for case two example one that is minimized by ESC.
seen in Figure 82. The error metric is minimized as well and can be seen in Figure 83. The localization error is minimized and can be seen in Figure 84.

![Swarm Trajectories](image)

**Figure 81:** Trajectory of the swarm sensors “with obstruction” using ESC for case two example two.

### 6.6 Monte Carlo Implementation

A Monte Carlo experimental approach was taken to simulate the two different cases, mentioned above, where the initial positions of the mobile sensors are chosen randomly for each simulation run. This was used to verify the ESC algorithms, to prove that the error metric and localization error are minimized on average by adapting the sensors’ positions. For the two cases mentioned above, the standard deviation, median, and mean error metric and localization error were calculated for five hundred randomized experiments. In each
Figure 82: Potential functions “with obstruction” for case two example two.

Figure 83: Error metric “with obstruction” for case two example two that is minimized by ESC.
simulation, four mote positions were chosen randomly. The standard deviation, median, and the mean for the error metric and localization error were then calculated for the initial and final simulations. Tables 15 and 16, shown below, show the calculated values for both experimental cases. The simulation time interval is $[0, 50]$ time units.

<table>
<thead>
<tr>
<th>Experiment Case</th>
<th>Mean ($\hat{e}(0)$)</th>
<th>Std. ($\hat{e}(0)$)</th>
<th>Median ($\hat{e}(0)$)</th>
<th>Mean ($\hat{e}(50)$)</th>
<th>Std. ($\hat{e}(50)$)</th>
<th>Median ($\hat{e}(50)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One sensor moving</td>
<td>2.5625</td>
<td>0.5056</td>
<td>2.5020</td>
<td>2.3045</td>
<td>0.4387</td>
<td>2.3242</td>
</tr>
<tr>
<td>Four sensors moving</td>
<td>2.6061</td>
<td>0.5156</td>
<td>2.5708</td>
<td>2.4941</td>
<td>0.4698</td>
<td>2.4082</td>
</tr>
</tbody>
</table>

Table 15: Monte Carlo study for 500 simulations, where $\hat{e}(0)$ means error metric for initial simulation, and $\hat{e}(50)$ means error metric for final simulations.

Figure 84: Localization error “with obstruction” for case two example two that is minimized by ESC.
<table>
<thead>
<tr>
<th>Experiment Case</th>
<th>Mean ($e(0)$)</th>
<th>Std. ($e(0)$)</th>
<th>Median ($e(0)$)</th>
<th>Mean ($e(50)$)</th>
<th>Std. ($e(50)$)</th>
<th>Median ($e(50)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One sensor moving</td>
<td>2.4100</td>
<td>0.9284</td>
<td>2.8469</td>
<td>2.2743</td>
<td>0.7531</td>
<td>2.4280</td>
</tr>
<tr>
<td>Four sensors moving</td>
<td>2.2236</td>
<td>0.3424</td>
<td>2.0634</td>
<td>2.0878</td>
<td>0.6275</td>
<td>1.9794</td>
</tr>
</tbody>
</table>

Table 16: Monte Carlo study for 500 simulations, where $e(0)$ means localization error for initial simulation, and $e(50)$ means localization error for final simulations.

As shown in Table 15, the initial mean for case one and case two were 2.5626 and 2.6061, respectively. The final mean for case one and case two were 2.3045 and 2.4941, respectively. The final values were lower than the initial values, which suggest that the ESC algorithms operated as intended. The same holds true for the initial and final standard deviations, and the initial and final medians as well.

Moreover, Figure 85 shows the plot of the five hundred calculated values of the mean error metric, standard deviation, and median for case one. This plot holds true, as well, for case 2 and can be see in Figure 86.

Figure 87 shows the plot of the five hundred calculated values of the mean localization error, standard deviation, and median for case one. This plot holds true, as well, for case 2 and can be see in Figure 88.

Figures 89 and 90 show the contour plot of the error metric and the localization error, respectively, for case one, where mote number four moved on a grid spanning the space $[-2.5, 2.5] \times [0, 5]$ meters, at intervals of 5 cm along each dimension, and the other three motes are fixed. Mote number one is at (0, 0), mote two at (-2.5, -2.5) and mote three at (0, 5) in the Cartesian plane. These plots show that the error metric is sufficiently correlated with the localization error, although it is not perfect. The mismatch between error metric
Figure 85: Case one Monte Carlo study: (a) mean error metric; (b) standard deviation error metric; (c) median error metric.
Figure 86: Case two Monte Carlo study: (a) mean error metric; (b) standard deviation error metric; (c) median error metric.
Figure 87: Case one Monte Carlo study: (a) mean localization error; (b) standard deviation localization error; (c) median localization error.
Figure 88: Case two Monte Carlo study: (a) mean localization error; (b) standard deviation localization error; (c) median localization error.
and actual error is the main reason why even though the ESC mechanism does manage to minimize the potential, it does not always minimize the error.

Figure 89: Contour plot for error metric, where the sign indicates the position of minimum metric value.
Figure 90: Contour plot for localization error, where the sign indicates the position of minimum error value.
CHAPTER VII

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In recent years, location estimation in WSN has raised a lot of interest from researchers. The idea of having sensing data without any information of the location does not make sense. Existing techniques, such as GPS are, usually, inappropriate in large scale networks, due to the increase in the cost and size of the nodes. For that reason, a variety of location estimation techniques has been proposed for WSNs. Recent advances in wireless communications and electronics have enabled the development of low-cost, low-power and multi-functional sensors that are small in size and communicate in short distances. Cheap, smart sensors, networked through wireless links and deployed in large numbers, provide unprecedented opportunities for monitoring and controlling homes, cities, and the environment.

WSNs is a kind of ad-hoc network. WSN is a network of many small sensing and communicating devices called sensor nodes or motes, which can sense the environment and communicate the information gathered from the monitored field through wireless links. Sensor networks are low cost and low power devices, designed to collect data, do some local computation and transmit partially processed data via radio frequency or bluetooth.
WSNs are being used in a variety of ways: from reconnaissance and detection in the military to biomedical applications toward environmental sensing applications and a variety of commercial endeavors. Most WSNs applications require knowing or measuring locations of thousands of sensors accurately. For example, sensing data without knowing the sensor location is often meaningless. Localization is the process of determining the positions of nodes in an ad hoc network. It is an important problem that has attracted much attention recently.

In WSNs, the distance between sensor nodes can be estimated by measuring RSSI. The wireless RSSI based localization techniques have attracted significant research interest for their simplicity. The RSSI based localization techniques can be divided into two categories: the distance estimation based and the RSSI profiling based techniques. The PLE is a key parameter in the distance estimation based localization algorithms, where distance is estimated from the RSSI.

Position-Adaptive radar concepts have been formulated and investigated at the AFRL. This research introduced a PADF method using mobile sensor networks and presents the latest experimental results in the localization of a non-cooperative sensor node using static and mobile sensor networks. PADF is based on the formulation and investigation of path-loss (PLE) based metrics that are measured and estimated across multiple platforms in order to robotically/intelligently adapt the location of each platform. This approach shows potential for accurate emitter localization in challenging embedded multi-path environments (i.e., urban environments).

The results indicate the potential for developing intelligent PADF systems that incorporate novel and advanced processing features such as the capability to delimit or select the useful data during dynamic PADF operations. Further pursuit of this type of PADF research
shows potential for the development of PADF systems that may demonstrate a degree of robustness and localization accuracy via development and implementation of advanced distributed self data-pruning processes. In addition, some novel platform concepts are under consideration for future tiers of research including PADF concepts with launchable projectiles and wirelessly remote-controlled integrated and ejectable RC helicopter blades and RF apertures.

We have experimentally demonstrated that RSSI can effectively be used for localization of hidden emitters in wireless sensor networks environment. Localization accuracy is improved as the number of sensor nodes and their spatial distribution/spread increases. We first reviewed the most relevant WSNs and localization algorithms. Then, we surveyed the existing body of literature on these fields. Furthermore, initial results on PADF were presented. We have introduced the concept of PADF applied to localization of hidden, non-cooperative emitters where static and mobile nodes cooperate in the localization mission.

We provided a summary of recent experimental results in localization of a non-cooperative sensor node using static and mobile sensor networks. In this study we used IRIS wireless sensor nodes. In order to localize the transmitter, we used the RSSI data to approximate the distance from the transmitter to the revolving receivers. We provided an algorithm for on-line estimation of the PLE that is used in modeling the distance based on RSSI measurements. The emitter position estimation was calculated based on surrounding sensors’ RSSI values using LSE. The PADF method was tested on a number of different configurations in the laboratory via the design and implementation of four IRIS wireless sensor nodes as receivers and one hidden sensor as a transmitter during the localization phase. The robustness of detecting the transmitter’s position was initiated by getting the RSSI data through experiments and then data manipulation in MATLAB determined the robustness of each
node and ultimately of each configuration. The parameters that were used in the functions are the median values of RSSI and rms values. From the result, it was determined which configurations possess high robustness. High values obtained from the robustness function indicated high robustness, while low values indicated lower robustness.

Finally, successful implementation of the detection system initiated the integration of a sensor position control system in the feedback loop so as to make the process autonomous using ESC. We presented the experimental performance analysis on the application aspect. We applied ESC schemes by using the swarm source seeking approach, where we designed a control law for each individual sensor using extremum seeking methods that can minimize the error metric by adapting the sensor positions in real-time, thereby minimizing the unknown estimation error. As a result, we achieved source seeking and collision avoidance.

7.2 Future Work and Open Problems

Currently we adopted this concept and integrated wireless sensor nodes to MAVs, thus creating mobile nodes that can adapt their position to localize an RF source in three dimensional space. This will be vital in detecting frequency signatures of harmful entities, for instance explosives.

Some specific near-term plans include the integration and demonstration of IRIS sensor nodes onto multiple TREX 600 RC helicopters (see Figures 91 and 92 [306]) platforms (and/or combination of similar platforms) to demonstrate novel approaches to PADF.

Other than implementation, open problems that remain include the following:

• Improving the error metric function, potentially by using function approximators such as neural networks.
Figure 91: Sample TREX 600 RC helicopter platform recently acquired by IDCAST to support PADF research via integration of onboard IRIS sensor motes.

- Studying theoretically the “estimation limits”, that is, the best possible level of accuracy achievable given a fixed number of sensors and the complexity of the environment.

- Application of the cooperative localization method explored in this dissertation to problems not involving RF sensors, but instead for instance chemical and other types of signals.
Figure 92: Integration and demonstration of IRIS sensor nodes onto multiple RC helicopters.
BIBLIOGRAPHY


We have developed a Graphical User Interface (GUI) using MATLAB that allows for easy implementation of the PADF concepts. The GUI inputs the cartesian coordinates of known nodes, along with the hidden emitter, and gives a prediction or estimation as to where the hidden node could be. Based on the location of actual hidden emitter compared to estimated hidden emitter, we determine an absolute error, which shows the accuracy of the localization technique. For a proof of concept, we know the location of the hidden emitter. Four $x$ and $y$ positions are used to estimate the transmitter’s position. On the other side of the loop, the IRIS nodes themselves gather the RSS from emitter to receiver and forward that information to the base station wirelessly, which is then sent to the same MATLAB station via USB. The GUI merges the position data of the nodes as well as its corresponding RSSI measurements, and performs the LSE in order to estimate the location of the hidden emitter.

### A.1 How To Run The Code

1. Open MATLAB, change to the directory where the code is.

2. Type ’padf_gui’ in MATLAB’s command window.
3. When the GUI opens, make sure the TX mote number 0 is on, and that the receiver motes number 1-4 are on and in the desired positions.

4. Enter the positions of the RX motes and the TX motes in the GUI. When ready, click on the 'Estimate Position’ button. Wait till data acquisition is done and the estimate appears.

5. If desired, click the 'Save data’ button, which produces a mat file called ‘rssi_data (number).dat’, where number is the current experiment number.

6. Other parameters you can change in the GUI:

   a: Ranges for x and y axes. These are used in the determination of when an estimate is reliable or not.

   b: The number of samples desired.

   c: The 'Fudge' factor used to make a guess of when an RX mote is behind an obstruction.

7. If you need to modify the GUI: type ‘guide’ in the MATLAB command window, then open the 'padf_gui.fig’ file.

**A.2 Padf.Gui.m**

```matlab
function varargout = padf_gui(varargin)     
  % PADF_GUI M-file for padf_gui.fig    
  % PADF_GUI, by itself, creates a new    
  % PADF_GUI or raises the existing      
  % singleton*.                        
  % H = PADF_GUI returns the handle     
  % to a new PADF_GUI or the handle to  
  % the existing singleton*.           
  % PADF_GUI('CALLBACK',hObject,eventData,handles,...)  
  % calls the local                     
  % function named CALLBACK in PADF_GUI.M 
  % with the given input arguments.     
  % PADF_GUI('Property','Value',...) creates a new PADF_GUI or raises the  
  % existing singleton*. Starting from the left, property value pairs are  
  % applied to the GUI before padf_gui_OpeningFcn gets called. An        
  % unrecognized property name or invalid value makes property application  
  % stop. All inputs are passed to padf_gui_OpeningFcn via varargin.       
```

193
See GUI Options on GUIDE’s Tools menu. Choose "GUI allows only one instance to run (singleton)."

Author: Huthaifa Al Issa, Raul Ordonez
Date started: 5-Apr-2010
Last modified: 14-May-2010
See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help padf_gui
Last Modified by GUIDE v2.5 11-May-2010 10:50:55
Begin initialization code — DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @padf_gui_OpeningFcn, ...
    'gui_OutputFcn', @padf_gui_OutputFcn, ...
    'gui_LayoutFcn', [], ..., ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
End initialization code — DO NOT EDIT
--- Executes just before padf_gui is made visible.
function padf_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to padf_gui (see VARARGIN)
% Choose default command line output for padf_gui
handles.output = hObject;
% Add fields to handle object for error, metric and experiment number
% tracking
handles.exp_number = 0;
handles.error = [];
handles.metric = [];
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes padf_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = padf_gui_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
% --- Executes on mouse press over axes background.
function padf_fig_ButtonDownFcn(hObject, eventdata, handles)
% hObject handle to padf_fig (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Determine whether val is a valid position
val = str2double(get(hObject,'String'));
if isnumeric(val) && length(val)==1 && ...
    val >= x_axis(1) && ...
    val <= x_axis(2)
    % Enable the Plot button with its original name
    set(handles.estimate_pos,'String','Estimate position')
    set(handles.estimate_pos,'Enable','on')
else
    % Disable the Plot button and change its string to say why
    set(handles.estimate_pos,'String','Cannot estimate')
    set(handles.estimate_pos,'Enable','off')
    % Restore focus to the edit text box after error
    uicontrol(hObject)
end
set(handles.show_msg,'String','');
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
% function m2_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to m2_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% function edit2_Callback(hObject, eventdata, handles)
% hObject handle to edit2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit2 as text
% str2double(get(hObject,'String')) returns contents of edit2 as a double
% --- Executes during object creation, after setting all properties.
% function edit2_CreateFcn(hObject, eventdata, handles)
% hObject handle to edit2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in estimate_pos.
function estimate_pos_Callback(hObject, eventdata, handles)
% hObject handle to estimate_pos (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.save_button,'Enable','off');
handles.exp_number = handles.exp_number + 1;
[y, Fs, nbits, readinfo] = wavread('purr.wav');
m2 = [str2double(get(handles.m2_x,'String')) str2double(get(handles.m2_y,'String'))];
m3 = [str2double(get(handles.m3_x,'String')) str2double(get(handles.m3_y,'String'))];
m4 = [str2double(get(handles.m4_x,'String')) str2double(get(handles.m4_y,'String'))];
m5 = [str2double(get(handles.m5_x,'String')) str2double(get(handles.m5_y,'String'))];
hm = [str2double(get(handles.hm_x,'String')) str2double(get(handles.hm_y,'String'))];
x_axis = eval(get(handles.x_axis,'String'));
y_axis = eval(get(handles.y_axis,'String'));
% Plot mote positions
plot(handles.padf_fig,m2(1), m2(2), 'go', 'LineWidth', 5)
% compute offset for mote label positions
off_x = (x_axis(2)-x_axis(1))/20;
off_y = (y_axis(2)-y_axis(1))/20;
% text(handles.padf_fig, m2(1)+off_x, m2(2)+off_y,'2','FontSize',16,'Color','Green')
% text(handles.padf_fig, m3(1)+off_x, m3(2)+off_y,'3','FontSize',16,'Color','Green')
% text(handles.padf_fig, m4(1)+off_x, m4(2)+off_y,'4','FontSize',16,'Color','Green')
% text(handles.padf_fig, m5(1)+off_x, m5(2)+off_y,'5','FontSize',16,'Color','Green')
axis(handles.padf_fig, [x_axis y_axis])
grid(handles.padf_fig)

% Perform position estimate
set(handles.show_msg,'String','Acquiring data...')
pause(1)
num_samples = str2double(get(handles.num_samples,'String'));
handles.data(handles.exp_number).rssi_data = read_basestation(num_samples);
% load off_line_data_analysis/experiments_14_may_2010_withtop/rssi_data4.mat
% handles.data(handles.exp_number).rssi_data = rssi_data;
[D, cross_PLE, avg_TX_toall] = process_data_func(handles.data(handles.exp_number).rssi_data);
T = [m2; m3; m4; m5];
handles.data(handles.exp_number).T = T;
fudge = str2double(get(handles.fudge,'String'));
% used to discriminate between yes or no blockage present
% The larger this number, the less likely it is to detect a % true blockage, but also the less likely it is to have a false % positive
% t1=1; t2=1;
for j=1:6 % 6 distance combinations for 4 motes
    if t2<4 t2=t2+1;
    end
    sizeT(j) = norm(T(t2,:)-T(t1,:));
    if t2=4 t2=t1+1;
    t1=t1+1;
    end
end
max_sep = max(sizeT)-min(sizeT); % not currently used
D' Mean(sizeT)*fudge;
RSSI_edge=avg_TX_toall(2,1)
discard=0;
for j=1:4
    if D(j)>max_sep*fudge;
        D(j)=D(j)*0.75;
    end
    if D(j) > x_axis(2)-x_axis(1) | D(j) > y_axis(2)-y_axis(1)
        % At least one distance estimate is too large, so discard experiment
        discard=1;
    end
end
distance_weight = mean(cross_PLE)./cross_PLE;
A.mat = [ 2*(T(2,1)-T(1,1)) 2*(T(2,2)-T(1,2))
            2*(T(3,1)-T(1,1)) 2*(T(3,2)-T(1,2))
            2*(T(4,1)-T(1,1)) 2*(T(4,2)-T(1,2))
            2*(T(3,1)-T(2,1)) 2*(T(3,2)-T(2,2))
            2*(T(4,1)-T(2,1)) 2*(T(4,2)-T(2,2))
            2*(T(4,1)-T(3,1)) 2*(T(4,2)-T(3,2)) ];
A.mat_wgt(:,1) = A.mat(:,1).*distance_weight(:,1);
A.mat_wgt(:,2) = A.mat(:,2).*distance_weight(:,1);
B.mat = -1*(T(1,1)-T(2,1)-T(3,1)-T(4,1))
        (T(1,1)-T(2,1)-T(3,1)-T(4,1))
        (T(1,1)-T(2,1)-T(3,1)-T(4,1))
        (T(1,1)-T(2,1)-T(3,1)-T(4,1))
        (T(1,1)-T(2,1)-T(3,1)-T(4,1));
C.pos = (A.mat_wgt.'*A.mat_wgt').\A.mat_wgt.'*B.mat;
% Draw circles based on distance estimates
ttl=';
for j = 1:4
    fh = @(x,y) sqrt((x-T(j,1)).^2 + (y-T(j,2)).^2 - D(j)^2);
    ezplot(handles.padf_fig,fh);
end
plot(handles.padf_fig, C.pos(1),C.pos(2), 'mx', 'LineWidth',5)
axis equal
xlabel(handles.padf_fig,'x - coordinate')
ylabel(handles.padf_fig,'y - coordinate')
handles.error(handles.exp_number) = norm(C.pos-[hm(1);hm(2)]);
ttl = sprintf('Results for experiment #%d.', ...
                handles.exp_number);
% Evaluate and plot performance metric
for j=1:size(A.mat,1)
    mtrc_distance_weight(j) = norm(A.mat(j,:)/2);
end
% compute distance from each mote to estimate
for j=1:4
    D_est(j)=norm(T(j,:)-C.pos');
    handles.metric(handles.exp_number) = ...
        3.*(maxD./(10+minD)).^((1/2))+2./abs(mean_cross_PLE).^(1/2)+1.:(std_cross_PLE_wgt.^((1/2))+...
        plot(handles.error_metric_fig, [1:1:handles.exp_number],...
        handles.error, 'r-o','LineWidth',2);
        handles.RSSI_edge(handles.exp_number) = RSSI_edge;
        handle = 'unreliable results';
        if discard
            set(handles.show_msg,'String','unreliable results')
        else
            set(handles.save_button,'Enable','on')
        end
        sound(y, Fs);
        % Update handles structure
        guidata(hObject, handles);
        function m2_y_Callback(hObject, eventdata, handles)
            % hObject handle to m2_y (see GCBO)
            % eventdata reserved - to be defined in a future version of MATLAB
            % handles structure with handles and user data (see GUIDATA)
            val = str2double(get(hObject,'String'));
            % Determine whether val is a valid position
            if isnumeric(val) && length(val)==1 && ...
                val >= y_axis(1) && ...
                val <= y_axis(2)
                    % Enable the Plot button with its original name
                set(handles.estimate_pos,'String','Estimate position')
                    % Restore focus to the edit text box after error
                uicontrol(hObject)
            end
        end
        set(handles.show_msg,'String','')
        set(handles.save_button,'Enable','off')
        % --- Executes during object creation, after setting all properties.
        function m2_y_CreateFcn(hObject, eventdata, handles)
            % hObject handle to m2_y (see GCBO)
            % eventdata reserved - to be defined in a future version of MATLAB
            % handles empty - handles not created until after all CreateFcns called
            % Hint: edit controls usually have a white background on Windows.
            if ispc && isequal(get(hObject,'BackgroundColor'), ...
            get(0,'defaultUicontrolBackgroundColor'))
                set(hObject,'BackgroundColor','white');
            end
        function m3_x_Callback(hObject, eventdata, handles)
    % hObject handle to m3_x (see GCBO)
    197
% eventdata reserved - to be defined in a future version of MATLAB
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
x_axis = eval(get(handles.x_axis,'String'));
if isnumeric(val) && length(val)==1 &&
    val >= x_axis(1) &&
    val <= x_axis(2)
    % Enable the Plot button with its original name
    set(handles.estimate_pos,'String','Estimate position')
    set(handles.estimate_pos,'Enable','on')
else
    set(hObject,'String','Invalid entry');
    % Disable the Plot button and change its string to say why
    set(handles.estimate_pos,'String','Cannot estimate')
    set(handles.estimate_pos,'Enable','off')
    % Restore focus to the edit text box after error
    uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function m3_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to m3_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function m3_y_Callback(hObject, eventdata, handles)
% hObject handle to m3_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
y_axis = eval(get(handles.y_axis,'String'));
if isnumeric(val) && length(val)==1 &&
    val >= y_axis(1) &&
    val <= y_axis(2)
    % Enable the Plot button with its original name
    set(handles.estimate_pos,'String','Estimate position')
    set(handles.estimate_pos,'Enable','on')
else
    set(hObject,'String','Invalid entry');
    % Disable the Plot button and change its string to say why
    set(handles.estimate_pos,'String','Cannot estimate')
    set(handles.estimate_pos,'Enable','off')
    % Restore focus to the edit text box after error
    uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function m3_y_CreateFcn(hObject, eventdata, handles)
% hObject handle to m3_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function m4_x_Callback(hObject, eventdata, handles)
% hObject handle to m4_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
x_axis = eval(get(handles.x_axis,'String'));
if isnumeric(val) && length(val)==1 &&
    val >= x_axis(1) &&
    val <= x_axis(2)
    % Enable the Plot button with its original name
    set(handles.estimate_pos,'String','Estimate position')
    set(handles.estimate_pos,'Enable','on')
else
    set(hObject,'String','Invalid entry');
end
function m4_y_Callback(hObject, eventdata, handles)
% hObject handle to m4_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
y_axis = eval(get(handles.y_axis,'String'));
if isnumeric(val) && length(val)==1 &&
    val >= y_axis(1) &&
    val <= y_axis(2)
    % Enable the Plot button with its original name
    set(handles.estimate_pos,'String','Estimate position')
    set(handles.estimate_pos,'Enable','on')
ext
% Disable the Plot button and change its string to say why
set(handles.estimate_pos,'String','Cannot estimate')
set(handles.estimate_pos,'Enable','off')
% Restore focus to the edit text box after error
uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function m4_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to m4_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
get(0,'defaultUicontrolBackgroundColor')),...
set(hObject,'BackgroundColor','white');
end
function m4_y_Callback(hObject, eventdata, handles)
% hObject handle to m4_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
y_axis = eval(get(handles.y_axis,'String'));
if isnumeric(val) && length(val)==1 && ... val >= y_axis(1) && val <= y_axis(2)
set(hObject,'String',... ['Invalid entry']);
set(handles.estimate_pos,'String','Estimate position')
set(handles.estimate_pos,'Enable','on')
else
set(hObject,'String',...
['Invalid entry']);
% Disable the Plot button and change its string to say why
set(handles.estimate_pos,'String','Cannot estimate')
set(handles.estimate_pos,'Enable','off')
% Restore focus to the edit text box after error
uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function m4_y_CreateFcn(hObject, eventdata, handles)
% hObject handle to m4_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),... get(0,'defaultUicontrolBackgroundColor')),...
set(hObject,'BackgroundColor','white');
end
function m5_x_Callback(hObject, eventdata, handles)
% hObject handle to m5_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
x_axis = eval(get(handles.x_axis,'String'));
if isnumeric(val) && length(val)==1 && ... val >= x_axis(1) && val <= x_axis(2)
set(hObject,'String',... ['Invalid entry']);
set(handles.estimate_pos,'String','Estimate position')
set(handles.estimate_pos,'Enable','on')
else
set(hObject,'String',...
['Invalid entry']);
% Disable the Plot button and change its string to say why
set(handles.estimate_pos,'String','Cannot estimate')
set(handles.estimate_pos,'Enable','off')
% Restore focus to the edit text box after error
uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function m5_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to m5_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ... 
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function m5_y_Callback(hObject, eventdata, handles)
    % hObject handle to m5_y (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    val = str2double(get(hObject,'String'));
    if isnumeric(val) && length(val)==1 && ... 
       val >= y_axis(1) && ... 
       val <= y_axis(2)
        % Enable the Plot button with its original name
        set(handles.estimate_pos,'String','Estimate position');
        set(handles.estimate_pos,'Enable','on')
    else
        % 'Invalid entry')
        % Disable the Plot button and change its string to say why
        set(handles.estimate_pos,'String','Cannot estimate')
        set(handles.estimate_pos,'Enable','off')
        % Restore focus to the edit text box after error
        uicontrol(hObject)
    end

set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function m5_y_CreateFcn(hObject, eventdata, handles)
    % hObject handle to m5_y (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called
    % Hint: edit controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),... 
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function x_axis_Callback(hObject, eventdata, handles)
    % hObject handle to x_axis (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    x_axis = eval(get(handles.x_axis,'String'));
    if isa(x_axis,'double') && length(x_axis) > 2 && ... 
       min(diff(x_axis)) < 0
        % x_axis is not a number
        set(handles.show_msg,'String','x_axis is not numeric')
        return
    elseif length(x_axis) < 2
        % x_axis is not a number
        set(handles.show_msg,'String','x_axis must be vector')
        return
    elseif length(x_axis) > 2
        % x_axis is too long a vector to plot clearly
        set(handles.show_msg,'String','x_axis is too long')
        return
    elseif min(diff(x_axis)) < 0
        % x_axis is not monotonically increasing
        set(handles.show_msg,'String','x_axis must increase')
        return
    end
    % All OK; Enable the Plot button with its original name
    set(handles.estimate_pos,'Enable','on')
return
end

% --- Executes during object creation, after setting all properties.
function x_axis_CreateFcn(hObject, eventdata, handles)
    % hObject handle to x_axis (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called
    % Hint: edit controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),... 
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function x_axis_CreateFcn(hObject, eventdata, handles)
    % hObject handle to x_axis (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    x_axis = eval(get(handles.x_axis,'String'));
    if isa(x_axis,'double') && length(x_axis) > 2 && ... 
       min(diff(x_axis)) < 0
        % x_axis is not a number
        set(handles.show_msg,'String','x_axis is not numeric')
        return
    elseif length(x_axis) < 2
        % x_axis is not a number
        set(handles.show_msg,'String','x_axis must be vector')
        return
    elseif length(x_axis) > 2
        % x_axis is too long a vector to plot clearly
        set(handles.show_msg,'String','x_axis is too long')
        return
    elseif min(diff(x_axis)) < 0
        % x_axis is not monotonically increasing
        set(handles.show_msg,'String','x_axis must increase')
        return
    end
    % All OK; Enable the Plot button with its original name
    set(handles.estimate_pos,'Enable','on')
return
end

catch EM
    % Cannot evaluate expression user typed
    set(handles.show_msg,'String','Cannot plot')
    return
end

set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function x_axis_CreateFcn(hObject, eventdata, handles)
    % hObject handle to x_axis (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called
    % Hint: edit controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),... 
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

200
set(hObject,'BackgroundColor','white');
end
function y_axis_Callback(hObject, eventdata, handles)
% hObject handle to y_axis (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Test that the time vector is not too large, is not scalar,
% and increases monotonically. First fail if EVAL cannot parse it.
% Disable the Plot button ... until proven innocent
set(handles.estimate_pos,'Enable','off')
try
  y_axis = eval(get(handles.y_axis,'String'));
  if isnumeric(y_axis)
    % y_axis is a number
  else
    set(handles.show_msg,'String','y_axis is not numeric')
  end
  if length(y_axis) < 2
    % y_axis is not a vector
  else
    if length(y_axis) > 2
      % y_axis is too long a vector to plot clearly
    else
      if min(diff(y_axis)) < 0
        % y_axis is not monotonically increasing
      else
        % All OK; Enable the Plot button with its original name
        set(handles.estimate_pos,'Enable','on')
        return
      end
    end
  end
catch EM
  % Cannot evaluate expression user typed
  set(handles.estimate_pos,'String','Cannot plot')
  % Give the edit text box focus so user can correct the error
  uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function y_axis_CreateFcn(hObject, eventdata, handles)
% hObject handle to y_axis (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
                get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end
function hm_x_Callback(hObject, eventdata, handles)
% hObject handle to hm_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
x_axis = eval(get(handles.x_axis,'String'));
if isnumeric(val) && length(val)==1 && ...
  val >= x_axis(1) && ...
  val <= x_axis(2)
  % All OK; Enable the Plot button with its original name
  set(handles.estimate_pos,'String','Estimate position')
  set(handles.estimate_pos,'Enable','on')
else
  set(hObject,'String','Invalid entry');
  % Disable the Plot button and change its string to say why
  set(handles.estimate_pos,'String','Cannot estimate')
  set(handles.estimate_pos,'Enable','off')
  % Restore focus to the edit text box after error
  uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','on')
% --- Executes during object creation, after setting all properties.
function hm_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to hm_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% determine whether val is a valid position
x_axis = eval(get(handles.x_axis,'String'));
if isnumeric(val) && length(val)==1 && ...
  val >= x_axis(1) && ...
  val <= x_axis(2)
  % All OK; Enable the Plot button with its original name
  set(handles.estimate_pos,'String','Estimate position')
  set(handles.estimate_pos,'Enable','on')
else
  set(hObject,'String','Invalid entry');
  % Disable the Plot button and change its string to say why
  set(handles.estimate_pos,'String','Cannot estimate')
  set(handles.estimate_pos,'Enable','off')
  % Restore focus to the edit text box after error
  uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','on')
% --- Executes during object creation, after setting all properties.
function hm_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to hm_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
                get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end
201
function hm_y_Callback(hObject, eventdata, handles)
% hObject handle to hm_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
y_axis = eval(get(handles.y_axis,'String'));
if isnumeric(val) && length(val)==1 && ...
  val >= y_axis(1) && ...
  val <= y_axis(2)
% Enable the Plot button with its original name
set(handles.estimate_pos,'String','Estimate position')
set(handles.estimate_pos,'Enable','on')
else
  set(hObject,'String',...
       ['Invalid entry']);
  % Disable the Plot button and change its string to say why
  set(handles.estimate_pos,'String','Cannot estimate')
  set(handles.estimate_pos,'Enable','off')
  % Restore focus to the edit text box after error
  uicontrol(hObject)
end
set(handles.show_msg,'String','')
set(handles.save_button,'Enable','off')
% --- Executes during object creation, after setting all properties.
function hm_y_CreateFcn(hObject, eventdata, handles)
% hObject handle to hm_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in save_button.
function save_button_Callback(hObject, eventdata, handles)
% hObject handle to save_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
filename = strcat('rssi_data', num2str(handles.exp_number));
rssi_data=handles.data(handles.exp_number).rssi_data;
T=handles.data(handles.exp_number).T;
error=handles.error(handles.exp_number);
metric=handles.metric(handles.exp_number);
fudge = str2double(get(handles.fudge,'String'));
RSSI_edge = handles.RSSI_edge(handles.exp_number);
hidden_mote = [str2double(get(handles.hm_x,'String')) ...
               str2double(get(handles.hm_y,'String'))]
save(filename, 'rssi_data', 'T', 'error', 'metric', 'fudge', 'hidden_mote', 'RSSI_edge')
set(handles.estimate_pos,'Enable','on')
set(handles.show_msg,'String','Done saving.

function fudge_Callback(hObject, eventdata, handles)
% hObject handle to fudge (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
val = str2double(get(hObject,'String'));
% Determine whether val is a valid position
if isnumeric(val) && length(val)==1 && ...
  val > 0 && ...
  val <= 3
  % Enable the Plot button with its original name
  set(handles.estimate_pos,'String','Estimate position')
  set(handles.estimate_pos,'Enable','on')
else
  set(hObject,'String',
    ['Invalid entry']);
  % Disable the Plot button and change its string to say why
  set(handles.estimate_pos,'String','Cannot estimate')
  set(handles.estimate_pos,'Enable',
    'off')
  % Restore focus to the edit text box after error
  uicontrol(hObject)
end
set(handles.show_msg,'String','

% --- Executes during object creation,
% after setting all properties. %function fudge_CreateFcn(hObject, eventdata, handles)
% hObject handle to fudge (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB % handles empty - handles not created
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
  get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end

A.3 Process Data Func Code

function [D, cross_PLE, avg_TXtoall] = process_data_func(rssi_data)
%% Description
% This function receives RSSI data in the parameter 'rssi_data', which is
% written by the java program RssiDemo.
% The file has three columns per line. First one is the sender node number.
% The second one is the receiver (forwarding) node number. The third one is
% the RSSI value measured by the receiver node.
% The function separates the data corresponding to each sender-receiver
% pair, then returns PLE and other values.
% Author: Huthiifa Al Issa, Raul Ordonez, Atindra Mitra
% Date started: 1-Feb-2010
% Last modified: 10-May-2010
%% Initialize and read data
hidden_tx = 10; % node_id used for the "hidden" transmitter node
%% Separate data by node pairs (sender-receiver), going line by line
sender_count=0; % how many senders have been recorded so far
data = struct('sender',[],'receiver',[],'rssi',[],'dBm',[]);
senders = [];
% It duplicates the information in data.sender, except that
% in this structure, each sender is recorded individually
% together with all its associated receivers. But this data
% structure is not easily searchable, hence the need for
% the array.
for k = 1:length(rssi_data)
  found_sender = find(senders==rssi_data(k,1));
  if isempty(found_sender) % is it a sender node not yet recorded?
    sender_count = sender_count+1;
    senders(sender_count) = rssi_data(k,1);
    data(sender_count).sender = rssi_data(k,1);
    data(sender_count).receiver = rssi_data(k,2); % pad with -1 to denote empty space
    data(sender_count).rssi = rssi_data(k,3);
  else % OK, it's a known sender, now organize by receiver
    found_receiver = find(data(found_sender).receiver == rssi_data(k,2));
    if isempty(found_receiver) % is it a receiver not yet recorded?
      data(found_sender).receiver = rssi_data(k,2);
    else % OK, it's a known receiver, now organize by sender
      found_sender = find(data(found_sender).receiver == rssi_data(k,2));
      if isempty(found_sender) % is it a sender node not yet recorded?
        % pad with -1 to denote empty space
  
203
end
else
    idx = find(data(found_sender).rssi(found_receiver,:)==-1);
    % find all -1 paddings for current node
    if isempty(idx)
        % OK, no padding found, so pad all other nodes
        data(found_sender).rssi(:,size(data(found_sender).rssi,2)+1) = -1;
        % Now, record rssi value for the found receiver of current
        % sender
        data(found_sender).rssi(found_receiver,size(data(found_sender).rssi,2))...
            = rssi_data(k,3);
    else
        % record data in left-most position
        data(found_sender).rssi(found_receiver,min(idx)) = rssi_data(k,3);
    end
end
end
end

%% Convert RSSI to dBm
% Use this if RssiBase is configured to return PHY_ED_LEVEL (received
% message energy level)
for k=1:sender_count
    data(k).dBm = data(k).rssi - 91;
end
% Use this if RssiBase is configured to return true RSSI values
% for k=1:sender_count
% data(k).dBm = 3*data(k).rssi - 91;
% end

%% Order data by sender, then by receiver, for easier handling
% Always put hidden tx in first position
idx = find(senders==hidden_tx);
temp = senders(1);
senders(1)=senders(idx);
senders(idx)=temp;
temp = data(1);
data(1) = data(idx);
data(idx) = temp;
% Now do the rest of the senders
for k = 2:sender_count-1
    [y,idx]=min(senders(k:end));
    idx = idx+k-1;
    % Assuming we never have more than 255 nodes working at once
    temp = senders(k);
    senders(k)=senders(idx);
    senders(idx)=temp;
    temp = data(k);
    data(k) = data(idx);
    data(idx) = temp;
end
% Now that we ordered by senders, let’s order each entry by receivers
for j = 1:sender_count
    for k = 1:length(data(j).receiver)-1
        [y,idx]=min(data(j).receiver(k:end));
        idx = idx+k-1;
        temp = data(j).receiver(k);
        data(j).receiver(k) = data(j).receiver(idx);
        data(j).receiver(idx) = temp;
        temp = data(j).rssi(k,:);
        data(j).rssi(k,:) = data(j).rssi(idx,:);
        data(j).rssi(idx,:) = temp;
    end
end
%% Compute outputs
PLE = -13.82;
%load dist_model;
a=2.218065227302679e-15;
b=8.303884930269744;
c=2.361192808137663;
j=1; % only look at TX to receiver data
for k=1:length(data(j).receiver)
    PL = median(10.^(data(j).dBm(k,find(data(j).rssi(k,:)==-1))/10) );
    PL_dB = (10*log10(PL));
    D(k,1) = 10^((PL_dB/(10*PLE)));
    D(k,1) = mean( a*(-data(j).dBm(k,find(data(j).rssi(k,:)==-1))).b + c);
    avg_TXtoall(1,k) = data(j).receiver(k);
    avg_TXtoall(2,k) = mean(data(j).dBm(k,find(data(j).rssi(k,:)==-1)));
end
% this choice looks at data from node 2 to 3
from_node=2;
to_node=1;
PL = median(10.^(data(from_node).dBm(to_node,find(data(from_node)...
    .rssi(to_node,:)==-1))/10) );
PL_dB = 10*log10(PL);
PLE23 = \log_{10}\left( 10^{\frac{PL\_dB}{10}} \right);
% this choice looks at data from node ID 2 to 4
from_node = 2;
to_node = 2;
PL = \text{median}( 10^{\frac{\text{data(from_node).dBm(to_node, find(data(from_node)...}}}{10}} )/10 ) );
PL\_dB = 10^{\log_{10}( PL )};
PLE24 = 10^{\frac{PL\_dB}{10}};
% this choice looks at data from node ID 2 to 5
from_node = 2;
to_node = 3;
PL = \text{median}( 10^{\frac{\text{data(from_node).dBm(to_node, find(data(from_node)...}}}{10}} )/10 ) );
PL\_dB = 10^{\log_{10}( PL )};
PLE25 = 10^{\frac{PL\_dB}{10}};
% this choice looks at data from node ID 3 to 4
from_node = 3;
to_node = 2;
PL = \text{median}( 10^{\frac{\text{data(from_node).dBm(to_node, find(data(from_node)...}}}{10}} )/10 ) );
PL\_dB = 10^{\log_{10}( PL )};
PLE34 = 10^{\frac{PL\_dB}{10}};
% this choice looks at data from node ID 3 to 5
from_node = 3;
to_node = 3;
PL = \text{median}( 10^{\frac{\text{data(from_node).dBm(to_node, find(data(from_node)...}}}{10}} )/10 ) );
PL\_dB = 10^{\log_{10}( PL )};
PLE35 = 10^{\frac{PL\_dB}{10}};
% this choice looks at data from node ID 4 to 5
from_node = 4;
to_node = 3;
PL = \text{median}( 10^{\frac{\text{data(from_node).dBm(to_node, find(data(from_node)...}}}{10}} )/10 ) );
PL\_dB = 10^{\log_{10}( PL )};
PLE45 = 10^{\frac{PL\_dB}{10}};
cross\_PLE=[PLE23 PLE24 PLE25 PLE34 PLE35 PLE45]';
cross\_PLE=PLE1;

A.4 Read Basestation Code

function data = read_basestation(num_samples)
%% Description
% This function reads the USB port where the Iris base station is connected
% and directly reads the packets arriving. It extracts three items:
% from_node, to_node and rssi_value.
%% It returns an array with as many rows as the argument num_samples
% specifies.
% Author: Huthaifa Al Issa, Raul Ordonez
% Date started: 27-Apr-2010
% Last modified: 27-Apr-2010
% Initialize USB port
s = serial('COM4', 'BaudRate', 57600);
fopen(s);
counter = 0;
PACKETSIZE=22; % this size was determined by trial and error
data = [ ];
%% Read data, throwing away any bad packets
while (counter < num_samples)
  if (s.BytesAvailable > PACKETSIZE)
    packet = fread(s, PACKETSIZE);
    from_node = packet(14);
    to_node = packet(7);
    rssi = packet(12);
    if from_node == 10 & to_node == 2
      counter = counter + 1;
    % only increase counter for 10 to 2 packet
    end
    % the positions of these items was determined by trial and error, and
    % a lot of observations
    if from_node < 2 | from_node > 10 | to_node < 2 | to_node > 10
      fclose(s);
      fopen(s);
      counter = counter - 1;
      disp('Bad packet')
    else
      data = [data; from_node to_node rssi];
      msg = sprintf('Sample %d. From: %d To: %d RSSI:%d',
                       counter, from_node, to_node, rssi);
      disp(msg)
  end
end
fclose(s);
%Author: Huthaifa Al Issa, Raul Ordonez
clc
s = serial('/dev/tty.usbserial-XBSDY4SBB','BaudRate',57600);
fopen(s);
counter = 0;
PACKETSIZE=22;
while(counter<100)
    if (s.BytesAvailable > PACKETSIZE)
        packet = fread(s,PACKETSIZE);
        counter = counter+1;
        from_node = packet(14);
        to_node = packet(7);
        rssi = packet(12);
        [from_node to_node rssi]
        if from_node < 2 | from_node > 10 | to_node < 2 | to_node > 10
            disp('Bad packet')
            fclose(s);
            fopen(s);
        end
    end
end
fclose(s);

A.5 Try_Serial_Reading Code

A.6 ABC values

a=2.18065227302679e-15;\ 
b=8.303884930269744;\ 
c=2.361192808137663;
B.1 IRIS Mote Program: About This Software

USE:
1. Install the ’SendingMote’ program in a single mote to function as the “hidden” transmitter, to be located by cooperative RSSI value measurements. I use NODE_ID=10 for this mote in my experiments.
2. Install ’RssiBase’ on a base station attached to a computer via USB. This mote must have NODE_ID=1.
3. Install ’ForwardingMote’ on one or more motes that act as RSSI sensors. These motes measure the RSSI value as they receive the packages broadcast by ’SendingMote’, and then forward the package to the base station but now with the measured RSSI value inserted. Also, different from v1, now each forwarding mote also broadcasts to all other forwarding motes, which also measure the inter-mote RSSI values and then send to the base station.

AUTHOR:
Raul Ordonez, Huthaifa Al Issa Dept. Electrical and Computer Engineering University of Dayton
DATE:
25-Feb-2010 11-Mar-2010

B.1.1 How_To.Make_Nc_Code

**NOTE**
Use com6 to program if using the standalone (black) base station. Then, use
export MOTECOM=serial@COM7:iris
IF using the individual USB programming board with separate IRIS radio, use com8 to program, and
export MOTECOM=serial@COM9:iris
To program, use
make iris install mib520,com6
or (to give a specific ID to the node)
make iris install,NODE_ID mib520,com6

/* * Copyright (c) 2008 Dimas Abreu Dutra * All rights reserved */
/** * @author Dimas Abreu Dutra */
/** * @author Raul Ordonez, Huthaifa AL Issa * Dept. Electrical and Computer Engineering * University of Dayton * Started: 26-Jan-2010 * @date 28-Feb-2010 */

B.1.2 ReadME.BEFORE.Making

MAKE NOTES:
Be sure to use
make iris install,NODE_ID mib520,com8
with a different NODE_ID for each forwarding mote.
Whatever NODE_ID is chosen, it **must** be different from the following:
* 1 (reserved for base station mote NODE_ID)
* any sending mote NODE_ID
In all my tests, I use node ID’s in the range 2 to 5 for the forwarding motes.
B.1.3 RssiDemoMessages Code

```c
/*
 * Copyright (c) 2008 Dimas Abreu Dutra
 * All rights reserved
 */
/**
 * @author Dimas Abreu Dutra
 */
/**
 * @author Raul Ordonez, Huthaifa Al Issa
 * Dept. Electrical and Computer Engineering
 * University of Dayton
 * Started: 26-Jan-2010
 * @date 28-Feb-2010
 */
#ifndef RSSIDEMOMESSAGES_H__
#define RSSIDEMOMESSAGES_H__
enum {
    AM_RSSI_MSG = 6,
    AM_FORWARDTOTBASE = 6,
    AM_BASE_ID = 1,
    SEND_INTERVAL_MS = 1000,
    FWD_INTERVAL_MS = 1300
};
typedef nx_struct RssiMsg{
    nx_int16_t rssi;
    nx_uint16_t sender_id;
    nx_uint16_t forwarder_id;
    nx_bool forwarded;
    nx_uint16_t counter;
} RssiMsg;
#endif //RSSIDEMOMESSAGES_H__
```

B.2 ForwardingMote

B.2.1 ForwardToBaseAppC Code

```c
/**
 * Waits for a message to arrive from sender mote, then writes its own node id
 * and measured RSSI value in the message, and forwards it to the base station
 * mote for archival on the PC.
 * @author Raul Ordonez and Huthaifa Al Issa
 * Dept. Electrical and Computer Engineering
 * University of Dayton
 * Started: 26-Jan-2010
 * @date 25-Feb-2010
 */
#include <Timer.h>
#include "RssiDemoMessages.h"
configuration ForwardToBaseAppC {
    implementation {
        components MainC;
        components LedsC;
        components new TimerMillis() as Timer0;
        components ForwardToBaseC as App;
        components ActiveMessageC;
        components new AMSenderC(AM_RSSI_MSG);
        components new AMReceiverC(AM_RSSI_MSG);
        #ifdef __CC2420_H__
        components CC2420ActiveMessageC;
        App -> CC2420ActiveMessageC.CC2420Packet;
        #elif defined(PLATFORM_IRIS)
        components RF230ActiveMessageC;
        App -> RF230ActiveMessageC.PacketRSSI;
        #elif defined(TDA5250_MESSAGE_H)
        components Tda5250ActiveMessageC;
        App -> Tda5250ActiveMessageC.Tda5250Packet;
        #endif
        App.Boot -> MainC;
        App.Leds -> LedsC;
        App.Timer0 -> Timer0;
        App.Packet -> AMSenderC;
        App.AMPacket -> AMSenderC;
        App.AMControl -> ActiveMessageC;
        App.AMSend -> AMSenderC;
        App.Receive -> AMReceiverC;
    }
```
B.2.2 ForwardToBaseC Code

/**
 * Waits for a message to arrive from sender mote, then writes its own node id
 * and measured RSSI value in the message, and forwards it to the base station
 * mote for archival on the PC.
 * @author Raul Ordonez and Huthaifa Al Issa
 * Dept. Electrical and Computer Engineering
 * University of Dayton
 * Started: 26-Jan-2010
 * @date 25-Feb-2010
 */

#include "RssiDemoMessages.h"

module ForwardToBaseC {
  uses interface Boot;
  uses interface Leds;
  uses interface Packet;
  uses interface Timer<TMilli> as Timer0;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
  #ifdef __CC2420_H__
    uses interface CC2420Packet;
  #elif defined(TDA5250_MESSAGE_H)
    uses interface Tda5250Packet;
  #else
    uses interface PacketField<uint8_t> as PacketRSSI;
  #endif
}

implementation {
  uint16_t getRssi(message_t *msg);
  message_t pkt;
  bool busy = FALSE;

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();

    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();

    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  task void toggle() {
    call Leds.led0Toggle();
  }

  event void Boot.booted() {
    call AMControl.start();
  }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(FWD_INTERVAL_MS);
      post toggle();
    } else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
    // Broadcast packages to all other forwarding nodes so they can measure the
    // inter-mote RSSI values.
    event void Timer0.fired() {
      if (!busy) {
        RssiMsg* btrpkt = (RssiMsg*)(call Packet.getPayload(&pkt, sizeof(RssiMsg)));
        if (btrpkt == NULL) {
          return;
        }

        btrpkt->sender_id = TOS_NODE_ID;
        btrpkt->counter = 0;
        btrpkt->forwarded = FALSE;
        if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(RssiMsg)) == SUCCESS) {
          busy = TRUE;
        } else {
          busy = FALSE;
        }
      }
    }

    event void AMSend.sendDone(message_t* msg, error_t err) {
      busy = FALSE;
    }

    event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
RssiMsg* btrpkt = (RssiMsg*)payload;
// check if the message received is "fresh" or whether it has already been forwarded
// by some other mote
if (btrpkt->forwarded) {
    return msg;
}
// OK, we have a fresh message, so fill in the information before forwarding
btrpkt->forwarder_id = TOS_NODE_ID;
btrpkt->forwarded = TRUE;
btrpkt->rssi = getRssi(msg);
setLeds(btrpkt->counter);
// Now we are ready to send the message to the base station for archival
if (call AMSend.send(AM_BROADCAST_ADDR, msg, sizeof(RssiMsg)) == SUCCESS) {
    busy = TRUE;
    return msg;
}
#endif

#include "CC2420_H"
uint16_t getRssi(message_t *msg){
    return (uint16_t) call CC2420Packet.getRssi(msg);
}
#elif defined(CC1K_RADIO_MSG_H)
uint16_t getRssi(message_t *msg){
    cc1000_metadata_t *md = (cc1000_metadata_t*) msg->metadata;
    return md->strength_or_preamble;
}
#elif defined(PLATFORM_IRIS)
uint16_t getRssi(message_t *msg){
    if (call PacketRSSI.isSet(msg))
        return (uint16_t) call PacketRSSI.get(msg);
    else
        return 0xFFFF;
}
#elif defined(TDA5250_MESSAGE_H)
uint16_t getRssi(message_t *msg){
    return call Tda5250Packet.getSnr(msg);
}
#else
#error Radio chip not supported! This demo currently works only \n    for motes with CC1000, CC2420, RF230 or TDA5250 radios.
#endif

B.3 Build Iris Code

B.3.1 App

#define nx_struct struct
#define nx_union union
#define dbg(mode, format, ...) ((void)0)
#define dbg_clear(mode, format, ...) ((void)0)
#define dbg_active(mode) 0

#include "C:\Crossbow\cygwin\usr\local\avr\include\stdint.h" 3
typedef int int8_t __attribute((__mode__(__QI__)));
typedef unsigned int uint8_t __attribute((__mode__(__QI__)));
typedef int int16_t __attribute((__mode__(__HI__)));
typedef unsigned int uint16_t __attribute((__mode__(__HI__)));
typedef int int32_t __attribute((__mode__(__SI__)));
typedef unsigned int uint32_t __attribute((__mode__(__SI__)));
typedef int int64_t __attribute((__mode__(__DI__)));
typedef unsigned int uint64_t __attribute((__mode__(__DI__)));
#line 135
typedef int16_t intnptr_t;
typedef uint16_t uintnptr_t;
#line 152
typedef int8_t int_least8_t;
typedef uint8_t uint_least8_t;
typedef int16_t int_least16_t;
typedef uint16_t uint_least16_t;
typedef int32_t int_least32_t;
typedef uint32_t uint_least32_t;
typedef int64_t int_least64_t;
typedef uint64_t uint_least64_t;
#line 200
typedef int8_t int_fast8_t;
typedef uint8_t uint_fast8_t;
typedef int16_t int_fast16_t;
typedef uint16_t uint_fast16_t;
typedef int32_t int_fast32_t;
typedef uint32_t uint_fast32_t;
typedef int64_t int_fast64_t;
typedef uint64_t uint_fast64_t;

typedef int64_t intmax_t;
typedef uint64_t uintmax_t;

typedef int32_t int_farptr_t;
typedef uint32_t uint_farptr_t;

static __inline uint8_t __nesc_ntoh_uint8(const void *source);
static __inline uint8_t __nesc_hton_uint8(void *target, uint8_t value);
static __inline uint8_t __nesc_ntoh_leuint8(const void *source);
static __inline uint8_t __nesc_hton_leuint8(void *target, uint8_t value);
static __inline int8_t __nesc_ntoh_int8(const void *source);
static __inline int8_t __nesc_hton_int8(void *target, int8_t value);
static __inline uint16_t __nesc_ntoh_uint16(const void *source);
static __inline uint16_t __nesc_hton_uint16(void *target, uint16_t value);
static __inline uint16_t __nesc_ntoh_leuint16(const void *source);
static __inline uint16_t __nesc_hton_leuint16(void *target, uint16_t value);
static __inline int16_t __nesc_ntoh_int16(const void *source);
static __inline int16_t __nesc_hton_int16(void *target, int16_t value);
static __inline uint32_t __nesc_ntoh_uint32(const void *source);
static __inline uint32_t __nesc_hton_uint32(void *target, uint32_t value);
static __inline uint32_t __nesc_ntoh_leuint32(const void *source);
static __inline uint32_t __nesc_hton_leuint32(void *target, uint32_t value);
static __inline int32_t __nesc_ntoh_int32(const void *source);
static __inline int32_t __nesc_hton_int32(void *target, int32_t value);

typedef struct { char data[1]; } __attribute__((packed)) nx_int8_t;
typedef int8_t __nesc_nxbase_nx_int8_t;

typedef struct { char data[2]; } __attribute__((packed)) nx_int16_t;
typedef int16_t __nesc_nxbase_nx_int16_t;

typedef struct __nesc_unnamed4242 {
    int quot;
    int rem;
} div_t;

typedef struct __nesc_unnamed4243 {
    long quot;
    long rem;
} ldiv_t;

extern void *memset(void *, int, size_t);

typedef struct __nesc_unnamed4242 {
    int quot;
    int rem;
} __attribute__((packed)) nxle_int8_t;

typedef struct __nesc_unnamed4243 {
    long quot;
    long rem;
} __attribute__((packed)) nxle_int16_t;

typedef unsigned int size_t;

typedef unsigned int size_t;
typedef struct __nesc_unnamed4247 {
    uint8_t porf : 1;
    uint8_t extrf : 1;
    uint8_t borf : 1;
    uint8_t wdrf : 1;
    uint8_t jtrf : 1;
    uint8_t resv1 : 3;
} Atm128_MCUSR_t;

typedef struct __nesc_unnamed4248 {
    uint8_t srw00 : 1;
    uint8_t srw01 : 1;
    uint8_t srw10 : 1;
    uint8_t srw11 : 1;
    uint8_t srl : 3;
    uint8_t sre : 1;
} Atm128_XMCRA_t;

typedef struct __nesc_unnamed4249 {
    uint8_t xmm : 3;
    uint8_t resv1 : 4;
    uint8_t xmbk : 1;
} Atm128_XMCRB_t;

enum __nesc_unnamed4250 {
    ATM128_ADC_VREF_OFF = 0,
    ATM128_ADC_VREF_AVCC = 1,
    ATM128_ADC_VREF_RSVD,
    ATM128_ADC_VREF_2_56 = 3
};

enum __nesc_unnamed4251 {
    ATM128_ADC_RIGHT_ADJUST = 0,
    ATM128_ADC_LEFT_ADJUST = 1
};

enum __nesc_unnamed4252 {
    ATM128_ADC_SNGL_ADC0 = 0,
    ATM128_ADC_SNGL_ADC1,
    ATM128_ADC_SNGL_ADC2,
    ATM128_ADC_SNGL_ADC3,
    ATM128_ADC_SNGL_ADC4,
    ATM128_ADC_SNGL_ADC5,
    ATM128_ADC_SNGL_ADC6,
    ATM128_ADC_SNGL_ADC7,
    ATM128_ADC_DIFF_ADC00_10x,
    ATM128_ADC_DIFF_ADC10_10x,
    ATM128_ADC_DIFF_ADC00_200x,
    ATM128_ADC_DIFF_ADC10_200x,
    ATM128_ADC_DIFF_ADC22_10x,
    ATM128_ADC_DIFF_ADC32_10x,
    ATM128_ADC_DIFF_ADC22_200x,
    ATM128_ADC_DIFF_ADC32_200x,
    ATM128_ADC_DIFF_ADC01_1x,
    ATM128_ADC_DIFF_ADC11_1x,
    ATM128_ADC_DIFF_ADC21_1x,
    ATM128_ADC_DIFF_ADC31_1x,
    ATM128_ADC_DIFF_ADC41_1x,
    ATM128_ADC_DIFF_ADC51_1x,
    ATM128_ADC_DIFF_ADC61_1x,
    ATM128_ADC_DIFF_ADC71_1x,
    ATM128_ADC_DIFF_ADC02_1x,
    ATM128_ADC_DIFF_ADC12_1x,
    ATM128_ADC_DIFF_ADC22_1x,
    ATM128_ADC_DIFF_ADC32_1x,
    ATM128_ADC_DIFF_ADC42_1x,
    ATM128_ADC_DIFF_ADC52_1x,
    ATM128_ADC_SNGL_1_23,
    ATM128_ADC_SNGL_GND
};

typedef struct __nesc_unnamed4253 {
    uint8_t mux : 5;
    uint8_t adlar : 1;
    uint8_t refs : 2;
} Atm128Admux_t;

enum __nesc_unnamed4254 {
    ATM128_ADC_PRESCALE_2 = 0,
    ATM128_ADC_PRESCALE_2b,
    ATM128_ADC_PRESCALE_4,
    ATM128_ADC_PRESCALE_8,
    ATM128_ADC_PRESCALE_16,
    ATM128_ADC_PRESCALE_32,
    ATM128_ADC_PRESCALE_64,
    ATM128_ADC_PRESCALE_128,
    ATM128_ADC_PRESCALE
};
enum __nesc_unnamed4255 {
    ATM128_ADC_ENABLE_OFF = 0,
    ATM128_ADC_ENABLE_ON
};
enum __nesc_unnamed4256 {
    ATM128_ADC_START_CONVERSION_OFF = 0,
    ATM128_ADC_START_CONVERSION_ON
};
enum __nesc_unnamed4257 {
    ATM128_ADC_FREE_RUNNING_OFF = 0,
    ATM128_ADC_FREE_RUNNING_ON
};
enum __nesc_unnamed4258 {
    ATM128_ADC_INT_FLAG_OFF = 0,
    ATM128_ADC_INT_FLAG_ON
};
enum __nesc_unnamed4259 {
    ATM128_ADC_INT_ENABLE_OFF = 0,
    ATM128_ADC_INT_ENABLE_ON
};
typedef struct __nesc_unnamed4260 {
    uint8_t adps : 3;
    uint8_t adie : 1;
    uint8_t adif : 1;
    uint8_t adate : 1;
    uint8_t adsc : 1;
    uint8_t aden : 1;
} Atm128Adcsra_t;
typedef struct __nesc_unnamed4261 {
    uint8_t adts : 3;
    uint8_t mux5 : 1;
    uint8_t resv1 : 2;
    uint8_t acme : 1;
    uint8_t resv2 : 1;
} Atm128Adcsrb_t;
typedef uint8_t Atm128_ADCH_t;
typedef uint8_t Atm128_ADCL_t;
#define TMilli int notUsed;
#define T32khz int notUsed;
#define TMicro int notUsed;

enum __nesc_unnamed4265 {
    ATM128_CLK8_OFF = 0x0,
    ATM128_CLK8_NORMAL = 0x1,
    ATM128_CLK8_DIVIDE_8 = 0x2,
    ATM128_CLK8_DIVIDE_32 = 0x3,
    ATM128_CLK8_DIVIDE_64 = 0x4,
    ATM128_CLK8_DIVIDE_128 = 0x5,
    ATM128_CLK8_DIVIDE_1024 = 0x7
};
enum __nesc_unnamed4266 {
    ATM128_CLK16_OFF = 0x0,
    ATM128_CLK16_NORMAL = 0x1,
    ATM128_CLK16_DIVIDE_8 = 0x2,
    ATM128_CLK16_DIVIDE_64 = 0x3,
    ATM128_CLK16_DIVIDE_128 = 0x4,
    ATM128_CLK16_DIVIDE_256 = 0x6,
    ATM128_CLK16_DIVIDE_1024 = 0x7
};
enum __nesc_unnamed4267 {
    AVR_CLOCK_OFF = 0,
    AVR_CLOCK_ON = 1,
    AVR_CLOCK_DIVIDE_8 = 2
};
enum __nesc_unnamed4268 {
enum __nesc_unnamed4269 {
    ATM128_TIMER_COMPARE_NORMAL = 0,
    ATM128_TIMER_COMPARE_TOGGLE,
    ATM128_TIMER_COMPARE_CLEAR,
    ATM128_TIMER_COMPARE_SET
};

enum __nesc_unnamed4270 {
    ATM128_WAVE8_NORMAL = 0,
    ATM128_WAVE8_PWM,
    ATM128_WAVE8_CTC,
    ATM128_WAVE8_PWM_FAST
};

enum __nesc_unnamed4271 {
    ATM128_COMPARE_OFF = 0,
    ATM128_COMPARE_TOGGLE,
    ATM128_COMPARE_CLEAR,
    ATM128_COMPARE_SET
};

typedef union __nesc_unnamed4272 {
    uint8_t flat;
    struct __nesc_unnamed4273 {
        uint8_t wgm00 : 1;
        uint8_t wgm01 : 1;
        uint8_t resv1 : 2;
        uint8_t com0b0 : 1;
        uint8_t com0b1 : 1;
        uint8_t com0a0 : 1;
        uint8_t com0a1 : 1;
    } bits;
} Atm128_TCCR0A_t;

typedef union __nesc_unnamed4274 {
    uint8_t flat;
    struct __nesc_unnamed4275 {
        uint8_t cs00 : 1;
        uint8_t cs01 : 1;
        uint8_t cs02 : 2;
        uint8_t wgm02 : 1;
        uint8_t resv1 : 2;
        uint8_t foc0b : 1;
        uint8_t foc0a : 1;
    } bits;
} Atm128_TCCR0B_t;

typedef union __nesc_unnamed4276 {
    uint8_t flat;
    struct __nesc_unnamed4277 {
        uint8_t toie0 : 1;
        uint8_t ocie0a : 1;
        uint8_t ocie0e : 1;
        uint8_t resv1 : 5;
    } bits;
} Atm128_TIMSK0_t;

typedef union __nesc_unnamed4278 {
    uint8_t flat;
    struct __nesc_unnamed4279 {
        uint8_t tov0 : 1;
        uint8_t ocf0a : 1;
        uint8_t ocf0b : 1;
        uint8_t resv1 : 5;
    } bits;
} Atm128_TIFR0_t;
typedef union __nesc_unnamed4280 {
    uint8_t flat;
    struct __nesc_unnamed4281 {
        uint8_t tcr2bub : 1;
        uint8_t tcr2aub : 1;
        uint8_t ocr2bub : 1;
        uint8_t ocr2aub : 1;
        uint8_t tcn2ub : 1;
        uint8_t as2 : 1;
        uint8_t exclk : 1;
        uint8_t resv1 : 1;
    } bits;
} Atm128_ASSR_t;
#line 216
#line 206
typedef union __nesc_unnamed4282 {
    uint8_t flat;
    struct __nesc_unnamed4283 {
        uint8_t wgm20 : 1;
        uint8_t wgm21 : 1;
        uint8_t resv1 : 2;
        uint8_t comb : 2;
        uint8_t coma : 2;
    } bits;
} Atm128_TCCR2A_t;
#line 229
#line 219
typedef union __nesc_unnamed4284 {
    uint8_t flat;
    struct __nesc_unnamed4285 {
        uint8_t cs : 3;
        uint8_t wgm22 : 1;
        uint8_t resv1 : 2;
        uint8_t foc2b : 1;
        uint8_t foc2a : 1;
    } bits;
} Atm128_TCCR2B_t;
#line 241
#line 232
typedef union __nesc_unnamed4286 {
    uint8_t flat;
    struct __nesc_unnamed4287 {
        uint8_t toie : 1;
        uint8_t ociea : 1;
        uint8_t ocieb : 1;
        uint8_t resv1 : 5;
    } bits;
} Atm128_TIMSK2_t;
#line 253
#line 244
typedef union __nesc_unnamed4288 {
    uint8_t flat;
    struct __nesc_unnamed4289 {
        uint8_t tov : 1;
        uint8_t ocrfa : 1;
        uint8_t ocrfb : 1;
        uint8_t resv1 : 5;
    } bits;
} Atm128_TIFR2_t;
#line 266
#line 257
typedef union __nesc_unnamed4289 {
    uint8_t flat;
    struct __nesc_unnamed4291 {
        uint8_t wgm01 : 2;
        uint8_t comc : 2;
        uint8_t comb : 2;
        uint8_t coma : 2;
    } bits;
} Atm128_TCCRA_t;
#line 279
#line 269
typedef union __nesc_unnamed4292 {
    uint8_t flat;
    struct __nesc_unnamed4293 {
        uint8_t cs : 3;
        uint8_t wgm23 : 2;
        uint8_t resv1 : 1;
        uint8_t lces : 1;
        uint8_t icnc : 1;
    } bits;
} Atm128_TCCRB_t;
#line 282
#line 281
typedef union __nesc_unnamed4294 {
    uint8_t flat;
struct __nesc_unnamed4295 {
  uint8_t resv1 : 5;
  uint8_t focc : 1;
  uint8_t focb : 1;
  uint8_t foca : 1;
} bits;
} Atm128_TCCRC_t;
#line 306
#line 294
typedef union __nesc_unnamed4296 {
  uint8_t flat;
  struct __nesc_unnamed4297 {
    uint8_t toie : 1;
    uint8_t ociea : 1;
    uint8_t ocieb : 1;
    uint8_t ociec : 1;
    uint8_t resv1 : 1;
    uint8_t icie : 1;
    uint8_t resv2 : 2;
  } bits;
} Atm128_TIMSK_t;
#line 321
#line 309
typedef union __nesc_unnamed4298 {
  uint8_t flat;
  struct __nesc_unnamed4299 {
    uint8_t tov : 1;
    uint8_t ocfa : 1;
    uint8_t ocfb : 1;
    uint8_t ocfc : 1;
    uint8_t resv1 : 1;
    uint8_t icf : 1;
    uint8_t resv2 : 2;
  } bits;
} Atm128_TIFR_t;
#line 333
#line 324
typedef union __nesc_unnamed4300 {
  uint8_t flat;
  struct __nesc_unnamed4301 {
    uint8_t psrsync : 1;
    uint8_t psrasy : 1;
    uint8_t resv1 : 5;
    uint8_t tsm : 1;
  } bits;
} Atm128_GTCCR_t;
typedef uint8_t Atm128_TCNT1H_t;
typedef uint8_t Atm128_TCNT1L_t;
typedef uint8_t Atm128_TCNT3H_t;
typedef uint8_t Atm128_TCNT3L_t;
typedef uint8_t Atm128_TCNT3H_t;
typedef uint8_t Atm128_TCNT3L_t;
typedef uint8_t Atm128_TCNT4H_t;
typedef uint8_t Atm128_TCNT4L_t;
typedef uint8_t Atm128_TCNT5H_t;
typedef uint8_t Atm128_TCNT5L_t;
typedef uint8_t Atm128_TCNT5H_t;
typedef uint8_t Atm128_TCNT5L_t;
typedef uint8_t Atm128_OCR1AH_t;
typedef uint8_t Atm128_OCR1AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_OCR3AH_t;
typedef uint8_t Atm128_OCR3AL_t;
typedef uint8_t Atm128_ICR1H_t;
typedef uint8_t Atm128_ICR1L_t;
typedef uint8_t Atm128_ICR3H_t;
typedef uint8_t Atm128_ICR3L_t;
typedef uint8_t Atm128_ICR4H_t;
typedef uint8_t Atm128_ICR4L_t;
typedef uint8_t Atm128_ICR5H_t;
typedef uint8_t Atm128_ICR5L_t;

typedef struct __nesc_unnamed4302 {
}
# line 74
T64khz;
typedef struct __nesc_unnamed4303 {
}
# line 75
T128khz;
typedef struct __nesc_unnamed4304 {
}
# line 76
T2mhz;
typedef struct __nesc_unnamed4305 {
}
# line 77
T4mhz;
# line 147
typedef TMicro TOne;
typedef TMicro TThree;
typedef uint32_t counter_one_overflow_t;
typedef uint16_t counter_three_overflow_t;
enum __nesc_unnamed4306 {
    MICA_PRESCALER_ONE = ATM128_CLK16_DIVIDE_8,
    MICA_DIVIDE_ONE_FOR_32KHZ_LOG2 = 5,
    MICA_PRESCALER_THREE = ATM128_CLK16_DIVIDE_8,
    MICA_DIVIDE_THREE_FOR_MICRO_LOG2 = 0,
    EXT_STANDBY_T0_THRESHOLD = 12
};
enum __nesc_unnamed4307 {
    PLATFORM_MHZ = 8
};
# 55
"C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\platforms\micaz\hardware.h"
enum __nesc_unnamed4308 {
    CHANNEL_THERMISTOR = ATM128_ADC_SNGL_ADC1
};
enum __nesc_unnamed4309 {
    PLATFORM_BAUDRATE = 57600L
};
# 21
"C:\Crossbow\cygwin\home\ordonez\rssi_ordonez_v2\RssiDemoMessages.h"
enum __nesc_unnamed4310 {
    AM_RSSIMSG = 6,
    AM_FORWARDTOBASE = 6,
    AM_BASE_ID = 1,
    SEND_INTERVAL_MS = 1000,
    FWD_INTERVAL_MS = 1300
};
# line 29
typedef nx_struct RssiMsg {
    nx_int16_t rssi;
    nx_uint16_t sender_id;
    nx_uint16_t forwarder_id;
    nx_bool forwarded;
    nx_uint16_t counter;
} __attribute__((packed)) RssiMsg;
# 32
"C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\types\Leds.h"
enum __nesc_unnamed4311 {
    LEDS_LED0 = 1 << 0,
    LEDS_LED1 = 1 << 1,
    LEDS_LED2 = 1 << 2,
    LEDS_LED3 = 1 << 3,
    LEDS_LED4 = 1 << 4,
    LEDS_LED5 = 1 << 5,
    LEDS_LED6 = 1 << 6,
    LEDS_LED7 = 1 << 7
};
# 42
"C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154Packet.h"
# line 27
typedef nx_struct ieee154_header_t {
    nxle_uint8_t length;
    nxle_uint16_t fcf;
    nxle_uint8_t dst;
    nxle_uint16_t destpan;
    nxle_uint16_t dest;
    nxle_uint16_t src;
    nxle_uint8_t network;
    nxle_uint8_t type;
} __attribute__((packed)) ieee154_header_t;
# line 45

218
typedef nx_struct ieee154_footer_t {
  nxle_uint16_t crc;
} __attribute__((packed)) ieee154_footer_t;

enum ieee154_fcf_enums {
  IEEE154_FCF_FRAME_TYPE = 0,
  IEEE154_FCF_SECURITY_ENABLED = 3,
  IEEE154_FCF_FRAME_PENDING = 4,
  IEEE154_FCF_ACK_REQ = 5,
  IEEE154_FCF_INTRAPAN = 6,
  IEEE154_FCF_DEST_ADDR_MODE = 10,
  IEEE154_FCF_SRC_ADDR_MODE = 14
};

enum ieee154_fcf_type_enums {
  IEEE154_TYPE_BEACON = 0,
  IEEE154_TYPE_DATA = 1,
  IEEE154_TYPE_ACK = 2,
  IEEE154_TYPE_MAC_CMD = 3,
  IEEE154_TYPE_MASK = 7
};

enum ieee154_fcf_addr_mode_enums {
  IEEE154_ADDR_NONE = 0,
  IEEE154_ADDR_SHORT = 2,
  IEEE154_ADDR_EXT = 3,
  IEEE154_ADDR_MASK = 3
};

#define __nesc_unnamed4312 {
  AM_BROADCAST_ADDR = 0xffff
};

#define __nesc_unnamed4313 {
  TOS_AM_GROUP = 0x22,
  TOS_AM_ADDRESS = 1
};

#define __nesc_unnamed4314 {
  HDLC_FLAG_BYTE = 0x7e,
  HDLC_CTLESC_BYTE = 0x7d
};

#define __nesc_unnamed4315 {
  TOS_SERIAL_ACTIVE_MESSAGE_ID = 0,
  TOS_SERIAL_CC1000_ID = 1,
  TOS_SERIAL_802_15_4_ID = 2,
  TOS_SERIAL_UNKNOWN_ID = 255
};

#define __nesc_unnamed4316 {
  SERIAL_PROTO_ACK = 67,
  SERIAL_PROTO_PACKET_ACK = 68,
  SERIAL_PROTO_PACKET_NOACK = 69,
  SERIAL_PROTO_PACKET_UNKNOWN = 255
};

enum ieee154_fcf_addr_mode_enums {
  IEEE154_ADDR_NONE = 0,
  IEEE154_ADDR_SHORT = 2,
  IEEE154_ADDR_EXT = 3,
  IEEE154_ADDR_MASK = 3
};

#define __nesc_unnamed4317 {
  TOS_SERIAL_ACTIVE_MESSAGE_ID = 0,
  TOS_SERIAL_CC1000_ID = 1,
  TOS_SERIAL_802_15_4_ID = 2,
  TOS_SERIAL_UNKNOWN_ID = 255
};

#define __nesc_unnamed4318 {
  SERIAL_PROTO_ACK = 67,
  SERIAL_PROTO_PACKET_ACK = 68,
  SERIAL_PROTO_PACKET_NOACK = 69,
  SERIAL_PROTO_PACKET_UNKNOWN = 255
};

#define __nesc_unnamed4319 {
  TOS_SERIAL_ACTIVE_MESSAGE_ID = 0,
  TOS_SERIAL_CC1000_ID = 1,
  TOS_SERIAL_802_15_4_ID = 2,
  TOS_SERIAL_UNKNOWN_ID = 255
};

#define __nesc_unnamed4320 {
  SERIAL_PROTO_ACK = 67,
  SERIAL_PROTO_PACKET_ACK = 68,
  SERIAL_PROTO_PACKET_NOACK = 69,
  SERIAL_PROTO_PACKET_UNKNOWN = 255
};

typedef struct radio_stats {
  uint8_t version;
  ...
typedef nx struct serial_header {
    nx_am_addr_t dest;
    nx_am_addr_t src;
    nx_uint8_t length;
    nx_am_group_t group;
    nx_am_id_t type;
} __attribute__((packed)) serial_header_t;

typedef nx struct serial_packet {
    serial_header_t header;
    nx_uint8_t data[];
} __attribute__((packed)) serial_packet_t;

typedef nx struct serial_metadata {
    nx_uint8_t ack;
} __attribute__((packed)) serial_metadata_t;

typedef union message_header {
    rf230packet_header_t rf230;
    serial_header_t serial;
} message_header_t;

typedef union message_footer {
    rf230packet_footer_t rf230;
} message_footer_t;

typedef union message_metadata {
    rf230packet_metadata_t rf230;
} message_metadata_t;

enum rf230_registers_enum {
    RF230_TRX_STATUS = 0x01,
    RF230_TRX_STATE = 0x02,
    RF230_TRX_CTRL_0 = 0x03,
    RF230_PHY_TX_PWR = 0x05,
    RF230_PHY_RSSI = 0x06,
    RF230_PHY_ED_LEVEL = 0x07,
    RF230_PHY_CC_CCA = 0x08,
    RF230_CCA_THRES = 0x09,
    RF230_IRQ_MASK = 0x0E,
    RF230_IRQ_STATUS = 0x0F,
    RF230_VREG_CTRL = 0x10,
    RF230_BATMON = 0x11,
    RF230_XOSC_CTRL = 0x12,
    RF230_PLL_CF = 0x1A,
    RF230_PLL_DCU = 0x1B,
    RF230_PART_NUM = 0x1C,
    RF230_VERSION_NUM = 0x1D,
    RF230_MAN_ID_0 = 0x1E,
    RF230_MAN_ID_1 = 0x1F,
    RF230_SHORT_ADDR_0 = 0x20,
    RF230_SHORT_ADDR_1 = 0x21,
    RF230_PAN_ID_0 = 0x22,
    RF230_PAN_ID_1 = 0x23,
    RF230_IEEE_ADDR_0 = 0x24,
    RF230_IEEE_ADDR_1 = 0x25,
    RF230_IEEE_ADDR_2 = 0x26,
    RF230_IEEE_ADDR_3 = 0x27,
    RF230_IEEE_ADDR_4 = 0x28,
    RF230_IEEE_ADDR_5 = 0x29,
    RF230_IEEE_ADDR_6 = 0x2A,
    RF230_IEEE_ADDR_7 = 0x2B,
    RF230_XAH_CTRL = 0x2C,
RF230_CSMA_SEED_0 = 0x2D,
RF230_CSMA_SEED_1 = 0x2E
}

enum rf230_trx_register_enums {
  RF230_CCA_DONE = 1 << 7,
  RF230_CCA_STATUS = 1 << 6,
  RF230_TRX_STATUS_MASK = 0x1F,
  RF230_P_ON = 0,
  RF230_BUSY_RX = 1,
  RF230_BUSY_TX = 2,
  RF230_RX_ON = 6,
  RF230_TRX_OFF = 8,
  RF230_PLL_ON = 9,
  RF230_SLEEP = 15,
  RF230_BUSY_RX_AACK = 16,
  RF230_BUSY_TX_AACK = 17,
  RF230_RX_AACK_ON = 22,
  RF230_TX_AACK_ON = 25,
  RF230_RX_ON_NOCLOCK = 28,
  RF230_AACK_ON_NOCLOCK = 29,
  RF230_BUSY_RX_AACK_NOCLOCK = 30,
  RF230_STATE_TRANSITION_IN_PROGRESS = 31,
  RF230_TRAC_STATUS = 3 << 5,
  RF230_TRAC_CHANNEL_ACCESS_FAILURE = 3 << 5,
  RF230_TRAC_NO_ACK = 5 << 5,
  RF230_TRX_CMD_MASK = 0x1F,
  RF230_NOP = 0,
  RF230_TX_START = 2,
  RF230_FORCE_TRX_OFF = 3
};

enum rf230_phy_register_enums {
  RF230_TX_AUTO_CRC_ON = 1 << 7,
  RF230_TX_PWR_MASK = 0x0F,
  RF230_TX_PWR_DEFAULT = 0,
  RF230_CCA_REQUEST = 1 << 7,
  RF230_CCA_MODE_0 = 0 << 5,
  RF230_CCA_MODE_1 = 1 << 5,
  RF230_CCA_MODE_2 = 2 << 5,
  RF230_CCA_MODE_3 = 3 << 5,
  RF230_CHANNEL_DEFAULT = 11,
  RF230_CHANNEL_MASK = 0x1F,
  RF230_CCA_CS_THRES_SHIFT = 4,
  RF230_CCA_ED_THRES_SHIFT = 0
};

enum rf230_irq_register_enums {
  RF230_IRQ_BAT_LOW = 1 << 7,
  RF230_IRQ_TRX_UR = 1 << 6,
  RF230_IRQ_TRX_END = 1 << 3,
  RF230_IRQ_RX_START = 1 << 2,
  RF230_IRQ_PLL_UNLOCK = 1 << 1,
  RF230_IRQ_PLL_LOCK = 1 << 0
};

enum rf230_control_register_enums {
  RF230_AVREG_EXT = 1 << 7,
  RF230_AVDD_OK = 1 << 6,
  RF230_DVREG_EXT = 1 << 3,
  RF230_DVDD_OK = 1 << 2,
  RF230_BATMON_OK = 1 << 5,
  RF230_BATMON_VHR = 1 << 4,
  RF230_BATMON_VTH_MASK = 0x0F,
  RF230_XTAL_MODE_OFF = 0 << 4,
  RF230_XTAL_MODE_EXTERNAL = 4 << 4,
  RF230_XTAL_MODE_INTERNAL = 15 << 4
};

enum rf230_pll_register_enums {
  RF230_PLL_CF_START = 1 << 7,
  RF230_PLL_DCU_START = 1 << 7
};

enum rf230_spi_command_enums {
  RF230_CMD_REGISTER_READ = 0x80,
  RF230_CMD_REGISTER_WRITE = 0xC0,
  RF230_CMD_REGISTER_MASK = 0x3F,
  RF230_CMD_FRAME_READ = 0x20,
  RF230_CMD_FRAME_WRITE = 0x60,
  RF230_CMD_SRAM_READ = 0x00,
  RF230_CMD_SRAM_WRITE = 0x40
};

#define __nesc_unnamed4317 {
  RF230_TRX_CTRL_0_VALUE = 0,
  RF230_CCA_MODE_VALUE = RF230_CCA_MODE_3,
  RF230_CCA_THRES_VALUE = 0xC7
};

typedef TOne TRF230;
typedef nx_int32_t timesync_relative_t;
typedef uint32_t timesync_absolute_t;

enum nesc_unnamed4318 {
    ATM128_SPI_CLK_DIVIDE_4 = 0,
    ATM128_SPI_CLK_DIVIDE_16 = 1,
    ATM128_SPI_CLK_DIVIDE_64 = 2,
    ATM128_SPI_CLK_DIVIDE_128 = 3
};

typedef struct __nesc_unnamed4319 {
    uint8_t spie : 1;
    uint8_t spe : 1;
    uint8_t dord : 1;
    uint8_t mstr : 1;
    uint8_t cpol : 1;
    uint8_t cpha : 1;
    uint8_t spr : 2;
} Atm128SPIControl_s;

typedef union __nesc_unnamed4320 {
    uint8_t flat;
    Atm128SPIControl_s bits;
} Atm128SPIControl_t;

typedef struct __nesc_unnamed4321 {
    uint8_t spif : 1;
    uint8_t wcol : 1;
    uint8_t rsvd : 5;
    uint8_t spi2x : 1;
} Atm128SPIStatus_s;

typedef union __nesc_unnamed4322 {
    uint8_t flat;
    Atm128SPIStatus_s bits;
} Atm128SPIStatus_t;

typedef uint8_t Atm128_SPDR_t;

typedef union __nesc_unnamed4323 {
    uint8_t spif : 1;
    uint8_t wcol : 1;
    uint8_t rsvd : 5;
    uint8_t spi2x : 1;
} Atm1281AlarmAsyncP$0$AlarmSprecision_tag;

typedef uint32_t Atm1281AlarmAsyncP$0$AlarmSsize_type;

typedef union __nesc_unnamed4324 {
    uint8_t spif : 1;
    uint8_t wcol : 1;
    uint8_t rsvd : 5;
    uint8_t spi2x : 1;
} Atm1281AlarmAsyncP$0$CounterSprecision_tag;

typedef uint32_t Atm1281AlarmAsyncP$0$CounterSsize_type;

typedef uint8_t HplAtm1281Timer2AsyncP$CompareSsize_type;

typedef TMilli /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$precision_tag;

typedef /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$AlarmSprecision_tag;

typedef uint32_t /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$AlarmSsize_type;

typedef /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$CounterSprecision_tag;

typedef uint8_t /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$CounterSsize_type;

typedef uint8_t HplAtm1281Timer2AsyncP$CompareSsize_type;

typedef TMilli /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$precision_tag;

typedef /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$AlarmSprecision_tag;

typedef uint32_t /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$AlarmSsize_type;

typedef /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$CounterSprecision_tag;

typedef uint8_t /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$CounterSsize_type;

typedef uint8_t HplAtm1281Timer2AsyncP$CompareSsize_type;

typedef TMilli /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$precision_tag;

typedef /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$AlarmSprecision_tag;

typedef uint32_t /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$AlarmSsize_type;

typedef /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$CounterSprecision_tag;

typedef uint8_t /*HilTimerMilliC.AlarmToTimerC*/
    AlarmToTimerC$0$CounterSsize_type;

222
typedef TMilli /*HilTimerMilliC.VirtualizeTimerC*/ VirtualizeTimerC$0$precision_tag;
typedef /*HilTimerMilliC.VirtualizeTimerC*/ VirtualizeTimerC$0$Timer$precision_tag /*HilTimerMilliC.VirtualizeTimerC*/;
typedef /*HilTimerMilliC.CounterToLocalTimeC*/ CounterToLocalTimeC$0$Counter$size_type;
typedef TMilli /*HilTimerMilliC.CounterToLocalTimeC*/ CounterToLocalTimeC$0$LocalTime$precision_tag;
typedef TMilli /*HilTimerMilliC.CounterToLocalTimeC*/ CounterToLocalTimeC$0$CounterFrom$precision_tag;
typedef TMilli /*HilTimerMilliC.CounterToLocalTimeC*/ CounterToLocalTimeC$0$Counter$precision_tag;
typedef /*HilTimerMilliC.VirtualizeTimerC*/ VirtualizeTimerC$0$TimerFrom$precision_tag;
typedef /*HilTimerMilliC.VirtualizeTimerC*/ VirtualizeTimerC$0$Timer$precision_tag;
typedef /*HilTimerMilliC.CounterToLocalTimeC*/ CounterToLocalTimeC$0$Counter$precision_tag;
typedef /*HilTimerMilliC.CounterToLocalTimeC*/ CounterToLocalTimeC$0$Counter$size_type;
TransformCounterC$0$Counter$size_type;
typedef TMicro /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$precision_tag;
typedef /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$precision_tag
/*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$LocalTime$precision_tag;
typedef /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$precision_tag
/*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$Counter$precision_tag;
typedef uint32_t /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$Counter$size_type;
typedef TRF230 RadioAlarmP$Alarm$precision_tag;
typedef uint16_t RadioAlarmP$Alarm$size_type;
typedef TMilli TrafficMonitorLayerP$Timer$precision_tag;
typedef uint16_t RandomMlcgC$SeedInit$parameter;
typedef TMicro RF230LayerP$BusyWait$precision_tag;
typedef uint16_t RF230LayerP$BusyWait$size_type;
typedef TMilli RF230LayerP$PacketTimestamp$precision_tag;
typedef uint32_t RF230LayerP$PacketTimestamp$size_type;
typedef uint8_t RF230LayerP$PacketTransmitPower$value_type;
typedef TRF230 RF230LayerP$LocalTime$precision_tag;
typedef uint8_t RF230LayerP$PacketTimeSyncOffset$value_type;
typedef uint8_t RF230LayerP$PacketLinkQuality$value_type;
typedef HplRF230P$Capture$size_type;
typedef TOne /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$frequency_tag;
typedef uint16_t /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$timer_size;
typedef /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$Alarm$precision_tag;
typedef /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$Alarm$size_type;
typedef /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$size_type;
typedef /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Timer$timer_size;
enum /*HplRF230C.AlarmC*/AlarmOne16C$0$__nesc_unnamed4324 {
AlarmOne16C$0$COMPARE_ID = 0U
};
typedef TMicro BusyWaitMicroC$BusyWait$precision_tag;
typedef uint16_t TMicro BusyWaitMicroC$BusyWait$size_type;
enum AMQueueP$__nesc_unnamed4325 {
AMQueueP$NUM_CLIENTS = 1U
};
static void HplAtm128GeneralIOC.PortA.Bit4(clr(void);
static void HplAtm128GeneralIOC.PortA.Bit6
HplAtm128GeneralIOC.PortA.Bit6(set(void);
static void HplAtm128GeneralIOC.PortA.Bit6
HplAtm128GeneralIOC.PortA.Bit6(clr(void);
static void HplAtm128GeneralIOC.PortB.Bit0
HplAtm128GeneralIOC.PortB.Bit0(makeOutput(void);
static void HplAtm128GeneralIOC.PortB.Bit0
HplAtm128GeneralIOC.PortB.Bit0(set(void);
static void HplAtm128GeneralIOC.PortB.Bit0
HplAtm128GeneralIOC.PortB.Bit0(clr(void);
static void HplAtm128GeneralIOC.PortB.Bit2
HplAtm128GeneralIOC.PortB.Bit2(makeOutput(void);
static void HplAtm128GeneralIOC.PortB.Bit3
HplAtm128GeneralIOC.PortB.Bit3(makeInput(void);
static void HplAtm128GeneralIOC.PortB.Bit7
HplAtm128GeneralIOC.PortB.Bit7(makeOutput(void);
static void HplAtm128GeneralIOC.PortB.Bit7
HplAtm128GeneralIOC.PortB.Bit7(set(void);
static void HplAtm128GeneralIOC.PortB.Bit7
HplAtm128GeneralIOC.PortB.Bit7(clr(void);
static void HplAtm128GeneralIOC.PortD.Bit4
HplAtm128GeneralIOC.PortD.Bit4(makeInput(void);
static void HplAtm128GeneralIOC.PortD.Bit6
HplAtm128GeneralIOC.PortD.Bit6(makeInput(void);
static void HplAtm128GeneralIOC.PortD.Bit6
HplAtm128GeneralIOC.PortD.Bit6(clr(void);
static void HplAtm128GeneralIOC.PortD.Bit6
HplAtm128GeneralIOC.PortD.Bit6(set(void);
static void HplAtm128GeneralIOC.PortD.Bi
HplAtm128GeneralIOC.PortD.Bi(set(void);
static void HplAtm128GeneralIOC.PortD.Bi
HplAtm128GeneralIOC.PortD.Bi(clr(void);
static void HplAtm128GeneralIOC.PortD.Bi
HplAtm128GeneralIOC.PortD.Bi(makeOutput(void);
static void LedsP$Leds$led0Off(void);
static void LedsP$Leds$led0Toggle(void);
static void LedsP$Leds$led1On(void);
static void LedsP$Leds$led1Off(void);
static void LedsP$Leds$led2Off(void);
static void LedsP$Leds$led0On(void);
static void /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Timer$stop(void);
static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer$runTask(void);
static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$TimerFrom$fired(void);
static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$default$fired(uint8_t arg_0x1a7c0cb8);
static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$startPeriodic(uint8_t arg_0x1a7c0cb8,
uint32_t arg_0x1a5d0490);
static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$stop(uint8_t arg_0x1a7c0cb8);
static void /*HilTimerMilliC.CounterToLocalTimeC*/
CounterToLocalTimeC$0$Counter$overflow(void);
static void ForwardToBaseC$AMControl$startDone(error_t arg_0x1a894678);
static void ForwardToBaseC$AMControl$stopDone(error_t arg_0x1a8994678);
static void ForwardToBaseC$AMControl$booted(void);
static void ForwardToBaseC$AMControl$sendDone(message_t *arg_0x1a8821c0, error_t arg_0x1a882348);
static void RF230ActiveMessageP$TrafficMonitorConfig$timerTick(void);
static void RF230ActiveMessageP$TrafficMonitorConfig$checkPacket(message_t *arg_0x1a954b08);
static void RF230ActiveMessageP$TrafficMonitorConfig$createAckPacket(message_t *arg_0x1a945e08, message_t *arg_0x1a944010);
static bool RF230ActiveMessageP$SoftwareAckConfig$
requiresAckWait(message_t *arg_0x1a948ef0);

static bool RF230ActiveMessageP$SoftwareAckConfig$
verifyAckPacket(message_t *arg_0x1a945120, message_t *arg_0x1a9452d0);

static uint16_t RF230ActiveMessageP$SoftwareAckConfig$
getAckTimeout(void);

static void RF230ActiveMessageP$SoftwareAckConfig$
setAckReceived(message_t *arg_0x1a947450, bool arg_0x1a9475d8);

static bool RF230ActiveMessageP$SoftwareAckConfig$
requiresAckReply(message_t *arg_0x1a9458d0);

static void RF230PacketP$PacketRSSI$clear
(message_t *arg_0x1a8a8718);

static RF230PacketP$PacketRSSI$value_type RF230PacketP$PacketRSSI$get
(message_t *arg_0x1a8a8190);

static bool RF230PacketP$PacketRSSI$clear
(message_t *arg_0x1a8a8190);

static void RF230PacketP$PacketRSSI$set
(message_t *arg_0x1a8a8718, RF230PacketP$PacketRSSI$Value_type arg_0x1a8a8718);

static void RF230PacketP$PacketRSSI$clear
(message_t *arg_0x1a8a8718);

static void *RF230PacketP$Packet$getPayload
(message_t *arg_0x1a862358, uint8_t arg_0x1a8624e0);
static void HplAtm1281Timer1P$Timer$setScale(uint8_t arg_0x1a670da8);
static void HplAtm1281Timer1P$Timer$set(HplAtm1281Timer1P$Timer$timer_size arg_0x1a671120);
static void HplAtm1281Timer1P$Timer$start(void);

static error_t /*InitOneP.InitOne*/ Atm128TimerInitC$0$Init$init(void);
static void /*CounterOne16C.NCounter*/ Atm128CounterC$0$Counter$size_type /*CounterOne16C.NCounter*/ Atm128CounterC$0$Counter$get(void);
static bool /*CounterOne16C.NCounter*/ Atm128CounterC$0$Counter$isOverflowPending(void);
static void /*CounterOne16C.NCounter*/ Atm128CounterC$0$Timer$overflow(void);

static void /*LocalTimeMicroC.TransformCounterC*/ TransformCounterC$0$CounterFrom$overflow(void);
static uint32_t /*LocalTimeMicroC.CounterToLocalTimeC*/ CounterToLocalTimeC$1$LocalTime$get(void);
static void /*LocalTimeMicroC.CounterToLocalTimeC*/ CounterToLocalTimeC$1$Counter$overflow(void);

static void RadioAlarmP$RadioAlarm$default$fired(uint8_t arg_0x1abccd98);
static void RadioAlarmP$RadioAlarm$wait(uint8_t arg_0x1abccd98, uint16_t arg_0x1a995d20);
static void RadioAlarmP$RadioAlarm$cancel(uint8_t arg_0x1abccd98);
static bool RadioAlarmP$RadioAlarm$isFree(uint8_t arg_0x1abccd98);
static uint16_t RadioAlarmP$RadioAlarm$getNow(uint8_t arg_0x1abccd98);
static void RadioAlarmP$Alarm$fired(void);
static void RadioAlarmP$Tasklet$run(void);
static void TaskletC$Tasklet$schedule(void);
static void MessageBufferLayerP$stateDoneTask$runTask(void);
static void MessageBufferLayerP$sendTask$runTask(void);
static void MessageBufferLayerP$deliverTask$runTask(void);
static void MessageBufferLayerP$RadioSend$ready(void);
static void MessageBufferLayerP$RadioSend$sendDone(error_t arg_0x1ac23360);
static void MessageBufferLayerP$Tasklet$run(void);
static void UniqueLayerP$SubSend$sendDone(message_t *arg_0x1abe2010, error_t arg_0x1abe2198);
static void UniqueLayerP$Neighborhood$evicted(uint8_t arg_0x1ac93270);
static void UniqueLayerP$Init$init(void);
static void NeighborhoodP$NeighborhoodFlag$clearAll(uint8_t arg_0x1acba860);
static bool NeighborhoodP$NeighborhoodFlag$get(uint8_t arg_0x1ac900b8);
static void NeighborhoodP$Neighborhood$insertNode(am_addr_t arg_0x1ac95c40);
static bool UniqueLayerP$SubReceive$header(message_t *arg_0x1ac3d3e0);
static error_t UniqueLayerP$Send$send(message_t *arg_0x1abe5ef8, uint8_t arg_0x1abe4090);
static void TrafficMonitorLayerP$SubSend$ready(void);
static void TrafficMonitorLayerP$TrafficMonitorConfig$channelError(void);
static void TrafficMonitorLayerP$SubReceive$receive(message_t *arg_0x1ac3d3e0);
static void TrafficMonitorLayerP$SubSend$sendDone(error_t arg_0x1ac23360);
static void TrafficMonitorLayerP$RadioSend$sendDone(message_t *arg_0x1ac3d3e0);
static void TrafficMonitorLayerP$RadioSend$sendDone(error_t arg_0x1ac23360);
static void TrafficMonitorLayerP$RadioReceive$receive(message_t *arg_0x1ac3d3e0);
static error_t TrafficMonitorLayerP$RadioState$turnOn(void);
#line 58
static void TrafficMonitorLayerP$SubState$done(void);
static void TrafficMonitorLayerP$Neighborhood$evicted(uint8_t arg_0x1ac93270);
static void TrafficMonitorLayerP$Tasklet$run(void);
static error_t TrafficMonitorLayerP$RadioSend$send(message_t *arg_0x1ac24c88);
static void TrafficMonitorLayerP$startStopTimer$runTask(void);
static void RandomCollisionLayerP$SubSend$ready(void);
static void RandomCollisionLayerP$SubSend$sendDone(error_t arg_0x1ac23360);
static message_t *RandomCollisionLayerP$SubReceive$receive(message_t *arg_0x1ac3dab8);
static bool RandomCollisionLayerP$SubReceive$header(message_t *arg_0x1ac3d3ab8);
static void RandomCollisionLayerP$CalcNextRandom$runTask(void);
static void SoftwareAckLayerP$SubSend$sendDone(error_t arg_0x1ac23360);
static message_t *SoftwareAckLayerP$SubReceive$receive(message_t *arg_0x1ac3dab8);
static bool SoftwareAckLayerP$SubReceive$header(message_t *arg_0x1ac3d3ab8);
static void SoftwareAckLayerP$SoftwareInit$init(void);
static void RF230LayerP$RadioState$turnOn(void);
static void RF230LayerP$RadioAlarm$fired(void);
static void RF230LayerP$RadioSend$send(message_t *arg_0x1ac24c88);
static void RF230LayerP$RadioReceive$receive(message_t *arg_0x1ac3dab8);
static void RF230LayerP$RadioCCA$default$done(error_t arg_0x1ac3a920);
static void RF230LayerP$RadioState$turnOn(void);
static void RF230LayerP$RadioState$turnOn(void);
static void RF230LayerP$RadioState$turnOn(void);
static bool Atm128SpiP$Resource$isOwner(
  uint8_t arg_0x1ae4d8e0);
static void HplAtm128SpiP$SPI$sleep(void);
static void HplAtm128SpiP$SPI$initMaster(void);
static void HplAtm128SpiP$SPI$setMasterBit(bool arg_0x1ae43438);
static void HplAtm128SpiP$SPI$enableInterrupt(bool arg_0x1ae46c70);
static uint8_t HplAtm128SpiP$SPI$read(void);
static void HplAtm128SpiP$SPI$setMasterDoubleSpeed(bool arg_0x1ae40db0);
static void HplAtm128SpiP$SPI$setClock(uint8_t arg_0x1ae42c58);
static void HplAtm128SpiP$SPI$setClockPolarity(bool arg_0x1ae43c90);
static void HplAtm128SpiP$SPI$write(uint8_t arg_0x1ae46218);
static void HplAtm128SpiP$SPI$enableSpi(bool arg_0x1ae45480);
static void HplAtm128SpiP$SPI$setClockPhase(bool arg_0x1ae42490);
static bool FcfsResourceQueueC$0$FcfsQueue$isEmpty(void);
static bool FcfsResourceQueueC$0$FcfsQueue$isEnqueued(resource_client_id_t arg_0x1af2f7f8);
static resource_client_id_t FcfsResourceQueueC$0$FcfsQueue$dequeue(void);
static void SimpleArbiterP$0$ResourceRequested$default$requested(uint8_t arg_0x1af22688);
static void SimpleArbiterP$0$ResourceRequested$default$immediateRequested(uint8_t arg_0x1af22688);
static void SimpleArbiterP$0$ResourceConfigure$default$unconfigure(uint8_t arg_0x1af20188);
static void SimpleArbiterP$0$ResourceConfigure$default$configure(uint8_t arg_0x1af20188);
static error_t SimpleArbiterP$0$Resource$release(uint8_t arg_0x1af23d10);
static error_t SimpleArbiterP$0$Resource$immediateRequest(uint8_t arg_0x1af23d10);
```c
" \system\AMQueueImplP.nc"
uint8_t arg_0x1b019b68,
# 89 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Send.nc"
message_t *arg_0x1abe2010, error_t arg_0x1abe2198);
# 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\TaskBasic.nc"
static void */AMQueueP.AMQueueImplP*/
AMQueueImplP$0$errorTask$runTask(void);
#line 64
static void */AMQueueP.AMQueueImplP*/
AMQueueImplP$0$CancelTask$runTask(void);
# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Init.nc"
static error_t PlatformP$MoteInit$init(void);
#line 51
static error_t PlatformP$MeasureClock$init(void);
# 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\platforms\mica\PlatformP.nc"
static inline void PlatformP$power_init(void);
static inline error_t PlatformP$Init$init(void);
# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Init.nc"
static error_t MotePlatformP$SubInit$init(void);
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GeneralIO.nc"
static void MotePlatformP$Serial1DPin$makeInput(void);
#line 30
static void MotePlatformP$Serial1DPin$clr(void);
# 26 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\platforms\micaz\MotePlatformP.nc"
static inline void MotePlatformP$platformInit$init(void);
static inline error_t MotePlatformP$platformInit$init(void);
# 46 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void */HplAtm128General10C.PortA.Bit0*/
HplAtm128General10PinP$0$IO$set(void);
static __inline void */HplAtm128General10PinP$0$IO$clr(void);
static __inline void */HplAtm128General10PinP$0$IO$makeOutput(void);
#line 52
static __inline void */HplAtm128General10PinP$0$IO$makeInput(void);
static __inline void */HplAtm128General10PinP$0$IO$toggle(void);
static __inline void */HplAtm128General10PinP$0$IO$makeOutput(void);
```
HplAtm128GeneralIOPinP$11$IO$makeInput(void);
#line 46
static __inline void / *HplAtm128GeneralIOC.PortB.Bit7*/
HplAtm128GeneralIOPinP$15$IO$set(void);
static __inline void / *HplAtm128GeneralIOC.PortB.Bit7*/
HplAtm128GeneralIOPinP$15$IO$clr(void);
static __inline void / *HplAtm128GeneralIOC.PortB.Bit7*/
HplAtm128GeneralIOPinP$15$IO$makeOutput(void);
#line 47
static __inline void / *HplAtm128GeneralIOC.PortD.Bit4*/
HplAtm128GeneralIOPinP$28$IO$clr(void);
static __inline void / *HplAtm128GeneralIOC.PortD.Bit4*/
HplAtm128GeneralIOPinP$28$IO$makeInput(void);
#line 47
static __inline void / *HplAtm128GeneralIOC.PortD.Bit6*/
HplAtm128GeneralIOPinP$30$IO$clr(void);
static __inline void / *HplAtm128GeneralIOC.PortD.Bit6*/
HplAtm128GeneralIOPinP$30$IO$makeInput(void);

enum MeasureClockC$__nesc_unnamed4326 {
    MeasureClockC$MAGIC = 488 / (16 / PLATFORM_MHZ)
};
uint16_t MeasureClockC$cycles;
static inline error_t MeasureClockC$Init$init(void);

# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Init.nc"
static error_t RealMainP$SoftwareInit$init(void);

# 49 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Boot.nc"
static void RealMainP$Boot$booted(void);

# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Init.nc"
static error_t RealMainP$PlatformInit$init(void);

# 46 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\RealMainP.nc"
int main(void) ;

# 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\SchedulerBasicP.nc"
static void SchedulerBasicP$TaskBasic$runTask(uint8_t arg_0x1a1206b8);

# 59 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\SchedulerBasicP.nc"
enum SchedulerBasicP$__nesc_unnamed4327 {
    SchedulerBasicP$NUM_TASKS = 12U,
    SchedulerBasicP$NO_TASK = 255
};


t static inline void SchedulerBasicP$Scheduler$init(void);
st static bool SchedulerBasicP$Scheduler$runNextTask(void);

t static bool SchedulerBasicP$Scheduler$runNextTask(void);

t static bool SchedulerBasicP$Scheduler$runNextTask(void);

# 86 __inline uint8_t SchedulerBasicP$Scheduler$init(void);
# 113 __inline uint8_t SchedulerBasicP$Scheduler$runNextTask(void);
# 138 __inline void SchedulerBasicP$Scheduler$init(void);
    static __inline uint8_t SchedulerBasicP$Scheduler$runNextTask(void);
# 146 __inline void SchedulerBasicP$Scheduler$runNextTask(void);
    static bool SchedulerBasicP$Scheduler$runNextTask(void);

# 84 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\SchedulerBasicP.nc"
static mcu_power_t McuSleepC$McuPowerOverride$lowestState(void);

# 80 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm1281\McuSleepC.nc"
const uint8_t McuSleepC$atm128PowerBits[ATM128_POWER_DOWN + 1] = {
    0,
    1 << 1,
    1 << 1
};
static void /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$stop(void);
static void /*AlarmCounterMilliP.Atm128AlarmAsyncC.
Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$startAt(uint32_t nt0, uint32_t ndt);
static inline uint32_t /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$getNow(void);
static inline uint32_t /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$getAlarm(void);
static inline void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$overflow(void);
static void HplAtm1281Timer2AsyncP$Compare$fired(void);
static void HplAtm1281Timer2AsyncP$Timer$overflow(void);
static inline uint8_t HplAtm1281Timer2AsyncP$Timer$get(void);
static inline void HplAtm1281Timer2AsyncP$TimerCtrl$setControlA(uint8_t x);
static inline void HplAtm1281Timer2AsyncP$TimerCtrl$setControlB(uint8_t x);
static inline uint8_t HplAtm1281Timer2AsyncP$TimerCtrl$getInterruptFlag(void);
static inline void HplAtm1281Timer2AsyncP$Compare$start(void);
static inline uint8_t HplAtm1281Timer2AsyncP$Compare$get(void);
static inline void HplAtm1281Timer2AsyncP$Compare$set(uint8_t t);
static __inline void HplAtm1281Timer2AsyncP$stabiliseTimer2(void);
static inline mcu_power_t HplAtm1281Timer2AsyncP$McuPowerOverride$lowestState(void);

void __vector_13(void) __attribute((signal)) ;
void __vector_15(void) __attribute((signal)) ;

/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$get(void);
# 62 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\Atm1281AlarmAsyncC.Atm1281AlarmAsyncP/*.nsc"
uint8_t /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$set;
uint32_t /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$t0;
#line 63
uint32_t /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$dt;
uint32_t /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$base;
enum /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$_nesc_unnamed4328 {
  Atm1281AlarmAsyncP$0$MINDT = 2,
  Atm1281AlarmAsyncP$0$MAXT = 230
};
static void /*AlarmCounterMilliP.Atm128AlarmAsyncC.
Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setInterrupt(void);
static inline error_t /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Init$init(void);
#line 101
static inline void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setOcr2A(uint8_t n);
#line 117
static void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setInterrupt(void);
#line 176
static inline void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$fired(void);
#line 188
static uint32_t /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$getNow(void);
#line 231
static inline void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$stop(void);
static void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$startAt(uint32_t nt0, uint32_t ndt);
static inline uint32_t /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$getAlarm(void);
static inline void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$overflow(void);
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Init$init(void);
static inline void HplAtm1281Timer2AsyncP$TimerAsync$setTimer2Asynchronous(void);
#line 258
static inline int HplAtm1281Timer2AsyncP$TimerAsync$compareABusy(void);
#line 258
static / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$,Alarm$size_type /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$getNow(void);
#line 92
static void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$size_type /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$startAt(/*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$size_type arg_0x1a631548,
/*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$size_type arg_0x1a6316d8);
#line 105
static void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$size_type /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$stop(void);
#line 110
static void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$startOneShotAt(uint32_t arg_0x1a5e3738, uint32_t arg_0x1a5e38c8);
#line 67
static void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$stop(void);
#line 61
static void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$size_type /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$startAt(uint32_t t0, uint32_t dt, bool oneshot);
#line 105
static void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$stop(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$fired(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$fired$runTask(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Timer$fired(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Timer$stop(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$stop(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$fired(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$fired$runTask(void);
static inline void / *HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$size_type /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Alarm$startAt(uint32_t t0, uint32_t dt, bool oneshot);

# 72 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\TaskBasic.nc"
static error_t / *HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer$postTask(void);

# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\VirtualizeTimerC.nc"
enum /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$__nesc_unnamed4330 {
    #line 60
    VirtualizeTimerC$0$updateFromTimer = 0U
}

# 63 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\VirtualizeTimerC.nc"
enum /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$__nesc_unnamed4330 {
    #line 60
    VirtualizeTimerC$0$updateFromTimer = 1U
}
typedef int /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$__nesc_sillytask_updateFromTimer[/*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer];

enum /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$__nesc_unnamed4331 {
  VirtualizeTimerC$0$NUM_TIMERS = 2,
  VirtualizeTimerC$0$END_OF_LIST = 255
};

typedef struct /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$__nesc_unnamed4332 {
  uint32_t t0;
  uint32_t dt;
  bool isoneshot : 1;
  bool isrunning : 1;
  bool _reserved : 6;
} /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer_t;

/*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer_t /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$m_timers[/*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$NUM_TIMERS];

static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$fireTimers(uint32_t now);

static inline void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer$runTask(void);

static inline void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$TimerFrom_fired(void);

static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$startTimer(uint8_t num, uint32_t t0, uint32_t dt, bool isoneshot);

static inline void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$startPeriodic(uint8_t num, uint32_t dt);

static inline void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$stop(uint8_t num);

static inline void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$default_fired(uint8_t num);

# 47 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\CounterToLocalTimeC.nc"
static inline void /*HilTimerMilliC.CounterToLocalTimeC*/
CounterToLocalTimeC$0$Counter$overflow(void);

# 53 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\Timer.nc"
static void ForwardToBaseC$Timer0$startPeriodic(uint32_t arg_0x1a5d0490);

# 83 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\SplitControl.nc"
static error_t ForwardToBaseC$AMControl$start(void);

# 115 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Packet.nc"
static void *ForwardToBaseC$Packet$getPayload(message_t *arg_0x1a862358, uint8_t arg_0x1a8624e0);

# 69 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\AMSend.nc"
static error_t ForwardToBaseC$AMSend$send(am_addr_t arg_0x1a884010, uint8_t arg_0x1a884198);

# 35 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\PacketField.nc"
static ForwardToBaseC$PacketRSSI$value_type ForwardToBaseC$PacketRSSI$get(message_t *arg_0x1a8a8190);

# 29 static bool ForwardToBaseC$PacketRSSI$isSet(message_t *arg_0x1a890c68);

# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\TaskBasic.nc"
static error_t ForwardToBaseC$Toggle$postTask(void);

# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Leds.nc"
static void ForwardToBaseC$Leds$led0Off(void);
static void ForwardToBaseC$Leds$led0Toggle(void);
static void ForwardToBaseC$Leds$led1On(void);
static void ForwardToBaseC$Leds$led1Off(void);

# 45 static void ForwardToBaseC$Leds$led2Off(void);

# 78 static void ForwardToBaseC$Leds$led0On(void);

# 53 "C:\Crossbow\cygwin\home\ordonez\rssi_ordonez_v2\ForwardingMote\ForwardToBaseC.nc"
enum ForwardToBaseC$__nesc_unnamed4333 {
typedef int ForwardToBaseC$__nesc_sillytask_toggle[ForwardToBaseC$toggle];
static inline uint16_t ForwardToBaseC$getRssi(message_t *msg);
message_t ForwardToBaseC$pkt;
bool ForwardToBaseC$busy = FALSE;
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
message_t ForwardToBaseC$pkt;
bool ForwardToBaseC$busy = FALSE;
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
#line 53 ForwardToBaseC$toggle = 2U
static inline uint16_t ForwardToBaseC$getRssi(message_t *msg);
message_t ForwardToBaseC$pkt;
bool ForwardToBaseC$busy = FALSE;
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
static inline void ForwardToBaseC$setLeds(uint16_t val);
static inline void ForwardToBaseC$toggle$runTask(void);
static inline void ForwardToBaseC$Boot$booted(void);
static inline void ForwardToBaseC$AMControl$startDone(error_t err);
static inline void ForwardToBaseC$AMControl$stopDone(error_t err);
static inline void ForwardToBaseC$Timer0$fired(void);
static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err);
static inline message_t *ForwardToBaseC$receive(message_t *msg, void *payload, uint8_t len);
getMeta(message_t *msg);
static inline uint8_t RF230ActiveMessageP$RF230Config$getHeaderLength(void);
static inline uint8_t RF230ActiveMessageP$RF230Config$getMaxLength(void);
static inline bool RF230ActiveMessageP$RF230Config$requiresRssiCca(message_t *msg);
static inline bool RF230ActiveMessageP$SoftwareAckConfig$requiresAckWait(message_t *msg);
static inline bool RF230ActiveMessageP$SoftwareAckConfig$isAckPacket(message_t *data, message_t *ack);
static inline bool RF230ActiveMessageP$SoftwareAckConfig$verifyAckPacket(message_t data, message_t *ack);
static inline bool RF230ActiveMessageP$SoftwareAckConfig$requiresAckReply(message_t *msg);
static inline void RF230ActiveMessageP$SoftwareAckConfig$createAckPacket(message_t *data, message_t *ack);
static void RF230ActiveMessageP$SoftwareAckConfig$setAckReceived(message_t *msg, bool acked);
static inline uint16_t RF230ActiveMessageP$SoftwareAckConfig$getAckTimeout(void);
static inline void RF230ActiveMessageP$UniqueConfig$reportChannelError(void);
static inline uint8_t RF230ActiveMessageP$UniqueConfig$getSequenceNumber(message_t *msg);
static inline void RF230ActiveMessageP$UniqueConfig$setSequenceNumber(message_t *msg, uint8_t dsn);
static inline am_addr_t RF230ActiveMessageP$UniqueConfig$getSender(message_t *msg);
static inline void RF230ActiveMessageP$UniqueConfig$reportChannelError(void);
static inline error_t RF230ActiveMessageP$ActiveMessageConfig$checkPacket(message_t *msg);

#line 184
enum RF230ActiveMessageP$__nesc_unnamed4334 {
    RF230ActiveMessageP$TRAFFIC_UPDATE_PERIOD = 100,
    RF230ActiveMessageP$TRAFFIC_MAX_BYTES = (uint16_t )(RF230ActiveMessageP$TRAFFIC_UPDATE_PERIOD * 100UL / 32)
};
static inline uint16_t RF230ActiveMessageP$TrafficMonitorConfig$getUpdatePeriod(void);
static inline am_addr_t RF230ActiveMessageP$TrafficMonitorConfig$getSender(message_t *msg);
static inline void RF230ActiveMessageP$TrafficMonitorConfig$timerTick(void);

# 229
static inline uint16_t RF230ActiveMessageP$RandomCollisionConfig$getMinimumBackoff(void);
static inline uint16_t RF230ActiveMessageP$RandomCollisionConfig$getInitialBackoff(message_t *msg);
static inline uint16_t RF230ActiveMessageP$RandomCollisionConfig$getCongestionBackoff(message_t *msg);
static inline uint16_t RF230ActiveMessageP$RandomCollisionConfig$getTransmitBarrier(message_t *msg);

# 250
static inline void RF230ActiveMessageP$RadioAlarm$fired(void);

# 291
static inline void RF230ActiveMessageP$SlottedCollisionConfig$default$timerTick(void);

# 73 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154Packet.nc"
static void RF230PacketP$IEEE154Packet$createDataFrame(message_t *arg_0x1a972768);

# 49
static void RF230PacketP$IEEE154Packet$setLength(message_t *arg_0x1a975ee0, uint8_t arg_0x1a973088);
# 44
static uint8_t RF230PacketP$IEEE154Packet$getLength(message_t *arg_0x1a9759d0);

enum RF230PacketP$__nesc_unnamed4335 {
    RF230PacketP$PACKET_LENGTH_INCREASE = sizeof(rf230packet_header_t ) - 1
    + sizeof(ieee154_footer_t )
};
static __inline bool IEEE154PacketP$IEEE154Packet$ getAckRequired(message_t *msg);
#line 154
static __inline uint8_t IEEE154PacketP$IEEE154Packet$ getDSN(message_t *msg);
static __inline void IEEE154PacketP$IEEE154Packet$ setDSN(message_t *msg, uint8_t dsn);
static __inline void IEEE154PacketP$IEEE154Packet$ setDestPan(message_t *msg, uint16_t pan);
static __inline uint16_t IEEE154PacketP$IEEE154Packet$ getDestAddr(message_t *msg);
static __inline void IEEE154PacketP$IEEE154Packet$ setDestAddr(message_t *msg, uint16_t addr);
static __inline uint8_t IEEE154PacketP$IEEE154Packet$ get6LowPan(message_t *msg);
static __inline void IEEE154PacketP$IEEE154Packet$ set6LowPan(message_t *msg, uint8_t network);
static __inline am_id_t IEEE154PacketP$IEEE154Packet$ getType(message_t *msg);
static __inline void IEEE154PacketP$IEEE154Packet$ setType(message_t *msg, am_id_t type);
static __inline bool IEEE154PacketP$IEEE154Packet$ requiresAckWait(message_t *msg);
static bool IEEE154PacketP$IEEE154Packet$ requiresAckReply(message_t *msg);
#line 238
static __inline am_addr_t IEEE154PacketP$AMPacket$ address(Void);
static __inline am_group_t IEEE154PacketP$AMPacket$ destination(Void);
static __inline void IEEE154PacketP$AMPacket$ setDestination(message_t *msg, am_addr_t addr);
static __inline am_addr_t IEEE154PacketP$AMPacket$ destination(Void);
static __inline void IEEE154PacketP$AMPacket$ setDestination(message_t *msg, am_addr_t addr);
static __inline am_id_t IEEE154PacketP$AMPacket$ isForMe(message_t *msg);
static __inline am_group_t IEEE154PacketP$AMPacket$ type(message_t *msg);
static __inline void IEEE154PacketP$AMPacket$ setType(message_t *msg, am_group_t grp);
# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\ActiveMessageAddressC.nc"
static am_addr_t ActiveMessageAddressC$amAddress(void);
static am_group_t ActiveMessageAddressC$amGroup(void);
#line 82
static am_addr_t ActiveMessageAddressC$amAddress(void);
static am_group_t ActiveMessageAddressC$amGroup(void);
#line 95
static am_addr_t ActiveMessageAddressC$amAddress(void);
# 49 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\timer\HplAtm128Compare.nc"
static void HplAtm1281Timer1P$CompareB$fired(void);
# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\timer\HplAtm128Timer1P.nc"
static inline uint16_t HplAtm1281Timer1P$Timer$get(void);
static inline void HplAtm1281Timer1P$Timer$set(uint16_t t);
static inline void HplAtm1281Timer1P$Timer$setScale(uint8_t s);
static inline bool HplAtm1281Timer1P$TimerCtrl$setControlB(uint8_t x);
#line 106
static inline void HplAtm1281Timer1P$TimerCtrl$setControlB(uint8_t x);
TransformCounterC$0$NUM_UPPER_BITS = 8 * sizeof(/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$to_size_type) - 8 * sizeof(/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$from_size_type) + 0,
TransformCounterC$0$OVERFLOW_MASK = /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$NUM_UPPER_BITS ? ((/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$upper_count_type)2 << ((/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$NUM_UPPER_BITS - 1)) - 1 : 0);

static /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$to_size_type /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$Counter$get(void);

#line 122
static inline void /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$CounterFrom$overflow(void);

# 53 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\timer\Counter.nc"
static /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$Counter$size_type /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$Counter$get(void);

# 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\timer\CounterToLocalTimeC.nc"
static inline uint32_t /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$LocalTime$get(void);

# 48 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioAlarm.nc"
static void RadioAlarmPSRadioAlarm$fired(uint8_t arg_0x1abccd98);

# 98 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\timer\Alarm.nc"
static RadioAlarmP$Alarm$size_type RadioAlarmP$Alarm$getNow(void);

#line 55
static void RadioAlarmPSAlarm$start
(RadioAlarmP$Alarm$size_type arg_0x1a633438);
static void RadioAlarmPSAlarm$stop(void);

# 48 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioAlarm.nc"
static void RadioAlarmP$Tasklet$run(void);

# 43 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioAlarmP.nc"
static inline void RadioAlarmPSRadioAlarm$default$fired(uint8_t id);
static inline void RadioAlarmPSRadioAlarm$wait(uint8_t id, uint16_t timeout);
static inline void RadioAlarmPSRadioAlarm$cancel(uint8_t id);

# 37 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\Tasklet.nc"
static void TaskletC$Tasklet$schedule(void);

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\TaskletC.nc"
static void TaskletC$Tasklet$schedule(void);

# 37 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\Tasklet.nc"
static void TaskletC$Tasklet$suspend(void);
static void TaskletC$Tasklet$resume(void);

static void TaskletC$Tasklet$schedule(void);

# 101
static void TaskletC$Tasklet$schedule(void);

# 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\\Send.nc"
static void ActiveMessageLayerC$SubSend$send(message_t *arg_0x1abe4090);

249
static void IEEE154NetworkLayerP$IEEE154Packet$set6LowPan(message_t *arg_0x1a982518, uint8_t arg_0x1a9826a0);
#line 167
static uint8_t IEEE154NetworkLayerP$IEEE154Packet$get6LowPan(message_t *arg_0x1a982010);
# 67 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Receive.nc"
static message_t *IEEE154NetworkLayerP$Receive$receive(message_t *arg_0x1a897260, void *arg_0x1a897400, uint8_t arg_0x1a897588);
# 48 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154NetworkLayerP.nc"
static inline error_t IEEE154NetworkLayerP$Send$send(message_t *msg, uint8_t len);
#line 69
static inline void IEEE154NetworkLayerP$SubSend$sendDone(message_t *msg, error_t error);
static inline message_t *IEEE154NetworkLayerP$SubReceive$receive(message_t *msg, void *payload, uint8_t len);
static inline message_t *IEEE154NetworkLayerP$NonTinyosReceive$default$receive(uint8_t network, message_t *msg, void *payload, uint8_t len);
# 92 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\SplitControl.nc"
static void MessageBufferLayerP$SplitControl$startDone(error_t arg_0x1a894678);
#line 117
static void MessageBufferLayerP$SplitControl$stopDone(error_t arg_0x1a892250);
# 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RadioState.nc"
static error_t MessageBufferLayerP$RadioStemSturnOn(void);
# 89 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Send.nc"
static void MessageBufferLayerP$Send$sendDone(message_t *arg_0x1a892198);
# 67 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Packet.nc"
static uint8_t MessageBufferLayerP$Packet$payloadLength(message_t *arg_0x1a865ea8);
#line 115
static void *MessageBufferLayerP$Packet$getPayload(message_t *arg_0x1a862358, uint8_t arg_0x1a8624e0);
#line 83
static void MessageBufferLayerP$Packet$maxPayloadLength(void);
# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\TaskBasic.nc"
static error_t MessageBufferLayerP$stateDoneTask$postTask(void);
# 67 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\TaskBasic.nc"
static error_t MessageBufferLayerP$sendTaskpostcodeTask(void);
# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RadioSend nc"
static error_t MessageBufferLayerP$RadioSend$send(message_t *arg_0x1a82c88);
# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\Tasklet.nc"
static void MessageBufferLayerP$Tasklet$suspend(void);
static void MessageBufferLayerP$Tasklet$resume(void);
# 101 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\MessageBufferLayerP.nc"
enum MessageBufferLayerP$__nesc_unnamed4339 {
  #line 101
  MessageBufferLayerP$stateDoneTask = 3U
};
#line 101
typedef int MessageBufferLayerP$_nesc_sillytask_stateDoneTask[MessageBufferLayerP$stateDoneTask];
#line 139
enum MessageBufferLayerP$_nesc_unnamed4340 {
  #line 139
  MessageBufferLayerP$sendTask = 4U
};
#line 139
typedef int MessageBufferLayerP$_nesc_sillytask_sendTask[MessageBufferLayerP$sendTask];
#line 267
enum MessageBufferLayerP$__nescUnnamed4341 {
    #line 267  
    MessageBufferLayerP$deliverTask = 5U  
};  
#line 267  
typedef int MessageBufferLayerP$__nesc_sillytask_deliverTask  
(MessageBufferLayerP$deliverTask);  
#line 52  
uint8_t MessageBufferLayerP$state;  
enum MessageBufferLayerP$__nescUnnamed4342 {
    MessageBufferLayerP$STATE_READY = 0,  
    MessageBufferLayerP$STATE_TX_PENDING = 1,  
    MessageBufferLayerP$STATE_TX_SEND = 2,  
    MessageBufferLayerP$STATE_TX_DONE = 3,  
    MessageBufferLayerP$STATE_TURN_ON = 4,  
    MessageBufferLayerP$STATE_TURN_OFF = 5  
};

static error_t MessageBufferLayerP$SplitControl$start(void);
static inline void MessageBufferLayerP$stateDoneTask$runTask(void);
static inline void MessageBufferLayerP$RadioState$done(void);
static inline void MessageBufferLayerP$RadioState$ready(void);

message_t *MessageBufferLayerP$txMsg;
error_t MessageBufferLayerP$txError;
uint8_t MessageBufferLayerP$retries;
enum MessageBufferLayerP$__nescUnnamed4343 {
    #line 137  
    MessageBufferLayerP$MAX_RETRIES = 5  
};

message_t MessageBufferLayerP$receiveQueueData
[MessageBufferLayerP$RECEIVE_QUEUE_SIZE];
message_t *MessageBufferLayerP$receiveQueue
[MessageBufferLayerP$RECEIVE_QUEUE_SIZE];
uint8_t MessageBufferLayerP$receiveQueueHead;
uint8_t MessageBufferLayerP$receiveQueueSize;
static inline error_t MessageBufferLayerP$SoftwareInit$init(void);
static inline bool MessageBufferLayerP$RadioReceive$header(message_t *msg);
static inline void MessageBufferLayerP$deliverTask$runTask(void);
static inline message_t *MessageBufferLayerP$RadioReceive$receive(message_t *msg);

error_t UniqueLayerP$SubSend$send(message_t *arg_0x1abe5ef8, uint8_t arg_0x1abe4090);
bool UniqueLayerP$NeighborhoodFlag$get(uint8_t arg_0x1ac900b8);
void UniqueLayerP$Neighborhood$insertNode(am_addr_t arg_0x1ac95c40);
bool UniqueLayerP$RadioReceive$header(message_t *arg_0x1ac3d3e0);

error_t UniqueLayerP$SoftwareInit$init(void);
bool UniqueLayerP$RadioReceive$header(message_t *arg_0x1ac3d3e0);
bool UniqueLayerP$RadioReceive$header(message_t *arg_0x1acb3d3e0);
bool UniqueLayerP$RadioReceive$header(message_t *arg_0x1a941cd8, uint8_t arg_0x1a941e60);
void UniqueLayerP$UniqueConfig$reportChannelError(void);

252
static uint8_t UniqueLayerP$UniqueConfig$getSequenceNumber(message_t *arg_0x1a9412c8);
static am_addr_t UniqueLayerP$UniqueConfig$getSender(message_t *arg_0x1a9417d8);
# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\UniqueLayerP.nc"
uint8_t UniqueLayerP$sequenceNumber;
static inline error_t UniqueLayerP$Init$init(void);
static inline error_t UniqueLayerP$Send$send(message_t *msg, uint8_t len);
static inline void UniqueLayerP$SubSend$sendDone(message_t *msg, error_t error);
static inline message_t *UniqueLayerP$SubReceive$header(message_t *msg);
uint8_t UniqueLayerP$receivedNumbers[5];
static inline message_t *UniqueLayerP$SubReceive$receive(message_t *msg);
#line 84
static inline bool UniqueLayerP$SubReceive$header(message_t *msg);
uint8_t UniqueLayerP$sequenceNumber;
static inline void UniqueLayerP$ NeighborP$evicted(uint8_t index);
static void NeighborhoodP$ NeighborP$evicted(uint8_t arg_0x1ac93270);
static uint8_t NeighborhoodP$ NeighborP$nodes[5];
uint8_t NeighborhoodP$ NeighborP$ages[5];
uint8_t NeighborhoodP$ NeighborP$flags[5];
uint8_t NeighborhoodP$ time;
uint8_t NeighborhoodP$ last;
static inline error_t NeighborhoodP$Init$init(void);
#line 83
static uint8_t NeighborhoodP$ NeighborP$insertNode(am_addr_t node);
#line 147
static __inline bool NeighborhoodP$ NeighborP$Flag$get(uint8_t bit, uint8_t index);
static __inline void NeighborhoodP$ NeighborP$Flag$set(uint8_t bit, uint8_t index);
static void NeighborhoodP$ NeighborP$Flag$clearAll(uint8_t bit);
static error_t TrafficMonitorLayerP$SubSend$send(message_t *arg_0x1ac24c88);
static uint16_t TrafficMonitorLayerP$ TrafficMonitorConfig$getUpdatePeriod(void);
static uint16_t TrafficMonitorLayerP$ TrafficMonitorConfig$getChannelTime(message_t *arg_0x1a9545e0);
static void TrafficMonitorLayerP$ RadioState$done(void);
static void TrafficMonitorLayerP$ RadioState$setReady(void);
static void TrafficMonitorLayerP$ RadioState$setStateDone(void);
static void TrafficMonitorLayerP$ RadioState$setState(void);
static void TrafficMonitorLayerP$ RadioState$turnOn(void);
static void TrafficMonitorLayerP$ RadioState$turnOff(void);
static void TrafficMonitorLayerP$ RadioState$suspend(void);
static void TrafficMonitorLayerP$ RadioState$resume(void);
static void TrafficMonitorLayerP$ Tasklet$suspend(void);
static void TrafficMonitorLayerP$ Tasklet$resume(void);
static void TrafficMonitorLayerP$ RadioSend$ready(void);
static uint16_t RandomCollisionLayerP$Config$
  getTransmitBarrier(message_t * arg_0x1a968ac0);
# 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
  \chips\rf230\RadioReceive.nc"
static message_t *RandomCollisionLayerP$RadioReceive$
  receive(message_t * arg_0x1ac3d3ab8);
#line 35
static bool RandomCollisionLayerP$RadioReceive$
  header(message_t * arg_0x1ac3d3eb6);
# 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
  \chips\rf230\RadioSend.nc"
static void RandomCollisionLayerP$RadioSend$ready(void);
#line 45
static void RandomCollisionLayerP$RadioSend$
  sendDone(error_t arg_0x1ac23360);
# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
  \interfaces\TaskBasic.nc"
static void RandomCollisionLayerP$calcNextRandom$,
  postTask(void);
# 67 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
  \chips\rf230\RandomCollisionLayerP.nc"
enum RandomCollisionLayerP$__nesc_unnamed4348 {
  RandomCollisionLayerP$calcNextRandom = 7U
};

typedef int RandomCollisionLayerP
  $__nesc_sillytask_calcNextRandom[RandomCollisionLayerP$calcNextRandom];
#line 46
uint8_t RandomCollisionLayerP$state;
enum RandomCollisionLayerP$__nesc_unnamed4349 {
  RandomCollisionLayerP$STATE_READY = 0,
  RandomCollisionLayerP$STATE_TX_PENDING_FIRST = 1,
  RandomCollisionLayerP$STATE_TX_PENDING_SECOND = 2,
  RandomCollisionLayerP$STATE_TX_SENDING = 3,
  RandomCollisionLayerP$STATE_BARRIER = 0x80
};
message_t *RandomCollisionLayerP$txMsg;
uint16_t RandomCollisionLayerP$txBarrier;
static inline void RandomCollisionLayerP$SubSend$ready(void);
static inline uint16_t RandomCollisionLayerP$nextRandom;
static inline void RandomCollisionLayerP$calcNextRandom$,
  RandomCollisionLayerP$getBackoff(uint16_t maxBackoff);
static inline void RandomCollisionLayerP$sendDone(error_t error);
static inline bool RandomCollisionLayerP$SubReceive$,
  header(message_t * msg);
static inline message_t *RandomCollisionLayerP$SubReceive$,
  receive(message_t * msg);
# 41 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
  \system\RandomMlcgC.nc"
uint32_t RandomMlcgC$seed;
static inline error_t RandomMlcgC$Init$,
  init(void);
static uint32_t RandomMlcgC$Random$rand32(void);
static uint16_t RandomMlcgC$Random$rand16(void);
static error_t SoftwareAckLayerP$SubSend$, send(message_t * arg_0x1ac24c88);
static void SoftwareAckLayerP$RadioAlarm$,
  wait(uint16_t arg_0x1a995d20);
static void SoftwareAckLayerP$RadioAlarm$,
  cancel(void);
static bool SoftwareAckLayerP$RadioAlarm$,
  isFree(void);
static message_t *SoftwareAckLayerP$RadioReceive$,,
  receive(message_t * arg_0x1ac3d3ab8);
static bool SoftwareAckLayerP$RadioReceive$,,
  header(message_t * arg_0x1ac3d3eb6);
static message_t *SoftwareAckLayerP$SoftwareAckConfig$,,
  reportChannelError(void);
# 33
255
static void SoftwareAckLayerP$SoftwareAckConfig$
createAckPacket(message_t *arg_0x1a945e08, message_t *arg_0x1a944010);
#line 38
static bool SoftwareAckLayerP$SoftwareAckConfig$
requiresAckWait(message_t *arg_0x1a948ef0);
#line 50
static bool SoftwareAckLayerP$SoftwareAckConfig$
isAckPacket(message_t *arg_0x1a947b00);
static bool SoftwareAckLayerP$SoftwareAckConfig$
verifyAckPacket(message_t *arg_0x1a945120, message_t *arg_0x1a9452d0);
#line 32
static uint16_t SoftwareAckLayerP$SoftwareAckConfig$
getAckTimeout(void);
static void SoftwareAckLayerP$SoftwareAckConfig$
setAckReceived(message_t *arg_0x1a947450, bool arg_0x1a9475d8);
#line 63
static bool SoftwareAckLayerP$SoftwareAckConfig$
requiresAckReply(message_t *arg_0x1a9458d0);
# 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioSend.nc"
static void SoftwareAckLayerP$RadioSend$ready(void);
#line 45
static void SoftwareAckLayerP$RadioSend$sendDone(error_t arg_0x1ac23360);
# 46 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SoftwareAckLayerP.nc"
uint8_t SoftwareAckLayerP$state;
enum SoftwareAckLayerP$__nesc_unnamed4350 {
SoftwareAckLayerP$STATE_READY = 0,
SoftwareAckLayerP$STATE_DATA_SEND = 1,
SoftwareAckLayerP$STATE_ACK_WAIT = 2,
SoftwareAckLayerP$STATE_ACK_SEND = 3
};
message_t *SoftwareAckLayerP$txMsg;
message_t SoftwareAckLayerP$ackMsg;
static inline void SoftwareAckLayerP$SubSend$ready(void);
static inline error_t SoftwareAckLayerP$RadioSend$send(message_t *msg);
#line 83
static inline void SoftwareAckLayerP$SubSend$sendDone(error_t error);
#line 110
static void SoftwareAckLayerP$RadioAlarm$fired(void);
static inline bool SoftwareAckLayerP$SubReceive$header(message_t *msg);
static inline message_t *SoftwareAckLayerP$SubReceive$receive(message_t *msg);
# 41 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioCCA.nc"
static void RF230LayerP$RadioCCA$done(error_t arg_0x1ac3a920);
static void RF230LayerP$PacketField$clear(message_t *arg_0x1a8a8718);
static void RF230LayerP$PacketField$set(message_t *arg_0x1a8a8d90, RF230LayerP$PacketField$value_type arg_0x1a8d9790);
static void RF230LayerP$PacketField$spiSplitRead(void);
static void RF230LayerP$PacketField$spiSplitReadWrite(uint8_t arg_0x1ad62980);
static void RF230LayerP$PacketField$spiSplitWrite(uint8_t arg_0x1ad62e90);
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioState.nc"
static void RF230LayerP$RadioState$done(void);
# 35 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GeneralIO.nc"
static void RF230LayerP$RSTN$makeOutput(void);
# 29

256
static __inline void RF230LayerP$downloadMessage(void);
# line 622
static inline void RF230LayerP$IRQ$captured(uint16_t time);
# line 635
static inline void RF230LayerP$serviceRadio(void);
# line 787
static inline void RF230LayerP$Tasklet$run(void);
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\GeneralIO.nc"
static void HplRF230P$PortCLKM$makeInput(void);
# line 30
static void HplRF230P$PortIRQ$clr(void);
# 79 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\atm128\timer\HplAtm128Capture.nc"
static inline void HplRF230P$Captured$setEdge(bool arg_0x1ab09b48);
# line 38
static HplRF230P$Capture$size_type HplRF230P$Capture$get(void);
# line 55
static void HplRF230P$Capture$reset(void);
static void HplRF230P$Capture$start(void);
static void HplRF230P$Capture$stop(void);
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\GeneralIO.nc"
static void HplRF230P$PortIRQ$makeInput(void);
# line 30
static void HplRF230P$PortIRQ$clr(void);
# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\GpioCapture.nc"
static void HplRF230P$Capture$setEdge(bool arg_0x1ad7c680);
# line 55
static void HplRF230P$Capture$reset(void);
static void HplRF230P$Capture$start(void);
static void HplRF230P$Capture$stop(void);
static void HplRF230P$PortIRQ$makeInput(void);
# line 30
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\GeneralIO.nc"
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static void HplRF230P$PortIRQ$makeInput(void);
static void HplRF230P$PortIRQ$clr(void);
static am_addr_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$AMPacket$destination(message_t *arg_0x1a875eb0);
#line 136
static am_id_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$AMPacket$setType(message_t *arg_0x1a872610);
#line 118 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\AMQueueImplP.nc"
enum /*AMQueueP.AMQueueImplP*/AMQueueImplP0S$nesc_unnamed4360 { AMQueueImplP0S_CancelTask = 10U };
#line 118
typedef int /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S__nesc_sillytask_CancelTask[/*AMQueueP.AMQueueImplP*/ AMQueueImplP0SCancelTask];
#line 161
typedef int /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S__nesc_sillytask_errorTask[/*AMQueueP.AMQueueImplP*/ AMQueueImplP0SerrorTask];
#line 49
typedef struct /*AMQueueP.AMQueueImplP*/
AMQueueImplP0Squeue_entry_t { message_t *msg; } /*AMQueueP.AMQueueImplP*/ AMQueueImplP0Squeue[1];
uint8_t /*AMQueueP.AMQueueImplP*/ AMQueueImplP0ScancelMask[1 / 8 + 1];
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$send(uint8_t clientId, message_t *msg, uint8_t len);
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$CancelTask$runTask(void);
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$errorTask$runTask(void);
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$AMSend$sendDone(am_id_t id, message_t *msg, error_t err);
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S$default$sendDone(uint8_t id, message_t *msg, error_t err);
static inline void __nesc_disable_interrupt(void)
{ __asm volatile ("cli");
#line 109
typedef int /*AMQueueP.AMQueueImplP*/
AMQueueImplP0S__nesc_sillystart-cancelTask;
 AMQueueImplP0S-cancelTask();
#line 118__nesc_atomic_t
__nesc_atomic_start(void)
{ volatile uint8_t *t (volatile uint8_t *)0x3F + 0x20);
#line 129__nesc_disable_interrupt();
__asm volatile ("": : "memory");
return result;
}
#line 135__nesc_atomic_end__nesc_atomic_t original_SREG
{ __asm volatile ("": : "memory");


static inline void SchedulerBasicP$Scheduler$init(void)
{
    /* atomic removed: atomic calls only */
    memset((void *)SchedulerBasicP$m_next,
            SchedulerBasicP$NO_TASK, sizeof SchedulerBasicP$m_next);
    SchedulerBasicP$m_head = SchedulerBasicP$NO_TASK;
    SchedulerBasicP$m_tail = SchedulerBasicP$NO_TASK;
}

inline static void RealMainP$Scheduler$init(void)
{
    SchedulerBasicP$Scheduler$init();
}

static inline error_t ecombine(error_t r1, error_t r2)
{
    return r1 == r2 ? r1 : FAIL;
}

static inline void PlatformP$power_init(void)
{
    /* atomic removed: atomic calls only */
    * (volatile uint8_t *)(0X35 + 0x20) = 1 << 0;
}

static inline void HplAtm1281Timer1P$TimerCtrl$setControlB(uint8_t x)
{
    * (volatile uint8_t *)0x81 = x;
}

static inline uint8_t HplAtm1281Timer1P$TimerCtrl$getControlB(void)
{
    return * (volatile uint8_t *)0x81;
}

static inline void HplAtm1281Timer1P$Timer$setScale(uint8_t s)
{
    Atm128_TCCRB_t x = (Atm128_TCCRB_t )HplAtm1281Timer1P$TimerCtrl$getControlB();
    x.bits.cs = s;
    HplAtm1281Timer1P$TimerCtrl$setControlB(x.flat);
}

inline static void Atm128TimerInitC$0$Timer$setScale(uint8_t arg_0x1a670da8)
{
    HplAtm1281Timer1P$Timer$setScale(arg_0x1a670da8);
}

inline static void Atm128TimerInitC$0$Timer$start(void)
{
    HplAtm1281Timer1P$Timer$start();
}

---

265
# 73 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\atm128\timer\HplAtm1281Timer1P.nc"
static inline void HplAtm1281Timer1P$Timer$set(uint16_t t)
#line 73
* (volatile uint16_t *)0x84 = t;

# 58 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\atm128\timer\HplAtm128Timer.nc"
#include "InitOneP.InitOne"/
 Atm128TimerInitC50$Timer$set(/*InitOneP.InitOne*/
 Atm128TimerInitC50$Timer$timer_size arg_0x1a671120); 
#line 58
HplAtm1281Timer1P$Timer$set(arg_0x1a671120);
}

# 58 
# 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\chips\atm128\timer\Atm128TimerInitC.nc"
static inline error_t /*InitOneP.InitOne*/
Atm128TimerInitC$0$Init$init(void)
#line 42
{ /* atomic removed: atomic calls only */
    /*InitOneP.InitOne*/Atm128TimerInitC$0$Timer$set(0);
    /*InitOneP.InitOne*/Atm128TimerInitC$0$Timer$start();
    /*InitOneP.InitOne*/Atm128TimerInitC$0$Timer$setScale(2);
    return SUCCESS;
}

# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\interfaces\Init.nc"
inline static error_t MotePlatformP$SubInit$init(void)
#line 51
{ unsigned char result;
#line 51
result = /*InitOneP.InitOne*/Atm128TimerInitC$0$Init$init();
#line 51
return result;
}

# 51 
# 47 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void /*HplAtm128GeneralIOC.PortA.Bit4*/
HplAtm128GeneralIOPinP$4$IO$clr(void)
#line 47
{ (volatile uint8_t *)34U &= ~(1 << 4);
}

# 30 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\interfaces\GeneralIO.nc"
inline static void MotePlatformP$SerialIdPin$clr(void)
#line 30
{ /*HplAtm128GeneralIOC.PortA.Bit4*/HplAtm128GeneralIOPinP$4$IO$clr();
}

# 30 
# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void /*HplAtm128GeneralIOC.PortA.Bit4*/
HplAtm128GeneralIOPinP$4$IO$makeInput(void)
#line 50
{ (volatile uint8_t *)33U &= ~(1 << 4);
}

# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\interfaces\GeneralIO.nc"
inline static void MotePlatformP$SerialIdPin$makeInput(void)
#line 33
{ /*HplAtm128GeneralIOC.PortA.Bit4*/HplAtm128GeneralIOPinP$4$IO$makeInput();
}

# 33 
# 26 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos 
\platforms\micaz\MotePlatformP.nc"
static inline error_t MotePlatformP$PlatformInit$init(void)
#line 26
{ (volatile uint8_t *)0x08 + 0x20) = 0;

inline static void LedsP$Led1$makeOutput(void)
{
    #line 35
    /*HplAtm128GeneralIOC.PortA.Bit1*/
    HplAtm128GeneralIOPinP$1$IO$makeOutput();
    #line 35
}

# 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void /*HplAtm128GeneralIOC.PortA.Bit2*/
HplAtm128GeneralIOPinP$2$IO$makeOutput(void)
{
    #line 52
    * (volatile uint8_t *)33U |= 1 << 2;
}

inline static void LedsP$Led0$makeOutput(void)
{
    #line 35
    /*HplAtm128GeneralIOC.PortA.Bit2*/
    HplAtm128GeneralIOPinP$2$IO$makeOutput();
    #line 35
}

#line 35
# 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\LedsP.nc"
static inline error_t LedsP$Init$init(void)
{
    /* atomic removed: atomic calls only */
    #line 46
    {
        LedsP$Led0$makeOutput();
        LedsP$Led1$makeOutput();
        LedsP$Led2$makeOutput();
        LedsP$Led0$set();
        LedsP$Led1$set();
        LedsP$Led2$set();
    }
    return SUCCESS;
}

# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Init.nc"
inline static error_t PlatformP$MoteInit$init(void)
{
    #line 51
    unsigned char result;
    #line 51
    result = LedsP$Init$init();
    #line 51
    result = ecombine(result, MotePlatformP$PlatformInit$init());
    #line 51
    return result;
}

# 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\platforms\iris\MeasureClockC.nc"
static inline error_t MeasureClockC$Init$init(void)
{
    /* atomic removed: atomic calls only */
    uint8_t now;
    #line 70
    uint8_t wraps;
    uint16_t start;
    * (volatile uint8_t *)0xB2 = 1 << 0;
    * (volatile uint8_t *)0xB6 = 1 << 5;
    * (volatile uint8_t *)0xB1 = (1 << 1) | (1 << 0);
    start = * (volatile uint16_t *)0x84;
    for (wraps = MeasureClockC$MAGIC / 2; wraps; )
    {
        uint16_t next = * (volatile uint16_t *)0x84;
        if (next < start) {wraps--;
    }
    now = * (volatile uint8_t *)0xB2;
while (* (volatile uint8_t *)0xB2 == now) ;
start = * (volatile uint16_t *)0x84;
now = * (volatile uint8_t *)0xB2;
while (* (volatile uint8_t *)0xB2 == now) ;
MeasureClockC$cycles = * (volatile uint16_t *)0x84;
MeasureClockC$cycles = (MeasureClockC$cycles - start + 16) >> 5;
* (volatile uint8_t *)0xB6 = * (volatile uint8_t *)0x81 =
* (volatile uint8_t *)0xB1 = 0;
* (volatile uint8_t *)0xB2 = 0;
* (volatile uint16_t *)0x84 = 0;
* (volatile uint8_t *)0x17 + 0x20) =
while (* (volatile uint8_t *)0xB6 & (((1 << 4) | (1 << 2)) | (1 << 0)) ;
)
return SUCCESS;
}

inline static error_t PlatformP$MeasureClock$init(void){
unsigned char result;
result = MeasureClockC$Init$init();
return result;
}

static inline error_t PlatformP$Init$init(void)
{
error_t ok;
ok = PlatformP$MeasureClock$init();
ok = ecombine(ok, PlatformP$MoteInit$init());
if (ok != SUCCESS) {
return ok;
}
PlatformP$power_init();
return SUCCESS;
}

static __inline void / *HplAtm128GeneralIOC.PortA.Bit6*/
HplAtm128GeneralIOPinP$6$IO$set(void)
{
* (volatile uint8_t *)34U |= 1 << 6;
}

inline static void RF230LayerP$RSTN$set(void){
/*HplAtm128GeneralIOC.PortA.Bit6*/
HplAtm128GeneralIOPinP$6$IO$set();
}

static __inline void / *HplAtm128GeneralIOC.PortB.Bit7*/
HplAtm128GeneralIOPinP$15$IO$clr(void)
{
* (volatile uint8_t *)33U &= ~(1 << 6);
}

inline static void RF230LayerP$RSTN$makeOutput(void){
/*HplAtm128GeneralIOC.PortA.Bit6*/
HplAtm128GeneralIOPinP$6$IO$set(void);
}

static __inline void / *HplAtm128GeneralIOC.PortA.Bit6*/
HplAtm128GeneralIOPinP$6$IO$clr(void)
{
* (volatile uint8_t *)33U &= ~(1 << 6);
}

inline static void RF230LayerP$RSTN$makeOutput(void){
/*HplAtm128GeneralIOC.PortA.Bit6*/
HplAtm128GeneralIOPinP$6$IO$set(void);
}

static __inline void / *HplAtm128GeneralIOC.PortB.Bit7*/
HplAtm128GeneralIOPinP$15$IO$clr(void)
static void RF230LayerP$SLP_TR$clr(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit7*/
    HplAtm128GeneralIOPinP$15$IO$clr();
}

static void RF230LayerP$SLP_TR$makeOutput(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit7*/
    HplAtm128GeneralIOPinP$15$IO$makeOutput();
}

static void RF230LayerP$RSTN$makeOutput(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit0*/
    HplAtm128GeneralIOPinP$8$IO$makeOutput();
}

static void RF230LayerP$RSTN$set(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit0*/
    HplAtm128GeneralIOPinP$8$IO$set();
}

static void RF230LayerP$SELN$makeOutput(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit0*/
    HplAtm128GeneralIOPinP$8$IO$makeOutput();
}

static void RF230LayerP$SELN$set(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit0*/
    HplAtm128GeneralIOPinP$8$IO$set();
}

error_t RF230LayerP$PlatformInit$init(void)
{
    RF230LayerP$SELN$makeOutput();
    RF230LayerP$SELN$set();
    RF230LayerP$SLP_TR$makeOutput();
    RF230LayerP$SLP_TR$clr();
    RF230LayerP$RSTN$makeOutput();
    RF230LayerP$RSTN$set();
    RF230LayerP$rssiClear = 0;
    RF230LayerP$rssiBusy = 90;
    return SUCCESS;
}

void HplAtm1281Timer1P$Capture$stop(void)
#line 151
#line 151
* (volatile uint8_t *)0x6F &= ~((1 << 5);
}

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Capture.nc"
inline static void HplRF230P$Capture$stop(void)
#
# 61
HplAtm128Timer1P$Capture$stop();
#
# 61
# 47 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void /\HplAtm128GeneralIOC.PortD.Bit4=/
HplAtm128GeneralIOPinP$28$IO$clr(void)
#
# 47
line 47
* (volatile uint8_t *)43U &= ~((1 << 4);
#
# 30 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GeneralIO.nc"
inline static void HplRF230P$PortIRQ$clr(void)
#
# 30
/HplAtm128GeneralIOC.PortD.Bit4//=
HplAtm128GeneralIOPinP$28$IO$clr();
#
# 30
# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void /\HplAtm128GeneralIOC.PortD.Bit6=/
HplAtm128GeneralIOPinP$30$IO$clr(void)
#
# 50
line 50
* (volatile uint8_t *)42U &= ~((1 << 6);
#
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GeneralIO.nc"
inline static void HplRF230P$PortIRQ$makeInput(void)
#
# 33
/HplAtm128GeneralIOC.PortD.Bit4//=
HplAtm128GeneralIOPinP$28$IO$makeInput();
#
# 33
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GeneralIO.nc"
inline static void HplRF230P$PortCLKM$clr(void)
#
# 33
/HplAtm128GeneralIOC.PortD.Bit6//=
HplAtm128GeneralIOPinP$30$IO$clr();
#
# 33
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GeneralIO.nc"
inline static void HplRF230P$PortCLKM$makeInput(void)
#
# 33
/HplAtm128GeneralIOC.PortD.Bit6//=
HplAtm128GeneralIOPinP$30$IO$makeInput();
#
# 33
# 47 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\platforms\iris\chips\rf230\HplRF230P.nc"
static inline error_t HplRF230P$PlatformInit$init(void)
{
    HplRF230P$PortCLKM$makeInput();
    HplRF230P$PortCLKM$clr();
    HplRF230P$PortIRQ$makeInput();
    HplRF230P$PortIRQ$clr();
    HplRF230P$Capture$stop();
    return SUCCESS;
}

inline static error_t RealMainP$PlatformInit$init(void){
    unsigned char result;
    result = HplRF230P$PlatformInit$init();
    result = ecombine(result, RF230LayerP$PlatformInit$init());
    result = ecombine(result, PlatformP$Init$init());
    return result;
}

inline static bool RealMainP$Scheduler$runNextTask(void){
    unsigned char result;
    result = SchedulerBasicP$Scheduler$runNextTask();
    return result;
}

static inline void ForwardToBaseC$AMSend$sendDone(message_t *msg, error_t err)
{
    ForwardToBaseC$busy = FALSE;
}

static inline void / *ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
AMQueueEntryP$0$AMSend$sendDone(message_t *arg_0x1a8821c0, error_t arg_0x1a882348){
    ForwardToBaseC$AMSend$sendDone(arg_0x1a8821c0, arg_0x1a882348);
}

static inline void / *ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
AMQueueEntryP$0$Send$sendDone(message_t *m, error_t err)
{
    /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
    AMQueueEntryP$0$AMSend$sendDone(m, err);
}

static inline void / *AMQueueP.AMQueueImplP*/
AMQueueImplP$0$Send$default$(uint8_t id, message_t *msg, error_t err)
{
}

static inline void / *AMQueueP.AMQueueImplP*/
AMQueueImplP$0$Send$(uint8_t id, message_t *msg, error_t err)
{

    switch (arg_0x1b019b68) {
    case 0U: 
        sendDone(uint8_t arg_0x1b0b19b68, message_t arg_0x1abe2198);
        break;
    default:
        sendDone(uint8_t arg_0x1b0b19b68, message_t arg_0x1abe2198);
        break;
    }
}
/*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
AMQueueEntryP$0$Send$sendDone(arg_0x1abe2010, arg_0x1abe2198);

#line 89
break;

#line 89
default:

#line 89
/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$Send$default$
sendDone(arg_0x1b019b68, arg_0x1abe2010, arg_0x1abe2198);

#line 89
break;

#line 89
}

#line 89

# 118 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\AMQueueImplP.nc"
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$CancelTask$runTask(void)
{
  uint8_t i;
  uint8_t j;
  uint8_t mask;
  uint8_t last;
  message_t *msg;
  for (i = 0; i < 1 / 8 + 1; i++) {
    if (/*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$cancelMask[i]) {
      for (mask = 1, j = 0; j < 8; j++) {
        if (/*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$cancelMask[i] & mask) {
          last = i * 8 + j;
          msg = /*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$queue[last].msg;
          /*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$queue[last].msg = (void *)0;
          /*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$cancelMask[i] &= ˜mask;
          /*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$Send$sendDone(last, msg, ECANCEL);
        }
        mask <<= 1;
      }
    }
  }
}

#line 161
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$errorTask$runTask(void)
{
  /*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$sendDone(/*AMQueueP.AMQueueImplP*/
      AMQueueImplP$0$current, /*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$queue[/*AMQueueP.AMQueueImplP*/
          AMQueueImplP$0$current].msg, FAIL);
}

# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\TaskBasic.nc"
inline static error_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$errorTask$postTask(void){
  unsigned char result;
  #line 56
  result = SchedulerBasicP$TaskBasic$postTask(/*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$errorTask);
  #line 56
  return result;
  #line 56
}

# 69 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\AMSend.nc"
inline static error_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$AMSend$send(am_id_t arg_0x1b0184e8, am_addr_t arg_0x1a885e28,
  message_t *arg_0x1a884010, uint8_t arg_0x1a884198){
  #line 69
}
unsigned char result;
#line 69
unsigned char result = ActiveMessageLayerC$AMSend$
    send(arg_0x1b0184e8, arg_0x1a885e28, arg_0x1a884010, arg_0x1a884198);
#line 69
return result;
#line 69
# 246 "C:\Crossbow\cygwin\lib\ncc\nesc_nx.h"
static __inline uint8_t __nesc_ntoh_leuint8(const void *source)
#line 246
    const uint8_t *base = source;
#line 248
    return base[0];
# 58 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154PacketP.nc"
static __inline ieee154_header_t *IEEE154PacketP$getHeader(message_t *msg)
    return (ieee154_header_t *)(msg->data - sizeof(ieee154_header_t ));
static __inline uint8_t IEEE154PacketP$IEEE154Packet$getLength(message_t *msg)
    return __nesc_ntoh_leuint8((unsigned char *)&IEEE154PacketP$getHeader(msg)->length);
# 44 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154PacketP.nc"
inline static uint8_t IEEE154PacketP$Packet$payloadLength(message_t *msg)
    return RF230PacketP$Packet$payloadLength(msg) - RF230PacketP$Packet$PAYLOAD_LENGTH_INCREASE;
# 67 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Packet.nc"
inline static uint16_t AMQueueImplP$AMQueueImplP$AMQueueImplP$Packet$payloadLength(message_t *msg)
    return __nesc_ntoh_leuint16((unsigned char *)&IEEE154PacketP$getHeader(msg)->length);
    return RF230PacketP$IEEE154PacketP$Packet$payloadLength(msg) - RF230PacketP$Packet$PAYMENT_INCREMENT;
    return AMQueueImplP$AMQueueImplP$AMQueueImplP$Packet$payloadLength(msg) - AMQueueImplP$AMQueueImplP$Packet$PAYLOAD_LENGTH_INCREMENT;
    return __nesc_ntoh_leuint16((unsigned char *)&IEEE154PacketP$getHeader(msg)->length);
static __inline am_addr_t IEEE154PacketP$AMPacket$destination(message_t *msg)
{
    return IEEE154PacketP$IEEE154Packet$getDestAddr(msg);
}

inline static am_addr_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$AMPacket$destination(message_t *arg_0x1a875eb0){
    unsigned short result;
    result = IEEE154PacketP$AMPacket$destination(arg_0x1a875eb0);
    return result;
}

static __inline am_id_t IEEE154PacketP$IEEE154Packet$getType(message_t *msg)
{
    return __nesc_ntoh_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$getHeader(msg)->type);
}

static __inline am_id_t IEEE154PacketP$AMPacket$type(message_t *msg)
{
    return IEEE154PacketP$IEEE154Packet$getType(msg);
}

inline static am_id_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$AMPacket$type(message_t *arg_0x1a872610){
    unsigned char result;
    result = IEEE154PacketP$AMPacket$type(arg_0x1a872610);
    return result;
}

static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$nextPacket(void)
{
    uint8_t i;
    /*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$current = ((/*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$current + 1) % 1;
    for (i = 0; i < 1; i++) {
        if (/*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$queue[/*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$current].msg == (void *)0 ||
        /*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$cancelMask[/*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$current / 8] & (1 << /*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$current % 8))
        {
            /*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$current = (/*AMQueueP.AMQueueImplP*/
            AMQueueImplP$0$current + 1) % 1;
        } else {
            break;
        }
    }
    if (i >= 1) {
        /*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$current = 1;
    }
}

static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$tryToSend(void)
/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$nextPacket();
if (/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$current < 1) {
    error_t nextErr;
    message_t *nextMsg = /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$queue[/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$current].msg;
    am_id_t nextId = /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$AMPacket$type(nextMsg);
    am_addr_t nextDest = /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$AMPacket$destination(nextMsg);
    uint8_t len = /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$Packet$payloadLength(nextMsg);
    #line 174
    nextErr = /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$AMSend$send(nextId, nextDest, nextMsg, len);
    if (nextErr != SUCCESS) {
        /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$errorTask$postTask();
    }
}
# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230PacketP.nc"
static __inline rf230packet_metadata_t *RF230PacketP$
getMeta(message_t *msg)
{
    return (rf230packet_metadata_t *)msg->metadata;
}
# 281 "C:\Crossbow\cygwin\lib\nc\nesc_nx.h"
static __inline uint16_t __nesc_hton_leuint16(void *target, uint16_t value)
#line 281{
    uint8_t *base = target;
    #line 283
    base[0] = value;
    base[1] = value >> 8;
    return value;
}
# 93 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154PacketP.nc"
static __inline void IEEE154PacketP$IEEE154Packet$
createDataFrame(message_t *msg)
{
    __nesc_hton_leuint16((unsigned char *)&IEEE154PacketP$
getHeader(msg)->fcf, IEEE154PacketP$IEEE154_DATA_FRAME_VALUE);
}
# 73 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154Packet.nc"
inline static void RF230PacketP$IEEE154Packet$
createDataFrame(message_t *msg)
{
    RF230PacketP$IEEE154Packet$createDataFrame(msg);
    RF230PacketP$IEEE154Packet$clear(msg);
    RF230PacketP$IEEE154Packet$setMeta(msg)->flags = RF230_PACKET_CLEAR_METADATA;
}
# 54 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Packet.nc"
inline static void RF230ActiveMessageP$Packet$
clear(message_t *msg)
{
    RF230PacketP$IEEE154Packet$clear(msg);
}
# 54

276
inline static bool RF230ActiveMessageP$IEEE154Packet$isDataFrame(message_t *arg_0x1a972228){
    #line 67
    unsigned char result;
    #line 67
    #line 67
    result = IEEE154PacketP$IEEE154Packet$isDataFrame(arg_0x1a972228);
    #line 67
    #line 67
    return result;
    #line 67
}

# 166 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RF230ActiveMessageP.nc"
static inline error_t RF230ActiveMessageP$ActiveMessageConfig$checkPacket(message_t *msg)
{
    if (!RF230ActiveMessageP$IEEE154Packet$isDataFrame(msg)) {
        RF230ActiveMessageP$Packet$clear(msg);
        return SUCCESS;
    }
    return result;
}

# 31 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\ActiveMessageConfig.nc"
inline static error_t ActiveMessageLayerC$Config$checkPacket(message_t *arg_0x1a960140){
    #line 31
    unsigned char result;
    #line 31
    #line 31
    result = RF230ActiveMessageP$ActiveMessageConfig$checkPacket(arg_0x1a960140);
    #line 31
    #line 31
    return result;
    #line 31
}

# 189 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154PacketP.nc"
static __inline void IEEE154PacketP$AMPacket$setSource(message_t *msg, am_addr_t addr)
{
    IEEE154PacketP$IEEE154Packet$setSrcAddr(msg, addr);
}

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\ActiveMessageAddressC.nc"
static inline am_addr_t ActiveMessageAddress$amAddress(void)
{
    return ActiveMessageAddressC$amAddress();
}

# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\ActiveMessageAddress.nc"
inline static am_addr_t IEEE154PacketP$ActiveMessageAddress$amAddress(void){
    #line 50
    unsigned short result;
    #line 50
    #line 50
    result = ActiveMessageAddressC$ActiveMessageAddress$amAddress();
    #line 50
    #line 50
    return result;
    #line 50
}

# 238 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
static __inline am_addr_t IEEE154PacketP$AMPacket$address(void)
{
    return IEEE154PacketP$ActiveMessageAddress$amAddress();
}

static __inline am_addr_t ActiveMessageLayerC$AMPacket$address(void)
{
    unsigned short result;

    result = IEEE154PacketP$AMPacket$address();

    return result;
}

static __inline void IEEE154PacketP$IEEE154Packet$setDestPan(message_t *msg, uint16_t pan)
{
    __nesc_hton_leuint16((unsigned char *)&IEEE154PacketP$getHeader(msg)->destpan, pan);
}

static __inline void IEEE154PacketP$AMPacket$setGroup(message_t *msg, am_group_t grp)
{
    IEEE154PacketP$IEEE154Packet$setDestPan(msg, grp);
}

static inline am_group_t ActiveMessageAddressC$ActiveMessageAddress$amGroup(void)
{
    am_group_t myGroup;

    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();

    myGroup = ActiveMessageAddressC$ActiveMessageAddress$amGroup();

    __nesc_atomic_end(__nesc_atomic); }

    return myGroup;
}

static inline am_group_t IEEE154PacketP$ActiveMessageAddress$amGroup(void)
{
    unsigned char result;

    result = ActiveMessageAddressC$ActiveMessageAddress$amGroup();

    return result;
}

static __inline am_group_t IEEE154PacketP$AMPacket$localGroup(void)
{
    return IEEE154PacketP$ActiveMessageAddress$amGroup();
}

static __inline am_group_t ActiveMessageLayerC$AMPacket$localGroup(void)
{
    unsigned char result;

    result = IEEE154PacketP$AMPacket$localGroup();

return result;
}

static __inline uint8_t __nesc_hton_leuint8(void *target, uint8_t value)
{
    uint8_t *base = target;
    base[0] = value;
    return value;
}

static __inline void IEEE154PacketP$IEEE154Packet$setType(message_t *msg, am_id_t type)
{
    __nesc_hton_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$.getHeader(msg)->type, type);
}

static __inline void IEEE154PacketP$AMPacket$setType(message_t *msg, am_id_t type)
{
    IEEE154PacketP$IEEE154Packet$setType(msg, type);
}

static __inline void IEEE154PacketP$IEEE154Packet$setDestAddr(message_t *msg, am_addr_t addr)
{
    __nesc_hton_leuint16((unsigned char *)&IEEE154PacketP$IEEE154Packet$.getHeader(msg)->dest, addr);
}

static __inline void IEEE154PacketP$AMPacket$setDestination(message_t *msg, uint16_t addr)
{
    IEEE154PacketP$AMPacket$setDestAddr(msg, addr);
}

static __inline void IEEE154PacketP$AMPacket$sendTask$postTask(void)
{
    unsigned char result;
    result = SchedulerBasicP$TaskBasic$postTask(IEEE154PacketP$AMPacket$sendTask);
    return result;
}

static __inline void IEEE154PacketP$IEEE154Packet$setLength(message_t *msg, uint8_t length)
{
    __nesc_hton_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$.getHeader(msg)->length, length);
}

#### Interface Functions

- IEEE154PacketP$IEEE154Packet$sendTask$postTask
- IEEE154PacketP$IEEE154Packet$setDestAddr
- IEEE154PacketP$IEEE154Packet$setLength

#### Type Definitions

- $message_t$
- $am_id_t$
- $am_addr_t$
- $uint8_t$
- $uint16_t$

#### Includes

- `ncc\nesc_nx.h`
- `AMPacket.nc`
- `IEEE154PacketP.nc`
- `TaskBasic.nc`

### Notes
- The code snippet includes various functions that handle networking and message passing, typical for embedded systems.
- The `__nesc_hton_leuint8` function is used for byte order conversion from little to host byte order.
- The `setType` and `setDestination` functions are used to set type and destination addresses in packets.
- The `sendTask$postTask` function schedules a task to be executed by the scheduler.

---

279
inline static void RF230PacketP$IEEE154Packet$setLength(message_t *arg_0x1a975ee0, uint8_t arg_0x1a973088) { 
    IEEE154PacketP$IEEE154Packet$setLength(arg_0x1a975ee0, arg_0x1a973088); 
} 

inline static void RF230PacketP$Packet$setPayloadLength(message_t *msg, uint8_t len) { 
    RF230PacketP$Packet$setLength(msg, len + RF230PacketP$PACKET_LENGTH_INCREASE); 
} 

static uint8_t RF230PacketP$Packet$maxPayloadLength(void) { 
    return 28; 
} 

inline static error_t UniqueLayerP$SubSend$send(message_t *arg_0x1abe5ef8, uint8_t arg_0x1abe4090) { 
    unsigned char result; 
    result = MessageBufferLayerP$Send$send(arg_0x1abe5ef8, arg_0x1abe4090); 
    return result; 
} 

inline static error_t MessageBufferLayerP$Send$send(message_t *msg, uint8_t len) { 
    if (len > MessageBufferLayerP$Packet$maxPayloadLength()) { 
        return EINVAL; 
    } else { 
        unsigned char result; 
        result = MessageBufferLayerP$Packet$setPayloadLength(msg, len); 
        MessageBufferLayerP$txMsg = msg; 
        MessageBufferLayerP$state = MessageBufferLayerP$STATE_TX_PENDING; 
        MessageBufferLayerP$retries = 0; 
        MessageBufferLayerP$sendTask$postTask(); 
        return SUCCESS; 
    } 
} 

inline static void RF230PacketP$SubSend$send(message_t *arg_0x1abe5ef8, uint8_t arg_0x1abe4090) { 
    unsigned char result; 
    result = MessageBufferLayerP$Send$send(arg_0x1abe5ef8, arg_0x1abe4090); 
    return result; 
} 

unsigned char MessageBufferLayerP$Packet$maxPayloadLength(void) { 
    unsigned char result; 
    result = RF230PacketP$Packet$maxPayloadLength(); 
    return result; 
} 

unsigned char MessageBufferLayerP$Packet$getPayloadLength(void) { 
    return MessageBufferLayerP$Packet$maxPayloadLength(); 
} 

unsigned char MessageBufferLayerP$Packet$getPayloadLength(void) { 
    return MessageBufferLayerP$Packet$maxPayloadLength(); 
} 


static inline void IEEE154PacketP$IEEE154Packet$
setDSN(message_t *msg, uint8_t dsn)
{
    __nesc hton_leuint8((unsigned char *)&IEEE154PacketP$
    getHeader(msg)->dsn, dsn);
}

# 129 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154PacketPnc"
inline static void RF230ActiveMessageP$IEEE154PacketP$
setDSN(message_t * arg_0x1a986500, uint8_t arg_0x1a98688)
{
    IEEE154PacketP$IEEE154PacketP$setDSN(arg_0x1a986500, arg_0x1a98688);
}

# 129
# 129
# 129

# 149 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessagePnc"
inline static void
RF230ActiveMessageP$UniqueConfig$
setSequenceNumber(message_t *msg, uint8_t dsn)
{
    RF230ActiveMessageP$IEEE154PacketP$setDSN(msg, dsn);
}

# 41 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\UniqueLayerPnc"
inline static void UniqueLayerP$UniqueConfig$
setSequenceNumber(message_t * arg_0x1a982518, uint8_t arg_0x1a9826a0)
{
    RF230ActiveMessageP$UniqueConfig$
    setSequenceNumber(arg_0x1a982518, arg_0x1a9826a0);
}

# 58 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\UniqueLayerPnc"
static inline error_t UniqueLayerP$Send$send(message_t *msg, uint8_t len)
{
    UniqueLayerP$UniqueConfig$
    setSequenceNumber(msg, ++UniqueLayerP$sequenceNumber);
    return UniqueLayerP$SubSend$send(msg, len);
}

# 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154NetworkLayerPnc"
inline static error_t IEEE154NetworkLayerP$Send$send(message_t *msg, uint8_t len)
{
    IEEE154NetworkLayerP$IEEE154PacketP$set6LowPan(msg, 0x3f);
    return IEEE154NetworkLayerP$SubSend$send(msg, len);
}

# 201 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154NetworkLayerPnc"
static inline void IEEE154NetworkLayerP$IEEE154PacketP$
set6LowPan(message_t *msg, uint8_t network)
{
    __nesc hton_leuint8((unsigned char *)&IEEE154PacketP$
    getHeader(msg)->network, network);
}

# 172 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154PacketPnc"
inline static void IEEE154NetworkLayerP$IEEE154PacketP$
set6LowPan(message_t * arg_0x1a982518, uint8_t arg_0x1a9826a0)
{
    IEEE154PacketP$IEEE154PacketP$set6LowPan(arg_0x1a982518, arg_0x1a9826a0);
}

# 172
# 172
# 48 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154NetworkLayerPnc"
static inline error_t IEEE154NetworkLayerP$Send$send(message_t *msg, uint8_t len)
{
    IEEE154NetworkLayerP$IEEE154PacketP$set6LowPan(msg, 0xf3);
    return IEEE154NetworkLayerP$SubSend$send(msg, len);
}

# 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Sendnc"
inline static error_t ActiveMessageLayerC$SubSend$send(message_t *arg_0x1abe5ef8, uint8_t arg_0x1abe4090)
unsigned char result;
result = IEEE154NetworkLayerP$Send$(arg_0x1abe5ef8, arg_0x1abe4090);
return result;
}

static inline bool SchedulerBasicP$isWaiting(uint8_t id)
{
    return SchedulerBasicP$m_next[id] != SchedulerBasicP$NO_TASK || SchedulerBasicP$m_tail == id;
}

static inline bool SchedulerBasicP$pushTask(uint8_t id)
{
    if (!SchedulerBasicP$isWaiting(id))
    {
        if (SchedulerBasicP$m_head == SchedulerBasicP$NO_TASK)
        {
            SchedulerBasicP$m_head = id;
            SchedulerBasicP$m_tail = id;
        }
        else
        {
            SchedulerBasicP$m_next[SchedulerBasicP$m_tail] = id;
            SchedulerBasicP$m_tail = id;
        }
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

inline static void RF230LayerP$Tasklet$schedule()
{
    TaskletC$Tasklet$schedule();
}

inline static error_t RF230LayerP$SpiResource$release(void)
{
    unsigned char result;
    result = Atm128SpiP$Resource$release(0U);
    return result;
}

static __inline void / *HplAtm128GeneralIOC.PortB.Bit7*/
HplAtm128GeneralIOPinP$15$IO$set(void)
{
    * (volatile uint8_t *)37U |= 1 << 7;
}

inline static void RF230LayerP$SLP_TR$set(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit7*/
    HplAtm128GeneralIOPinP$15$IO$set();
}

static __inline uint8_t HplRF230P$HplRF230$spiSplitRead(void)
{
    while (!(* (volatile uint8_t *)(0x2D + 0x20) & 0x80));
    return * (volatile uint8_t *)(0X2E + 0x20);
}
inline static uint8_t RF230LayerP$HplRF230$spiSplitRead(void){
  unsigned char result;
  result = HplRF230P$HplRF230$spiSplitRead();
  return result;
}

static __inline uint8_t HplRF230P$HplRF230$spiSplitReadWrite(uint8_t data){
  uint8_t b;
  while (!(* (volatile uint8_t *)(0x2D + 0x20) & 0x80));
  b = * (volatile uint8_t *)(0X2E + 0x20);
  * (volatile uint8_t *)(0X2E + 0x20) = data;
  return b;
}

inline static uint8_t RF230LayerP$HplRF230$spiSplitReadWrite(uint8_t arg_0x1ad62980){
  unsigned char result;
  result = HplRF230P$HplRF230$spiSplitReadWrite(arg_0x1ad62980);
  return result;
}

static __inline void HplRF230P$HplRF230$spiSplitWrite(uint8_t data){
  * (volatile uint8_t *)(0X2E + 0x20) = data;
}

inline static void RF230LayerP$SELN$clr(void){
  /*HplAtm128GeneralIOC.PortB.Bit0*/
  HplAtm128GeneralIOPinP$8$IO$clr();
}

static __inline void HplAtm128GeneralIOPinP$8$IO$clr(void){
  * (volatile uint8_t *)37U &= ~(1 << 0);
}

inline static void RF230LayerP$SELN$set(void){
  /*HplAtm128GeneralIOC.PortB.Bit0*/
  HplAtm128GeneralIOPinP$8$IO$set();
}

static __inline void RF230LayerP$SELN$set(void){
  for (i = 0; i < 8; i++)
    RF230LayerP$SELN$set();
}

static __inline void RF230LayerP$SELN$clr(void){
  for (i = 0; i < 8; i++)
    RF230LayerP$SELN$set();
    RF230LayerP$SELN$clr();
}

static __inline void RF230LayerP$SELN$set(void){
  for (i = 0; i < 8; i++)
    RF230LayerP$SELN$set();
    RF230LayerP$SELN$set();
}

static __inline void RF230LayerP$SELN$set(void){
  for (i = 0; i < 8; i++)
    RF230LayerP$SELN$set();
    RF230LayerP$SELN$set();
static inline uint8_t RF230ActiveMessageP$RF230Config$getDefaultChannel(void) {
    return 11;
}

inline static uint8_t RF230LayerP$RF230Config$getDefaultChannel(void) {
    // Code
    return result;
}

static __inline void BusyWaitMicroC$BusyWait$wait(uint16_t dt) {
    if (dt) {
        __asm volatile ("1: sbiw %0,1
                      adiw %0,1
                      sbiw %0,1
                      brne 1b" : "+w"(dt));
    }
}

inline static void RF230LayerP$RSTN$clr(void) {
    // Code
}

static inline void RF230LayerP$initRadio(void) {
    RF230LayerP$BusyWait$wait(510);
    RF230LayerP$RSTN$clr();
    RF230LayerP$SLP_TR$clr();
    RF230LayerP$BusyWait$wait(6);
    RF230LayerP$RSTN$set();
    RF230LayerP$writeRegister(RF230_TRX_CTRL_0, RF230_TRX_CTRL_0_VALUE);
    RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_TRX_OFF);
    RF230LayerP$writeRegister(RF230_IRQ_MASK,
                                (RF230_IRQ_TRX_UR | RF230_IRQ_PLL_LOCK) |
                                RF230_IRQ_TRX_END | RF230_IRQ_RX_START);  // Code
}
(RF230_PHY_TX_PWR | RF230_TX_AUTO_CRC_ON | RF230_TX_PWR_DEFAULT);
RF230LayerP$txPower = RF230_TX_PWR_DEFAULT;
RF230LayerP$channel = RF230LayerP$RF230Config$getDefaultChannel() &
RF230_CHANNEL_MASK;
RF230LayerP$writeRegister
(RF230_PHY_CC_CCA, RF230_CCA_MODE_VALUE | RF230LayerP$channel);
RF230LayerP$SLP_TR$set();
RF230LayerP$state = RF230LayerP$STATE_SLEEP;
}

static inline void RF230LayerP$SpiResource$granted(void)
{
RF230LayerP$SELN$makeOutput();
RF230LayerP$SELN$set();
if (RF230LayerP$state == RF230LayerP$STATE_P_ON)
{
RF230LayerP$initRadio();
RF230LayerP$SpiResource$release();
}
else
{
RF230LayerP$Tasklet$schedule();
}
}

static inline void Atm128SpiP$Resource$default$granted(uint8_t id)
{
switch (arg_0x1ae4d8e0) {
case 0U:
RF230LayerP$SpiResource$granted();
break;
default:
Atm128SpiP$Resource$default$granted(arg_0x1ae4d8e0);
break;
}
}

static inline void Atm128SpiP$ResourceArbiter$granted(uint8_t id)
{
Atm128SpiP$Resource$granted(id);
}

static inline void /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$Resource$configure(uint8_t arg_0x1af20188){
/*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$ResourceConfigure$default$configure(arg_0x1af20188);
}
SimpleArbiterP$0$grantedTask$runTask(void)
#line 154
(__nesc_atomic_t *)__nesc_atomic = __nesc_atomic_start();

#line 155
{__nesc_atomic = __nesc_atomic_start();
  /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId =
  /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$reqResId;
  /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state =
  /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_BUSY;
}
#line 158
__nesc_atomic_end(__nesc_atomic);
}/__line 158
/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceConfigure$configure(
  /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$Resource$granted(
  /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
static inline void /*Atm128SpiC.Arbiter.Arbiter*/
  SimpleArbiterP$0$ResourceConfigure$default$unconfigure(const uint8_t id)
#line 172
{__nesc_temp = id;
  result = SchedulerBasicP$TaskBasic$postTask(
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$grantedTask);
  return result;
}

static inline resource_client_id_t /*Atm128SpiC.Arbiter.Queue*/
  FcfsResourceQueueC$0$FcfsQueue$dequeue(void)
#line 58
{__nesc_atomic = __nesc_atomic_start();
  if (/*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$qHead !=
      /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$NO_ENTRY) {
    __nesc_temp = /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$qHead;
    if (/*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$qTail ==
        /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$NO_ENTRY) {
      unsigned char __nesc_temp = id;
      return __nesc_temp;
    }
    /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$qTail =
        /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$qHead;
    __nesc_atomic_end(__nesc_atomic);
  }
  __nesc_temp = id;
  return __nesc_temp;
unsigned char __nesc_temp = /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$NO_ENTRY;
return __nesc_temp;

result = /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$FcfsQueue$dequeue();
return result;

result = /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$FcfsQueue$isEmpty();
return result;

bool released = FALSE;
{ __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state == /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_BUSY && /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId == id) {
  if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$QueueisEmpty() == FALSE) {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$reqResId = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$Queue$dequeue();
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_GRANTING;
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$grantedTask$postTask();
  } else {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$NO_RES;
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_IDLE;
  }
  } else { /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$grantedTask$postTask();

bool released = FALSE;
if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state == /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_BUSY && /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId == id) {
  if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$QueueisEmpty() == FALSE) {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$reqResId = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$Queue$dequeue();
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_GRANTING;
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$grantedTask$postTask();
  } else {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$NO_RES;
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_IDLE;
  }
}
released = TRUE;
}

#line 112
__nesc_atomic_end(__nesc_atomic); }
if (released == TRUE) {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceConfigure$
    unconfigure(id);
    return SUCCESS;
}
return FAIL;

#include "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Resource.nc"
inline static  error_t Atm128SpiP$ResourceArbiter$
    release(uint8_t arg_0x1ae49b10){
    #line 110
    unsigned char result;
    #line 110
    result = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$Resource$
        release(arg_0x1ae49b10);
    #line 110
    return result;
    #line 110
}

#include "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\ArbiterInfo.nc"
inline static bool Atm128SpiP$ArbiterInfo$inUse(void){
    #line 80
    unsigned char result;
    #line 80
    result = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$
        ArbiterInfo$inUse();
    #line 80
    return result;
    #line 80
}

#include "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm1281\McuSleepC.nc"
static inline void McuSleepC$McuPowerState$update(void)
#line 142{ }

#include "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\HplAtm128SpiP.nc"
static inline void HplAtm128SpiP$SPI$sleep(void)
#line 94{ }

#include "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\Atm128Spi.nc"
inline static void Atm128SpiP$Spi$enableSpi(bool arg_0x1ae45480){
    #line 99
    HplAtm128SpiP$SPI$enableSpi(arg_0x1ae45480);
    #line 99
}

#include "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\Atm128Spi.nc"
static inline void Atm128SpiP$stopSpi(void)
#line 119{ 
    Atm128SpiPSpi$enableSpi(FALSE);
    /* atomic removed: atomic calls only */
Atm128SpiP$Spi$sleep();
)
Atm128SpiP$McuPowerState$update();
)

# 44 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\McuPowerState.nc"
inline static void HplAtm128SpiP$McuPowerState$update(void){
#line 44
McuSleepC$McuPowerState$update();
#line 44

# 260 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessageP.nc"
static inline void RF230ActiveMessageP$RadioAlarm$fired(void)
#line 260
{

# 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioSend.nc"
inline static void RandomCollisionLayerP$RadioSend$sendDone(error_t arg_0x1ac23360)
#line 45
TrafficMonitorLayerP$SubSend$sendDone(arg_0x1lac23360);
#line 45

# 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessageP.nc"
static inline uint16_t RF230ActiveMessageP$RandomCollisionConfig$getCongestionBackoff(message_t *msg)
{
    return (uint16_t )
        (2240 * (737280UL / 8 / 32 * (1 << MICA_DIVIDE_ONE_FOR_32KHZ_LOG2) / 1000000.0));
}

# 35 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RandomCollisionConfig.nc"
inline static unsigned short RF230ActiveMessageP$RandomCollisionConfig$getCongestionBackoff(message_t *arg_0x1a9681f8)
{
    unsigned short result;
    result = RF230ActiveMessageP$RandomCollisionConfig$getCongestionBackoff(arg_0x1a9681f8);
    return result;
}

# 43 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SoftwareAckLayerP.nc"
inline static void SoftwareAckLayerP$SoftwareAckConfig$setAckReceived(message_t *msg)
{
    RF230ActiveMessageP$SoftwareAckConfig$setAckReceived(msg);
}

# 37 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioSend.nc"
inline static error_t SoftwareAckLayerP$SubSend$send(message_t *msg)
{
    unsigned char result;
    result = RF230LayerP$RadioSend$send(msg);
    return result;
}

# 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SoftwareAckLayerP.nc"
static inline error_t SoftwareAckLayerP$RadioSend$send(message_t *msg)
{
    error_t error;
    if (SoftwareAckLayerP$state == SoftwareAckLayerP$STATE_READY)
    {
        if ((error = SoftwareAckLayerP$SubSend$send(msg)) == SUCCESS)
        {
            SoftwareAckLayerP$state = SoftwareAckLayerP$STATE_BUSY;
            SoftwareAckLayerP$lastError = error;
        }
    }
    return error;
}

289
SoftwareAckLayerP$SoftwareAckConfig$
setAckReceived(msg, FALSE);
SoftwareAckLayerP$state = SoftwareAckLayerP$
STATE_DATA_SEND;
SoftwareAckLayerP$txMsg = msg;
}

else {
  error = EBUSY;
  return error;
}

#define 37 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RadioSend.nc"
inline static error_t RandomCollisionLayerP$SubSend$
send(message_t *arg_0x1ac24c88)
{
  unsigned char result;
  result = SoftwareAckLayerP$RadioSend$send(arg_0x1ac24c88);
  return result;
}

#define 195 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\MessageBufferLayerP.nc"
static inline void MessageBufferLayerP$RadioSend$ready(void)
{
  if (MessageBufferLayerP$state == MessageBufferLayerP$
STATE_TX_PENDING) {
    MessageBufferLayerP$sendTask$postTask();
  }
}

#define 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RadioSend.nc"
inline static void TrafficMonitorLayerP$RadioSend$ready(void)
{
  MessageBufferLayerP$RadioSend$ready();
}

#define 81 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\TrafficMonitorLayerP.nc"
static inline void TrafficMonitorLayerP$SubSend$ready(void)
{
  TrafficMonitorLayerP$RadioSend$ready();
}

#define 70 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\timer\HplAtm128Timer1P.nc"
static inline uint16_t HplAtm128Timer1P$Timer$get(void)
{
  return * (volatile uint16_t *)0x84;
}

#define 53 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\timer\Atm128AlarmC.nc"
static inline / *HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Timer$get(void)
{
  unsigned short result;
  result = HplAtm128Timer1P$Timer$get();
  return result;
}
# line 53
| return /*HplRF230C.AlarmC.NAlarm*/ Atm128AAlarmC$0$HplAtm128Timer$get(); |
# 98 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \lib\timer\Alarm.nc"
| inline static RadioAlarmP$Alarm$size_type RadioAlarmP$Alarm$getNow(void) |
| unsigned short result; |
| # line 98 |
| result = /*HplRF230C.AlarmC.NAlarm*/ Atm128AlarmC$0$Alarm$getNow(); |
| # line 98 |
| return result; |
| # line 98 |
| # 64 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RadioAlarmP.nc" |
| static __inline uint16_t RadioAlarmP$RadioAlarm$getNow(uint8_t id) |
| { return RadioAlarmP$Alarm$getNow(); } |
| # 53 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RandomCollisionLayerP.nc" |
| static inline void RandomCollisionLayerP$RadioAlarm$fired(void) |
| { error_t error; int16_t delay; for (;;) ; delay = (int16_t )RandomCollisionLayerP$txBarrier - RandomCollisionLayerP$RadioAlarm$getNow(); if (RandomCollisionLayerP$state == RandomCollisionLayerP$STATE_BARRIER) |
| RandomCollisionLayerP$state = RandomCollisionLayerP$STATE_READY; RandomCollisionLayerP$RadioSend$ready(); return; } |
| else { |
| # line 115 |
| if (RandomCollisionLayerP$state & RandomCollisionLayerP$STATE_BARRIER & delay > 0) { error = EBUSY; } |
| else { |
| # line 118 |
| error = RandomCollisionLayerP$SubSend$send(RandomCollisionLayerP$txMsg); |
| } |
| # line 120 |
| if (error != SUCCESS) |
| { if ((RandomCollisionLayerP$state & RandomCollisionLayerP$STATE_BARRIER) == RandomCollisionLayerP$STATE_TX_PENDING_FIRST) |
| RandomCollisionLayerP$state = RandomCollisionLayerP$STATE_BARRIER |
| RandomCollisionLayerP$STATE_TX_PENDING_SECOND; RandomCollisionLayerP$RadioAlarm$wait |
| RandomCollisionLayerP$getBackoff |
| RandomCollisionLayerP$Config$ |
getCongestionBackoff(RandomCollisionLayerP$txMsg));
}
else{
  if (RandomCollisionLayerP$state &
      RandomCollisionLayerP$STATE_BARRIER && delay > 0)
    { RandomCollisionLayerP$state =
      RandomCollisionLayerP$STATE_BARRIER;
      RandomCollisionLayerP$RadioAlarm$wait(delay);
    }
  else{
    RandomCollisionLayerP$state =
      RandomCollisionLayerP$STATE_READY;
  }
  RandomCollisionLayerP$RadioSend$sendDone(error);
}
else{
  RandomCollisionLayerP$state =
      RandomCollisionLayerP$STATE_TX_SENDING;
}
#line 144
static inline void RandomCollisionLayerP$
SubSend$sendDone(error_t error)
{
  for (; 0; ) ;
  RandomCollisionLayerP$state =
      RandomCollisionLayerP$STATE_READY;
  RandomCollisionLayerP$RadioSend$sendDone(error);
}
inline static void SoftwareAckLayerP$
RadioSend$sendDone(error_t arg_0x1ac23360){
#line 45
RandomCollisionLayerP$SubSend$sendDone(arg_0x1ac23360);
}
#line 45
# 124 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\TrafficMonitorLayerP.nc"
static inline void TrafficMonitorLayerP$TrafficMonitorConfig$channelError(void)
{
  if (TrafficMonitorLayerP$errorAverage < 255) {
    ++TrafficMonitorLayerP$errorAverage;
  }
  TrafficMonitorLayerP$TrafficMonitorConfig$channelError();
}
# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\TrafficMonitorConfig.nc"
inline static void RF230ActiveMessageP$TrafficMonitorConfig$channelError(void){
#line 50
RF230ActiveMessageP$TrafficMonitorConfig$channelError();
}
#line 50
# 137 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessageP.nc"
static inline void RF230ActiveMessageP$SoftwareAckConfig$reportChannelError(void)
{
  RF230ActiveMessageP$TrafficMonitorConfig$channelError();
  RF230ActiveMessageP$SoftwareAckConfig$reportChannelError();
}
# 74 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SoftwareAckConfig.nc"
inline static void SoftwareAckLayerP$SoftwareAckConfig$reportChannelError(void){
#line 74
RF230ActiveMessageP$SoftwareAckConfig$reportChannelError();
}
#line 74
# 110 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SoftwareAckLayerP.nc"
static inline void SoftwareAckLayerP$RadioAlarm$fired(void)
{
  for (; 0; ) ;
  SoftwareAckLayerP$SoftwareAckConfig$reportChannelError();
  SoftwareAckLayerP$state = SoftwareAckLayerP$STATE_READY;
  SoftwareAckLayerP$RadioSend$sendDone(SUCCESS);
}
# 537 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
static inline void RF230LayerP$RadioCCA$default$done(error_t error) {
    RF230LayerP$RadioCCA$done(arg_0x1ac3a920);
}

static __inline uint8_t RF230LayerP$readRegister(uint8_t reg) {
    for (; 0; ) ;
    RF230LayerP$SELN$clr();
    RF230LayerP$HplRF230$spiSplitWrite(RF230_CMD_REGISTER_READ | reg);
    RF230LayerP$HplRF230$spiSplitReadWrite(0);
    reg = RF230LayerP$HplRF230$spiSplitRead();
    RF230LayerP$SELN$set();
    return reg;
}

static inline void RF230LayerP$RadioAlarm$fired(void) {
    if (RF230LayerP$state == RF230LayerP$STATE_SLEEP_2_TRX_OFF) {
        RF230LayerP$state = RF230LayerP$STATE_TRX_OFF;
    } else {
        RF230LayerP$cmd = RF230LayerP$CMD_CCA;
        uint8_t cca;
        for (; 0; ) ;
        RF230LayerP$cmd = RF230LayerP$CMD_NONE;
        cca = RF230LayerP$readRegister(RF230_TRX_STATUS);
        for (; 0; ) ;
        RF230LayerP$RadioCCA$done(cca & RF230_CCA_DONE ? cca & RF230_CCA_STATUS ? SUCCESS : EBUSY : FAIL);
        RF230LayerP$Tasklet$schedule();
    }
}

static inline void RadioAlarmP$RadioAlarm$default$fired(uint8_t id) {
    ...
}

inline static void RadioAlarmP$RadioAlarm$fired(uint8_t arg_0x1abccd98) {
    switch (arg_0x1abccd98) {
    case 0U:
        RF230ActiveMessageP$RadioAlarm$fired();
        break;
    case 1U:
        RandomCollisionLayerP$RadioAlarm$fired();
        break;
    case 2U:
        SoftwareAckLayerP$RadioAlarm$fired();
        break;
    case 3U:
        RF230LayerP$RadioAlarm$fired();
        break;
    default:
    ...
}
RadioAlarmP$RadioAlarm$default$fired(arg_0x1abccd98);

static inline void RadioAlarmP$Tasklet$run(void)
{
    if (RadioAlarmP$state == RadioAlarmP$STATE_FIRED)
    {
        RadioAlarmP$state = RadioAlarmP$STATE_READY;
        RadioAlarmP$RadioAlarm$fired(RadioAlarmP$alarm);
    }
}

static inline void MessageBufferLayerP$Tasklet$run(void)
{
}

static inline void TrafficMonitorLayerP$Tasklet$run(void)
{
}

static __inline bool RadioAlarmP$RadioAlarm$isFree(uint8_t id)
{
    return RadioAlarmP$state == RadioAlarmP$STATE_READY;
}

static inline void RandomCollisionLayerP$SubSend$ready(void)
{
    if (RandomCollisionLayerP$state == RandomCollisionLayerP$STATE_READY &&
        RandomCollisionLayerP$RadioAlarm$isFree()) {
        RandomCollisionLayerP$RadioSend$ready();
    }
}

static inline void SoftwareAckLayerP$SubSend$ready(void)
{
    if (SoftwareAckLayerP$state == SoftwareAckLayerP$STATE_READY) {
        SoftwareAckLayerP$RadioSend$ready();
    }
}

static inline void RF230LayerP$RadioSend$ready(void)
{
    SoftwareAckLayerP$SubSend$ready();
}

}
unsigned char result = SchedulerBasicP$TaskBasic$postTask(MessageBufferLayerP$stateDoneTask);
return result;
}

static inline void MessageBufferLayerP$RadioState$done(void)
{
MessageBufferLayerP$stateDoneTask$postTask();
}

inline static void TrafficMonitorLayerP$RadioState$done(void)
{
TrafficMonitorLayerP$RadioState$done();
}

inline static error_t TrafficMonitorLayerP$startStopTimer$postTask(void)
{
unsigned char result = SchedulerBasicP$TaskBasic$postTask(TrafficMonitorLayerP$startStopTimer);
return result;
}

static inline void RF230LayerP$IRQ$disable(void)
{
RF230LayerP$Capture$stop();
}

static inline void HplRF230P$IRQ$disable(void)
{
HplRF230P$Capture$stop();
}

static inline void HplRF230P$Capture$stop(void)
{

}

static inline void RF230LayerP$changeChannel(void)
{
for (; 0; ) ;
for (; 0; ) ;
if (RF230LayerP$isSpiAcquired())
{
RF230LayerP$writeRegister(RF230_PHY_CC_CCA, RF230_CCA_MODE_VALUE | RF230LayerP$channel);
if (RF230LayerP$state == RF230LayerP$STATE_RX_ON)
{
RF230LayerP$state = RF230LayerP$STATE_TRX_OFF_2_RX_ON;
}
else {
RF230LayerP$cmd = RF230LayerP$CMD_SIGNAL_DONE;
}

}
HplRF230P$IRQ$disable();
#line 55
# 145 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm1281Timer1P.nc"
static inline void HplAtm1281Timer1P$Capture$start(void)
#line 145
# 58 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Capture.nc"
inline static void HplRF230P$Capture$start(void){
#line 58
HplAtm1281Timer1P$Capture$start();
#line 58
#line 58
# 139 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm1281Timer1P.nc"
static inline void HplAtm1281Timer1P$Capture$reset(void)
#line 139
# 55 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Capture.nc"
inline static void HplRF230P$Capture$reset(void)
#line 55
HplAtm1281Timer1P$Capture$reset();
#line 55
# 135 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm1281Timer1P.nc"
static inline void HplAtm1281Timer1P$Capture$setEdge(bool up)
#line 135
# 79 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Capture.nc"
inline static void HplRF230P$Capture$setEdge(bool arg_0x1ab09b48)
#line 79
HplAtm1281Timer1P$Capture$setEdge(arg_0x1ab09b48);
#line 79
# 68 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\platforms\iris\chips\rf230\HplRF230P.nc"
static inline error_t HplRF230P$IRQ$captureRisingEdge(void)
{
    HplRF230P$Capture$setEdge(TRUE);
    HplRF230P$Capture$reset();
    HplRF230P$Capture$start();
    return SUCCESS;
}
# 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\GpioCapture.no"
inline static error_t RF230LayerP$IRQ$captureRisingEdge(void)
{
    unsigned char result;
    result = HplRF230P$IRQ$captureRisingEdge();
    return result;
}
# 69 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\Atm128AlarmC.nc"
static inline void */HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$Alarm$start();
# 55 \"C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\lib\\timer\\Alarm.nc\" 
inline static void RadioAlarmP$RadioAlarm$wait(uint8_t id, uint16_t timeout) 
for (; 0; ) ;
RadioAlarmP$alarm = id;
RadioAlarmP$state = RadioAlarmP$STATE_WAIT;
RadioAlarmP$RadioAlarm$start(timeout);

# 38 \"C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\rf230\\RF230LayerP.nc\" 
inline static void RF230LayerP$RadioAlarm$wait(uint16_t arg_0x1a995d20) {
  for (; 0; ) ;
  RF230LayerP$RadioAlarm$wait(3U, arg_0x1a995d20);
}
inline static bool RF230LayerP$RadioAlarm$isFree(void) {
  unsigned char result;
  result = RadioAlarmP$RadioAlarm$isFree(3U);
  return result;
}
# line 315 \"C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\rf230\\RF230LayerP.nc\" 
static __inline void RF230LayerP$changeState(void) {
  if (0) {
      RF230LayerP$SLP_TR$clr();
      RF230LayerP$RadioAlarm$wait(RF230LayerP$SLEEP_WAKEUP_TIME);
      RF230LayerP$state = RF230LayerP$STATE_SLEEP_2_TRX_OFF;
    } else { ...
  } else { ...
  RF230LayerP$cmd = RF230LayerP$CMD_TURNON && RF230LayerP$state = RF230LayerP$STATE_TRX_OFF & RF230LayerP$isSpiAcquired()) {
    for (; 0; ) ;
    RF230LayerP$readRegister(RF230_IRQ_STATUS);
    RF230LayerP$IRQ_captureRisingEdge();
    RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_RX_ON);
    RF230LayerP$state = RF230LayerP$STATE_TRX_OFF_2_RX_ON;
  } else { ...
  }
}
RF230LayerP$state = RF230LayerP$STATE_TRX_OFF;
}
}

#line 336
if (RF230LayerP$cmd == RF230LayerP$CMD_TURNOFF &&
RF230LayerP$state == RF230LayerP$STATE_TRX_OFF) {
RF230LayerP$SLP_TR$set();
RF230LayerP$state = RF230LayerP$STATE_SLEEP;
RF230LayerP$cmd = RF230LayerP$CMD_SIGNAL_DONE;
} else {
#line 342
if (RF230LayerP$cmd == RF230LayerP$CMD_STANDBY &&
RF230LayerP$state == RF230LayerP$STATE_TRX_OFF) {
RF230LayerP$cmd = RF230LayerP$CMD_SIGNAL_DONE;
}
}

inline static error_t MessageBufferLayerP$deliverTask$postTask(void){
unsigned char result;
#line 56
result = SchedulerBasicP$TaskBasic$postTask(MessageBufferLayerP$deliverTask);
#line 56
return result;
#line 56
}

static inline message_t *MessageBufferLayerP$RadioReceive$receive(message_t *msg)
{
message_t *m;
{ __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
  if (MessageBufferLayerP$receiveQueueSize >=
      MessageBufferLayerP$RECEIVE_QUEUE_SIZE) {m = msg;
  } else {
    uint8_t index = MessageBufferLayerP$receiveQueueHead +
        MessageBufferLayerP$receiveQueueSize;
    #line 309
    if (index >= MessageBufferLayerP$RECEIVE_QUEUE_SIZE) {
      index -= MessageBufferLayerP$RECEIVE_QUEUE_SIZE;
    }
    m = MessageBufferLayerP$receiveQueue[index];
    MessageBufferLayerP$receiveQueue[index] = msg;
    ++MessageBufferLayerP$receiveQueueSize;
    MessageBufferLayerP$deliverTask$postTask();
  }
  #line 318
  __nesc_atomic_end(__nesc_atomic); }
  return m;
}

# 42 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinysos-2.x\\tos\\chips\\rf230\\MessageBufferLayerP.nc"
inline static message_t *UniqueLayerP$RadioReceive$receive(message_t *arg_0x1ac3dab8){
#line 42 nx_struct message_t +result;
#line 42
result = MessageBufferLayerP$RadioReceive$receive(arg_0x1ac3dab8);
#line 42
return result;
#line 42
}

# 152 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinysos-2.x\\tos\\chips\\rf230\\NeighborhoodP.nc"
static __inline void NeighborhoodP$NeighborhoodFlag$set(uint8_t bit, uint8_t index)
{
  NeighborhoodP$flags[index] |= 1 << bit;
}

298
inline static void UniqueLayerP$NeighborhoodFlag$ set(uint8_t arg_0x1ac90590) {
  NeighborhoodP$NeighborhoodFlag$set(0U, arg_0x1ac90590);
}

inline static void RF230ActiveMessageP$UniqueConfig$ reportChannelError(void) {
  RF230ActiveMessageP$TrafficMonitorConfig$channelError();
}

inline static bool NeighborhoodP$NeighborhoodFlag$get(uint8_t arg_0x1ac900b8) {
  unsigned char result;
  result = NeighborhoodP$NeighborhoodFlag$get(0U, arg_0x1ac900b8);
  return result;
}

static inline uint8_t IEEE154PacketP$IEEE154Packet$getDSN(message_t *msg) {
  return __nesc_ntoh_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$getHeader(msg)->dsn);
}

static inline uint8_t RF230ActiveMessageP$UniqueConfig$getSequenceNumber(message_t *msg) {
  RF230ActiveMessageP$UniqueConfig$reportChannelError();
  return RF230ActiveMessageP$IEEE154Packet$getDSN(msg);
}

static inline uint8_t IEEE154PacketP$IEEE154Packet$getDSN(message_t *msg) {
  return __nesc_ntoh_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$getHeader(msg)->dsn);
}

static inline uint8_t RF230ActiveMessageP$UniqueConfig$getSequenceNumber(message_t *msg) {
  return RF230ActiveMessageP$IEEE154Packet$getDSN(msg);
}
getSequenceNumber(arg_0x1a9412c8);
#line 31
#line 31
return result;
#line 31

getSrcAddr(message_t *msg)
{ return __nesc_ntoh_leuint16((unsigned char *)&IEEE154PacketP$getHeader(msg)->src); }

RF230ActiveMessageP$IEEE154Packet$ getSrcAddr(message_t *arg_0x1a983388)
{ unsigned short result;
  result = IEEE154PacketP$IEEE154Packet$getSrcAddr(arg_0x1a983388);
  return result;
}

RF230ActiveMessageP$UniqueConfig$ getSender(message_t *msg)
{ return RF230ActiveMessageP$IEEE154Packet$getSrcAddr(msg); }

UniqueLayerP$UniqueConfig$ getSender(message_t *arg_0x1a9417d8)
{ unsigned short result;
  result = RF230ActiveMessageP$UniqueConfig$getSender(arg_0x1a9417d8);
  return result;
}

insertNode(am_addr_t arg_0x1ac95c40)
{ unsigned char result;
  result = NeighborhoodP$Neighborhood$insertNode(arg_0x1ac95c40);
  return result;
}

receive(message_t *msg)
{ uint8_t index = UniqueLayerP$Neighborhood$insertNode(UniqueLayerP$UniqueConfig$getSender(msg));
  uint8_t dsn = UniqueLayerP$UniqueConfig$getSequenceNumber(msg);
  if (UniqueLayerP$NeighborhoodFlag$get(index))
  { uint8_t diff = dsn - UniqueLayerP$receivedNumbers[index];
    if (diff == 0)
    { UniqueLayerP$UniqueConfig$reportChannelError();
      return msg;
    }
  }
  else
  { UniqueLayerP$NeighborhoodFlag$set(index);
  }
}
UniqueLayerP$receivedNumbers[index] = dsn;
return UniqueLayerP$RadioReceive$receive(msg);
}

inline static message_t *TrafficMonitorLayerP$RadioReceive$receive(message_t *arg_0x1ac3dab8)
{
    nx_struct message_t *result;
    result = UniqueLayerP$SubReceive$receive(arg_0x1ac3dab8);
    return result;
}

inline static void TrafficMonitorLayerP$NeighborhoodFlag$set(uint8_t arg_0x1ac90590)
{
    NeighborhoodP$NeighborhoodFlag$set(1U, arg_0x1ac90590);
}

inline static bool TrafficMonitorLayerP$NeighborhoodFlag$get(uint8_t arg_0x1ac900b8)
{
    unsigned char result;
    result = NeighborhoodP$NeighborhoodFlag$get(1U, arg_0x1ac900b8);
    return result;
}

static inline am_addr_t RF230ActiveMessageP$TrafficMonitorConfig$getSender(message_t *msg)
{
    return RF230ActiveMessageP$IEEE154Packet$getSrcAddr(msg);
}

inline static am_addr_t TrafficMonitorLayerP$TrafficMonitorConfig$getSender(message_t *arg_0x1a954b08)
{
    unsigned short result;
    result = RF230ActiveMessageP$TrafficMonitorConfig$getSender(arg_0x1a954b08);
    return result;
}

inline static uint8_t TrafficMonitorLayerP$Neighborhood$insertNode(am_addr_t arg_0x1ac95c40)
{
    unsigned char result;
    result = NeighborhoodP$Neighborhood$insertNode(arg_0x1ac95c40);
    return result;
}

inline static uint16_t TrafficMonitorLayerP$TrafficMonitorConfig$getChannelTime(message_t *arg_0x1a9545e0)
{
    unsigned short result;
    result = RF230ActiveMessageP$TrafficMonitorConfig$getChannelTime(arg_0x1a9545e0);
    return result;
}
result = RF230ActiveMessageP$TrafficMonitorConfig$getChannelTime(arg_0x1a9545e0);
return result;

static inline message_t *TrafficMonitorLayerP$SubReceive$receive(message_t *msg) {
  uint8_t index;
  TrafficMonitorLayerP$rxAverage += TrafficMonitorLayerP$TrafficMonitorConfig$getChannelTime(msg);
  index = TrafficMonitorLayerP$Neighborhood$insertNode(TrafficMonitorLayerP$TrafficMonitorConfig$getSender(msg));
  if (!TrafficMonitorLayerP$NeighborhoodFlag$get(index)) {
    if (TrafficMonitorLayerP$neighborCount < TrafficMonitorLayerP$TRAFFIC_MONITOR_UINT8_MAX) {
      ++TrafficMonitorLayerP$neighborCount;
      TrafficMonitorLayerP$NeighborhoodFlag$set(index);
    }
  }
  return TrafficMonitorLayerP$RadioReceive$receive(msg);
}

inline static message_t *RandomCollisionLayerP$RadioReceive$receive(message_t *msg) {
  result = TrafficMonitorLayerP$SubReceive$receive(msg);
  return result;
}

inline static bool RF230ActiveMessageP$IEEE154Packet$requiresAckReply(message_t *msg)
{
  unsigned char result;
  result = IEEE154PacketP$IEEE154Packet$requiresAckReply(msg);
  return result;
}

inline static uint16_t RF230ActiveMessageP$RadioAlarm$getNow(void) {
  unsigned short result;
  result = RadioAlarmP$RadioAlarm$getNow(0U);
  return result;
}

inline static rf230activemessagep$randomcollisionconfig$gettransmitbarrier(message_t *msg) {
  uint16_t time;
  time = RF230ActiveMessageP$RadioAlarm$getNow(void);
  if (RF230ActiveMessageP$IEEE154Packet$requiresAckReply(msg)) {
    time = time + (((((32 * (-5 + 16 + 11 + 5) * (7372800UL / 8 / 32)) / 1000000.0) / (1 << MICA_DIVIDE_ONE_FOR_32KHZ_LOG2)) / 1000000.0));
  }
}
else {
    #line 255
    time += (uint16_t )(32 * (-5 + 5) * (7372800UL / 8 / 32 *
    (1 << MICA_DIVIDE_ONE_FOR_32KHZ_LOG2) / 1000000.0));
    }
    return time;
}

# 46 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RandomCollisionConfig.nc"
inline static uint16_t RandomCollisionLayerP$Config$getTransmitBarrier(message_t *arg_0x1a968ac0){
    #line 46
    unsigned short result;
    #line 46
    #line 46
    result = RF230ActiveMessageP$RandomCollisionConfig$getTransmitBarrier(arg_0x1a968ac0);
    #line 46
    #line 46
    return result;
    #line 46
}

# 157 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RandomCollisionLayerP.nc"
static inline message_t *RandomCollisionLayerP$SubReceive$receive(message_t *msg){
    int16_t delay;
    RandomCollisionLayerP$txBarrier = RandomCollisionLayerP$Config$getTransmitBarrier(msg);
    delay = RandomCollisionLayerP$txBarrier - RandomCollisionLayerP$RadioAlarm$getNow();
    if (delay > 0)
    {
        if (RandomCollisionLayerP$state ==
            RandomCollisionLayerP$STATE_READY)
        {
            RandomCollisionLayerP$RadioAlarm$wait(delay);
            RandomCollisionLayerP$state =
            RandomCollisionLayerP$STATE_BARRIER;
        } else {
            RandomCollisionLayerP$state |=
            RandomCollisionLayerP$STATE_BARRIER;
        }
    }
    return RandomCollisionLayerP$RadioReceive$receive(msg);
}

# 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RadioReceive.nc"
inline static message_t *SoftwareAckLayerP$RadioReceive$receive(message_t *arg_0x1ac3dab8){
    #line 42
    nx_struct message_t *result;
    #line 42
    #line 42
    result = RandomCollisionLayerP$SubReceive$receive(arg_0x1ac3dab8);
    #line 42
    #line 42
    return result;
    #line 42
}

# 111 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154PacketP.nc"
static __inline void IEEE154PacketP$IEEE154Packet$createAckReply(message_t *data, message_t *ack)
{
    ieee154_header_t *header = IEEE154PacketP$getHeader(ack);
    ieee154_header_t *header = IEEE154PacketP$getHeader(ack);
    __nesc_hton_leuint8((unsigned char *)&header->length, IEEE154PacketP$IEEE154_ACK_FRAME_LENGTH);
    __nesc_hton_leuint16((unsigned char *)&header->fcf, IEEE154PacketP$IEEE154_ACK_FRAME_VALUE);
    __nesc_hton_leuint8((unsigned char *)&header->dsn, IEEE154PacketP$IEEE154_ACK_FRAME_VALUE);
    __nesc_ntoh_leuint8((unsigned char *)&
    IEEE154PacketP$getHeader(data)->dsn);
    }
}

# 92 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154Packet.nc"
inline static void RF230ActiveMessageP$IEEE154Packet$createAckReply(message_t *arg_0x1a9719e8, message_t *arg_0x1a9719e8){
    #line 92
    IEEE154PacketP$IEEE154Packet$createAckReply( arg_0x1a971838, arg_0x1a9719e8);

    }
static inline void RF230ActiveMessageP$SoftwareAckConfig$createAckPacket(message_t *data, message_t *ack)
{
    RF230ActiveMessageP$IEEE154Packet$createAckReply(data, ack);
}

inline static void SoftwareAckLayerP$SoftwareAckConfig$createAckPacket(message_t *arg_0x1a945e08, message_t *arg_0x1a944010)
{
    RF230ActiveMessageP$SoftwareAckConfig$createAckPacket(arg_0x1a945e08, arg_0x1a944010);
}

static inline bool RF230ActiveMessageP$SoftwareAckConfig$requiresAckReply(message_t *msg)
{
    return RF230ActiveMessageP$IEEE154Packet$requiresAckReply(msg);
}

inline static bool SoftwareAckLayerP$SoftwareAckConfig$requiresAckReply(message_t *arg_0x1a9458d0)
{
    unsigned char result;
    result = RF230ActiveMessageP$SoftwareAckConfig$requiresAckReply(arg_0x1a9458d0);
    return result;
}

static inline void HplAtm1281Timer1P$CompareA$stop(void)
{
    * (volatile uint8_t *)0x6F &= ~((1 << 1));
}

inline static void / *HplRF230C.AlarmC.NAlarm*/ Atm128AlarmC$0$HplAtm128Compare$stop(void)
{
    Atm128AlarmC$0$Alarm$stop();
}

inline static void RadioAlarmP$Alarm$stop(void)
{
    /*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$Alarm$stop();
}

static inline void RadioAlarmP$RadioAlarm$cancel(uint8_t id)
{
    for (; 0; ) ;
    RadioAlarmP$Alarm$state = RadioAlarmP$STATE_READY;
}

# line 43 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RadioAlarmP.nc"
static inline void RadioAlarmP$RadioAlarm$cancel(uint8_t id)
{
    for (; 0; ) ;
    RadioAlarmP$Alarm$state = RadioAlarmP$STATE_READY;
}
inline static void SoftwareAckLayerP$RadioAlarm$cancel(void){
    #line 43
    RadioAlarmP$RadioAlarm$cancel(2U);
    #line 43
    #line 48 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
    \chips\rf230\IEEE154PacketP.nc"
    static __inline bool IEEE154PacketP$IEEE154Packet$isAckFrame(message_t *msg)
    {
        IEEE154PacketP$IEEE154_ACK_FRAME_VALUE;
    }
    # 79 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
    \chips\rf230\IEEE154Packet.nc"
    inline static bool RF230ActiveMessageP$IEEE154Packet$ isAckFrame(message_t *arg_0x1a972d18)
    {
        unsigned char result;
        result = IEEE154PacketP$IEEE154Packet$isAckFrame(arg_0x1a972d18);
        return result;
    }
    #line 79
    #line 79
    #line 79
    # 104 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
    \chips\rf230\RF230ActiveMessageP.nc"
    static inline bool RF230ActiveMessageP$SoftwareAckConfig$ isAckPacket(message_t *msg)
    {
        return RF230ActiveMessageP$IEEE154Packet$isAckFrame(msg);
    }
    # 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
    \chips\rf230\SoftwareAckLayerP.nc"
    inline static bool SoftwareAckLayerP$SoftwareAckConfig$ isAckPacket(message_t *arg_0x1a947b00)
    {
        unsigned char result;
        result = SoftwareAckLayerP$SoftwareAckConfig$ isAckPacket(arg_0x1a947b00);
        return result;
    }
    #line 50
    unsigned char ack = SoftwareAckLayerP$SoftwareAckConfig$isAckPacket(msg);
    if (!ack) {
        SoftwareAckLayerP$RadioAlarmScancel();
        SoftwareAckLayerP$SoftwareAckConfig$setAckReceived
        SoftwareAckLayerP$state = SoftwareAckLayerP$STATE_ACK_WAIT;
    }
    if (SoftwareAckLayerP$SoftwareAckConfig$ requiresAckReply(msg))
    {
        SoftwareAckLayerP$SoftwareAckConfig$ createAckPacket(msg, &SoftwareAckLayerP$ackMsg);
        if (SoftwareAckLayerP$SubSend$send(&SoftwareAckLayerP$ackMsg) == SUCCESS) {
            SoftwareAckLayerP$state = SoftwareAckLayerP$STATE_ACK_SEND;
        }
        else {
            SoftwareAckLayerP$SubReceive$receive(msg);
        }
    }
    else {
        SoftwareAckLayerP$RadioSend$sendDone(SUCCESS);
    }
    SoftwareAckLayerP$state = SoftwareAckLayerP$STATE_ACK_WAIT;
}
}
return SoftwareAckLayerP$RadioReceive$receive(msg);

inline static message_t *RF230LayerP$RadioReceive$
receive(message_t *arg_0x1ac3dab8)
{
  #line 42
  nx_struct message_t *result;
  #line 42
  #line 42
  result = SoftwareAckLayerP$SubReceive$receive(arg_0x1ac3dab8);
  #line 42
  #line 42
  return result;
  #line 42
}

static inline void RF230PacketP$PacketLinkQuality$
set(message_t *msg, uint8_t value)
{
  RF230PacketP$getMeta(msg)->lqi = value;
}

inline static void RF230LayerP$PacketLinkQuality$
set(message_t *arg_0x1a8a8c00,RF230LayerP$PacketLinkQuality$
value_type arg_0x1a8a8d90)
{
  #line 46
  RF230PacketP$PacketLinkQuality$
  set(arg_0x1a8a8c00, arg_0x1a8a8d90);
  #line 46
}

static __inline uint16_t HplRF230P$HplRF230$
crcByte(uint16_t crc, uint8_t data)
{
  return _crc_ccitt_update(crc, data);
}

inline static uint16_t RF230LayerP$HplRF230$
crcByte(uint16_t arg_0x1ad638f8, uint8_t arg_0x1ad63a80)
{
  unsigned short result;
  result = HplRF230P$HplRF230$
crcByte(arg_0x1ad638f8, arg_0x1ad63a80);
  return result;
}
return result;

static inline bool MessageBufferLayerP$RadioReceive$ header(message_t *msg) {
  __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
  notFull = MessageBufferLayerP$receiveQueueSize < MessageBufferLayerP$RECEIVE_QUEUE_SIZE;
  __nesc_atomic_end(__nesc_atomic);
  return notFull;
}

unsigned char result;
result = MessageBufferLayerP$RadioReceive$ header(arg_0x1ac3d3e0);
return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

result = UniqueLayerP$SubReceive$header(arg_0x1ac3d3e0);
return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;

return result;
inline static bool SoftwareAckLayerP$RadioReceive$ header(message_t *arg_0x1ac3d3e0) {
    #line 35
    unsigned char result;
    #line 35
    #line 35
    result = RandomCollisionLayerP$SubReceive$header(arg_0x1ac3d3e0);
    #line 35
    #line 35
    return result;
    #line 35
    #line 35
    #line 120 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\IEEE154PacketP.nc"
static __inline bool IEEE154PacketP$IEEE154Packet$ verifyAckReply(message_t *data, message_t *ack) {
    ieee154_header_t *header = IEEE154PacketP$getHeader(ack);
    return __nesc_ntoh_leuint8((unsigned char *)&header->dsn) ==
    __nesc_ntoh_leuint8((unsigned char *)&
    IEEE154PacketP$getHeader(data)->dsn) &&
    __nesc_ntoh_leuint16((unsigned char *)&header->fcf) &
    IEEE154PacketP$IEEE154_ACK_FRAME_MASK ==
    IEEE154PacketP$IEEE154_ACK_FRAME_VALUE;
} # 98 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\IEEE154Packet.nc"
inline static bool RF230ActiveMessageP$IEEE154Packet$ verifyAckReply(message_t *arg_0x1a989010, message_t *arg_0x1a9891c0) {
    #line 98
    unsigned char result;
    #line 98
    #line 98
    result = IEEE154PacketP$IEEE154Packet$verifyAckReply(arg_0x1a989010, arg_0x1a9891c0);
    #line 98
    #line 98
    return result;
    #line 98
}
# 109 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RF230ActiveMessageP.nc"
static inline bool RF230ActiveMessageP$SoftwareAckConfig$ verifyAckPacket(message_t *data, message_t *ack) {
    #line 98
    unsigned char result;
    #line 98
    #line 98
    result = RF230ActiveMessageP$IEEE154Packet$verifyAckReply(data, ack);
    #line 98
    #line 98
    return RF230ActiveMessageP$IEEE154Packet$verifyAckReply(data, ack);
} # 57 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\SoftwareAckConfig.nc"
static inline bool SoftwareAckLayerP$SoftwareAckConfig$ verifyAckPacket(message_t *arg_0x1a945120, message_t *arg_0x1a9452d0) {
    #line 57
    unsigned char result;
    #line 57
    #line 57
    result = RF230ActiveMessageP$SoftwareAckConfig$ verifyAckPacket(arg_0x1a945120, arg_0x1a9452d0);
    #line 57
    #line 57
    return result;
    #line 57
}
# 120 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\SoftwareAckLayerP.nc"
static inline bool SoftwareAckLayerP$SubReceive$ header(message_t *msg) {
    if (SoftwareAckLayerP$SoftwareAckConfig$ isAckPacket(msg)) { return SoftwareAckLayerP$state ==
    SoftwareAckLayerP$STATE_ACK_WAIT &&
    SoftwareAckLayerP$SoftwareAckConfig$ verifyAckPacket(SoftwareAckLayerP$txMsg, msg); }
else { #line 125
    return SoftwareAckLayerP$RadioReceive$ header(msg);
}
# 35 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RadioReceive.nc"
inline static bool RF230LayerP$RadioReceive$
header(message_t *arg_0x1ac3d3e0){
    #line 35
    unsigned char result;
    #line 35
    #line 35
    result = SoftwareAckLayerP$SubReceive$header(arg_0x1ac3d3e0);
    #line 35
    #line 35
    return result;
    #line 35
}

# 75 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos
\\chips\\rf230\\RF230ActiveMessageP.nc"
static inline uint8_t RF230ActiveMessageP$RF230Config$getHeaderLength(void)
    { return 7; }

# 55 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos
\\chips\\rf230\\RF230Config.nc"
inline static uint8_t RF230LayerP$RF230Config$getHeaderLength(void){
    #line 55
    unsigned char result;
    #line 55
    #line 55
    result = RF230ActiveMessageP$RF230Config$getHeaderLength();
    #line 55
    #line 55
    return result;
    #line 55
}

# 63 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos
\\chips\\rf230\\IEEE154PacketP.nc"
static __inline ieee154_header_t *IEEE154PacketP$IEEE154Packet$getHeader(message_t *msg)
    { return IEEE154PacketP$IEEE154Packet$getHeader(msg); }

# 39 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos
\\chips\\rf230\\IEEE154Packet.nc"
inline static void IEEE154PacketP$IEEE154Packet$setLength(message_t *msg)
    { return (uint8_t *)(RF230ActiveMessageP$IEEE154Packet$getHeader(msg) + 1); }

# 46 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos
\\chips\\rf230\\RF230ActiveMessageP.nc"
inline static void RF230ActiveMessageP$RF230Config$setLength(message_t *msg)
    { return (uint8_t *)(RF230ActiveMessageP$IEEE154Packet$setLength(msg) - 1); }
setLength(arg_0x1a975ee0, arg_0x1a973088);
#line 49

# 60 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessageP.nc"
static inline void RF230ActiveMessageP$RF230Config$
setLength(message_t *msg, uint8_t len)
{
    RF230ActiveMessageP$IEEE154Packet$setLength(msg, len);
}

# 39 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230Config.nc"
inline static void RF230LayerP$RF230Config$
setLength(message_t *msg, uint8_t arg_0x1a934500)
{
    RF230ActiveMessageP$RF230Config$
    setLength(arg_0x1a934378, arg_0x1a934500);
}

# 39

# 81 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessageP.nc"
static inline uint8_t RF230ActiveMessageP$RF230Config$
getMaxLength(void)
{
    return sizeof(rf230packet_header_t) - 1 + 28 +
    sizeof(ieee154_footer_t);
}

# 59 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\platforms\iris\chips\rf230\HplRF230P.nc"
static __inline uint8_t HplRF230P$HplRF230$
spiWrite(uint8_t data)
{
    * (volatile uint8_t *)(0X2E + 0x20) = data;
    while (!(* (volatile uint8_t *)(0x2D + 0x20) & 0x80));
    return * (volatile uint8_t *)(0X2E + 0x20);
}

# 541 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230LayerP.nc"
static __inline void RF230LayerP$downloadMessage(void)
{
    uint8_t length;
    uint16_t crc;
    RF230LayerP$SELN$clr();
    RF230LayerP$HplRF230P$spiWrite(RF230_CMD_FRAME_READ);
    length = RF230LayerP$HplRF230P$spiWrite(0);
    if (length >= 3 && length <= RF230LayerP$RF230Config$
        getMaxLength())
    {
        uint8_t read;
        uint8_t *data;
        RF230LayerP$HplRF230P$spiSplitWrite(0);
RF230LayerP$RF230Config$setLength(RF230LayerP$rxMsg, length);
data = RF230LayerP$RF230Config$getPayload(RF230LayerP$rxMsg);
crc = 0;
length -= 2;
read = RF230LayerP$RF230Config$getHeaderLength();
if (length < read) {
    read = length;
    length -= read;
do {
    crc = RF230LayerP$HplRF230$crcByte(crc, * data++ = RF230LayerP$HplRF230$spiSplitReadWrite(0));
} while (--read != 0);
if (RF230LayerP$RadioReceive$header(RF230LayerP$rxMsg))
    { while (length-- != 0)
        if (crc == 0) {
            RF230LayerP$PacketLinkQuality$set(RF230LayerP$rxMsg, RF230LayerP$HplRF230$spiSplitRead());
        } else {
            crc = 1;
        }
    } else {
        #line 593
        crc = 1;
    } RF230LayerP$SELN$set();
RF230LayerP$state = RF230LayerP$STATE_RX_ON;
#line 613
RF230LayerP$cmd = RF230LayerP$CMD_NONE;
if (crc == 0) {
    RF230LayerP$rxMsg = RF230LayerP$RadioReceive$receive(RF230LayerP$rxMsg);
} else {
    #line 593
    crc = 1;
} # 227 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RF230PacketP.nc"
static inline void RF230PacketP$PacketRSSI$clear(message_t *msg)
{
    RF230PacketP$getMeta(msg)->flags &= ~RF230PACKET_RSSI;
}
# 40 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\PacketField.nc"
inline static void RF230LayerP$PacketRSSI$clear(message_t *arg_0x1a8a8718)
{
    #line 40
    RF230PacketP$PacketRSSI$clear(arg_0x1a8a8718);
}
# 40
# 132 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\PacketField.nc"
static inline void RF230PacketP$PacketRSSI$set(message_t *msg, uint8_t value)
{
    RF230PacketP$getMeta(msg)->flags &= ~RF230PACKET_TXPOWER;
    RF230PacketP$PacketRSSI$set(msg) = RF230PACKET_RSSI;
    RF230PacketP$PacketRSSI$power = value;
}
# 46 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\PacketField.nc"
inline static void RF230LayerP$PacketRSSI$set(message_t *arg_0x1a8a8c00, RF230LayerP$PacketRSSI$value_type arg_0x1a8a8d90)
{
    #line 46
    RF230PacketP$PacketRSSI$set(arg_0x1a8a8c00, arg_0x1a8a8d90);
}
# 46
# 132 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\ActiveMessageP.nc"
static inline uint16_t RF230ActiveMessageP$SoftwareAckConfig$getAckTimeout(void)
{
    return (uint16_t )(800 * (7372800UL / 8 / 32 * (1 << MICA_DIVIDE_ONE_FOR_32KHZ_LOG2) / 1000000.0));
}
inline static uint16_t SoftwareAckLayerP$SoftwareAckConfig$getAckTimeout(void) {
    result = RF230ActiveMessageP$SoftwareAckConfig$getAckTimeout();
    return result;
}

inline static void SoftwareAckLayerP$RadioAlarm$wait(uint16_t arg_0x1a995d20) {
    RadioAlarmP$RadioAlarm$wait(2U, arg_0x1a995d20);
}

inline static bool SoftwareAckLayerP$RadioAlarm$isFree(void) {
    result = RadioAlarmP$RadioAlarm$isFree(2U);
    return result;
}

static __inline bool IEEE154PacketP$IEEE154Packet$getAckRequired(message_t *msg) {
    return __nesc_ntoh_leuint16((unsigned char *)&IEEE154PacketP$IEEE154Packet$getHeader(msg)->fcf) & (1 << IEEE154_FCF_ACK_REQ);
}

static inline bool IEEE154PacketP$IEEE154Packet$requiresAckWait(message_t *msg) {
    return IEEE154PacketP$IEEE154Packet$getAckRequired(msg) && IEEE154PacketP$IEEE154Packet$isDataFrame(msg) && IEEE154PacketP$IEEE154Packet$getDestAddr(msg) != 0xFFFF;
}

inline static bool RF230ActiveMessageP$IEEE154Packet$requiresAckWait(message_t *msg) {
    return RF230ActiveMessageP$IEEE154Packet$requiresAckWait(msg);
}

inline static bool RF230ActiveMessageP$SoftwareAckConfig$requiresAckWait(message_t *msg) {
    return RF230ActiveMessageP$SoftwareAckConfig$requiresAckWait(msg);
requiresAckWait(arg_0x1a948ef0);
#line 38
line 38
return result;
#line 38

#if 83 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SoftwareAckLayerP.nc"
static inline void SoftwareAckLayerP$SubSend$sendDone(error_t error)
 |
{ SoftwareAckLayerP$state = SoftwareAckLayerP$STATE_ACK_SEND
 |
 if (SoftwareAckLayerP$state == SoftwareAckLayerP$STATE_ACK_SEND)
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
# line 53 # 42 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \\lib\timer\CounterToLocalTimeC.nc"
static inline uint32_t /*LocalTimeMicroC.CounterToLocalTimeC*/
CounterToLocalTimeC$1$LocalTime$get(void)
{
    return /*LocalTimeMicroC.CounterToLocalTimeC*/
    CounterToLocalTimeC$1$Counter$get();
}
# 50 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \\lib\timer\LocalTime.nc"
inline static uint32_t RF230LayerP$LocalTime$get(void)
{
    unsigned long result;
    result = /*LocalTimeMicroC.CounterToLocalTimeC*/
    CounterToLocalTimeC$1$LocalTime$get();
    return result;
}
#line 635 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \\chips\rf230\RF230LayerP.nc"
static inline void RF230LayerP$serviceRadio(void)
{
    if (RF230LayerP$isSpiAcquired())
    {
        uint16_t time;
        uint32_t time32;
        uint8_t irq;
        uint8_t temp;
        { __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
            time = RF230LayerP$capturedTime;
            __nesc_atomic_end(__nesc_atomic); }
        RF230LayerP$radioIrrq = FALSE;
        irq = RF230LayerP$readRegister(RF230_IRQ_STATUS);
        if (irq & RF230_IRQ_PLL_LOCK)
        {
            if (RF230LayerP$cmd == RF230LayerP$CMD_TURNON ||
                RF230LayerP$cmd == RF230LayerP$CMD_CHANNEL)
            {
                for (; 0; ) ;
                RF230LayerP$state = RF230LayerP$STATE_RX_ON;
                RF230LayerP$cmd = RF230LayerP$CMD_SIGNAL_DONE;
            }
            else {
                if (RF230LayerP$cmd == RF230LayerP$CMD_TRANSMIT)
                {
                    for (; 0; ) ;
                }
                else {
                    for (; 0; ) ;
                }
            }
        }
        if (irq & RF230_IRQ_RX_START)
        {
            if (RF230LayerP$cmd == RF230LayerP$CMD_CCA)
            {
                RF230LayerP$radioCCA$done(FAIL);
                RF230LayerP$cmd = RF230LayerP$CMD_NONE;
            }
            if (RF230LayerP$cmd == RF230LayerP$CMD_NONE)
            {
                for (; 0; ) ;
                if (irq == RF230_IRQ_RX_START)
                {
                    temp = RF230LayerP$readRegister(RF230_PHY_RSSI) &
                        RF230_RSSI_MASK;
                    RF230LayerP$rssiBusy +=
                        temp - (RF230LayerP$rssiBusy >> 2); }
            }
        }
        if (irq == RF230_IRQ_RX_START)
        {
            time32 = RF230LayerP$LocalTime$get();
            time32 += (int16_t )time - RF230LayerP$RX_SFD_DELAY) - (int16_t )time32;
            314
RF230LayerP$PacketTimeStamp$
set(RF230LayerP$rxMsg, time32);
}
else {
RF230LayerP$PacketTimeStamp$
clear(RF230LayerP$rxMsg);
RF230LayerP$cmd = RF230LayerP$CMD_RECEIVE;
} else {
for (; 0; ) ;
}

if (irq & RF230_IRQ_TRX_END)
{
if (RF230LayerP$cmd == RF230LayerP$CMD_TRANSMIT)
{
for (; 0; ) ;
RF230LayerP$state = RF230LayerP$STATE_RX_ON;
RF230LayerP$cmd = RF230LayerP$CMD_NONE;
RF230LayerP$RadioSend$sendDone(SUCCESS);
for (; 0; ) ;
}
else {
#line 745
if (RF230LayerP$cmd == RF230LayerP$CMD_RECEIVE)
{
for (; 0; ) ;
if (irq == RF230_IRQ_TRX_END) {
RF230LayerP$PacketRSSI$ set(RF230LayerP$rxMsg,
RF230LayerP$readRegister(RF230_PHY_ED_LEVEL));
} else {
#line 752
RF230LayerP$PacketRSSI$ clear(RF230LayerP$rxMsg);
}
if (RF230LayerP$state == RF230LayerP$STATE_PLL_ON_2_RX_ON)
{
for (; 0; ) ;
RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_RX_ON);
RF230LayerP$state = RF230LayerP$STATE_RX_ON;
} else {
RF230LayerP$rssiClear += (RF230LayerP$readRegister(RF230_PHY_RSSI) & RF230_RSSI_MASK)
- (RF230LayerP$rssiClear >> 2);
}
RF230LayerP$cmd = RF230LayerP$CMD_DOWNLOAD;
} else {
for (; 0; ) ;
}
}
}

#line 745
if (RF230LayerP$cmd == RF230LayerP$CMD_RECEIVE)
{
for (; 0; ) ;
if (irq == RF230_IRQ_TRX_END) {
RF230LayerP$PacketRSSI$ set(RF230LayerP$rxMsg,
RF230LayerP$readRegister(RF230_PHY_ED_LEVEL));
} else {
#line 752
RF230LayerP$PacketRSSI$ clear(RF230LayerP$rxMsg);
}
if (RF230LayerP$state == RF230LayerP$STATE_PLL_ON_2_RX_ON)
{
for (; 0; ) ;
RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_RX_ON);
RF230LayerP$state = RF230LayerP$STATE_RX_ON;
} else {
RF230LayerP$rssiClear += (RF230LayerP$readRegister(RF230_PHY_RSSI) & RF230_RSSI_MASK)
- (RF230LayerP$rssiClear >> 2);
}
RF230LayerP$cmd = RF230LayerP$CMD_DOWNLOAD;
} else {
for (; 0; ) ;
}
}

static inline void RF230LayerP$Tasklet$run(void)
{
if (RF230LayerP$radioIrq) {
RF230LayerP$serviceRadio();
}
if (RF230LayerP$cmd != RF230LayerP$CMD_NONE)
{
if (RF230LayerP$cmd == RF230LayerP$CMD_DOWNLOAD) {
RF230LayerP$downloadMessage();
} else {
#line 796
if (RF230LayerP$CMD_TURNOFF <= RF230LayerP$cmd && RF230LayerP$cmd <= RF230LayerP$CMD_TURNON) {
RF230LayerP$changeState();
} else {
#line 798
if (RF230LayerP$cmd == RF230LayerP$CMD_CHANNEL) {
RF230LayerP$changeChannel();
}
}
#line 801
if (RF230LayerP$cmd == RF230LayerP$CMD_SIGNAL_DONE) {
}
RF230LayerP$cmd = RF230LayerP$CMD_NONE;
RF230LayerP$RadioState$done();
}

if (RF230LayerP$cmd == RF230LayerP$CMD_NONE && RF230LayerP$state ==
RF230LayerP$STATE_RX_ON && !RF230LayerP$radioIrq) {
RF230LayerP$RadioSend$ready();
}

if (RF230LayerP$cmd == RF230LayerP$CMD_NONE) {
RF230LayerP$SpiResource$release();
}

inline static void TaskletC$Tasklet$run(void){

RF230LayerP$Tasklet$run();
TrafficMonitorLayerP$Tasklet$run();
MessageBufferLayerP$Tasklet$run();
RadioAlarmP$Tasklet$run();
}

static inline uint8_t / *Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$Resource$isOwner(uint8_t id)
{

__nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
{
if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId ==
id && /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state ==
/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_BUSY) {
unsigned char __nesc_temp =
TRUE;
}

__nesc_atomic_end(__nesc_atomic);
return __nesc_temp;
}
else
{
unsigned char __nesc_temp =
FALSE;

__nesc_atomic_end(__nesc_atomic);
return __nesc_temp;
}
}

inline static bool Atm128SpiP$ResourceArbiter$ isOwner(uint8_t arg_0x1ae49b10){
unsigned char result;
result = /*Atm128SpiC.Arbiter.Arbiter*/ SimpleArbiterP$0$Resource$ isOwner(arg_0x1ae49b10);
return result;
}
inline static bool RF230LayerP$SpiResource$isOwner(void) {
    unsigned char result;
    result = Atm128SpiP$Resource$isOwner(0U);
    return result;
}

static inline void /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$ResourceRequested$default$immediateRequested(uint8_t id) {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceRequested$immediateRequested(/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
    if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state == /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_IDLE) {
        /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_BUSY;
        /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId = id;
        /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceConfigure$configure(/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
        __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        unsigned char __nesc_temp = SUCCESS;
        __nesc_atomic_end(__nesc_atomic);
        return __nesc_temp;
    }
    unsigned char __nesc_temp = FAIL;
    __nesc_atomic_end(__nesc_atomic);
    return __nesc_temp;
}

error_t Atm128SpiP$ResourceArbiter$immediateRequest(uint8_t arg_0x1ae49b10) {
    unsigned char result;
    result = Atm128SpiP$Resource$isOwner(0U);
    return result;
}

static inline error_t /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$Resource$immediateRequest(uint8_t id) {
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceRequested$default$immediateRequested(arg_0x1af22688);
    /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceConfigure$configure(/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state == /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_IDLE) {
        /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_BUSY;
        /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId = id;
        /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$ResourceConfigure$configure(/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
        unsigned char __nesc_temp = SUCCESS;
        __nesc_atomic_end(__nesc_atomic);
        return __nesc_temp;
    }
    unsigned char __nesc_temp = FAIL;
    __nesc_atomic_end(__nesc_atomic);
    return __nesc_temp;
}

error_t /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$Resource$immediateRequest(uint8_t arg_0x1ae49b10) {
    unsigned char result;
    result = Atm128SpiP$Resource$isOwner(0U);
    return result;
}
#line 87 result = /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$.Resource$immediateRequest(arg_0x1ae49b10);
#line 87 #line 87 return result;
#line 87 #line 87

static inline error_t Atm128SpiP$Resource$immediateRequest(uint8_t id)
#line 305 {
    error_t result = Atm128SpiP$ResourceArbiter$immediateRequest(id);
    #line 307
    if (result == SUCCESS) {
        Atm128SpiP$startSpi();
    }
    return result;
}
# 87 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\Resource.nc"
inline static error_t RF230LayerP$SpiResource$immediateRequest(void) {
    #line 87 unsigned char result;
    #line 87 #line 87
    result = Atm128SpiP$Resource$immediateRequest(0U);
    #line 87 #line 87 return result;
    #line 87
}
# 156 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\spi\HplAtm128SpiP.nc"
static inline void HplAtm128SpiP$SPI$setMasterBit(bool isMaster) {
    #line 156
    if (isMaster) {
        * (volatile uint8_t *)(0x2C + 0x20) |= 1 << 4;
    } else {
        * (volatile uint8_t *)(0x2C + 0x20) &= ~(1 << 4);
    }
}
# 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void /*HplAtm128GeneralIOC.PortB.Bit1*/HplAtm128GeneralIOPinP$9$IO$makeOutput(void) {
    #line 52
    * (volatile uint8_t *)36U |= 1 << 1;
}
# 35 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\GeneralIO.nc"
inline static void HplAtm128GeneralIOPinP$SCK$makeOutput(void) {
    #line 35
    /*HplAtm128GeneralIOC.PortB.Bit3*/HplAtm128GeneralIOPinP$MISO$makeInput(void) {
    #line 33
    * (volatile uint8_t *)36U &= ~(1 << 3);
}
# 33 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\interfaces\GeneralIO.nc"
inline static void HplAtm128GeneralIOPinP$SCK$makeOutput(void) {
    #line 33
    /*HplAtm128GeneralIOC.PortB.Bit3*/HplAtm128GeneralIOPinP$MISO$makeInput(void) {
    #line 33
    # 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static inline void HplAtm128GeneralIOC.PortB.Bit2$makeOutput(void)
{
    (volatile uint8_t *)36U |= 1 << 2;
}

inline static void HplAtm128GeneralIOC.PortB.Bit2$makeOutput();

inline static void HplAtm128SpiP$MOSI$makeOutput(void)
{
    /*HplAtm128GeneralIOC.PortB.Bit2*/
    HplAtm128GeneralIOPinP$10$IO$makeOutput();
}

inline static void HplAtm128SpiP$SPI$initMaster(void)
{
    HplAtm128SpiP$MOSI$makeOutput();
    HplAtm128SpiP$MISO$makeInput();
    HplAtm128SpiP$SCK$makeOutput();
    HplAtm128SpiP$SPI$setMasterBit(TRUE);
}

inline static void Atm128SpiP$Spi$initMaster(void){
    HplAtm128SpiP$SPI$initMaster();
}

inline static void Atm128SpiP$Spi$setMasterDoubleSpeed(bool on)
{
    if (on) {
        (volatile uint8_t *)(0x2D + 0x20) |= 1 << 0;
    } else {
        (volatile uint8_t *)(0x2D + 0x20) &= ~ (1 << 0);
    }
}

inline static void Atm128SpiP$Spi$setClockPolarity(bool highWhenIdle)
{
    if (highWhenIdle) {
        (volatile uint8_t *)(0x2C + 0x20) |= 1 << 3;
    } else {
        (volatile uint8_t *)(0x2C + 0x20) &= ~ (1 << 3);
    }
}

inline static void Atm128SpiP$Spi$setClockPhase(bool sampleOnTrailing)
{
    if (sampleOnTrailing) {
        (volatile uint8_t *)(0x2C + 0x20) |= 1 << 2;
    }
}
else {
    *(volatile uint8_t *)(0x2C + 0x20) &= ~((1 << 2);  
}

#endif
unsigned char result;
result = /*Atm128SpiC.Arbiter.Queue*/
FcfsResourceQueueC$0$FcfsQueue$enqueue(arg_0x1af2d208);
return result;
}

static inline void /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$ResourceRequested$default$requested(uint8_t id)
{
/*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$ResourceRequested$requested
(/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$resId);
{ __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    if /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$state == /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$RES_IDLE) {
/*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$state = /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$RES_GRANTING;
/*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$reqResId = id;
/*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$grantedTask$postTask();
    { unsigned char __nesc_temp =
        SUCCESS;
        __nesc_atomic_end(__nesc_atomic);
        return __nesc_temp;
    }
}
    { unsigned char __nesc_temp =
        /*Atm128SpiC.Arbiter.Arbiter*/
SimpleArbiterP$0$Queue$enqueue(id);
        __nesc_atomic_end(__nesc_atomic);
        return __nesc_temp;
    }
}
unsigned char result;
result = SimpleArbiterP$0$Resource$request(arg_0x1ae49b10);
return result;
}

result = HplAtm1281Timer1P$Timer$test();
return result;

result = HplAtm1281Timer1P$Timer$get();
result = HplAtm1281Timer1P$Timer$get();
result = HplAtm1281Timer1P$Timer$get();


```c
return /*CounterOne16C.NCounter*/Atm128CounterC$0$Timer$test();
}

# 60 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\Timer\Counter.nc"
inline static bool /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$CounterFrom$IsOverflowPending( void )
{ # line 60
  unsigned char result;
  # line 60
  result = /*CounterOne16C.NCounter*/Atm128CounterC$0$Counter$IsOverflowPending();
  # line 60
  return result;
  # line 60
}

# 194 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm1281\timer\HplAtm1281Timer1P.nc"
static inline void HplAtm1281Timer1P$CompareA$set(uint16_t t)
{ # line 194
  * (volatile uint16_t *)0x88 = t;
}

# 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Compare.nc"
inline static void /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$set(/*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$size_type arg_0x1a6924f0)
{ # line 45
  HplAtm1281Timer1P$CompareA$set(arg_0x1a6924f0);
  # line 45
}

# 140 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm1281\timer\HplAtm1281Timer1P.nc"
static inline void HplAtm1281Timer1P$CompareA$reset( void )
{ # line 140
  * (volatile uint8_t *)(0x16 + 0x20) = 1 << 1;
}

# 53 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Compare.nc"
inline static void /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$reset( void )
{ # line 53
  HplAtm1281Timer1P$CompareA$reset();
  # line 53
}

# 146 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Compare.nc"
static inline void HplAtm1281Timer1P$CompareA$start( void )
{ # line 146
  * (volatile uint8_t *)0x6F |= 1 << 1;
}

# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\IEEE154Packet.nc"
inline static bool /*IEEE154PacketP$IEEE154Packet$getAckRequired( message_t *arg_0x1a989768)
{ # line 56
  unsigned char result;
  # line 56
  result = IEEE154PacketP$IEEE154Packet$getAckRequired(arg_0x1a989768);
  # line 56
  return result;
  # line 56
}
```

323
static inline void MessageBufferLayerP$RadioSend$sendDone(error_t error)
{
    for (; 0; )
    {
        __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        #line 174
        MessageBufferLayerP$txError = error;
        #line 174
        __nesc_atomic_end(__nesc_atomic);
    }
    MessageBufferLayerP$sendTask$postTask();
}

inline static void TrafficMonitorLayerP$RadioSend$sendDone(error_t arg_0x1ac23360){
    #line 45
    MessageBufferLayerP$RadioSend$sendDone(arg_0x1ac23360);
    #line 45
}

static __inline rf230packet_metadata_t *RF230ActiveMessageP$getMeta(message_t *msg)
{
    return (rf230packet_metadata_t *)msg->metadata;
}

static inline bool RF230PacketP$PacketTransmitPower$isSet(message_t *msg)
{
    return RF230PacketP$PacketTransmitPower$getMeta(msg)->flags & RF230PACKET_TXPOWER;
}

inline static bool RF230LayerP$PacketTransmitPower$isSet(message_t *arg_0x1a890c68){
    #line 29
    unsigned char result;
    #line 29
    #line 29
    result = RF230PacketP$PacketTransmitPower$isSet(arg_0x1a890c68);
    #line 29
    return result;
    #line 29
}

inline static uint8_t RF230PacketP$PacketTransmitPower$get(message_t *msg)
{
    return RF230PacketP$PacketTransmitPower$getMeta(msg)->power;
}

inline static RF230LayerP$PacketTransmitPower$value_type RF230LayerP$PacketTransmitPower$get(message_t *arg_0x1a8a8190){
    #line 35
    unsigned char result;
    #line 35
    #line 35
    result = RF230PacketP$PacketTransmitPower$get(arg_0x1a8a8190);
    #line 35
    return result;
    #line 35
}

inline static bool RF230ActiveMessageP$RF230Config$requiresRssiCca(message_t *msg)
{
    return RF230ActiveMessageP$IEEE154Packet$isNewDataFrame(msg);
}

inline static bool RF230LayerP$RF230Config$requiresRssiCca(message_t *arg_0x1a932c88){
    #line 72
    return RF230ActiveMessageP$IEEE154Packet$isNewDataFrame(msg);
    #line 72
}
unsigned char result;
#line 72
result = RF230ActiveMessageP$RF230Config$
requiresRssiCca(arg_0x1a932c88);
#line 72
return result;
#line 72

result = RF230ActiveMessageP$RF230Config$
requiresRssiCca(arg_0x1a932c88);
#line 72
return result;
#line 72

static inline bool RF230PacketP$PacketTimeSyncOffset$
isSet(message_t *msg)
{
    return RF230PacketP$PacketTimeSyncOffset$
    isSet(arg_0x1a890c68);
}

inline static bool RF230LayerP$PacketTimeSyncOffset$
isSet(message_t *arg_0x1a890c68)
{
    unsigned char result;
    result = RF230PacketP$PacketTimeSyncOffset$
    isSet(arg_0x1a890c68);
    return result;
}

static inline uint8_t RF230PacketP$PacketTimeSyncOffset$
get(message_t *msg)
{
    return RF230PacketP$IEEE154Packet$getLength(msg) -
    RF230PacketP$PACKET_LENGTH_INCREASE -
    sizeof(timesync_absolute_t);
}

inline static RF230LayerP$PacketTimeSyncOffset$
value_type RF230LayerP$PacketTimeSyncOffset$
get(message_t *arg_0x1a8a8190)
{
    unsigned char result;
    result = RF230PacketP$PacketTimeSyncOffset$get(arg_0x1a8a8190);
    return result;
}

inline static uint16_t RF230LayerP$RadioAlarm$getNow(void)
{
    unsigned short result;
    result = RadioAlarmP$RadioAlarm$getNow(3U);
    return result;
}

inline static uint8_t RF230ActiveMessageP$IEEE154Packet$getLength
(message_t *arg_0x1a9759d0)
{
    unsigned char result;
    result = IEEE154PacketP$IEEE154Packet$getLength(arg_0x1a9759d0);
    return result;
}

unsigned char result;

static inline uint8_t RF230ActiveMessageP$IEEE154Packet$getLength(msg)
{
    return RF230ActiveMessageP$RF230Config$getLength(msg);
}

inline static uint8_t RF230LayerP$RF230Config$getLength(message_t *arg_0x1a938d68){
    unsigned char result;
    result = RF230ActiveMessageP$RF230Config$getLength(arg_0x1a938d68);
    return result;
}

static __inline uint32_t __nesc_hton_uint32(void *target, uint32_t value)
{
    const uint8_t *base = target;
}

static __inline int32_t __nesc_hton_int32(void *target, int32_t value)
{
    return __nesc_hton_uint32(target, value);
}

static __inline uint32_t __nesc_ntoh_uint32(const void *source)
{
    return __nesc_ntoh_int32(source);
}

static __inline int32_t __nesc_ntoh_int32(const void *source)
{
    return __nesc_ntoh_uint32(source);
}


inline static error_t RandomCollisionLayerP$calcNextRandom$postTask(void) {
    unsigned char result;
    result = SchedulerBasicP$TaskBasic$postTask(RandomCollisionLayerP$calcNextRandom);
    return result;
}

static inline uint16_t RF230ActiveMessageP$RandomCollisionConfig$getMinimumBackoff(void) {
    return (uint16_t )(320 *
        (7372800UL / 8 / 32 * (1 << MICA_DIVIDE_ONE_FOR_32KHZ_LOG2)
        / 1000000.0));
}

inline static uint16_t RandomCollisionLayerP$Config$getMinimumBackoff(void) {
    unsigned short result;
    result = RF230ActiveMessageP$RandomCollisionConfig$getMinimumBackoff();
    return result;
}

static inline void Atm128SpiP$SpiPacket$sendDone(void *txbuffer, void *rxbuffer, uint16_t _length, error_t _success) {
    Atm128SpiP$SpiPacket$default$sendDone(txbuffer, rxbuffer, _length, _success);
}

void Atm128SpiP$zeroTask$runTask(void) {
    uint16_t myLen;
    uint8_t *tx;
    uint8_t *rx;
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    myLen = Atm128SpiP$len;
    tx = Atm128SpiP$txBuffer;
    rx = Atm128SpiP$rxBuffer;
    Atm128SpiP$txBuffer = (void *)0;
    Atm128SpiP$rxBuffer = (void *)0;
    Atm128SpiP$len = 0;
    Atm128SpiP$pos = 0;
    Atm128SpiP$spiPacket$sendDone(tx, rx, myLen, SUCCESS);
}
static inline uint16_t RandomMlcgC$Random$rand16(void)
{
    return (uint16_t )RandomMlcgC$Random$rand32();
}

inline static uint16_t RandomCollisionLayerP$Random$rand16(void)
{
    unsigned short result;
    result = RandomMlcgC$Random$rand16();
    return result;
}

static inline void RandomCollisionLayerP$calcNextRandom$runTask(void)
{
    uint16_t a = RandomCollisionLayerP$Random$rand16();
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    RandomCollisionLayerP$nextRandom = a;
    __nesc_atomic_end(__nesc_atomic); }

static inline void / *HilTimerMilliC.VirtualizeTimerC*/VirtualizeTimerC$0$Timer$stop(uint8_t num)
{
    /*HilTimerMilliC.Virtualize TimerC*/ VirtualizeTimerC$0$m_timers[num].isrunning = FALSE;
}

inline static void TrafficMonitorLayerP$Timer$stop(void)
{
    /*HilTimerMilliC.VirtualizeTimerC*/ VirtualizeTimerC$0$Timer$stop(1U);
}

static inline uint16_t RF230ActiveMessageP$TrafficMonitorConfig$getUpdatePeriod(void)
{
    return RF230ActiveMessageP$TrafficMonitorConfig$getUpdatePeriod();
}

static inline uint16_t RF230ActiveMessageP$TrafficMonitorConfig$getUpdatePeriod(void)
{
    return RF230ActiveMessageP$TrafficMonitorConfig$getUpdatePeriod();
}
Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Counter$get();

#line 98 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\Timer\Alarms.nc"
inline static /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerCS$0$Alarm$size_type /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerCS$0$Alarm$getNow(void){
#line 98 unsigned long result;
#line 98 result = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Alarm$getNow();
#line 98 return result;
#line 98
#line 98
# 85 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\Timer\AlarmToTimerC.nc"
static inline uint32_t /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Timer$getNow(void){
#line 86 return /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Timer$getNow();
#line 125 unsigned long result;
#line 125 result = /*HilTimerMilliC.AlarmToTimerC*/
AlarmToTimerC$0$Timer$getNow();
#line 125 return result;
#line 125
# 143 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\Timer\VirtualizeTimerC.nc"
static inline void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerCS$0$Timer$startPeriodic(uint8_t num, uint32_t dt){
#line 143 /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$TimerFrom$getNow(), dt, FALSE);
}
# 53 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\Timer\TaskBasic.nc"
inline static error_t /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer$postTask(void){
# 229 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\TaskBasic\TaskBasic.nc"
inline void TrafficMonitorLayerP$StartStopTimer$runTask(void)
if (TrafficMonitorLayerP$radioCmd ==
TrafficMonitorLayerP$RADIO_CMD_TURNON) {
TrafficMonitorLayerP$StartStopTimer$StartPeriodic(TrafficMonitorLayerP$TrafficMonitorConfig$getUpdatePeriod());
} else {
if (TrafficMonitorLayerP$radioCmd ==
TrafficMonitorLayerP$RADIO_CMD_TURNOFF) {
TrafficMonitorLayerP$Stop();
}
unsigned char result;

result = SchedulerBasicP$TaskBasic$postTask
          /*HilTimerMilliC.VirtualizeTimerC*/
          VirtualizeTimerC$0$updateFromTimer);

return result;

// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//chips\atm1281\timer\HplAtm1281Timer2AsyncP.nc
static inline uint8_t HplAtm1281Timer2AsyncP$TimerCtrl$getInterruptFlag(void)
{
    return * (volatile uint8_t *)(0x17 + 0x20);
}

// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//chips\atm1281\timer\HplAtm1281Timer2AsyncP.nc
inline static
uint8_t /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$TimerCtrl$getInterruptFlag(void)
{
    unsigned char result;
    result = HplAtm1281Timer2AsyncP$TimerCtrl$getInterruptFlag();
    return result;
}

// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//interfaces\Packet.nc
inline static void *MessageBufferLayerP$Packet$getPayload(message_t *msg, uint8_t arg_0x1a8624e0)
{
    void *result;
    result = RF230PacketP$Packet$payloadLength(arg_0x1a865ea8);
    return result;
}

// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//interfaces\Packet.nc
inline static void *MessageBufferLayerP$Packet$getPayload(message_t *msg, uint8_t len)
{
    if (len > 28) {
        return (void *)0;
    }
    return msg->data;
}

// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//IEEE154NetworkLayerP.nc
static inline message_t *IEEE154NetworkLayerP$NonTinyosReceive$default$receive
          (uint8_t network, message_t *msg, void *payload, uint8_t len)
{
    return msg;
}
inline static message_t *IEEE154NetworkLayerP$NonTinyosReceive$ receive(uint8_t arg_0x1ac03a10, message_t *arg_0x1a897260, void *arg_0x1a897400, uint8_t arg_0x1a897588)
{
    nx_struct message_t *result;
    result = IEEE154NetworkLayerP$NonTinyosReceive$default$ receive(arg_0x1ac03a10, arg_0x1a897260, arg_0x1a897400, arg_0x1a897588);
    return result;
}

static inline message_t *ActiveMessageLayerC$Snoop$default$ receive(am_id_t id, message_t *msg, void *payload, uint8_t len)
{
    return msg;
}

inline static message_t *ActiveMessageLayerC$Snoop$ receive(am_id_t arg_0x1abe93d0, message_t *arg_0x1a897260, void *arg_0x1a897400, uint8_t arg_0x1a897588)
{
    nx_struct message_t *result;
    result = ActiveMessageLayerC$Snoop$default$ receive(arg_0x1abe93d0, arg_0x1a897260, arg_0x1a897400, arg_0x1a897588);
    return result;
}

static __inline uint16_t __nesc_ntoh_uint16(const void *source)
{
    const uint8_t *base = source;
    return ((uint16_t )base[0] << 8) | base[1];
}

static __inline uint16_t __nesc_hton_uint16(void *target, uint16_t value)
{
    uint8_t *base = target;
    base[1] = value;
    base[0] = value >> 8;
    return value;
}

static __inline int16_t __nesc_hton_int16(void *target, int16_t value)
{
    __nesc_hton_uint16(target, value);
    return value;
}

static __inline uint8_t __nesc_hton_uint8(void *target, uint8_t value)
{
    uint8_t *base = target;
    base[0] = value;
    return value;
}

static __inline int8_t __nesc_hton_int8(void *target, int8_t value)
{
    __nesc_hton_uint8(target, value);
    return value;
}
#line 257
    return value;
}

static __inline uint8_t __nesc_ntoh_uint8(const void *source)
#line 235
{
    const uint8_t *base = source;
    return base[0];
}

static __inline int8_t __nesc_ntoh_int8(const void *source)
#line 235
{
    return __nesc_ntoh_uint8(source);
}

inline static error_t ForwardToBaseC$AMSend$send
(arg_0x1a885e28, arg_0x1a884010, arg_0x1a884198)
{
    unsigned char result;
    result = /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
        AMQueueEntryP$0$AMSend$send
    (arg_0x1a885e28, arg_0x1a884010, arg_0x1a884198);
    return result;
}

# 69 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.1\tos \interfaces\AMSend.nc"
inline static void LedsP$Leds$led2Off(void)
{
    return;
}

# 83 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.1\tos \interfaces\Leds.nc"
inline static void LedsP$Leds$led2On(void)
{
    return;
}

# 47 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.1\tos \chips\atm128\pins\HplAtm128GeneralIOPinP.nc"
static __inline void HplAtm128GeneralIOPinP$0$IO$clr(void)
{
    * (volatile uint8_t *)34U &= ~1 << 0;
}

# 30 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.1\tos \interfaces\GeneralIO.nc"
inline static void LedsP$Leds$led2$clr(void)
{
    /HplAtm128GeneralIOPortA.Bit0*/
        HplAtm128GeneralIOPortA.Bit0;

    /HplAtm128GeneralIOPortA.Bit0*/
        HplAtm128GeneralIOPortA.Bit0;

    /HplAtm128GeneralIOPortA.Bit0*/
        HplAtm128GeneralIOPortA.Bit0;

    /HplAtm128GeneralIOPortA.Bit0*/
        HplAtm128GeneralIOPortA.Bit0;

    /HplAtm128GeneralIOPortA.Bit0*/
        HplAtm128GeneralIOPortA.Bit0;
static inline void LedsP$Leds$led1Off(void)
{
    LedsP$Led1$set();
}

inline static void ForwardToBaseC$Leds$led1Off(void)
{
    LedsP$Leds$led1Off();
}

static __inline void /*HplAtm128GeneralIOC.PortA.Bit1*/
HplAtm128GeneralIOPinP$1$IO$clr(void)
{
    * (volatile uint8_t *)34U &= ¬(1 << 1);
}

inline static void LedsP$Led0$clr(void)
{
    /*HplAtm128GeneralIOC.PortA.Bit2*/
HplAtm128GeneralIOPinP$2$IO$clr(void);
}

static inline void LedsP$Leds$led0Off(void)
{
    LedsP$Led0$set();
}

inline static void ForwardToBaseC$Leds$led0Off(void)
{
    LedsP$Leds$led0Off();
}

static __inline void /*HplAtm128GeneralIOC.PortA.Bit2*/
HplAtm128GeneralIOPinP$2$IO$clr(void)
{
    * (volatile uint8_t *)34U &= ¬(1 << 2);
}
static inline void LedsP$Leds$led0On(void)
{
    LedsP$Led0$clr();
}

inline static void ForwardToBaseC$Leds$led0On(void)
{
    LedsP$Leds$led0On();
}

static inline void ForwardToBaseC$Leds$led0Off(void)
{
    LedsP$Leds$led0Off();
}

inline static void ForwardToBaseC$Leds$led1On(void)
{
    LedsP$Leds$led1On();
}

inline static void ForwardToBaseC$Leds$led1Off(void)
{
    LedsP$Leds$led1Off();
}

inline static void ForwardToBaseC$Leds$led2On(void)
{
    LedsP$Leds$led2On();
}

inline static void ForwardToBaseC$Leds$led2Off(void)
{
    LedsP$Leds$led2Off();
}

static inline uint8_t RF230PacketP$PacketRSSI$get(message_t *msg)
{
    return RF230PacketP$getMeta(msg)->power;
}

static inline bool RF230PacketP$PacketRSSI$isSet(message_t *msg)
{
    return RF230PacketP$getMeta(msg)->flags & RF230PACKET_RSSI;
}

static inline uint8_t RF230PacketP$PacketRSSI$getValue(message_t *msg)
{
    return RF230PacketP$PacketRSSI$get(msg);
}

static inline char RF230PacketP$PacketRSSI$getValueType(message_t *msg)
{
    return (char)RF230PacketP$PacketRSSI$getValue(msg);
}
static inline uint16_t ForwardToBaseC$getRssi(message_t *msg) {
    if (ForwardToBaseC$PacketRSSI$isSet(msg)) {
        return (uint16_t)ForwardToBaseC$PacketRSSI$get(msg);
    } else {
        return 0xFFFF;
    }
}

static inline message_t *ForwardToBaseC$Receive$receive(message_t *msg, void *payload, uint8_t len) {
    RssiMsg *btrpkt = (RssiMsg *)payload;
    if (__nesc_ntoh_int8((unsigned char *)&btrpkt->forwarded)) {
        return msg;
    } else {
        __nesc_hton_uint16((unsigned char *)&btrpkt->forwarder_id, TOS_NODE_ID);
        __nesc_hton_int8((unsigned char *)&btrpkt->forwarded, TRUE);
        __nesc_hton_int16((unsigned char *)&btrpkt->rssi, ForwardToBaseC$getRssi(msg));
        ForwardToBaseC$setLeds(__nesc_ntoh_uint16((unsigned char *)&btrpkt->counter));
        if (ForwardToBaseC$AMSend$send(AM_BROADCAST_ADDR, msg, sizeof(RssiMsg)) == SUCCESS) {
            ForwardToBaseC$busy = TRUE;
        }
        return msg;
    }
}

static inline message_t *ActiveMessageLayerC$Receive$default$receive(am_id_t id, message_t *msg, void *payload, uint8_t len) {
    return msg;
}

inline static message_t *ActiveMessageLayerC$Receive$receive(am_id_t arg_0x1abeada8, message_t *arg_0x1a897260, void *arg_0x1a897400, uint8_t arg_0x1a897588) {
    nx_struct message_t *result;
    switch (arg_0x1abeada8) {
    case 6:
        result = ForwardToBaseC$Receive$receive(arg_0x1a897260, arg_0x1a897400, arg_0x1a897588);
        break;
    default:
        result = ActiveMessageLayerC$Receive$default$receive(arg_0x1abeada8, arg_0x1a897260, arg_0x1a897400, arg_0x1a897588);
        break;
    }
    return result;
}

static __inline bool IEEE154PacketP$AMPacket$isForMe(message_t *msg) {
    am_addr_t addr = IEEE154PacketP$AMPacket$destination(msg);
    return false; // Fill in the logic to check if the packet is for this mote.
return addr == IEEE154PacketP$AMPacket$address() || addr == AM_BROADCAST_ADDR;
}

inline static bool ActiveMessageLayerC$AMPacket$isForMe(message_t *arg_0x1a873ca0){
  unsigned char result;
  result = IEEE154PacketP$AMPacket$isForMe(arg_0x1a873ca0);
  return result;
}

inline static am_id_t ActiveMessageLayerC$AMPacket$type(message_t *arg_0x1a872610){
  unsigned char result;
  result = IEEE154PacketP$AMPacket$type(arg_0x1a872610);
  return result;
}

static inline message_t *ActiveMessageLayerC$SubReceive$receive(message_t *msg, void *payload, uint8_t len){
  am_id_t type = ActiveMessageLayerC$AMPacket$type(msg);
  msg = ActiveMessageLayerC$AMPacket$isForMe(msg) ?
      ActiveMessageLayerC$Receive$receive(type, msg, payload, len) :
      ActiveMessageLayerC$Snoop$receive(type, msg, payload, len);
  return msg;
}

inline static message_t *IEEE154NetworkLayerP$Receive$receive(message_t *arg_0x1a897260, void *arg_0x1a897400, uint8_t arg_0x1a897588){
  result = ActiveMessageLayerC$SubReceive$receive(arg_0x1a897260, arg_0x1a897400, arg_0x1a897588);
  return result;
}

static __inline uint8_t IEEE154PacketP$IEEE154Packet$get6LowPan(message_t *msg){
  return __nesc_ntoh_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$getHeader(msg)->network);
}

inline static uint8_t IEEE154NetworkLayerP$IEEE154Packet$get6LowPan(message_t *arg_0x1a982010){
  unsigned char result;
  result = IEEE154PacketP$IEEE154Packet$get6LowPan(arg_0x1a982010);
  return result;
}

static _inline uint8_t IEEE154PacketP$IEEE154Packet$getLowPan(message_t *msg){
  return __nesc_ntoh_leuint8((unsigned char *)&IEEE154PacketP$IEEE154Packet$getHeader(msg)->network);
}

inline static uint8_t IEEE154NetworkLayerP$IEEE154Packet$getLowPan(message_t *arg_0x1a982010){
  unsigned char result;
  result = IEEE154PacketP$IEEE154Packet$getLowPan(arg_0x1a982010);
  return result;
}
static inline message_t *IEEE154NetworkLayerP$SubReceive$
receive(message_t *msg, void *payload, uint8_t len)
{
    uint8_t network = IEEE154NetworkLayerP$IEEE154Packet$
    get6LowPan(msg);
    #line 77
    if (network == 0x3f) {
        return IEEE154NetworkLayerP$Receive$
        receive(msg, payload, len);
    } else {
        #line 80
        return IEEE154NetworkLayerP$NonTinyosReceive$
        receive(network, msg, payload, len);
    }
}

inline static message_t *MessageBufferLayerP$Receive$
receive(message_t *arg_0x1a897260, void *arg_0x1a897400, uint8_t arg_0x1a897588){
    #line 67nx_struct message_t *result;
    #line 67
    #line 67
    result = IEEE154NetworkLayerP$SubReceive$
    receive(arg_0x1a897260, arg_0x1a897400, arg_0x1a897588);
    #line 67
    #line 67
    return result;
    #line 67
    #line 67
    # 267 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
    \chips\rf230\MessageBufferLayerP.nc"
static inline void MessageBufferLayerP$deliverTask$runTask(void)
{
    for (; ; )
    {
        message_t *msg;
        __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        if (MessageBufferLayerP$receiveQueueSize == 0) {
            __nesc_atomic_end(__nesc_atomic);
            __nesc_atomic_end(__nesc_atomic);
            return;
        }
        msg = MessageBufferLayerP$receiveQueue
        [MessageBufferLayerP$receiveQueueHead];
        __nesc_atomic_end(__nesc_atomic);
        msg = MessageBufferLayerP$receiveReceive
        (msg, MessageBufferLayerP$Packet$payloadLength());
        __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        MessageBufferLayerP$receiveQueue
        [MessageBufferLayerP$receiveQueueHead] = msg;
        if (++MessageBufferLayerP$receiveQueueHead >=
            MessageBufferLayerP$RECEIVE_QUEUE_SIZE) {
            MessageBufferLayerP$receiveQueueHead = 0;
        } -MessageBufferLayerP$receiveQueueSize;
        __nesc_atomic_end(__nesc_atomic);
    }
}

#pragma warning(disable:4189)

static void /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
AMQueueEntryP$0$AMPacket$
setDestination(message_t *arg_0x1a874a90, am_addr_t arg_0x1a874c20){
    #line 92
    IEEE154PacketP$AMPacket$setDestination(arg_0x1a874a90, arg_0x1a874c20);
    #line 92
    #line 92
    #line 151
    inline static void /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/
    AMQueueEntryP$0$AMPacket$setTyp
e(message_t *arg_0x1a872ba8, am_id_t arg_0x1a872d30);
#line 151
IEEE154PacketP$AMPacket$setType(arg_0x1a872ba8, arg_0x1a872d30);
#line 151
#line 151
#line 83 C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Packet.nc
inline static void /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$Packet$setPayloadLength
(message_t *arg_0x1a864548, uint8_t *arg_0x1a8646d0){
#line 83
RF230PacketP$Packet$
setPayloadLength(arg_0x1a864548, arg_0x1a8646d0);
#line 83
#line 83
# 82 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\system\AMQueueImplP.nc"
static inline error_t /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$Send$send(uint8_t clientId, message_t *msg,
uint8_t len)
#line 83
if (clientId >= 1) {
    return FAIL;
}
if (%/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$queue[clientId].msg != (void*)0) {
    return EBUSY;
} else {
    AMQueueImplP$0$Send$send(0U, arg_0x1abe5ef8, arg_0x1abe4090);
}
if (%/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$current >= 1) {
    error_t err;
    am_id_t amId = %/*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$AMPacket$type(msg);
    am_addr_t dest = %/*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$AMPacket$destination(msg);
    err = %/*AMQueueP.AMQueueImplP*/
    AMQueueImplP$0$AMSend$send(amId, dest, msg, len);
    if (err != SUCCESS) {
        /*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$current = clientId;
        %/*AMQueueP.AMQueueImplP*/
        AMQueueImplP$0$queue[clientId].msg = (void*)0;
        return err;
    }
    else{
        return SUCCESS;
    }
}
static inline void /*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$SendDone(am_id_t id, message_t *msg, error_t err)
#line 181
if (%/*AMQueueP.AMQueueImplP*/
AMQueueImplP$0$current >= 1) { return;

338
if (/*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$queue[/*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$current].msg == msg) {
  /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$sendDone(/*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$current, msg, err);
} else {
  ...
}

#line 99 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\AMSend.nc"
inline static void ActiveMessageLayerC$AMSend$sendDone(am_id_t arg_0x1abea478, message_t *arg_0x1a8821c0, error_t arg_0x1a882348){
  /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$AMSend$sendDone(arg_0x1abea478, arg_0x1a8821c0, arg_0x1a882348);
}

#line 99 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\ActiveMessageLayerC.nc"
static __inline void ActiveMessageLayerC$SubSend$sendDone(message_t *msg, error_t error){
  ActiveMessageLayerC$AMSend$sendDone(ActiveMessageLayerC$AMPacket$type(msg), msg, error);
}

#line 89 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\Send.nc"
inline static void IEEE154NetworkLayerP$Send$sendDone(message_t *arg_0x1abe2010, error_t arg_0x1abe2198){
  IEEE154NetworkLayerP$SubSend$sendDone(arg_0x1abe2010, arg_0x1abe2198);
}

#line 89 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\IEEE154NetworkLayerP.nc"
static inline void IEEE154NetworkLayerP$SubSend$sendDone(message_t *msg, error_t error){
  IEEE154NetworkLayerP$Send$sendDone(msg, error);
}

#line 99 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\Send.nc"
inline static void UniqueLayerP$Send$sendDone(message_t *arg_0x1abe2010, error_t arg_0x1abe2198){
  UniqueLayerP$SubSend$sendDone(arg_0x1abe2010, arg_0x1abe2198);
}

#line 89 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\UniqueLayerP.nc"
static inline void UniqueLayerP$SubSend$sendDone(message_t *msg, error_t error){
  UniqueLayerP$Send$sendDone(msg, error);
}

#line 89 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\Send.nc"
inline static void MessageBufferLayerP$Send$sendDone(message_t *arg_0x1abe2010, error_t arg_0x1abe2198){
  MessageBufferLayerP$SubSend$sendDone(arg_0x1abe2010, arg_0x1abe2198);
}

#line 88 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\Tasklet.nc"
inline static void MessageBufferLayerP$Tasklet$resume(void){
  TaskletC$Tasklet$resume();
}

AMQueueP::AMQueueImplP::AMQueueImplP(AMQueueP::AMQueueImplP* queue, AMQueueP::AMQueueImplP* current)
{
  msg = 0;
  err = 0;
  error_t arg_0x1abe2198;
  message_t arg_0x1a8821c0;
  am_id_t arg_0x1abea478;
  /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$queue[/*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$current].msg = msg;
  /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$sendDone(/*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$current, msg, err);
}

RandomCollisionConfig$RandomCollisionConfig$RandomCollisionConfig$getInitialBackoff(message_t *msg)
return (uint16_t)(9920 * (7372800UL / 8 / 32 * (1 << MICA_DIVIDE_ONE_FOR_32KHZ_LOG2) / 1000000.0));

// 29 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RandomCollisionConfig.nc"
inline static uint16_t RandomCollisionLayerP$Config$getInitialBackoff(message_t* arg_0x1a955c98)
{ #line 29
  unsigned short result;
  #line 29
  result = RF230ActiveMessageP$RandomCollisionConfig$ getInitialBackoff(arg_0x1a955c98);
  #line 29
  return result;
  #line 29
}

# 87 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RandomCollisionLayerP.nc"
static inline error_t RandomCollisionLayerP$RadioSend$ send(message_t* msg)
{ if (RandomCollisionLayerP$state != RandomCollisionLayerP$STATE_READY || !RandomCollisionLayerP$RadioAlarm$isFree()) {
  return EBUSY;
}
  RandomCollisionLayerP$txMsg = msg;
  RandomCollisionLayerP$state = RandomCollisionLayerP$STATE_TX_PENDING_FIRST;
  RandomCollisionLayerP$RadioAlarm$wait(RandomCollisionLayerP$getBackoff(RandomCollisionLayerP$Config$getInitialBackoff(msg)));
  return SUCCESS;
}

// 37 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\TrafficMonitorLayerP.nc"
inline static error_t TrafficMonitorLayerP$SubSend$ send(message_t* arg_0x1ac24c88)
{ #line 37
  unsigned char result;
  #line 37
  result = TrafficMonitorLayerP$RadioSend$ send(arg_0x1ac24c88);
  #line 37
  return result;
  #line 37
}

// 37 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\RadioSend.nc"
inline static error_t MessageBufferLayerP$RadioSend$ send(message_t* arg_0x1ac24c88)
{ #line 37
  unsigned char result;
  #line 37
  result = TrafficMonitorLayerP$RadioSend$ send(arg_0x1ac24c88);
  #line 37
  return result;
  #line 37
}

// 83 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\rf230\TaskletC.nc"
static __inline void TaskletC$Tasklet$suspend(void)
{ __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
  TaskletC$state = TaskletC$STATE_SUSPENDED;
  __nesc_atomic_end(__nesc_atomic); }
# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \\chips\rf230\Tasklet.nc"
inline static void MessageBufferLayerP$Tasklet$suspend(void){
    TaskletC$Tasklet$suspend();
    #line 61
}
#line 61
# 139 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \\chips\rf230\MessageBufferLayerP.nc"
static inline void MessageBufferLayerP$sendTask$runTask(void)
{
    error_t error;
    for (; 0; ) ;
    { __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        error = MessageBufferLayerP$txError;
        __nesc_atomic_end(__nesc_atomic); }
    if ((MessageBufferLayerP$state == MessageBufferLayerP$STATE_TX_SEND &&
        error == SUCCESS) || ++MessageBufferLayerP$retries >
        MessageBufferLayerP$MAX_RETRIES) {
        MessageBufferLayerP$state =
        MessageBufferLayerP$STATE_TX_DONE;
    }
    else {
        MessageBufferLayerP$Tasklet$suspend();
        error = MessageBufferLayerP$RadioSend$
            send(MessageBufferLayerP$txMsg);
        if (error == SUCCESS) {
            MessageBufferLayerP$state =
            MessageBufferLayerP$STATE_TX_SEND;
        }
        else {
            if (MessageBufferLayerP$retries ==
                MessageBufferLayerP$MAX_RETRIES) {
                MessageBufferLayerP$state =
                MessageBufferLayerP$STATE_TX_DONE;
            }
            else {
                MessageBufferLayerP$state =
                MessageBufferLayerP$STATE_TX_PENDING;
            }
        }
        MessageBufferLayerP$Tasklet$resume();
    }
    if (MessageBufferLayerP$state ==
        MessageBufferLayerP$STATE_TX_DONE) {
        MessageBufferLayerP$state =
        MessageBufferLayerP$STATE_READY;
        MessageBufferLayerP$Send$sendDone
            (MessageBufferLayerP$txMsg, error);
    }
}
# 72 "C:\Crossbow\cygwin\home\ordonez\rssi_ordonez_v2 \\ForwardingMote\ForwardToBaseC.nc"
static inline void ForwardToBaseC$AMControl$stopDone(error_t err)
#line 72
{ }
# 117 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \\interfaces\SplitControl.nc"
inline static void MessageBufferLayerP$SplitControl$stopDone(error_t arg_0x1a892250){
    ForwardToBaseC$AMControl$stopDone(arg_0x1a892250);
#line 117
    ForwardToBaseC$AMControl$stopDone(arg_0x1a892250);
#line 117
    }
inline static error_t ForwardToBaseC$AMControl$start(void){
    unsigned char result;
    result = MessageBufferLayerP$SplitControl$start();
    return result;
}
inline static error_t ForwardToBaseC$toggle$postTask(void) {
    return result;
}

inline static void ForwardToBaseC$Timer0$startPeriodic(uint32_t arg_0x1a5d0490) {
    VirtualizeTimerC$0$Timer$startPeriodic(0U, arg_0x1a5d0490);
}

static inline void ForwardToBaseC$AMControl$startDone(error_t err) {
    if (err == SUCCESS) {
        ForwardToBaseC$Timer0$startPeriodic(FWD_INTERVAL_MS);
        ForwardToBaseC$toggle$postTask();
    } else {
        ForwardToBaseC$AMControl$start();
    }
}

inline static void MessageBufferLayerP$SplitControl$startDone(error_t arg_0x1a894678) {
    MessageBufferLayerP$stateDoneTask$runTask(void) {
        uint8_t s;
        s = MessageBufferLayerP$state;
        if (s == MessageBufferLayerP$STATE_TURN_ON || s == MessageBufferLayerP$STATE_TURN_OFF) {
            MessageBufferLayerP$state = MessageBufferLayerP$STATE_READY;
        } else {
            MessageBufferLayerP$SplitControl$stopDone(SUCCESS);
        }
    } else {
        MessageBufferLayerP$SplitControl$stopDone(SUCCESS);
    }
}

static inline error_t RF230LayerP$RadioState$turnOn(void) {
    if (RF230LayerP$cmd != RF230LayerP$CMD_NONE && RF230LayerP$RadioAlarm$isFree()) {
        return EBUSY;
    } else {
        RF230LayerP$RadioState$turnOn(0);
    }
}

if (RF230LayerP$state == RF230LayerP$STATE_RX_ON) {
    return EALREADY;
} else {
    RF230LayerP$cmd = RF230LayerP$CMD_TURNON;
}
RF230LayerP$Tasklet$schedule();
return SUCCESS;
}

inline static error_t TrafficMonitorLayerP$RadioState$turnOn(void){
    unsigned char result;
    result = RF230LayerP$RadioState$turnOn();
    return result;
}

inline static void / *HplAtm128GeneralIOC.PortA.Bit2*/
HplAtm128GeneralIOPinP$2$IO$toggle(void)
{
    volatile uint8_t *34U ^= 1 << 2;
}

inline static void LedsP$Led0$toggle(void){
    HplAtm128GeneralIOPinP$2$IO$toggle();
}

static inline void LedsP$Leds$led0Toggle(void)
{
    LedsP$Leds$led0Toggle();
}

static inline void ForwardToBaseC$Leds$led0Toggle(void)
{
    LedsP$Leds$led0Toggle();
}
VirtualizeTimerC$0$updateFromTimer$runTask(void)
{
    uint32_t now = /*HilTimerMilliC.VirtualizeTimerC*/
    VirtualizeTimerC$0$TimerFrom$getNow();
    int32_t min_remaining = (1UL << 31) - 1;
    bool min_remaining_set = FALSE;
    uint8_t num;
    /*HilTimerMilliC.VirtualizeTimerC*/
    VirtualizeTimerC$0$TimerFrom$stop();
    for (num = 0; num < /*HilTimerMilliC.VirtualizeTimerC*/
        VirtualizeTimerC$0$NUM_TIMERS; num++)
    {
        /*HilTimerMilliC.VirtualizeTimerC*/
        VirtualizeTimerC$0$Timer_t *timer = &/*HilTimerMilliC.VirtualizeTimerC*/
        VirtualizeTimerC$0$m_timers[num];
        if (timer->isrunning)
        {
            uint32_t elapsed = now - timer->t0;
            int32_t remaining = timer->dt - elapsed;
            if (remaining < min_remaining)
            {
                min_remaining = remaining;
                min_remaining_set = TRUE;
            }
        }
    }
    if (min_remaining_set)
    {
        if (min_remaining <= 0) {
            /*HilTimerMilliC.VirtualizeTimerC*/
            VirtualizeTimerC$0$fireTimers(now);
        }
        else {
            RF230PacketP$Packet$getPayload(arg_0x1a862358, arg_0x1a8624e0);
        }
    }
}

# 115 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Packet.nc"
inline static void *ForwardToBaseC$Packet$getPayload(message_t *arg_0x1a862358, uint8_t arg_0x1a8624e0){
    void *result;
    result = RF230PacketP$Packet$getPayload(arg_0x1a862358, arg_0x1a8624e0);
    return result;
}
# 76 "C:\Crossbow\cygwin\home\ ordinex\rssi_ordinex_v2
\ForwardingMote\ForwardToBaseC.nc"
static inline void ForwardToBaseC$Timer0$fired(void)
{
    if (!ForwardToBaseC$busy) {
        RssiMsg *btrpkt = (RssiMsg *)ForwardToBaseC$Packet$getPayload(&ForwardToBaseC$pkt, sizeof(RssiMsg ));
        if (btrpkt == (void *)0) {
            return;
        }
        __nesc_hton_uint16((unsigned char *)&
        btrpkt->counter, 0);
        __nesc_hton_uint16((unsigned char *)&
        btrpkt->sender_id, TOS_NODE_ID);
        if (ForwardToBaseC$SAMSend$send(AM_BROADCAST_ADDR, &ForwardToBaseC$pkt,
        sizeof(RssiMsg )) == SUCCESS) {
            ForwardToBaseC$busy = TRUE;
        }
    }
}

# 68 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\Tasklet.nc"
inline static void TrafficMonitorLayerP$Tasklet$resume(void){
    TrafficMonitorLayerP$Tasklet$resume(void);
TaskletC$Tasklet$resume();
#
# 291 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230ActiveMessageP.nc"
static inline void RF230ActiveMessageP$SlottedCollisionConfig$default$timerTick(void)
#
# 60 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\SlottedCollisionConfig.nc"
inline static void RF230ActiveMessageP$SlottedCollisionConfig$timerTick(void)
#
# 213 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\TrafficMonitorConfig.nc"
static inline void RF230ActiveMessageP$TrafficMonitorConfig$timerTick(void)
#
# 132 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\TrafficMonitorLayerP.nc"
static inline void TrafficMonitorLayerP$Tasklet$suspend(void)
#
# 147 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\TrafficMonitorLayerP.nc"
static inline void TrafficMonitorLayerP$Timer$fired(void)

TrafficMonitorLayerP$neighborCount - fraction;
TrafficMonitorLayerP$neighborhoodFlag$clearAll();
TrafficMonitorLayerP$neighborCount = 0;
TrafficMonitorLayerP$TrafficMonitorConfig$timerTick();
TrafficMonitorLayerP$Tasklet$resume();

# 193 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\VirtualizeTimerC.nc"
static inline void VirtualizeTimerC$0$Timer$default$fired(uint8_t num)
{
    switch (arg_0x1a7c0cb8)
    {
    case 0U:
        ForwardToBaseC$Timer0$fired();
        break;
    case 1U:
        TrafficMonitorLayerP$Timer$fired();
        break;
    default:
        /*HilTimerMilliC.VirtualizeTimerC*/
        VirtualizeTimerC$0$Timer$default$fired(arg_0x1a7c0cb8);
        break;
    }
}

# 72 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\Timer.nc"
inline static void VirtualizeTimerC$0$Timer$fired(uint8_t arg_0x1a7c0cb8)
{
    #line 72
    case 0U:
        ForwardToBaseC$Timer0$fired();
        break;
    case 1U:
        TrafficMonitorLayerP$Timer$fired();
        break;
    default:
        #line 72
        /*HilTimerMilliC.VirtualizeTimerC*/
        VirtualizeTimerC$0$Timer$default$fired(arg_0x1a7c0cb8);
        break;
    }
}

# 174 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm1281\timer\HplAtm1281Timer2AsyncP.nc"
static inline void HplAtm1281Timer2AsyncP$Compare$set(uint8_t t)
{
    /* atomic removed: atomic calls only */
    {
        while (* (volatile uint8_t *)0xB6 & (1 << 3));
        * (volatile uint8_t *)0xB3 = t;
    }
}

# 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\timer\HplAtm128Compare.nc"
inline static void
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$set(/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$size_type arg_0x1a6924f0){
    #line 45
    HplAtm1281Timer2AsyncP$Compare$set(arg_0x1a6924f0);
    #line 45
}

# 79 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm1281\timer\HplAtm1281Timer2AsyncP.nc"
static inline uint8_t HplAtm1281Timer2AsyncP$Timer$get(void)
{
    #line 79
    return * (volatile uint8_t *)0xB2;
}

# 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\timer\HplAtm128Timer.nc"
inline static unsigned char
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$size
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$get(void){
    #line 52
    unsigned char result;
    #line 52
    result = HplAtm1281Timer2AsyncP$Timer$get();
}
#line 52  return result;
#line 52

static inline int HplAtm1281Timer2AsyncP$TimerAsync$compareABusy(void) {
    return (* (volatile uint8_t *)0xB6 & (1 << 3)) != 0;
}

static inline int /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$TimerAsync$compareABusy(void) {
    #line 75
    int result;
    #line 75
    result = HplAtm1281Timer2AsyncP$TimerAsync$compareABusy();
    #line 75
    return result;
    #line 75

    #line 101
    #line 101
    #line 101
    /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$setOcr2A(uint8_t n) {
        #line 101
        while (/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$TimerAsync$compareABusy()) {
            #line 101
            if (n == /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$get()) {n++;
                #line 101
                if (/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$base + n + 1 < /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$base) {
                    n = -/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$base - 1;
                }
            }
        }
        #line 111
        /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/ Atm1281AlarmAsyncP$0$Compare$set(n);
    }

    #line 56
    #line 56
    unsigned char result;
    #line 56
    result = SchedulerBasicP$TaskBasic$postTask(/*HilTimerMilliC.AlarmToTimerC*/ AlarmToTimerC$0$fired);
    #line 56
    return result;
    #line 56

    #line 70
    #line 70
    /*HilTimerMilliC.AlarmToTimerC*/ AlarmToTimerC$0$fired$postTask(void){
    #line 70
    return result;
    #line 70

    #line 67
    /*HilTimerMilliC.AlarmToTimerC*/
# line 161
*(volatile uint8_t *)0x70 |= 1 << 1;
}

{C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Compare.nc}
inline static void /*AlarmCounterMilliP.
Atm128AlarmAsync.C.atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP0$Compare$start(void)
{# line 56
HplAtm1281Timer2AsyncP0$Compare$start();
# line 56

# line 56
# 117 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281Timer2AsyncP.nc"
static inline void HplAtm1281Timer2AsyncP$TimerCtrl$setControlB(uint8_t x)
{# line 117
  while (*(volatile uint8_t *)0xB6 & (1 << 0));
  *(volatile uint8_t *)0xB1 = ((Atm128_TCCR2B_t )x).flat;
}

{C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281Timer2AsyncP.nc}
inline static void /*AlarmCounterMilliP.
Atm128AlarmAsync.C.atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP0$TimerCtrl$setControlB(uint8_t arg_0x1a696350)
{# line 62
  HplAtm1281Timer2AsyncP0$TimerCtrl$setControlB(arg_0x1a696350);
# line 62

# line 62
# 111 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281Timer2AsyncP.nc"
static inline void HplAtm1281Timer2AsyncP$TimerCtrl$setControlA(uint8_t x)
{# line 111
  while (*(volatile uint8_t *)0xB6 & (1 << 1));
  *(volatile uint8_t *)0xB0 = ((Atm128_TCCR2A_t )x).flat;
}

{C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281Timer2AsyncP.nc}
inline static void /*AlarmCounterMilliP.
Atm128AlarmAsync.C.atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP0$TimerCtrl$setControlA(uint8_t arg_0x1a680e28)
{# line 61
  HplAtm1281Timer2AsyncP0$TimerCtrl$setControlA(arg_0x1a680e28);
# line 61

# line 61
# 246 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281Timer2AsyncP.nc"
static inline void HplAtm1281Timer2AsyncP$TimerAsync$setTimer2Asynchronous(void)
{# line 246
  *(volatile uint8_t *)0xB6 |= 1 << 5;
}

{C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281Timer2AsyncP.nc}
inline static void /*AlarmCounterMilliP.
Atm128AlarmAsync.C.atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP0$TimerAsync$setTimer2Asynchronous(void)
{# line 57
  HplAtm1281Timer2AsyncP0$TimerAsync$setTimer2Asynchronous();
# line 57

# line 57
# 78 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm1281\\timer\\HplAtm1281AlarmAsyncP.nc"
static inline error_t /*AlarmCounterMilliP.
Atm128AlarmAsync.C.atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP0$Init$init(void)
{# line 78
  /* atomic removed: atomic calls only */
  Atm128_TCCR2A_t x;
  Atm128_TCCR2B_t y;
  /*AlarmCounterMilliP.C.128AlarmAsync*/
Atm1281AlarmAsyncP$0$TimerAsync$setTimer2Asynchronous();
    x.flat = 0;
    x.bits.wgm21 = 1;
  }/Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$TimerCtrl$setControlA(x.flat);
    y.flat = 0;
    y.bits.cs = 3;
  }/Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$set(/*AlarmCounterMilliP.Atm128AlarmAsyncC*/ Atm1281AlarmAsyncP$0$MAXT);
Atm1281AlarmAsyncP$0$Compare$setStart();
  };
  }/AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setInterrupt();
return SUCCESS;
} # 247 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\MessageBufferLayerP.nc"
static inline error_t MessageBufferLayerP$SoftwareInit$init(void)
{
    uint8_t i;
    for (i = 0; i < MessageBufferLayerP$RECEIVE_QUEUE_SIZE; ++i)
        MessageBufferLayerP$receiveQueue[i] = MessageBufferLayerP$receiveQueueData + i;
return SUCCESS;
} # 52 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\UniqueLayerP.nc"
static inline error_t UniqueLayerP$Init$init(void)
{
    UniqueLayerP$sequenceNumber = TOS_NODE_ID << 4;
return SUCCESS;
} # 44 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\NeighborhoodP.nc"
static inline error_t NeighborhoodP$Init$init(void)
{
    uint8_t i;
    for (i = 0; i < 5; ++i)
        NeighborhoodP$nodes[i] = AM_BROADCAST_ADDR;
return SUCCESS;
} # 44 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\system\RandomMlcgC.nc"
static inline error_t RandomMlcgC$Init$init(void)
#line 44
/* atomic removed: atomic calls only */
#line 45 RandomMlcgC$seed = (uint32_t )((TOS_NODE_ID + 1);
return SUCCESS;
} # 78 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Resource.nc"
inline static error_t RF230LayerP$SpiResourceRequest(void)
#line 78 unsigned char result;
#line 78 #line 78 result = Atm128SpiP$ResourceRequest(0U);
#line 78 #line 78 return result;
#line 78 #line 204 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\RF230LayerP.nc"
static inline error_t RF230LayerP$SoftwareInit$init(void)
{
    return RF230LayerP$SpiResourceRequest();
} # 45 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\system\FcfsResourceQueueC.nc"
static inline error_t /*Atm128SpiC.Arbiter.Queue*/
FcfsResourceQueueC$0$Init$init(void)
#line 45 memset(/*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$resQ,
/*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$NO_ENTRY, sizeof /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$resQ);
return SUCCESS;

inline static error_t RealMainP$SoftwareInit$init(void){
  unsigned char result;
  result = /*Atm128SpiC.Arbiter.Queue*/FcfsResourceQueueC$0$Init$init();
  result = ecombine(result, RF230LayerP$SoftwareInit$init());
  result = ecombine(result, RandomMlcgC$Init$init());
  result = ecombine(result, NeighborhoodP$Init$init());
  result = ecombine(result, UniqueLayerP$Init$init());
  result = ecombine(result, MessageBufferLayerP$SoftwareInit$init());
  result = ecombine(result, /*AlarmCounterMilliP.Atm128AlarmAsyncC. Atm1281AlarmAsyncP*/Atm1281AlarmAsyncP$0$Init$init());
  return result;
}

static inline void ForwardToBaseC$AMControl$start();
static inline void ForwardToBaseC$Boot$booted(void){
  ForwardToBaseC$AMControl$start();
}

inline static void RealMainP$Boot$booted(void){
  ForwardToBaseC$Boot$booted();
}

uint8_t diff;
if (* (volatile uint8_t *)0x70 & ((1 << 1) | (1 << 0))) {
  while (* (volatile uint8_t *)0xB6 & (((1 << 4) | (1 << 3)) | (1 << 1)));
  diff = * (volatile uint8_t *)0xB3 - *(volatile uint8_t *)0xB2;
  if (diff < EXT_STANDBY_T0_THRESHOLD ||
      *(volatile uint8_t *)0xB2 > 256 - EXT_STANDBY_T0_THRESHOLD) {
    return ATM128_POWER_EXT_STANDBY;
  }
  return ATM128_POWER_SAVE;
} else {
  return ATM128_POWER_DOWN;
}

inline static mcu_power_t McuSleepC$McuPowerOverride$lowestState(void){
  unsigned char result;
  result = HplAtm1281Timer2AsyncP$McuPowerOverride$lowestState();
  return result;
}
# 54 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \interfaces\McuPowerOverride.nc"
inline static mcu_power_t McuSleepC$McuPowerOverride$lowestState(void){
  unsigned char result;
  result = HplAtm1281Timer2AsyncP$McuPowerOverride$lowestState();
  return result;
}
static inline mcu_power_t McuSleepC$getPowerState(void)
{
    if (((volatile uint8_t *)0x6E & ((1 << 1) | (1 << 2)) ||
         (volatile uint8_t *)0x6F & ((((1 << 5) | (1 << 1)) | (1 << 2)) | (1 << 3))) ||
         (volatile uint8_t *)0x71 & ((((1 << 5) | (1 << 1)) | (1 << 2)) | (1 << 3)))
    {
        return ATM128_POWER_IDLE;
    }
    else {
        if (((volatile uint8_t *)[uint16_t]['X', 0X80] & (1 << 7)) {
            return ATM128_POWER_IDLE;
        }
        else {
            if (((volatile uint8_t *)[uint16_t]['X', 0X88] & (1 << 0)) {
                return ATM128_POWER_ADC_NR;
            }
            else {
                return ATM128_POWER_DOWN;
            }
        }
    }
}

static inline mcu_power_t mcombine(mcu_power_t m1, mcu_power_t m2)
{
    return m1 < m2 ? m1 : m2;
}

uint8_t powerState;
powerState = mcombine(McuSleepC$getPowerState(),
    McuSleepC$McuPowerOverride$lowestState());
    + (volatile uint8_t *)[uint16_t]['X', 0X80] & 0X33 + 0X20) & 0Xf0) | (1 << 0)) | __extension__ ({
    #line 133
    uint16_t __addr16 = (uint16_t *)[uint16_t]['X', 0X80] & McuSleepC$atm128PowerBits[powerState];
    #line 133
    __asm
      "" : "z"("result" : __result;
      ;
    #line 134
    __asm volatile ("asm volatile ("seil");
    __asm volatile ("sleep" : : : "memory";
    __asm volatile ("cli");
* (volatile uint8_t *)(0x33 + 0x20) &= ~(1 << 0);
}

# 59 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\McuSleep.nc"
inline static void SchedulerBasicP$McuSleep$sleep(void){
    McuSleepP$McuSleep$sleep();
    #line 59
}

# 59 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\system\SchedulerBasicP.nc"
static __inline uint8_t SchedulerBasicP$popTask(void)
{
    if (SchedulerBasicP$m_head != SchedulerBasicP$NO_TASK)
    {
        uint8_t id = SchedulerBasicP$m_head;
        SchedulerBasicP$m_head = SchedulerBasicP$m_next[SchedulerBasicP$m_head];
        if (SchedulerBasicP$m_head == SchedulerBasicP$NO_TASK)
        {
            SchedulerBasicP$m_tail = SchedulerBasicP$NO_TASK;
            return id;
        }
        else
        {
            return SchedulerBasicP$NO_TASK;
        }
    }
    else{
        return SchedulerBasicP$NO_TASK;
    }
}

#line 138
static inline void SchedulerBasicP$Scheduler$taskLoop(void)
{
    for (; ; )
    {
        uint8_t nextTask;
        __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        while ((nextTask = SchedulerBasicP$popTask()) == SchedulerBasicP$NO_TASK)
        {
            SchedulerBasicP$McuSleep$sleep();
        }
        __nesc_atomic_end(__nesc_atomic);
    }
    SchedulerBasicP$TaskBasic$runTask(nextTask);
    #line 138
}

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\interfaces\Scheduler.nc"
inline static void RealMainP$Scheduler$taskLoop(void){
    SchedulerBasicP$Scheduler$taskLoop();
    #line 61
}

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\timer\CounterToLocalTimeC.nc"
inline void / *HilTimerMilliC.CounterToLocalTimeC*/
CounterToLocalTimeC$0$Counter$overflow(void)
{
}

# 71 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\lib\timer\CounterToLocalTimeC.nc"
static inline void / *HilTimerMilliC.CounterToLocalTimeC*/
CounterToLocalTimeC$0$Counter$overflow(void)
{
    (volatile uint8_t *)0xB0 = (volatile uint8_t *)0xB0;
    while (* (volatile uint8_t *)0xB6 & (1 << 1))
    {
        #line 71
    }
    #line 71
}

354
static __inline void __nesc_enable_interrupt(void)
#line 105
{ static volatile (*"sei");

asm volatile ("sei");
}

# 171 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm1281\timer\HplAtm1281Timer2AsyncP.nc"
static Inline uint8_t HplAtm1281Timer2AsyncP$Compare$get(void)
#line 171
{ return *(volatile uint8_t *)0xB3;
}

# 39 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Compare.nc"

inline static
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$get(void){
#line 39
unsigned char result;
#line 39
result = HplAtm1281Timer2AsyncP$Compare$get();
#line 39
return result;
#line 39
}

# 176 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm1281\timer\Atm1281AlarmAsyncP.nc"

static inline void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$fired(void)
#line 176
{ int overflowed;
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$base +=
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$get() + 1U;
overflowed = !/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$base;
__nesc_enable_interrupt();
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setInterrupt();
if (overflowed)
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Counter$overflow();
}

# 49 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Compare.nc"

inline static void HplAtm1281Timer2AsyncP$Compare$fired(void){
#line 49
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$fired();
#line 49
}

# 257 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm1281\timer\Atm1281AlarmAsyncP.nc"

inline static void /*AlarmCounterMilliP.
Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$overflow(void)
#line 257
{
}

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Timer.nc"
inline static void HplAtm1281Timer2AsyncP$Timer$overflow(void){
#line 61
/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$overflow();
#line 61
}

# 61 /*Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\Tasklet.nc"
inline static void RadioAlarmP$Tasklet$schedule(void){
#line 48
TaskletC$Tasklet$schedule();
#line 48
}

# 48 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\rf230\Tasklet.nc"
inline static void RadioAlarmP$Tasklet$schedule(void){
#line 48
TaskletC$Tasklet$schedule();
#line 48
}
static inline void RadioAlarmP$Alarm$fired(void)
{
  __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
  if (RadioAlarmP$state == RadioAlarmP$STATE_WAIT) {
    RadioAlarmP$state = RadioAlarmP$STATE_FIRED;
  }
  __nesc_atomic_end(__nesc_atomic);
  RadioAlarmP$Tasklet$schedule();
}

inline static void / *HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$Alarm$fired(void)
{
  RadioAlarmP$Alarm$fired();
}

static inline void / *HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$fired(void)
{
  __nesc_enable_interrupt();
  Atm128AlarmC$0$Alarm$fired();
}

inline static void HplAtm1281Timer1P$CompareA$fired(void)
{
  HplAtm1281Timer1P$CompareB$default$fired();
}

inline static void HplAtm1281Timer1P$CompareB$fired(void)
{
  HplAtm1281Timer1P$CompareC$default$fired();
}

static inline void RF230LayerP$IRQ$captured(uint16_t time)
{
  for (; 0; ) ;
  __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
  if (RadioAlarmP$state == RadioAlarmP$STATE_WAIT) {
    RadioAlarmP$state = RadioAlarmP$STATE_FIRED;
  }
  __nesc_atomic_end(__nesc_atomic);
  RadioAlarmP$Tasklet$schedule();
}

356
inline static void HplRF230P$IRQ$captured(uint16_t arg_0x1ad7c680) {
    RF230LayerP$IRQ$captured(arg_0x1ad7c680);
}

static inline uint16_t HplAtm1281Timer1P$Capture$get(void) {
    return * (volatile uint16_t *)0x86;
}

inline static HplRF230P$Capture$size_type HplRF230P$Capture$get(void) {
    unsigned short result;
    result = HplAtm1281Timer1P$Capture$get();
    return result;
}

static inline void HplRF230P$Capture$captured(uint16_t time) {
    time = HplRF230P$Capture$get();
    HplRF230P$IRQ$captured(time);
}

inline static void HplAtm1281Timer1P$Capture$captured(HplAtm1281Timer1P$Capture$size_type arg_0x1ab0a540) {
    HplRF230P$Capture$captured(arg_0x1ab0a540);
}

}
/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$CounterFromOverflow();
#line 71

#line 71
# 56 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\Atm128CounterC.nc"
static inline void */CounterOne16C.NCounter*/
Atm128CounterC$0$TimerOverflow( void )
{ /*CounterOne16C.NCounter*/
Atm128CounterC$0$CounterOverflow();
}

# 51 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\Atm128TimerInitC.nc"
static inline void */InitOneP.InitOne*/
Atm128TimerInitC$0$TimerOverflow( void )
#line 51

# 117 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\Atm128AlarmC.nc"
static inline void */HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128TimerOverflow( void )
#line 117

# 61 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\timer\HplAtm128Timer.nc"
inline static void HplAtm1281Timer1P$TimerOverflow( void )
{ /*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128TimerOverflow();
 /*InitOneP.InitOne*/
Atm128TimerInitC$0$TimerOverflow();
 /*CounterOne16C.NCounter*/
Atm128CounterC$0$TimerOverflow();
}

#line 61

# 98 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\HplAtm128SpiP.nc"
static inline uint8_t HplAtm128SpiP$SPIRead( void )
#line 98

return * (volatile uint8_t *)(0X2E + 0x20);

# 80 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\Atm128Spi.nc"
inline static uint8_t Atm128SpiP$SPIRead( void )
{ unsigned char result;
return result;
}

#line 80

# 99 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\HplAtm128SpiP.nc"
static inline void HplAtm128SpiP$SPIWrite( uint8_t d )
#line 99

*(volatile uint8_t *)(0X2E + 0x20) = d;

# 86 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
\chips\atm128\spi\Atm128Spi.nc"
inline static void Atm128SpiP$SPIWrite( uint8_t arg_0x1ae46218 )
{ HplAtm128SpiP$SPIWrite( arg_0x1ae46218 );
}

# 86

inline static void Atm128SpiP$SPIEnableInterrupt( bool arg_0x1ae46c70 )
{ HplAtm128SpiP$SPIEnableInterrupt( arg_0x1ae46c70 );
}

# 96

# 162 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\atm128\spi\Atm128SpiP.nc"
static inline error_t Atm128SpiP$sendNextPart(void)
{
    uint16_t end;
    uint16_t tmpPos;
    uint16_t myLen;
    uint8_t *tx;
    uint8_t *rx;
    /* atomic removed: atomic calls only */

    myLen = Atm128SpiP$len;
    tx = Atm128SpiP$txBuffer;
    rx = Atm128SpiP$rxBuffer;
    tmpPos = Atm128SpiP$pos;
    end = Atm128SpiP$pos + Atm128SpiP$SPI_ATOMIC_SIZE;
    end = end > Atm128SpiP$len ? Atm128SpiP$len : end;

    for (; tmpPos < end - 1; tmpPos++) {
        uint8_t val;

        if (tx != (void *)0) {
            val = Atm128SpiP$SpiByte$write(tx[tmpPos]);
        } else {
            val = Atm128SpiP$SpiByte$write(0);
        }

        if (rx != (void *)0) {
            rx[tmpPos] = val;
        }
    }

    Atm128SpiP$Spi$enableInterrupt(TRUE);

    /* atomic removed: atomic calls only */

    if (tx != (void *)0) {
        Atm128SpiP$Spi$write(tx[tmpPos]);
    } else {
        Atm128SpiP$Spi$write(0);
    }

    Atm128SpiP$pos = tmpPos;
}
return SUCCESS;
}

static inline void Atm128SpiP$Spi$dataReady(uint8_t data)
{
    bool again;
    /* atomic removed: atomic calls only */

    if (Atm128SpiP$rxBuffer != (void *)0) {
        Atm128SpiP$rxBuffer[Atm128SpiP$pos] = data;
    } else {
       discard = Atm128SpiP$Spi$read();
    }

    Atm128SpiP$Spi$enableInterrupt(FALSE);
    /* atomic removed: atomic calls only */

    if (again) {
        Atm128SpiP$sendNextPart();
    } else {
        discard = Atm128SpiP$Spi$read();
    }
}
Atm128SpiP$SpiPacket$sendDone(tx, rx, myLen, SUCCESS);
}

# 92 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\chips\\atm128\\spi\\Atm128Spi.nc"
inline static void HplAtm128SpiP$Spi$dataReady(uint8_t arg_0x1ae46728)
{
    Atm128SpiP$Spi$dataReady(arg_0x1ae46728);
}

# 92 #line 92
HplAtm128SpiP$SPI$dataReady(arg_0x1ae46728);

# 52 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\system\\RealMainP.nc"
int main(void)
{

    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();

    __nesc_atomic_t __nesc_atomic_t __nesc_atomic_end(__nesc_atomic);
    __nesc_enable_interrupt();
    RealMainP$Boot$booted();
    RealMainP$Scheduler$taskLoop();
    return -1;
}

# 123 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\system\\SchedulerBasicP.nc"
static bool SchedulerBasicP$Scheduler$runNextTask(void)
{
    uint8_t nextTask;
    /* atomic removed: atomic calls only */
    #line 127
    nextTask = SchedulerBasicP$popTask();
    if (nextTask == SchedulerBasicP$NO_TASK)
    {
        __nesc_temp =
        #line 131 FALSE;
        __nesc_enable_interrupt();
        RealMainP$Boot$booted();
        return -1;
    }
    #line 134
    SchedulerBasicP$TaskBasic$runTask(nextTask);
    return TRUE;
}

static void SchedulerBasicP$TaskBasic$default$runTask(uint8_t id)
{
}

# 64 "C:\\Crossbow\\cygwin\\opt\\MoteWorks\\tinyos-2.x\\tos\\interfaces\\TaskBasic.nc"
static void SchedulerBasicP$TaskBasic$runTask(uint8_t arg_0x1a1206b8)
{
    switch (arg_0x1a1206b8)
    {
        #line 64 case /*HillTimerMilliC.AlarmToTimerC*/
        #line 64 AlarmToTimerC$0$fired:
            #line 64
            #line 64 SchedulerBasicP$TaskBasic$runTask(arg_0x1a1206b8);
            #line 64 break;
            #line 64 /\*HillTimerMilliC.AlarmToTimerC*/
            #line 64 AlarmToTimerC$0$fired$runTask();
            #line 64 break;
            #line 64 /\*HillTimerMilliC.VirtualizeTimerC*/
            #line 64 VirtualizeTimerC$0$updateFromTimer();
            #line 64 break;
            #line 64 /\*HillTimerMilliC.VirtualizeTimerC*/
            #line 64 VirtualizeTimerC$0$updateFromTimer$runTask();
            #line 64 break;

360
case ForwardToBaseC$toggle:
    ForwardToBaseC$toggle$runTask();
    break;

case MessageBufferLayerP$stateDoneTask:
    MessageBufferLayerP$stateDoneTask$runTask();
    break;

case MessageBufferLayerP$sendTask:
    MessageBufferLayerP$sendTask$runTask();
    break;

case MessageBufferLayerP$deliverTask:
    MessageBufferLayerP$deliverTask$runTask();
    break;

case TrafficMonitorLayerP$startStopTimer:
    TrafficMonitorLayerP$startStopTimer$runTask();
    break;

case RandomCollisionLayerP$calcNextRandom:
    RandomCollisionLayerP$calcNextRandom$runTask();
    break;

case Atm128SpiP$zeroTask:
    Atm128SpiP$zeroTask$runTask();
    break;

    case /*Atm128SpiC.Arbiter.Arbiter*/ SimpleArbiterP$0$grantedTask:
        /*Atm128SpiC.Arbiter.Arbiter*/ SimpleArbiterP$0$grantedTask$runTask();
        break;

    case /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$CancelTask:
        /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$CancelTask$runTask();
        break;

    case /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$errorTask:
        /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$errorTask$runTask();
        break;

    default:
        SchedulerBasicP$TaskBasic$default$runTask(arg_0x1a1206b8);
        break;

}

/*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$sendDone(uint8_t last, message_t *msg, error_t err)
{ static void /*AMQueueP.AMQueueImplP*/ AMQueueImplP$0$sendDone(uint8_t last, message_t *msg, error_t err)
SendSendDone(last, msg, err);

static error_t ActiveMessageLayerC$AMSend$(am_id_t id, am_addr_t addr, message_t *msg, uint8_t len)
{
    error_t error;
    error = ActiveMessageLayerC$Config$checkPacket(msg);
    if (error != SUCCESS) {
        return error;
    }
    ActiveMessageLayerC$AMPacket$setSource(msg, ActiveMessageLayerC$AMPacket$address());
    ActiveMessageLayerC$AMPacket$setGroup(msg, ActiveMessageLayerC$AMPacket$localGroup());
    ActiveMessageLayerC$AMPacket$setType(msg, id);
    ActiveMessageLayerC$AMPacket$setDestination(msg, addr);
    return ActiveMessageLayerC$SubSend$send(msg, len);
}

static am_addr_t ActiveMessageAddressC$amAddress(void)
{
    am_addr_t myAddr;
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    myAddr = ActiveMessageAddressC$addr;  
    __nesc_atomic_end(__nesc_atomic);  
    return myAddr;
}

static error_t SchedulerBasicP$TaskBasic$postTask(uint8_t id)
{
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    unsigned char __nesc_temp = SchedulerBasicP$pushTask(id) ? SUCCESS : EBUSY;
    __nesc_atomic_end(__nesc_atomic);
    return __nesc_temp;
}

static bool / *Atm128SpiC.Arbiter.Arbiter*/ SimpleArbiterP$0$ArbiterInfo$inUse(void)
{
    /* atomic removed: atomic calls only */
    return true;
}

error_t error = Atm128SpiP$ResourceArbiter$release(id);
    __nesc_atomic_end(__nesc_atomic);
    return error;
}

static bool /*Atm128SpiC.Arbiter.Arbiter*/ SimpleArbiterP$0$ArbiterInfo$inUse(void)
{
    /* atomic removed: atomic calls only */
    return true;
}
if (/*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$state == /*Atm128SpiC.Arbiter.Arbiter*/SimpleArbiterP$0$RES_IDLE) {
  unsigned char __nesc_temp =
  return __nesc_temp;
}
return TRUE;

static void HplAtm128SpiP$SPI$enableSpi(bool enabled) {
  if (enabled) {
    *(volatile uint8_t *)(0x2C + 0x20) |= 1 << 6;
    HplAtm128SpiP$Mcu$update();
  } else {
    *(volatile uint8_t *)(0x2C + 0x20) &= ~{(1 << 6); HplAtm128SpiP$Mcu$update();
  }
}

static void TaskletC$Tasklet$schedule(void) {
  { __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    if (TaskletC$state != 0)
      TaskletC$state |= 0x80;
    __nesc_atomic_end(__nesc_atomic);
    return;
  } TaskletC$doit();
}
static void TaskletC$doit(void) {
  for (; ; )
    TaskletC$Tasklet$run();
    __nesc_atomic_end(__nesc_atomic);
}

static bool RF230LayerP$isSpiAcquired(void) {
  if (RF230LayerP$SpiResource$isOwner()) {
    return TRUE;
  } else {
    if (RF230LayerP$SpiResource$immediateRequest() == SUCCESS)
      RF230LayerP$SELN$makeOutput();
      RF230LayerP$SELN$set();

  return FALSE;
  return TRUE;
return TRUE;
RF230LayerP$SpiResource$getRequest();
return FALSE;

# 105 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\atm128\spi\Atm128SpiP.nc"
static void Atm128SpiP$startSpi(void)
#line 105{
    Atm128SpiP$Spi$enableSpi(FALSE);
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
#line 107{
    Atm128SpiP$Spi$initMaster();
    Atm128SpiP$Spi$enableInterrupt(FALSE);
    Atm128SpiP$Spi)setMasterDoubleSpeed(TRUE);
    Atm128SpiP$Spi$setClockPolarity(FALSE);
    Atm128SpiP$Spi$setClockPhase(FALSE);
    Atm128SpiP$Spi$setClock(0);
    Atm128SpiP$Spi$enableSpi(TRUE);
}
#line 115
__nesc_atomic_end(__nesc_atomic); }
Atm128SpiP$McuPowerState$update();

# 115 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\atm128\spi\HplAtm128SpiP.nc"
static void HplAtm128SpiP$SPI$enableInterrupt(bool enabled)
#line 115{
if (enabled) {
    *(volatile uint8_t *)(0x2C + 0x20) |= 1 << 7;
    HplAtm128SpiP$Mcu$update();
}
else {
    *(volatile uint8_t *)(0x2C + 0x20) &= ~(1 << 7);
    HplAtm128SpiP$Mcu$update();
}

# 313 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \chips\atm128\spi\Atm128SpiP.nc"
static error_t Atm128SpiP$Resource$getRequest(uint8_t id)
#line 313{
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
#line 314{
    if (!Atm128SpiP$ArbiterInfo$inUse()) {
        Atm128SpiP$startSpi();
    }
#line 318
__nesc_atomic_end(__nesc_atomic); }
Atm128SpiP$ResourceArbiter$request(id);
}
# 69 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos \lib\timer\TransformCounterC.nc"
static / *LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$to_size_type
/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$Counter$get(void)
/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$to_size_type rv = 0;
#line 72
/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$Upper_count_type high = /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$Upper_count_type high = /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$m_upper;
/*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$Lower_count_type low = /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$Lower_count_type low = /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$CounterFrom$get();
#line 76
if /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$CounterFrom$IsOverflowPending())
    high++;
    low = /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$CounterFrom$get();
    }
/*LocalTimeMicroC.TransformCounterC*/

364
TransformCounterC$0$to_size_type high_to = high;
/\LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$to_size_type low_to = low; 
/\LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$LOW_SHIFT_RIGHT;
#line 90
rv = (high_to << /*LocalTimeMicroC.TransformCounterC*/
TransformCounterC$0$HIGH_SHIFT_LEFT) | low_to;
}
#line 92
__nesc_atomic_end(__nesc_atomic); }
return rv;

// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//\chips\rf230\RF230PacketP.nc
static void RF230PacketP$PacketTimeStampRadio$ set(message_t *msg, uint32_t value)
{
RF230PacketP$getMeta(msg)->flags |=
RF230PACKET_TIMESTAMP;
RF230PacketP$getMeta(msg)->timestamp = value;
}
// C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
//\chips\atm128\timer\Atm128AlarmC.nc
static void /*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$Alarm$startAt(/*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$timer_size t0,
/*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$timer_size dt)
#line 74{
{ __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();

/*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$timer_size now;
#line 83
/*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$timer_size elapsed;
#line 83
/*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$timer_size expires;
now = /*HplRF230C.AlarmC.NAlarm*/Atm128AlarmC$0$HplAtm128Timer$get();
elapsed = now + 3 - t0;
if (elapsed >= dt) {
expires = now + 3;
}
else {
expires = t0 + dt;
}
if (expires == 0) {
expires = 1;
}
/*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$set(expires - 1);
/*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$reset();
/*HplRF230C.AlarmC.NAlarm*/
Atm128AlarmC$0$HplAtm128Compare$start();
}
#line 107
__nesc_atomic_end(__nesc_atomic); }

static uint16_t RF230ActiveMessageP$TrafficMonitorConfig$getChannelTime(message_t *msg)
{
uint8_t len = RF230ActiveMessageP$IEEE154Packet$getLength(msg);
#line 195
RF230ActiveMessageP$RadioSend$sendDone(error_t error);
}
#line 205
return RF230ActiveMessageP$IEEE154Packet$getAckRequired(msg) ? len + 6 + 16 + 11 + 10 : len + 6 + 10;
}
}
setAckReceived(message_t *msg, bool acked)
{
  if (acked) {
    RF230ActiveMessageP$getMeta(msg)->flags |=
    RF230_PACKET_WAS_ACKED;
  }
  else {
    #line 129
    RF230ActiveMessageP$getMeta(msg)->flags &=
   ˜RF230_PACKET_WAS_ACKED;
  }
}

# 225 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\IEEE154PacketP.nc"
static bool IEEE154PacketP$IEEE154Packet
requiresAckReply(message_t *msg)
{
  return IEEE154PacketP$IEEE154Packet$getAckRequired(msg)
    && IEEE154PacketP$IEEE154Packet$isDataFrame(msg)
    && IEEE154PacketP$IEEE154Packet$getDestAddr(msg) ==
    IEEE154PacketP$ActiveMessageAddress$amAddress();
}

# 389 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RF230LayerP.nc"
static error_t RF230LayerP$RadioSend$send(message_t *msg)
{
uint16_t time;
uint8_t length;
uint8_t *data;
uint8_t header;
uint32_t time32;
void *timesync;
if (((RF230LayerP$cmd != RF230LayerP$CMD_NONE ||
RF230LayerP$state != RF230LayerP$STATE_RX_ON) ||
!RF230LayerP$isSpiAcquired()) || RF230LayerP$radioIrq) {
  return EBUSY;
}
length = (RF230LayerP$PacketTransmitPower$isSet(msg) ?
RF230LayerP$PacketTransmitPower$get(msg) : 0) &
RF230_TX_PWR_MASK;
if (length != RF230LayerP$txPower)
{
  RF230LayerP$txPower = length;
  RF230LayerP$writeRegister(RF230_PHY_TX_PWR, RF230_TX_AUTO_CRC_ON | RF230LayerP$txPower);
}

if (RF230LayerP$RF230Config$requiresRssiCca(msg) &
RF230LayerP$readRegister(RF230_PHY_RSSI) & RF230_RSSI_MASK) !=
RF230LayerP$rssiClear + RF230LayerP$rssiBusy >> 3) {
  return EBUSY;
}
RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_PLL_ON);
time32 = RF230LayerP$LocalTime$get();
timesync = RF230LayerP$PacketTimeSync-offset$isNew(msg) ?
msg->data + RF230LayerP$PacketTimeSync-offset$get(msg) : 0;
RF230LayerP$writeRegister(RF230_TRX_STATUS) &
RF230_TRX_STATUS_MASK) != RF230_PLL_ON)
{
  for (; 0; ) ;
  RF230LayerP$state = RF230LayerP$STATE_PLL_ON_2_RX_ON;
  return EBUSY;
}

RF230LayerP$SELN$clr();
RF230LayerP$RF230Config$setHeaderLength();
RF230LayerP$SELW$crcl();
for (0; ) ;
if (header > length) {
    header = length;
}
length -= header;
do {
    RF230LayerP$HplRF230$spiSplitReadWrite(* data++);
} while (--header != 0);
time32 += (int16_t)time - (int16_t)time32;
if (timesync != 0) {
    __nesc_hton_int32((unsigned char *)&*(timesync_absolute_t *)timesync, * (timesync_relative_t *)timesync - time32);
}
do {
    RF230LayerP$HplRF230$spiSplitReadWrite(* data++);
} while (--length != 0);
RF230LayerP$SPIEn$set();
RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_RX_ON);
if (timesync != 0) {
    * (timesync_absolute_t *)timesync = __nesc_ntoh_int32((unsigned char *)&*(timesync_relative_t *)timesync) + time32;
    RF230LayerP$PacketTimeStamp$set(msg, time32);
    RF230LayerP$state = RF230LayerP$STATE_BUSY_TX_2_RX_ON;
    RF230LayerP$cmd = RF230LayerP$CMD_TRANSMIT;
    return SUCCESS;
}
#line 489
RF230LayerP$writeRegister(RF230_TRX_STATE, RF230_RX_ON);
#line 494
if (timesync != 0) {
    *(timesync_absolute_t *)timesync = __nesc_ntoh_int32((unsigned char *)&*(timesync_relative_t *)timesync) + time32;
    RF230LayerP$PacketTimeStamp$set(msg, time32);
    RF230LayerP$state = RF230LayerP$STATE_BUSY_TX_2_RX_ON;
    RF230LayerP$cmd = RF230LayerP$CMD_TRANSMIT;
    return SUCCESS;
};
};
if ((NeighborhoodP$time & 0x7F) == 0x7F && maxAge >= 0x7F) {
    for (i = 0; i < 5; ++i) {
        if ((NeighborhoodP$ages[i] | 0x7F) != NeighborhoodP$time) {
            NeighborhoodP$ages[i] = NeighborhoodP$time & 0x80;
        }
    }
}
return NeighborhoodP$last;

# 73 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\chips\rf230\RandomCollisionLayerP.nc"
static uint16_t RandomCollisionLayerP$getBackoff(uint16_t maxBackoff) {
    uint16_t a;
    { __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
        a = RandomCollisionLayerP$nextRandom;
        RandomCollisionLayerP$nextRandom += 273;
    #line 81
    __nesc_atomic_end(__nesc_atomic); }
    RandomCollisionLayerP$calcNextRandom$postTask();
    return a % maxBackoff + RandomCollisionLayerP$Config$getMinimumBackoff();
}

# 58 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\system\RandomMlcgC.nc"
static uint32_t RandomMlcgC$Random$rand32(void) {
    uint32_t mlcg;
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    { tmpseed = (uint64_t )33614U * (uint64_t )RandomMlcgC$seed;
        q = tmpseed;
        q = q >> 1;
        p = tmpseed >> 32;
        mlcg = p + q;
        if (mlcg & 0x80000000) {
            mlcg = mlcg & 0x7FFFFFFF;
            mlcg++;
        }
    RandomMlcgC$seed = mlcg;
    __nesc_atomic_end(__nesc_atomic); }
    return mlcg;
}

# 133 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos\lib\timer\VirtualizeTimerC.nc"
static void /*HilTimerMilliC.VirtualizeTimerC*/VirtualizeTimerC$0$startTimer(uint8_t num, uint32_t t0, uint32_t dt, bool isoneshot) {
    /*HilTimerMilliC.VirtualizeTimerC*/VirtualizeTimerC$0$Timer_t *timer = &/*HilTimerMilliC.VirtualizeTimerC*/VirtualizeTimerC$0$m_timers[num];
    startTimer((uint8_t )t0, (uint32_t )dt, bool isoneshot);}
#line 188
{ uint32_t now;
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    uint8_t now8 = Atm128AlarmAsyncP$0$Timer$get();
    if (((Atm128_TIFR2_t )Atm128AlarmAsyncP$0$TimerCtrl$getInterruptFlag()).bits.ocfa) {
        now = Atm128AlarmAsyncP$0$base + Atm128AlarmAsyncP$0$Compare$get() + 1 + now8;
    } else {
        now = Atm128AlarmAsyncP$0$base + now8;
    }
    __nesc_atomic_end(__nesc_atomic); }
#line 206
return now;
}

static error_t AMQueueEntryP$0$AMSend$send(am_addr_t dest, message_t *msg, uint8_t len)
{
    /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/ AMQueueEntryP$0$AMPacket$setDestination(msg, dest);
    /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/ AMQueueEntryP$0$AMPacket$setType(msg, 6);
    return /*ForwardToBaseAppC.AMSenderC.AMQueueEntryP*/ AMQueueEntryP$0$Send$send(msg, len);
}

static void TaskletC$Tasklet$resume(void)
{
    __nesc_atomic_t __nesc_atomic = __nesc_atomic_start();
    if (--TaskletC$state != 0x80) {
        __nesc_atomic_end(__nesc_atomic);
        return;
    }
    TaskletC$state = 1;
    __nesc_atomic_end(__nesc_atomic);
    TaskletC$doit();
}

static error_t MessageBufferLayerP$SplitControl$start(void)
{
    error_t error;
    MessageBufferLayerP$Tasklet$suspend();
    if (MessageBufferLayerP$state != MessageBufferLayerP$STATE_READY) {
        error = EBUSY;
    } else {
        error = MessageBufferLayerP$RadioState$turnOn();
        if (error == SUCCESS) {
            MessageBufferLayerP$state = MessageBufferLayerP$STATE_TURN_ON;
            MessageBufferLayerP$Tasklet$resume();
            return error;
        }
    }
{
    __nesc_atomic_end(__nesc_atomic);
    TaskletC$doit();
}
static void /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$fireTimers(uint32_t now)
{
    uint8_t num;
    for (num = 0; num < /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$NUM_TIMERS; num++)
    {
        /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer_t *timer = &/*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$m_timers[num];
        if (timer->isrunning)
        {
            uint32_t elapsed = now - timer->t0;
            if (elapsed >= timer->dt)
            {
                if (timer->isoneshot) {
                    timer->isrunning = FALSE;
                } else {
                    timer->t0 += timer->dt;
                    /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$Timer$fired(num);
                    break;
                }
            }
        }
    }
    /*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer$postTask();
}

/*HilTimerMilliC.VirtualizeTimerC*/
VirtualizeTimerC$0$updateFromTimer$postTask();

static void /*AlarmCounterMilliP.Atm128AlarmAsyncC.
Atm1281AlarmAsyncP*/Atm1281AlarmAsyncP$0$Alarm$startAt(uint32_t nt0, uint32_t ndt)
#line 239{
    __nesc Atomic_t __nesc Atomic = __nesc Atomic_start();
    { __nesc Atomic_t __nesc Atomic = __nesc Atomic_start();
        /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$set = TRUE;
        /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$t0 = nt0;
        /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$dt = ndt;
    }
    __nesc Atomic_end(__nesc Atomic); }

//AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setInterrupt();

static void /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$setInterrupt(void)
#line 117{
    bool fired = FALSE;
    __nesc Atomic_t __nesc Atomic = __nesc Atomic_start();
    uint8_t interrupt_in = 1 +
    /AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Compare$get()
    /AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$Timer$get();
    uint8_t newOCR2A;
    uint8_t tIFr2 = /AlarmCounterMilliP.
Atm1281AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$TimerCtrl$InterruptFlag();
    #line 128
    if ((interrupt_in != 0 && interrupt_in <
    /AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
Atm1281AlarmAsyncP$0$MINDT) || tIFr2 & (1 << 1)) {
        if (interrupt_in < /AlarmCounterMilliP.Atm128AlarmAsyncC.
Atm1281AlarmAsyncP*/Atm1281AlarmAsyncP$0$MINDT) {
        } else {;
    }
    __nesc Atomic_end(__nesc Atomic);
    return;
}
if (!/*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
    Atm1281AlarmAsyncP$0$set) {
    newOcr2A = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
    Atm1281AlarmAsyncP$0$MAXT;;
} else {
    uint32_t now = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
    Atm1281AlarmAsyncP$0$counter$get();
    if ((uint32_t )
        (now - /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$t0) >=
        /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
        Atm1281AlarmAsyncP$0$dt)
        {
            /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$set = FALSE;
            fired = TRUE;
            newOcr2A = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$MAXT;
        }
    else {
        uint32_t alarm_in = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$t0 +
            /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$base;
        if (alarm_in > /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$MAXT) {
            newOcr2A = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
            Atm1281AlarmAsyncP$0$MAXT; } else {
            if ((uint8_t )alarm_in < /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
                Atm1281AlarmAsyncP$0$MINDT) {
                newOcr2A = /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
                Atm1281AlarmAsyncP$0$MINDT; } else {
                newOcr2A = alarm_in;
            }
        }
    }
    __nesc_atomic_end(__nesc_atomic);
    if (fired) {
        /*AlarmCounterMilliP.Atm128AlarmAsyncC.Atm1281AlarmAsyncP*/
        Atm1281AlarmAsyncP$0$Alarm$fired();
    }
}

__attribute__((signal)) void __vector_13(void)
{ /*HplAtm1281Timer2AsyncP.nc*/
    HplAtm1281Timer2AsyncP$stabiliseTimer2();
    HplAtm1281Timer2AsyncP$Compare$fired();
}

__attribute__((interrupt)) void __vector_17(void)
{ /*HplAtm1281Timer1P.nc*/
    HplAtm1281Timer1P$stabiliseTimer1();
}

# 222 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
 \chips\atm128\timer\HplAtm1281Timer2AsyncP.nc"
__attribute__((signal)) void __vector_13(void)
{ /*HplAtm1281Timer2AsyncP.nc*/
    HplAtm1281Timer2AsyncP$stabiliseTimer2();
    HplAtm1281Timer2AsyncP$Compare$fired();
}

# 206 "C:\Crossbow\cygwin\opt\MoteWorks\tinyos-2.x\tos
 \chips\atm128\timer\HplAtm1281Timer1P.nc"
__attribute__((interrupt)) void __vector_17(void)
{ /*HplAtm1281Timer1P.nc*/

371
B.4 InterceptBase

B.4.1 BaseStationC

/*
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 */

/* Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 * Permission to use, copy, modify, and distribute this software and its
documentation for any purpose, without fee, and without written agreement is
hereby granted, provided that the above copyright notice, the following
two paragraphs and the author appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/

Copyright (c) 2002-2003 Intel Corporation
All rights reserved.
This file is distributed under the terms in the attached INTEL-LICENSE
file. If you do not find these files, copies can be found by writing to
Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
94704. Attention: Intel License Inquiry.

The TinyOS 2.x base station that forwards packets between the UART
and radio. It replaces the GenericBase of TinyOS 1.0 and the
TOSBase of TinyOS 1.1.

On the serial link, BaseStation sends and receives simple active
messages (not particular radio packets): on the radio link, it
sends radio active messages, whose format depends on the network
stack being used. BaseStation will copy its compiled-in group ID to
messages moving from the serial link to the radio, and will filter
out incoming radio messages that do not contain that group ID.*/
BaseStation includes queues in both directions, with a guarantee that once a message enters a queue, it will eventually leave on the other interface. The queues allow the BaseStation to handle load spikes.

BaseStation acknowledges a message arriving over the serial link only if that message was successfully enqueued for delivery to the radio link.

The LEDs are programmed to toggle as follows:

- **RED Toggle**: Message bridged from serial to radio
- **GREEN Toggle**: Message bridged from radio to serial
- **YELLOW/BLUE Toggle**: Dropped message due to queue overflow in either direction

@copyright (c) 2000-2005 The Regents of the University of California.
All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation, without fee, and without written agreement is hereby granted, provided that the above copyright notice, the following two paragraphs and the author appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Copyright (c) 2002-2005 Intel Corporation
All rights reserved.

This file is distributed under the terms in the attached INTEL-LICENSE file. If you do not find these files, copies can be found by writing to Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA, 94704. Attention: Intel License Inquiry.

BaseStationP bridges packets between a serial channel and the radio. Messages moving from serial to radio will be tagged with the group ID compiled into the TOSBase, and messages moving from radio to
* serial will be filtered by that same group id.

```c
#include "AM.h"
#include "Serial.h"

module BaseStationP @safe() {
    uses {
        interface Boot;
        interface SplitControl as SerialControl;
        interface SplitControl as RadioControl;
        interface AMSend as UartSend[am_id_t id];
        interface Receive as UartReceive[am_id_t id];
        interface Packet as UartPacket;
        interface AMSend as RadioSend[am_id_t id];
        interface Receive as RadioReceive[am_id_t id];
        interface Packet as RadioPacket;
        interface Leds;
    }
    provides interface Intercept as RadioIntercept[am_id_t amid];
    provides interface Intercept as SerialIntercept[am_id_t amid];
}

implementation {
    enum {
        UART_QUEUE_LEN = 12,
        RADIO_QUEUE_LEN = 12,
    }
    message_t uartQueueBufs[UART_QUEUE_LEN];
    message_t *uartQueue[UART_QUEUE_LEN];
    uint8_t uartIn, uartOut;
    bool uartBusy, uartFull;
    message_t radioQueueBufs[RADIO_QUEUE_LEN];
    message_t *radioQueue[RADIO_QUEUE_LEN];
    uint8_t radioIn, radioOut;
    bool radioBusy, radioFull;
    task void uartSendTask();
    task void radioSendTask();
    void dropBlink() {
        call Leds.led2Toggle();
    }
    void failBlink() {
        call Leds.led2Toggle();
    }
    event void Boot.booted() {
        uint8_t i;
        for (i = 0; i < UART_QUEUE_LEN; i++)
            uartQueue[i] = &uartQueueBufs[i];
        uartIn = uartOut = 0;
        uartBusy = FALSE;
        uartFull = TRUE;
        for (i = 0; i < RADIO_QUEUE_LEN; i++)
            radioQueue[i] = &radioQueueBufs[i];
        radioIn = radioOut = 0;
        radioBusy = FALSE;
        radioFull = TRUE;
        call RadioControl.start();
        call SerialControl.start();
    }
    event void RadioControl.startDone(error_t error) {
        if (error == SUCCESS) {
            radioFull = FALSE;
        }
    }
    event void SerialControl.startDone(error_t error) {
        if (error == SUCCESS) {
            uartFull = FALSE;
        }
    }
    event void SerialControl.stopDone(error_t error) {}
    event void RadioControl.stopDone(error_t error) {}
    uint8_t count = 0;
    message_t* receive(message_t* msg, void* payload,
        uint8_t len, am_id_t id);
    event message_t *RadioSnoop.receive[am_id_t id](message_t *msg,
        void *payload, uint8_t len) {
        return receive(msg, payload, len, id);
    }
    event message_t *RadioReceive.receive[am_id_t id](message_t *msg,
        void *payload, uint8_t len) {
```
return receive(msg, payload, len, id);
}
message_t* receive(message_t *msg, void *payload, uint8_t len, am_id_t id) {
    message_t *ret = msg;
    if (!signal RadioIntercept.forward[id](msg, payload, len))
        return ret;
    atomic {
        if (uartFull)
            return ret;
        ret = uartQueue[uartIn];
        uartQueue[uartIn] = msg;
        uartIn = (uartIn + 1) % UART_QUEUE_LEN;
        if (uartIn == uartOut)
            uartFull = TRUE;
        if (!uartBusy)
            { post uartSendTask();
              uartBusy = TRUE;
            } else
            dropBlink();
    }
    uint8_t tmpLen;
task void uartSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr, src;
    message_t* msg;
    atomic
        if (uartIn == uartOut && !uartFull)
            { uartBusy = FALSE;
              return;
            }
    msg = uartQueue[uartOut];
    tmpLen = len = call RadioPacket.payloadLength(msg);
    id = call RadioAMPacket.type(msg);
    addr = call RadioAMPacket.destination(msg);
    src = call RadioAMPacket.source(msg);
    call UartAMPacket.setSource(msg, src);
    if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
        call Leds.led1Toggle();
    else{
        failBlink();
        post uartSendTask();
    }
    event void UartSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
        failBlink();
    else
        atomic
            if (msg == uartQueue[uartOut])
                { if (++uartOut >= UART_QUEUE_LEN)
                    uartOut = 0;
                    if (uartFull)
                        uartFull = FALSE;
                }
            post uartSendTask();
    }
    event message_t *UartReceive.receive[am_id_t id](message_t *msg, void *payload, uint8_t len) {
    message_t *ret = msg;
    bool reflectToken = FALSE;
    if (!signal SerialIntercept.forward[id](msg, payload, len))
        return ret;
    atomic
        if (!radioFull)
            { reflectToken = TRUE;
              ret = radioQueue[radioIn];
              radioQueue[radioIn] = msg;
              if (++radioIn >= RADIO_QUEUE_LEN)
                  radioIn = 0;
              if (radioIn == radioOut)
                  radioFull = TRUE;
              if (!radioBusy)
                  { post radioSendTask();
                    radioBusy = TRUE;
                  } else
                  dropBlink();
    }
    return receive(msg, payload, len, id);
}
task void radioSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr;
    message_t* msg;
    atomic
    if (radioIn == radioOut && !radioFull)
    {
        radioBusy = FALSE;
        return;
    }
    msg = radioQueue[radioOut];
    len = call UartPacket.payloadLength(msg);
    addr = call UartAMPacket.destination(msg);
    id = call UartAMPacket.type(msg);
    if (call RadioSend.send[id](addr, msg, len) == SUCCESS)
    {  
        call Leds.led0Toggle();
        failBlink();
        post radioSendTask();
    }
    else
    {
        failBlink();
        post radioSendTask();
    }
}

default event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
    { failBlink();
        atomic
        if (msg == radioQueue[radioOut])
        {
            if (++radioOut >= RADIO_QUEUE_LEN)
            radioOut = 0;
            if (radioFull) radioFull = FALSE;
        }
        post radioSendTask();
    }
    default event bool
    RadioIntercept.forward[am_id_t amid](message_t* msg, 
    void* payload, 
    uint8_t len) {  
    return TRUE;
    }
    default event bool
    SerialIntercept.forward[am_id_t amid](message_t* msg, 
    void* payload, 
    uint8_t len) {
    return TRUE;
    }
}

B.5 RssiBase

B.5.1 RssiBaseAppC Code

/*
 * Copyright (c) 2008 Dimas Abreu Dutra
 * All rights reserved
 */
/**
 * @author Dimas Abreu Dutra
 */
/*
 * Modified by Raul Ordonez and Huthaifa Al Issa
 * Dept. Eletrical and Computer Engineering
 * University of Dayton
 * Started: 26-Jan-2010
 * Last modified: 26-Jan-2010
 */
#include "RssiDemoMessages.h"
#include "message.h"
configuration RssiBaseAppC {
} implementation {
    components BaseStationC;
}
components RssiBaseC as App;
#ifdef __CC2420_H__
    components CC2420ActiveMessageC;
    App -> CC2420ActiveMessageC.CC2420Packet;
#else defined(PLATFORM_IRIS)
    components RF230ActiveMessageC;
    App -> RF230ActiveMessageC.PacketRSSI;
#else defined(TDA5250_MESSAGE_H)
    components Tda5250ActiveMessageC;
    App -> Tda5250ActiveMessageC.Tda5250Packet;
#endif
App-> BaseStationC.RadioIntercept[AM_RSSI_MSG];
}

B.5.2 RssiBaseC

/*@author Dimas Abreu Dutra
@date 1-Feb-2010*/

#include "RssiDemoMessages.h"

module RssiBaseC { uses interface Intercept as RssiMsgIntercept;
#ifdef __CC2420_H__
    uses interface CC2420Packet;
#else defined(TDA5250_MESSAGE_H)
    uses interface Tda5250Packet;
#else
    uses interface PacketField<uint8_t> as PacketRSSI;
#endif
}

implementation {
    uint16_t getRssi(message_t *msg);
    event bool RssiMsgIntercept.forward(message_t *msg, 
        void *payload, 
        uint8_t len) {
        RssiMsg *rssiMsg = (RssiMsg*) payload;
        // rssiMsg->rssi = getRssi(msg);
        if (rssiMsg->forwarded)
            return TRUE;
        else
            return FALSE;
    }
#ifdef __CC2420_H__
    uint16_t getRssi(message_t *msg) {
        return (uint16_t) call CC2420Packet.getRssi(msg);
    }
#else defined(CC1K_RADIO_MSG_H)
    uint16_t getRssi(message_t *msg) {
        cc1000_metadata_t *md = (cc1000_metadata_t*) msg->metadata;
        return md->strength_or_preamble;
    }
#else defined(PLATFORM_IRIS)
    uint16_t getRssi(message_t *msg) {
        if (call PacketRSSI.isSet(msg))
            return (uint16_t) call PacketRSSI.get(msg);
        else
            return 0xFFFF;
    }
#else defined(TDA5250_MESSAGE_H)
    uint16_t getRssi(message_t *msg) {
        return call Tda5250Packet.getSnr(msg);
    }
#else
    #error Radio chip not supported! This demo currently works only \ 
    for motes with CC1000, CC2420, RF230 or TDA5250 radios.
#endif
}
B.6 SendingMote

B.6.1 SendingMoteAppC

/\*[ Broadcasts messages to be received by the forwarding motes, which read
* RSSI values and then send over to the base station for recording.
* @author Raul Ordonez and Huthaifa Al Issa
* Dept. Electrical and Computer Engineering
* University of Dayton
* @date 26-Jan-2010
* Last modified: 1-Feb-2010
* 
*/
#include <Timer.h>
#include "RssiDemoMessages.h"
configuration SendingMoteAppC {
implementation {
    components MainC;
    components LedsC;
    components SendingMoteC as App;
    components new TimerMilliC() as Timer0;
    components ActiveMessageC;
    components new AMSenderC(AM_RSSIMSG);
    components new AMReceiverC(AM_RSSIMSG);
    App.Boot -> MainC;
    App.Leds -> LedsC;
    App.Timer0 -> Timer0;
    App.Packet -> AMSenderC;
    App.AMPacket -> AMSenderC;
    App.AMControl -> ActiveMessageC;
    App.AMSend -> AMSenderC;
    App.Receive -> AMReceiverC;
}
}

B.6.2 SendingMoteC

/\*[ Broadcasts messages to be received by the forwarding motes, which read
* RSSI values and then send over to the base station for recording.
* @author Raul Ordonez
* Dept. Electrical and Computer Engineering
* University of Dayton
* @date 26-Jan-2010
* Last modified: 28-Feb-2010
* 
*/
#include <Timer.h>
#include "RssiDemoMessages.h"
module SendingMoteC {
uses interface Boot;
uses interface Leds;
uses interface Timer<TMilli> as Timer0;
uses interface Packet;
uses interface AMPacket;
uses interface AMSend;
uses interface Receive;
uses interface SplitControl as AMControl;
}
implementation {
    uint16_t counter;
    message_t pkt;
    bool busy = FALSE;
    void setLeds(uint16_t val) {
        if (val & 0x01)
            call Leds.led0On();
        else
            call Leds.led0Off();
        if (val & 0x02)
            call Leds.led1On();
        else
            call Leds.led1Off();
        if (val & 0x04)
            call Leds.led2On();
        else
            call Leds.led2Off();
    }
    event void Boot.booted() {
        call AMControl.start();
    }
    event void AMControl.startDone(error_t err) {

    }

}
if (err == SUCCESS) {
    call Timer0.startPeriodic(SEND_INTERVAL_MS);
} else {
    call AMControl.start();
}

event void AMControl.stopDone(error_t err) {
}

event void Timer0.fired() {
    counter++;
    if (!busy) {
        RssiMsg* btrpkt = (RssiMsg*)(call Packet.getPayload(&pkt, sizeof(RssiMsg)));
        if (btrpkt != NULL) {
            return;
            btrpkt->sender_id = TOS_NODE_ID;
            btrpkt->counter = counter;
            btrpkt->forwarded = FALSE;
            if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(RssiMsg)) == SUCCESS) {
                busy = TRUE;
            }
        }
    }
}

event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
        busy = FALSE;
    }
}

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    if (len == sizeof(RssiMsg)) {
        RssiMsg* btrpkt = (RssiMsg*)payload;
        setLeds(btrpkt->counter);
    }
    return msg;
}
C.1 Robustness Code

C.1.1 Compute Median and Rms From Experimental Data

```matlab
%% Description
% This script compute the median and rms for configuration I and II
% Author: Huthaifa Al_Issa, Raul Ordonez,
% Date started: 8-10-2010
% Last modified: 10-11-2010
close all
clear all
clc
Huthaifa_2048
datanum = input('Data number needed: ');
for ii = 1:length(data(datanum).receiver)
    if datanum ˜= data(datanum).receiver(ii)
        inputdata = sprintf('Data%d%d',datanum,data(datanum).receiver(ii))
        StdDev = std(data(datanum).dBm(ii,:))
        meann = mean(data(datanum).dBm(ii,:))
        media = median(data(datanum).dBm(ii,:))
        Dsqr = (data(datanum).dBm(ii,:)).ˆ2;
        Root_mean = sqrt(1/length(data(1).dBm)*sum(Dsqr))
        disp('=================================')
    end
end
```

C.1.2 Huthaifa_2048 Code

```matlab
%% Description
% This script compute the robustness for configuration I and II.
% This script compute the fft for RSSI data Collect from a single experiment.
% consisting of 2048 samples,
% Author: Huthaifa Al_Issa, Raul Ordonez,
% Date started: 8-10-2010
% Last modified: 10-11-2010
clear, close all, clc
load rssi_data1
[D, cross_PLE, avg_TXtoall, data] = process_data_func(rssi_data);
N = 1024;
figure(1); clf;
hold on;
plot (data(1).dBm(1,N));grid
xlabel('Time (seconds)');
ylabel('Signal');
title('Data for Mote # (2)');
hold off;
figure(2);clf;
hold on;
plot (data(1).dBm(2,N));grid
```

380
xlabel('Time (seconds)');
ylabel('Signal');
title('Data for Mote # (3)');
hold off;
figure(3);clf;
hold on;
plot(data(1).dBm(3,1:N));grid
xlabel('Time (seconds)');
ylabel('Signal');
title('Data for Mote # (4)');
hold off;
figure(4);clf;
hold on;
plot(data(1).dBm(4,1:N));grid
xlabel('Time (seconds)');
ylabel('Signal');
title('Data for Mote # (5)');
hold off;

%% Use fft to compute the exponential Fourier series coefficients
Dn2 = fft(data(1).dBm(1,1:N)-mean(data(1).dBm(1,1:N)))/(2*N);
% need to scale by 1/(2*N) to get right values
Dn3 = fft(data(1).dBm(2,1:N)-mean(data(1).dBm(2,1:N)))/(2*N);
Dn4 = fft(data(1).dBm(3,1:N)-mean(data(1).dBm(3,1:N)))/(2*N);
Dn5 = fft(data(1).dBm(4,1:N)-mean(data(1).dBm(4,1:N)))/(2*N);

n = [-N/2 : N/2-0*1]; % range of index n
n = linspace(-.5,.5,N);
figure(5);clf;
subplot(2,1,1);
stem(n,abs(fftshift(Dn2)));
% use fftshift to get D(0) at the center of spectrum
hold on;
plot(n,abs(fftshift(Dn2)),'r--');
% plot envelope for better visualization
grid;
xlabel('n');
ylabel('|D_n|');
title('Exponential Fourier series coefficients(Dn2)');
subplot(2,1,2);
stem(n,angle(fftshift(Dn2))); % now, plot angles
hold on;
plot(n,angle(fftshift(Dn2)),'r--');
grid;
% ylim([-4 4]);
xlabel('n');
ylabel('angle D_n (rad)');
figure(6);clf;
subplot(2,1,1);
stem(n,abs(fftshift(Dn3)));
% use fftshift to get D(0) at the center of spectrum
hold on;
plot(n,abs(fftshift(Dn3)),'r--');
% plot envelope for better visualization
grid;
xlabel('n');
ylabel('|D_n|');
title('Exponential Fourier series coefficients(Dn3)');
subplot(2,1,2);
stem(n,angle(fftshift(Dn3))); % now, plot angles
hold on;
plot(n,angle(fftshift(Dn3)),'r--');
grid;
% ylim([-4 4]);
xlabel('n');
ylabel('angle D_n (rad)');
figure(7);clf;
subplot(2,1,1);
stem(n,abs(fftshift(Dn4)));
% use fftshift to get D(0) at the center of spectrum
hold on;
plot(n,abs(fftshift(Dn4)),'r--');
% plot envelope for better visualization
grid;
xlabel('n');
ylabel('|D_n|');
title('Exponential Fourier series coefficients(Dn4)');
subplot(2,1,2);
stem(n,angle(fftshift(Dn4))); % now, plot angles
hold on;
plot(n,angle(fftshift(Dn4)),'r--');
grid;
% ylim([-4 4]);
xlabel('n');
ylabel('angle D_n (rad)');

381
C.1.3 Robustness For Configuration I and II Code

%% Description
% This script compute the robustness for configuration I and II
%%
% Author: Huthaifa Al-Issa, Raul Ordonez,
% Date started: 8-10-2010
% Last modified: 10-11-2010

close all clear all
clf
rms_I_10_2 = [52.1567, 53.2752, 52.6817, 52.9821, 52.1895, 53.3020, 54.7527, 53.5966];
median_I_10_2 = [-51, -51, -51, -51, -51, -51, -51, -51];
rms_I_10_3 = [46.9246, 49.2691, 49.8293, 52.4256, 50.0050, 50.1750, 48.8601, 49.7731, 53.9702];
median_I_10_3 = [-46, -48, -49, -47, -49, -47, -49, -47, -49];
rms_I_10_4 = [62.2505, 62.9554, 62.2976, 63.8596, 63.7829, 63.5718, 62.3420, 63.1766, 64.7914];
median_I_10_4 = [-62, -63, -62, -64, -64, -62, -63, -64];
rms_I_10_5 = [59.8888, 60.6312, 59.3891, 59.8397, 59.3783, 59.1834, 59.7315, 58.5320, 58.4367];
median_I_10_5 = [-59, -59, -59, -60, -58.9, -59, -60, -58, -58];
Expnums = length(median_I_10_3);
error_I = [0.67615, 0.59429, 0.708, 0.7095, 0.70115, 0.67096, 0.2, 0.61426, 0.62821];

figure(1)
plot(1:Expnums, abs(median_I_10_2), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_I_10_2, 'rs')
title('Median and rms for I (0 to 1)'
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(2)
plot(1:Expnums, abs(median_I_10_3), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_I_10_3, 'rs')
title('Median and rms for I (0 to 2)'
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(3)
plot(1:Expnums, abs(median_I_10_4), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_I_10_4, 'rs')
title('Median and rms for I (0 to 3)'
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(4)
plot(1:Expnums, abs(median_I_10_5), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_I_10_5, 'rs')
title('Median and rms for I (0 to 4)'
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(13)
plot(1:Expnums, abs(median_I_10_2), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_I_10_2, 'rs')
title('Median and rms for I (0 to 1)'
xlabel('Experiment number');
ylabel('RSSI');
grid

%ylim([-3 6]);
xlabel('n');
ylabel('\angle D_n (rad)');
figure (13);clf;
subplot(4,1,1)
plot(data(1).dBm(1,1:N)),grid
title('Sensitivity (Dn2)');
subplot(4,1,2)
plot(data(1).dBm(2,1:N)),grid
title('Sensitivity (Dn3)');
subplot(4,1,3)
plot(data(1).dBm(3,1:N)),grid
title('Sensitivity (Dn4)');
subplot(4,1,4)
plot(data(1).dBm(4,1:N)),grid
title('Sensitivity (Dn5)');
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(4)
plot(1:Expnums, abs(median_I_10_5), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_I_10_5, 'rs')
title('Median and rms for I (0 to 4)');
legend('Median', 'RMS')
xlabel('Experiment number');
ylabel('RSSI');
grid

rms_II_10_2 = [51.5447, 51.5499, 52.2440, 51.4922, 50.6998, 51.3384, 49.5458, 49.0316, 49.7152];
median_II_10_2 = [-51, -51, -51, -50, -50, -49, -49, -49];

rms_II_10_3 = [54.9865, 54.9856, 54.9335, 56.4752, 56.9874, 58.3577, 56.3018, 55.6403, 57.2794];
median_II_10_3 = [-64, -64, -64, -61, -61, -61, -61, -61];

rms_II_10_4 = [55.3407, 61.1896, 63.9805, 59.5341];
median_II_10_4 = [-70, -70, -70, -69, -69, -69, -69, -69];

rms_II_10_5 = [55.3407, 61.1896, 63.9805, 59.5341];
median_II_10_5 = [-55, -55, -55, -55, -55, -55, -55, -55];

error_II = [0.89564, 2.7948, 1.0245, 3.1446, 4.1013, 0.41865, 0.50343, 1.0101, 0.34061];

figure(5)
plot(1:Expnums, abs(median_II_10_2), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_II_10_2, 'rs')
title('Median and rms for II (0 to 1)');
legend('Median', 'RMS')
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(6)
plot(1:Expnums, abs(median_II_10_3), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_II_10_3, 'rs')
title('Median and rms for II (0 to 2)');
legend('Median', 'RMS')
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(7)
plot(1:Expnums, abs(median_II_10_4), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_II_10_4, 'rs')
title('Median and rms for II (0 to 3)');
legend('Median', 'RMS')
xlabel('Experiment number');
ylabel('RSSI');
grid

figure(8)
plot(1:Expnums, abs(median_II_10_5), 'bo', 'linewidth', 3)
hold on
plot(1:Expnums, rms_II_10_5, 'rs')
title('Median and rms for II (0 to 4)');
legend('Median', 'RMS')
xlabel('Experiment number');
ylabel('RSSI');
grid

FC1 = (rms_I_10_2 + abs(median_I_10_2))./rms_I_10_2.^2;
FC2 = (rms_I_10_3 + abs(median_I_10_3))./rms_I_10_3.^2;
FC3 = (rms_I_10_4 + abs(median_I_10_4))./rms_I_10_4.^2;
FC4 = (rms_I_10_5 + abs(median_I_10_5))./rms_I_10_5.^2;

figure(9)
plot(1:Expnums, FC1, 'r-', 'linewidth', 3)
hold on
plot(1:Expnums, FC1, 'bo', 'linewidth', 3)
title('Configuration I (0 to 1)');
xlabel('Experiment number');
ylabel('Robustness');
grid

figure(10)
plot(1:Expnums, FC2, 'r-', 'linewidth', 3)
hold on
plot(1:Expnums, FC2, 'bo', 'linewidth', 3)
title('Configuration II (0 to 2)');
xlabel('Experiment number');
ylabel('Robustness');
grid
figure(11)
plot(1:Expnums,FC3,'r-','linewidth',3)
hold on
plot(1:Expnums,FC3,'bo','linewidth',3)
title('Configuration I (0 to 3)'
xlabel('Experiment number');
ylabel('Robustness');
grid
figure(12)
plot(1:Expnums,FC4,'r-','linewidth',3)
hold on
plot(1:Expnums,FC4,'bo','linewidth',3)
title('Configuration I (0 to 4)'
xlabel('Experiment number');
ylabel('Robustness');
grid
FE1 = (rms_II_10_2 + abs(median_II_10_2))/rms_II_10_2^2;
FE2 = (rms_II_10_3 + abs(median_II_10_3))/rms_II_10_3^2;
FE3 = (rms_II_10_4 + abs(median_II_10_4))/rms_II_10_4^2;
FE4 = (rms_II_10_5 + abs(median_II_10_5))/rms_II_10_5^2;
figure(13)
plot(1:Expnums,FE1,'r-','linewidth',3)
hold on
plot(1:Expnums,FE1,'bo','linewidth',3)
title('Configuration II (0 to 1)'
xlabel('Experiment number');
ylabel('Robustness');
grid
figure(14)
plot(1:Expnums,FE2,'r-','linewidth',3)
hold on
plot(1:Expnums,FE2,'bo','linewidth',3)
title('Configuration II (0 to 2)'
xlabel('Experiment number');
ylabel('Robustness');
grid
figure(15)
plot(1:Expnums,FE3,'r-','linewidth',3)
hold on
plot(1:Expnums,FE3,'bo','linewidth',3)
title('Configuration II (0 to 3)'
xlabel('Experiment number');
ylabel('Robustness');
grid
figure(16)
plot(1:Expnums,FE4,'r-','linewidth',3)
hold on
plot(1:Expnums,FE4,'bo','linewidth',3)
title('Configuration II (0 to 4)'
xlabel('Experiment number');
ylabel('Robustness');
RC = FC1 + FC2 + FC3 + FC4;
figure(17)
plot(1:Expnums,RC,'r-','linewidth',3)
hold on
plot(1:Expnums,RC,'bo','linewidth',3)
title('Robustness for whole configuration I'
xlabel('Experiment number');
ylabel('Robustness');
grid
RE = FE1 + FE2 + FE3 + FE4;
figure(18)
plot(1:Expnums,RE,'r-','linewidth',3)
hold on
plot(1:Expnums,RE,'bo','linewidth',3)
title('Robustness for whole configuration II'
xlabel('Experiment number');
ylabel('Robustness');
grid
figure(19)
hold on
plot(RC,'k-','linewidth', 3)
plot(RC, 'bo','linewidth', 3)
plot(RE,'r--','linewidth', 3)
plot(RE, 'bo','linewidth', 3)
legend('R-I','exp-#','R-II','exp-#')
title('Robustness for configurations I and II')
xlabel('Experiment Number');
ylabel('Robustness');
grid
figure(20)
hold on
grid
plot(new_metric_I_median,'ko-','linewidth', 2.5)
plot(error_I, 'gs--','linewidth', 2.5)
plot(new_metric_II_median,'bo-','linewidth', 2.5)
plot(error_II, 'rs--','linewidth', 2.5)
legend('Metric(I)','Error(I)','Metric(II)','Error(II)')
title('Position error and performance metric for I and II')
xlabel('Experiment number')
ylabel('Metric and error')

figure(21)
hold on
grid
plot(new_metric_I_median,'ko-','linewidth', 3)
plot(error_I, 'gs--','linewidth', 3)
legend('Metric_I','Error_I')
title('Metric and Error for I')
xlabel('Experiment number')
ylabel('Metric & Error values')

figure(22)
hold on
grid
plot(new_metric_II_median,'bo-','linewidth', 3)
plot(error_II, 'rs--','linewidth', 3)
legend('Metric_II','Error_II')
title('Metric and Error for II')
xlabel('Experiment number')
ylabel('Metric & Error')
APPENDIX D

EXTREMUM SEEKING CONTROL

D.1 Extremum Seeking Control Code

D.1.1 Without Obstruction Code

```matlab
% M-member n-dimensional CT-swarm
% The agents have point mass dynamics
% Extremum seeking control is used to force the vehicle converge
% to the minimum of the position
% The algorithm uses the positions of all the other
% members to update its position
% by Hutahifa Al_Issa and Raul Ordonez
clear all; close all;clc;
global M dt time k;
global a c;
global w ;
epsilon = 0.1;
global regsize regsizemin regnum flag;
% obstacleflag=0;
% parameter of swarming
M = 4;
a = 1;
c = sqrt(2)*a;  %3.3,3.2, 2
%sqrt(2.5)*c/2.5;
%b = 2/5;
% parameter of trust region
if flag==0
    regsize(1:M)=1*ones(1,M);
end
% if flag==1
% regsize(1:M)=1*ones(1,M);
% end
regsizemin=0.1;
regnum=2;
if flag==0
    % Potential function weight
    w=2;
    % Controller gain
    k=5;
    % optimizier interval
    dt = 0.005;
end
x0 = [0 -1 0 1 0; 0 0 -1 0 2.5; zeros(1, M+1)];
% x0 = [0 -1 0 rand(1,1) 0; 0 0 -1 rand(1,1) 2.5; zeros(1, M+1)];
% mote 4 is random position
% Ode solver
t0 = 0.0;
tf = 50;
time=0;
tspan = [t0 tf];
myoptions = odeset('OutputFcn','odeplot');
[t, x] = ode15s('extremumswarming_no2_derivative', tspan, x0, myoptions);
% position of vehicles
r=length(x);
x1=x(1:r,1);
y1=x(1:r,2);
x2=x(1:r,4);
y2=x(1:r,5);
```

386
\[ x_3 = x(1:r, 7); \]
\[ y_3 = x(1:r, 8); \]
\[ x_4 = x(1:r, 10); \]
\[ y_4 = x(1:r, 11); \]
\[ x_t = x(1:r, 13); \]
\[ y_t = x(1:r, 14); \]

\% Potential Function
\[ \text{den} = 2; \]
\[ \text{for } iii = 1 : \text{length}(t) \]
\[ \quad J(iii) = \text{metric}\_\text{potential}(x_4(iii), y_4(iii), x_1(iii), y_1(iii), \]
\[ \quad \quad \quad \quad x_2(iii), y_2(iii), x_3(iii), y_3(iii), x_t(iii), y_t(iii)) ; \]
\[ \quad J_{a(iii)} = 1 / (\epsilon + \text{norm}(x_4(iii) - x_1(iii)) + \text{norm}(x_4(iii) - x_2(iii)) + \text{norm}(x_4(iii) - x_3(iii))) ; \]
\[ \text{end} \]

\% plot simulation results
\% source seeking, formation control and obstacle avoidance
figure;
plot(x(1), y(1), 'o', ...
\[ \quad x_3(1), y_3(1), 'o', ... \]
\[ \quad x_2(1), y_2(1), 'o', ... \]
\[ \quad x_4(1), y_4(1), 'o', ... \]
\[ \quad 'LineWidth', 2, 'MarkerEdgeColor', 'b', ...
\[ \quad 'MarkerFaceColor', 'g', ... \]
\[ \quad 'MarkerSize', 12); \]
hold on;
plot(x(1:t), y(1:t), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerEdgeColor', 'b', ...
\[ \quad 'MarkerFaceColor', 'g', ... \]
\[ \quad 'MarkerSize', 12); \]
hold off;
plot(x(1), y(1), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerEdgeColor', 'b', ...
\[ \quad 'MarkerFaceColor', 'g', ... \]
\[ \quad 'MarkerSize', 12); \]
hold on;
plot(x(1), y(1), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerEdgeColor', 'b', ...
\[ \quad 'MarkerFaceColor', 'g', ... \]
\[ \quad 'MarkerSize', 12); \]
end;

\% Second Figure
hold on;
xlabel('x', 'FontSize', 18, 'FontWeight', 'bold');
ylabel('y', 'FontSize', 18, 'FontWeight', 'bold');
title('Swarm Trajectories', 'FontSize', 18, 'FontWeight', 'bold');
grid on
if flag == 0
saveas(gcf, 'swarmpath_noasmesc.eps', 'psc2');
end
axis tight
mov = avifile('movie_swarm_no_ob.avi', 'fps', 3);
for i = 1:length(t)
clf;
figure(2);
plot(x(1), y(1), 'o', ...
\[ \quad x_3(1), y_3(1), 'o', ... \]
\[ \quad x_4(1), y_4(1), 'o', ... \]
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
hold on;
plot(x(1), y(1), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
plot(x(1:t), y(1:t), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
if i == length(t)
plot(x(r), y(r), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
end
end

if flag == 0
saveas(gcf, 'swarmpath_noasmesc2.eps', 'psc2');
end
axis tight
mov = avifile('movie_swarm_no_ob.avi', 'fps', 3);
for i = 1:length(t)
clf;
figure(2);
plot(x(1), y(1), 'o', ...
\[ \quad x_3(1), y_3(1), 'o', ... \]
\[ \quad x_4(1), y_4(1), 'o', ... \]
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
hold on;
plot(x(1), y(1), 'o', ...
\[ \quad x_3(1), y_3(1), 'o', ... \]
\[ \quad x_4(1), y_4(1), 'o', ... \]
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
if i == length(t)
plot(x(r), y(r), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
end
end

if flag == 0
saveas(gcf, 'swarmpath_noasmesc3.eps', 'psc2');
end
axis tight
mov = avifile('movie_swarm_no_ob.avi', 'fps', 3);
for i = 1:length(t)
clf;
figure(2);
plot(x(1), y(1), 'o', ...
\[ \quad x_3(1), y_3(1), 'o', ... \]
\[ \quad x_4(1), y_4(1), 'o', ... \]
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
hold on;
plot(x(1), y(1), 'o', ...
\[ \quad x_3(1), y_3(1), 'o', ... \]
\[ \quad x_4(1), y_4(1), 'o', ... \]
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
if i == length(t)
plot(x(r), y(r), 'o', ...
\[ \quad 'LineWidth', 2, 'MarkerFaceColor', 'b', ...
\[ \quad 'MarkerSize', 12); \]
end
end

if flag == 0
saveas(gcf, 'swarmpath_noasmesc4.eps', 'psc2');
end
axis tight
windowsystem('mono');
D.1.2 Extremumswarming_No2_Derivative Code

function xdot = extremumswarming_no2_derivative(t, x)
    global M xp0 yp0 xs ys dt time k;
    global a b c;
    global w;
    global regsize regsizemin regnum;
    % variable matrix
    $ \mathbf{A} \mathbf{x} = \mathbf{x}_t$
    $\mathbf{x}_t, \mathbf{y}_t$  
    epsilon=0.1;
    sen = 3;
    z = zeros(sen, M+1);
    zdot = zeros(sen, M+1);
    for i=1:M+1,
        z(:, i) = x(sen*(i-1)+1:sen*i);
    end
    e = e4;
    for i = 1 : length(t)
        metric4 = compute_metric(x4(i),y4(i),x1(i),y1(i),x2(i),y2(i),x3(i),y3(i),xt(i),yt(i));
        end
    end
    metric = metric4;
    plot (t, metric, 'linewidth', 3)
    legend('J_{metric}')
    xlabel('t','FontSize',18,'FontWeight','bold');
    title('J_{metric}','FontSize',18,'FontWeight','bold');
    axis tight
    grid;
    figure
    plot (t, e, 'linewidth', 3)
    legend('e')
    xlabel('t','FontSize',18,'FontWeight','bold');
    title('e','FontSize',18,'FontWeight','bold');
    axis tight
    grid;
end
% keyboard
% target
xt=tar[1];
yt=tar[2];
tar_dot(1)=-.25*0;%0.25;
tar_dot(2)=-sin(0.25*t)*0;%sin(0.25*t);
for i=1:M
xp(i)=z(1,i);%
yp(i)=z(2,i);
eta(i)=z(3,i);
end
if t==0 | t-time>=dt
% Direct Search
xtemp=linspace(-0.5*regsize(1)+xp(1),0.5*regsize(1)+xp(1),regnum);
ytemp=linspace(-0.5*regsize(2)+yp(1),0.5*regsize(2)+yp(1),regnum);
[X1,Y1]=meshgrid(xtemp,ytemp);
xtemp=linspace(-0.5*regsize(2)+xp(2),0.5*regsize(2)+xp(2),regnum);
ytemp=linspace(-0.5*regsize(2)+yp(2),0.5*regsize(2)+yp(2),regnum);
[X2,Y2]=meshgrid(xtemp,ytemp);
xtemp=linspace(-0.5*regsize(3)+xp(3),0.5*regsize(3)+xp(3),regnum);
ytemp=linspace(-0.5*regsize(3)+yp(3),0.5*regsize(3)+yp(3),regnum);
[X3,Y3]=meshgrid(xtemp,ytemp);
% keyboard
xtemp=linspace(-0.5*regsize(4)+xp(4),0.5*regsize(4)+xp(4),regnum);
ytemp=linspace(-0.5*regsize(4)+yp(4),0.5*regsize(4)+yp(4),regnum);
[X4,Y4]=meshgrid(xtemp,ytemp);
J4=metric_potential(X4,Y4,xp(1),yp(1),xp(2),yp(2),xp(3),yp(3),xt,yt);
J4c=metric_potential(xp(4),yp(4),xp(1),yp(1),xp(2),yp(2),xp(3),yp(3),xt,yt);
[J4_min,J4_row,J4_col] = min2d(J4);
% Update as compass search
if J4_min==J4c
regsize(4)=max(regsize(4)/2,regsizemin);
end
xs(4)=X4(J4_row,J4_col);
ys(4)=Y4(J4_row,J4_col);
xp0=xp;
yp0=yp;
time=t;
end
% for i=1:M
% vx(i)=-k *(xp(i)-xs(i));
% vy(i)=-k *(yp(i)-ys(i));
% end
% for i=1:M
% if i==4
% vx(i)=-k*(xp(i)-xs(i));
% vy(i)=-k*(yp(i)-ys(i));
% else
% vx(i)=0;vy(i)=0;
% end
% for i=1:M
% vx(i)=-(1/dt) *(xp(i)-xs(i));
% vy(i)=-(1/dt) *(yp(i)-ys(i));
% end
% for i=1:M
% vx(i)=-(1/dt)*sign(xp(i)-xs(i));
% vy(i)=-(1/dt)*sign(yp(i)-ys(i));
% end
% vehicle dynamics (pointmass + high pass filter)
for i=1:M
% velocity, high pass filter
zdot(i+1:i+1) = [vx(i);vy(i);0];
end;
% Target dynamics
zdot(1:2, M+1) = tar_dot(1:);
% Return a column vector
for i=1:M+1
xdt(i+1:i+1)=zdot(:, i);
end;
xdt = xdt'; %

D.1.3 Min2d Code

function [Y, I_row, I_col] = min2d(X)
% [Y, I_row, I_col] = min2d(X)
% X Input 2D array
% Y The minimum point in the 2D array
Finding the minimum in the 2D array
NaN's are ignored when computing the minimum.
returns the indices of the minimum value in I_row, I_col

Author: Chunlei Zhang
University of Dayton
02/15/2002

[temp_col,I]=min(X,[],1); %Finding the minimum along the column
[Y,II]=min(temp_col); %Finding the final minimum in the temp 1d array
I_row=I(II);
I_col=II;

D.1.4 Metric Potential Code

function J = metric_potential(x1,y1,x2,y2,x3,x4,y3,y4,xt,yt)
global a c w wo;
global flag;
den = 2;
epsilon = 0.1;
[m,n] = size(x1); % or size(y1) because x1 and y1 have the same size
for dim1 = 1 : m
    for dim2 = 1 : n
        Rec = [x1(dim1,dim2), x2, x3, x4; y1(dim1,dim2), y2, y3, y4];
        Tran=[xt; yt];
        CPLE = [-5.2 -5.8 -6.6 -5.8 -6.4 -5.5 -6.8 -5.5924 -7.5 -6.7];
        PLE(1,2) = CPLE(1); PLE(2,1) = CPLE(1);
        PLE(1,3) = CPLE(2); PLE(3,1) = CPLE(2);
        PLE(1,4) = CPLE(3); PLE(4,1) = CPLE(3);
        PLE(1,5) = CPLE(4); PLE(5,1) = CPLE(4);
        PLE(2,3) = CPLE(5); PLE(3,2) = CPLE(5);
        PLE(2,4) = CPLE(6); PLE(4,2) = CPLE(6);
        PLE(2,5) = CPLE(7); PLE(5,2) = CPLE(7);
        PLE(3,4) = CPLE(8); PLE(4,3) = CPLE(8);
        PLE(3,5) = CPLE(9); PLE(5,3) = CPLE(9);
        PLE(4,5) = CPLE(10); PLE(5,4) = CPLE(10);
        Rec2 = [xt, x1(dim1,dim2), x2, x3, x4; yt, y1(dim1,dim2), y2, y3, y4];
        for ii = 1:5
            for jj = 1:5
                if jj == ii
                    continue;
                else
                    p1 = Rec2(:,ii);
                    p2 = Rec2(:,jj);
                    di = norm(p2 - p1);
                    dBm(ii,jj) = -32 + (PLE(ii,jj)) + 10*log10(di);
                end
            end
        end
        CPLE2 = CPLE(5:10);
        T = Rec';
        fudge = 0.8;
        x_axis = [-6 6]; y_axis = [-6 6];
        t1=1:t2=1;
        for j=1:6 % 6 distance combinations for 4 motes
            t2=t2+1;
        end
        sizeT(j) = norm(T(t2,:)-T(t1,:));
        if t2==4
            t2=t1+1;
        end
        end
        max_sep = max(sizeT)-min(sizeT); % not currently used
        a=2.2180653270302679e-15;
b=8.30384930269744;
c=2.361192808137663;
j = 1; % only look at TX to receiver data
        for k = 2:5
            D(k-1,1) = mean( a*dBm(k,j)' + b + c);
        end
        mean(sizeT)*fudge;
        RSSI_edge = transpose(dBm(2:5,1)); %avg_TXtoall(2,1)
discard=0;
        for j=1:4 %
            if D(j)>max_sep*fudge;
                if D(j)>mean(sizeT)*fudge;
                    D(j)=D(j)*0.75;
                end
            end
            %
        end
        %
40
if D(j) > x_axis(2) - x_axis(1) | D(j) > y_axis(2) - y_axis(1)
% At least one distance estimate is too large, so discard experiment
discard=1;
end
cross_PLE = CPLE;
distance_weight = mean(cross_PLE)./cross_PLE;
A.mat = [ 2*(T(2,1)-T(1,1)) 2*(T(2,2)-T(1,2))
2*(T(3,1)-T(1,1)) 2*(T(3,2)-T(1,2))
2*(T(4,1)-T(1,1)) 2*(T(4,2)-T(1,2))
2*(T(1,1)-T(4,1)) 2*(T(1,2)-T(4,2)) ];
A.mat_wgt(:,1) = A.mat(:,1).*distance_weight(:,1);
A.mat_wgt(:,2) = A.mat(:,2).*distance_weight(:,1);
B.mat = -[(T(1,1)^2-T(2,1)^2)+(T(1,2)^2-T(2,2)^2)+(D(2)^2-D(1)^2)
(T(1,1)^2-T(3,1)^2)+(T(1,2)^2-T(3,2)^2)+(D(3)^2-D(1)^2)
(T(1,1)^2-T(4,1)^2)+(T(1,2)^2-T(4,2)^2)+(D(4)^2-D(1)^2)
(T(2,1)^2-T(3,2)^2)+(T(2,2)^2-T(3,2)^2)+(D(3)^2-D(2)^2)
(T(2,1)^2-T(4,2)^2)+(T(2,2)^2-T(4,2)^2)+(D(4)^2-D(2)^2)
(T(3,1)^2-T(4,1)^2)+(T(3,2)^2-T(4,2)^2)+(D(4)^2-D(3)^2)];
C_hat = real((A.mat_wgt' * A.mat_wgt) \ A.mat_wgt' * B.mat);
for j=1:size(A.mat,1)
  mtrc_distance_weight(j) = norm(A.mat(j,:),2);
end
% compute distance from each mote to estimate
for j=1:4
  D_est(j) = norm(T(j,:) - C_hat');
end
% compute distance from each mote to estimate
D_est(j) = norm(T(j,:) - C_hat');
maxD = max(abs(D - D_est'));
minD = min(abs(D - D_est'));
mean_cross_PLE = mean(cross_PLE);
% keyboard
std_cross_PLE_wgt = std(mtrc_distance_weight .* cross_PLE);
metric(dim1, dim2) = 3*(maxD./(10+minD)) + 2./abs(mean_cross_PLE).^(1/2) + 1./(std_cross_PLE_wgt).^(1/2);
Ja(dim1, dim2) = 1 / (epsilon + norm([x1(dim1, dim2); y1(dim1, dim2)] - [x2; y2]) + norm([x1(dim1, dim2); y1(dim1, dim2)] - [x3; y3]) + norm([x1(dim1, dim2); y1(dim1, dim2)] - [x4; y4]));
end
end
if flag==0
% potential function for single vehicle
J_metric = w * Ja / 2;
end

D.1.5 Localization_Error Code

function error = localization_error(x1, y1, x2, y2, x3, y3, x4, y4, xt, yt)
[m, n] = size(x1); % or size(y1) because x1 and y1 have the same size
for dim1 = 1 : m
  for dim2 = 1 : n
    Rec = [x1(m, n), x2, x3, x4; y1(m, n), y2, y3, y4];
    % Set up transmitter [x;y]
    Tran = [xt; yt];
    CPLE = [-5.2 -5.8 -6.6 -5.8 -6.4 -6.5 -6.8 -5.5924 -7.5 -6.7];
    PLE(1,2) = CPLE(1); PLE(2,1) = CPLE(1);
    PLE(1,3) = CPLE(2); PLE(3,1) = CPLE(2);
    PLE(1,4) = CPLE(3); PLE(4,1) = CPLE(3);
    PLE(1,5) = CPLE(4); PLE(5,1) = CPLE(4);
    PLE(2,3) = CPLE(5); PLE(3,2) = CPLE(5);
    PLE(2,4) = CPLE(6); PLE(4,2) = CPLE(6);
    PLE(2,5) = CPLE(7); PLE(5,2) = CPLE(7);
    PLE(3,4) = CPLE(8); PLE(4,3) = CPLE(8);
    PLE(3,5) = CPLE(9); PLE(5,3) = CPLE(9);
    PLE(4,5) = CPLE(10); PLE(5,4) = CPLE(10);
    Rec2 = [xt, x1(m, n), x2, x3, x4; ...]
    yr, y1(m, n), y2, y3, y4];
    for ii = 1:5
      for jj = 1:5
        if jj == ii
          continue
        else
          p2 = Rec2(:, ii);
        end
dBm(ii, jj) = -32 + (PLE(ii, jj)) * 10*log10(di);
end
391
D.1.6 Compute Metric Code

```matlab
function metric = compute_metric(x1, y1, x2, y2, x3, y3, x4, y4, xt, yt)
global a c w wo;
global flag;
% J1t=(1/den) * ((x1-xt).^2+(y1-xt).^2)-a^2).
epsilon= 0.1;
[m,n] = size(x1); % or size(y1) because x1 and y1 have the same size
for dim1 = 1 : m
    for dim2 = 1 : n
        Rec = [x1(dim1,dim2), x2, x3, x4; ...]
        Tran=[xt; yt];
        CPLE = [-5.2 -5.8 -6.6 -5.8 -6.4 -6.5 -6.8 -5.5924 -7.5 -6.7];
PLE(1,2) = CPLE(1); PLE(2,1) = CPLE(1);
PLE(1,3) = CPLE(2); PLE(3,1) = CPLE(2);
PLE(1,4) = CPLE(3); PLE(4,1) = CPLE(3);
PLE(1,5) = CPLE(4); PLE(5,1) = CPLE(4);
PLE(2,3) = CPLE(5); PLE(3,2) = CPLE(5);
PLE(2,4) = CPLE(6); PLE(4,2) = CPLE(6);
PLE(2,5) = CPLE(7); PLE(5,2) = CPLE(7);
```
PLE(3,4) = CPLE(8); PLE(4,3) = CPLE(8);
PLE(3,5) = CPLE(9); PLE(5,3) = CPLE(9);
PLE(4,5) = CPLE(10); PLE(5,4) = CPLE(10);
Rec2 = [xt, x1(dim1,dim2), x2, x3, x4;...
yt, y1(dim1,dim2), y2, y3, y4];
for ii = 1:5
p1 = Rec2(:,ii);
%
if ii ˜= 1
%
p1 = Rec2(:,ii);
%
else
%
p1 = Tran;
%
end
for jj = 1:5
if jj == ii
continue
else
p2 = Rec2(:,jj);
di = norm(p2 - p1);
dBm(ii,jj) = -32 + (PLE(ii,jj))*10*log10(di);
end
end
end
CPLE2 = CPLE(5:10);
T = Rec’;
fudge = 0.8;
x_axis = [-6 6]; y_axis = [-6 6];
t1=1;t2=1;
for j=1:6
% 6 distance combinations for 4 motes
if t2<4
t2=t2+1;
end
sizeT(j) = norm(T(t2,:)-T(t1,:));
if t2==4
t2=t1+1;
t1=t1+1;
end
end
max_sep = max(sizeT)-min(sizeT);
% not currently used
% D’
%load dist_model;
a=2.218065227302679e-15;
b=8.303884930269744;
c=2.361192808137663;
j = 1;
% only look at TX to receiver data
for k = 2:5
D(k-1,1) = mean( a*dBm(k,j)ˆb + c);
end
mean(sizeT)*fudge;
RSSI_edge = transpose(dBm(2:5,1)); %avg_TXtoall(2,1)
discard=0;
for j=1:4
%
if D(j)>max_sep*fudge;
if D(j)>mean(sizeT)*fudge;
D(j)=D(j)*0.75;
end
if D(j) > x_axis(2)-x_axis(1) | D(j) > y_axis(2)-y_axis(1)
% At least one distance estimate is too large, so discard experiment
discard=1;
end
end
cross_PLE = CPLE2;
distance_weight = mean(cross_PLE)./cross_PLE;
A.mat = [ 2*(T(2,1)-T(1,1)) 2*(T(2,2)-T(1,2))
2*(T(3,1)-T(1,1)) 2*(T(3,2)-T(1,2))
2*(T(4,1)-T(1,1)) 2*(T(4,2)-T(1,2))
2*(T(3,1)-T(2,1)) 2*(T(3,2)-T(2,2))
2*(T(4,1)-T(2,1)) 2*(T(4,2)-T(2,2))
2*(T(4,1)-T(3,1)) 2*(T(4,2)-T(3,2)) ];
A.mat_wgt(:,1) = A.mat(:,1).*distance_weight(:,1);
A.mat_wgt(:,2) = A.mat(:,2).*distance_weight(:,1);
B.mat = -[ (T(1,1)ˆ2-T(2,1)ˆ2)+(T(1,2)ˆ2-T(2,2)ˆ2)+(D(2)ˆ2-D(1)ˆ2)
(T(1,1)ˆ2-T(3,1)ˆ2)+(T(1,2)ˆ2-T(3,2)ˆ2)+(D(3)ˆ2-D(1)ˆ2)
(T(1,1)ˆ2-T(4,1)ˆ2)+(T(1,2)ˆ2-T(4,2)ˆ2)+(D(4)ˆ2-D(1)ˆ2)
(T(2,1)ˆ2-T(3,1)ˆ2)+(T(2,2)ˆ2-T(3,2)ˆ2)+(D(3)ˆ2-D(2)ˆ2)
(T(2,1)ˆ2-T(4,1)ˆ2)+(T(2,2)ˆ2-T(4,2)ˆ2)+(D(4)ˆ2-D(2)ˆ2)
(T(3,1)ˆ2-T(4,1)ˆ2)+(T(3,2)ˆ2-T(4,2)ˆ2)+(D(4)ˆ2-D(3)ˆ2) ];
C_hat = real((A.mat_wgt’*A.mat_wgt)\A.mat_wgt’*B.mat);
for j=1:size(A.mat,1)
mtrc_distance_weight(j) = norm(A.mat(j,:)/2);
end
% compute distance from each mote to estimate
for j=1:4
D_est(j)=norm(T(j,:)-C_hat’);
end
maxD=max(abs(D-D_est’));
minD=min(abs(D-D_est’));
mean_cross_PLE=mean(cross_PLE);

393


std_cross_PLE_wgt = std(mtrc_distance_weight.*cross_PLE);
metric(dim1,dim2) = 3*(maxD./(10+minD)).*(1/2)+2./abs(mean_cross_PLE).*(1/2)+
1./(std_cross_PLE_wgt).*(1/1);
end
end