Interface and Performance Analysis of a Local Area Differential GPS
VHF Navigation Augmentation Broadcast System

A Thesis Presented to
The Faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirement for the Degree
Master of Science

by
Yao-Bang Steven Yang
June, 1997
Acknowledgment

The research presented in this thesis was supported by the Federal Aviation Administration (FAA) Office of Spectrum Policy and Management through TASC subcontract 00072-93062 and jointly funded by the FAA Satellite Program Office under University Research Grant 92-G-023.

A great thanks to Dr. Trent A. Skidmore, Assistant Professor of Electrical Engineering and Computer Science for the acceptance as the author’s advisor in the graduate program and support throughout.

The author wishes to extend his fullest gratitude to Dr. Frank van Graas, Associate Professor of Electrical Engineering and Computer Science, Mr. Maarten Uijt De Haag and Mr. David Quinet with the Avionics Engineering Center. Their help and remarks were vital to the completion of the research.

The author also wishes to acknowledge the following individuals for their assistance and contributions to this work:

Dr. Michael Braasch, Assistant Professor of Electrical Engineering and Computer Science, Dr. Frank van Graas, Associate Professor of Electrical Engineering and Computer Science, Dr. Larry Snyder, Professor of Mathematics are thanked for their help in reviewing this thesis and serving as members of the thesis defense. Mr. Larry Montgomery with the Raytheon Aircraft Montek company for his valuable suggestions.

Special thanks is given to the entire staff of the Avionics Engineering Center for their support, encouragement, hospitality, and making the author feel at home.

The author would like to express his profound appreciation of his family for their love, support and encouragement in all these years.
# TABLE OF CONTENTS

List of Tables

List of Figures

1. Introduction

2. Fundamentals of DGNSS and the VHF Data Link system
   2.1 GNSS characteristics
   2.2 NABS VDL characteristics
      2.2.1 Message Format and Pre-transmit Processing
      2.2.2 Carrier Frequency Range
      2.2.3 Power Budget
   2.3 Media Access Sublayer Using Time Division Multiple Access (TDMA)

3. Implementation of the VDL System
   3.1 System Set Up and Interfacing
      3.1.1 Interfacing by RS-232
      3.1.2 Synchronization of data packets in VDL -- 1 PPS
   3.2 Software implementation
      3.2.1 Transmitter Control Program
      3.2.2 Receiver Control Program

4. VDL System Performance Evaluation
   4.1 VDL System Laboratory Performance
      4.1.1 VDL Receiver Indicated Power
      4.1.2 Message Loss Rate and Receiver Sensitivity
      4.1.3 Received Message FEC Function

iii

iv

1

3

3

6

6

11

11

11

14

14

19

21

22

22

28

32

32

35

36

39
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>VDL message format</td>
<td>7</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Data encoding</td>
<td>10</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>VDL system power budget</td>
<td>34</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>VDL power inputs</td>
<td>34</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Received messages indicated average power</td>
<td>35</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Indicated header FEC by one time slot transmission</td>
<td>42</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Indicated header FEC by all time slot transmission</td>
<td>42</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Indicated application data FEC by one time slot transmission</td>
<td>42</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>Indicated application data FEC by all time slots transmission</td>
<td>43</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Local Area Differential GPS</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Message encoding block diagram</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Scrambler/Descrambler shift register block diagram</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>DPSK data modulation block diagram</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>VDL TDMA timing structure</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>VHF data link set up</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>PCB layout for voltage level converter</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>Characteristic of transceiver chips response to 1 PPS</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Null MODEM cable assignment</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>RS-232 interface assignment in the VDL system</td>
<td>21</td>
</tr>
<tr>
<td>3.6</td>
<td>VDL application data format for the VDL system bench test</td>
<td>24</td>
</tr>
<tr>
<td>3.7</td>
<td>Event chart of the transmitter control program host in QNX CPU</td>
<td>25</td>
</tr>
<tr>
<td>3.8</td>
<td>TDMA timing structure and data send time in QNX CPU</td>
<td>26</td>
</tr>
<tr>
<td>3.9</td>
<td>Flow chart of the VDL transmitter control program</td>
<td>27</td>
</tr>
<tr>
<td>3.10</td>
<td>Event chart of the receiver control program host in MS-DOS CPU</td>
<td>30</td>
</tr>
<tr>
<td>3.11</td>
<td>Flow chart of the VDL receiver control program</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>VDL system physical connections for laboratory bench test</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Message received by different power level (one time slot)</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Message received by different power level (all time slot)</td>
<td>37</td>
</tr>
<tr>
<td>4.4</td>
<td>Message loss rate using application data FEC = 4 (one time slot)</td>
<td>38</td>
</tr>
<tr>
<td>4.5</td>
<td>Message loss rate using application data FEC = 4 (all time slots)</td>
<td>38</td>
</tr>
<tr>
<td>4.6</td>
<td>Header FEC distribution by one time slot transmission</td>
<td>40</td>
</tr>
<tr>
<td>4.7</td>
<td>Header FEC distribution by all time slot transmission</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>Application data FEC distribution by one time slot transmission</td>
<td>41</td>
</tr>
<tr>
<td>4.9</td>
<td>Application data FEC distribution by all time slot transmission</td>
<td>41</td>
</tr>
<tr>
<td>4.10</td>
<td>Received data FEC performance at -50 dBm (one time slot)</td>
<td>44</td>
</tr>
<tr>
<td>4.11</td>
<td>Received data FEC performance at -50 dBm (all time slots)</td>
<td>44</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.12</td>
<td>VDL implementation for flight test</td>
<td>46</td>
</tr>
<tr>
<td>4.13</td>
<td>Flight test plans</td>
<td>47</td>
</tr>
<tr>
<td>4.14</td>
<td>Orbital flight path</td>
<td>48</td>
</tr>
<tr>
<td>4.15</td>
<td>Range from VDL transmitter antenna</td>
<td>48</td>
</tr>
<tr>
<td>4.16</td>
<td>Altitude above VDL transmitter antenna</td>
<td>49</td>
</tr>
<tr>
<td>4.17</td>
<td>Bering from VDL transmitter antenna</td>
<td>49</td>
</tr>
<tr>
<td>4.18</td>
<td>Normalized GPS time</td>
<td>51</td>
</tr>
<tr>
<td>4.19</td>
<td>Received message counter at orbital path</td>
<td>51</td>
</tr>
<tr>
<td>4.20</td>
<td>VDL receiver indicated power (flight test)</td>
<td>53</td>
</tr>
<tr>
<td>4.21</td>
<td>Application data FEC distribution (flight test)</td>
<td>54</td>
</tr>
<tr>
<td>4.22</td>
<td>Application data FEC histogram (flight test)</td>
<td>54</td>
</tr>
</tbody>
</table>
1. Introduction

The Local Area Augmentation System (LAAS) for aircraft precision approach and landing consists of four distinct functional segments. These are: 1) the space segment (navigation satellites), 2) the airborne subsystem, 3) the ground subsystem, and 4) the data broadcast system (often referred to as a data link).

The data broadcast subsystem is an important component of the Special CATegory - I (SCAT-I) [1] and the LAAS [2] which are under development by the industry for private use and the United States Federal Aviation Administration for public use, respectively. The data broadcast subsystem is responsible for transferring the differential correction, integrity, and path point information from the ground reference station to the airborne user to construct a robust precision approach system capable of eventually replacing the current Instrument Landing System (ILS).

This paper presents bench and flight test results of a SCAT-I/LAAS Navigation Augmentation Broadcast Signal (NABS) using a VHF Data Link (VDL) [3]. In this VDL, digital data is modulated by Differential 8-Phase Shift Keying (D8PSK) and broadcast in the VHF Omnirange (VOR) band (112.000 MHz - 117.95 MHz) [4] using time division multiple access (TDMA) protocol.
Following the bench tests, a series of flight trials were conducted at the Ohio University Airport on a Piper Saratoga aircraft. Parameters investigated during the flight trials included: 1) Service volume characteristics of the ground-based transmitter and antenna, and 2) The quality of signal reception using a horizontally polarized VOR/Localizer aircraft antenna. The results of this work aid in the validation of the efficacy and reliability of the VHF NABS functionality.

In chapter 2, fundamentals of the Global Navigation Satellite System (GNSS) and the VDL are described. In chapter 3, the implementation of VDL system and set-up details are illustrated. Chapter 4 includes discussions of laboratory and flight test results for the NABS VDL system. Conclusions and recommendations for future work are given in chapter 5.
2. Fundamentals of GNSS and the VHF Data Link System

Section 2.1 introduces the GNSS characteristics and describes differential GPS. Section 2.2 gives the details of the VDL message format, encoding and modulation, signal carrier frequency, and power budget. Section 2.3 describes the TDMA usage in the VDL system.

2.1 GNSS Characteristics

The GNSS is envisioned to be a world-wide position, velocity and time determination system. GNSS provides 24-hour, all-weather service to an unlimited number of users with appropriate receiving equipment. Two satellite constellations of GNSS are in operation. One is the Global Positioning System (GPS) developed by the United States Department of Defense. The other one is the Global Orbiting Navigation Satellite System (GLONASS) overseen and operated by Russia's Ministry of Defense.

The GPS is comprised of three segments: 1) Space Segment, 2) Control Segment, and 3) User Segment. Open-Loop GPS [5] positioning requires a minimum of four satellites to obtain position, velocity and timing measurements. It has a predictable horizontal plane accuracy on the order of 100 m (95%) and vertical accuracy in the order of 156 m (95%). However, ranging errors are generally grouped into the following six groups in making a position estimate [6]:
• User Range Error (URE): Unintentional satellite clock and orbital errors.
• Selective Availability (SA): Intentional satellite clock and orbital errors.
• Ionospheric group delays of GPS signals.
• Tropospheric group delay of GPS signals.
• Multipath: Errors caused by reflected signals, other than the direct signal.
• Receiver hardware/software measurement errors: Errors caused by thermal noise, inter-channel biases, receiver hardware or receiver software design

The concept of Differential GPS (DGPS) (Figure 2.1) [7] was developed to reduce most of the errors forenamed and increase accuracies to 1 - 10 meters. A Local Area DGPS (LADGPS) requires a GPS receiver at a known location (referred to as the reference station), a data link (or data broadcast), and mobile units. The reference station estimates error components of each satellite range measurement, forms a correction for each satellite and broadcasts the correction data through the data link to mobile units that are viewing the same satellites as the reference station. Common (correlated) errors that exist in both the reference station and mobile units are thus canceled. The primary non-common errors are multipath and errors internal to the user.
The GPS satellites provide positioning information to both the airborne and ground subsystems. The ground subsystem derives positioning information from the received signal, compares it with its known location to produce range corrections and transmits the ground-monitored differential correction, integrity and precision approach waypoint data by the VDL to the airborne subsystem for processing. The airborne subsystem corrects its pseudorange measurements to each satellite by the differential correction data received from the ground subsystem. DGPS can therefore support more accurate determination of position, velocity, and heading.
2.2 NABS VHF Data Link Characteristics

The NABS VHF data link includes a transmitter and a receiver. Messages are formatted, encoded, bit scrambled and modulated by using Differential 8-Phase Shift Keying (D8PSK) and are transmitted to the airborne subsystem (Figure 2.2).

![Figure 2.2 VDL message encoding block diagram](image)

The VDL receiver will receive transmitted message packets, demodulate and decode for navigation use. Details of the message format are shown in appendix A.

2.2.1 Message Format and Pre-transmit Processing

The format of the VHF data link physical layer contains a demodulator training sequence followed by application data and Forward Error Correction (FEC) (Table 2.1). A maximum data length of 1968 bits is encoded in a message packet and transmitted.
Table 2.1 VDL message format

<table>
<thead>
<tr>
<th>Physical Layer</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing tolerance (Early):</td>
<td>3</td>
</tr>
<tr>
<td>Training sequence:</td>
<td>85</td>
</tr>
<tr>
<td>Application Layer</td>
<td></td>
</tr>
<tr>
<td>Message Block Header:</td>
<td>48</td>
</tr>
<tr>
<td>Data Messages (variable):</td>
<td>1704 (Max)</td>
</tr>
<tr>
<td>Cyclic Redundancy Check:</td>
<td>24</td>
</tr>
<tr>
<td>Forward Error Correction:</td>
<td>62</td>
</tr>
<tr>
<td>Propagation guard:</td>
<td>39</td>
</tr>
<tr>
<td>Timing tolerance (Late):</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>1968 (Max)</td>
</tr>
</tbody>
</table>

The training sequence, application layer, and forward error correction are briefly discussed below.

- **Training sequence:**

  The training sequence is formed by 5 segments: 1) Transmitter Power Stabilization (12 bits), 2) Synchronization and Ambiguity Resolution (48 bits), 3) Reserved Symbol (3 bits), 4) Transmission Length (17 bits), and 5) Header FEC (5 bits) [8].

  The training sequence allows the airborne subsystem to demodulate the data messages properly. The definition of the physical layer modulation and training sequence includes some unnecessary functionality so as to be consistent with the ICAO Document AMCP/3-DP-8B.

- **Application Layer Data:**

  In SCAT-I, message Type 1 [9] is designed to provide differential correction data for a variable number of individual satellites (up to 35 SV’s). Its length is specified
to be a maximum data length of 222 bytes (1776 bits). This data also supports
ground subsystem integrity alarms. However, random messages are used in this
thesis to fill in the application layer and replace the Type-I message in experiments.
The format of the application message contains 3 blocks: 1) message header (48
bits), 2) message (variable length), and 3) Cyclic Redundancy Check (CRC) (24
bits).

- Error Correction Encoding [10]:

  For the application data, Forward Error Correction (FEC) is applied by a systematic
  Reed-Solomon (255,249) 2^8-ary code after the training sequence. This FEC shall be
  able to correct up to three codeword symbol (3 symbols × 8 bits/symbol = 24
  consecutive bits) errors or detect 4 codeword errors [11].

A pseudonoise (PN) scrambler (Figure 2.3) with a 15-stage generator register is
XORed with the formatted data stream, starting with the reserved symbol in order to aid
clock recovery.

![Scrambler/Descrambler shift register block diagram](image)
A D8PSK signal modulation is used in the VDL to allow a high degree of suppression of adjacent channel interference. The Differentially encoding Phase Shift Keying (DPSK) technique refers to a phase change of 45° from symbol to symbol as compared to the 180° phase shift for Binary Phase Shift Keying (BPSK). In D8PSK, the referenced signal is simply a 45° out-of-phase delayed version of the received signal [12]. In comparing the error performance of BPSK and DPSK with optimum demodulation, the DPSK requires at most 1dB more average bit energy to noise ratio \( (E_b/N_0) \) than BPSK to achieve the same Bit Error Rate (BER) [13]. However, DPSK is chosen for modulation in this system because the receiver does not require a carrier synchronization circuit.

The D8PSK modulation can be produced by combining two quadrature Radio Frequency signals (Figure 2.4). Based upon this design, the relationship between the In-phase and Quadrature-phase component in the transmitted signal shall be ±3° from quadrature and the amplitude equal within ±1 dB. The carrier component should be at least 20 dB less than the peak RF amplitude of the modulated signal.

The encoder encodes the scrambled data and converts into three separate binary streams X, Y and Z. At time \( k \), \((X_k Y_k Z_k)\) is represented by a change in phase as shown in Table 2.2, the absolute phase being \( \phi_k \). Where \( \phi_k = \phi_{k-1} + \Delta \phi_k \) is the accumulation of the \( \Delta \phi_i, i=1,\ldots, k \).
The VDL symbol rate is 10,500 symbols/sec -- a nominal bit rate of 31,500 bits/sec.

In Figure 2.4, pulse shaping filters were applied in the very end of data encoding to eliminate Inter Symbol Interference (ISI) [14]. The information is raise cosine filtered (roll-off factor = 0.6) before being transmitted.

Table 2.2 Data Encoding

<table>
<thead>
<tr>
<th>$X_k$, $Y_k$, $Z_k$</th>
<th>$\phi_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 0</td>
<td>0\pi/4</td>
</tr>
<tr>
<td>0, 0, 1</td>
<td>\pi/4</td>
</tr>
<tr>
<td>0, 1, 1</td>
<td>2\pi/4</td>
</tr>
<tr>
<td>0, 1, 0</td>
<td>3\pi/4</td>
</tr>
<tr>
<td>1, 1, 0</td>
<td>4\pi/4</td>
</tr>
<tr>
<td>1, 1, 1</td>
<td>5\pi/4</td>
</tr>
<tr>
<td>1, 0, 1</td>
<td>6\pi/4</td>
</tr>
<tr>
<td>1, 0, 0</td>
<td>7\pi/4</td>
</tr>
</tbody>
</table>
2.2.2 Carrier Frequency Range

The VDL transmission for SCAT-I has been assigned to the upper portion of Very High Frequency Omnirange (VOR) band (112.000 MHz to 117.950 MHz) with a channel spacing of 25 kHz. The interaction of VDL with the existing VOR system has been tested [15]. Work is ongoing to specify the criteria for the interaction of the systems.

2.2.3 Power Budget

The VDL transmitter is specified to radiate a power with a field strength of at least 200$\mu$V/m (-110 dBW/m$^2$)(-77 dBm) to a maximum +43 dBm (measured into a 50$\Omega$ load). This is within the defined operational coverage of such a facility. The VDL receiver, however, is specified to provide a -87 dBm signal sensitivity in a specified error rate. The sensitivity of receiver is 10 dB above the minimum required receiver field strength.

2.3 Media Access Sublayer Using Time Division Multiple Access (TDMA)

The Time Division Multiple Access (TDMA) is designed to share the same communication resource by assigning each of M signals or users the full spectral occupancy of the system for a short duration of time, referred to as a time slot. In a
TDMA application, time is segmented into intervals called frames, each frame is partitioned into assignable user time slots. Using TDMA, more capacity than is required by any single NABS ground station are offered by the physical layer. Spectrum efficiency is achieved by partitioning the total capacity offered by a single NABS frequency assignment to individual proximate ground stations.

The TDMA timing structure applied in VDL is based on a two-level hierarchy as shown in Figure 2.5. Two of DGPS frames are presented in one second GPS epoch. Each DGPS frame is 500 ms in duration. The first of these DGPS frames start at the beginning of the GPS one second epoch, and the second DGPS frame starts midway through the epoch. Since the GPS frame is 8 time divisions multiplexed, a frame will consist of 8 individual GPS time slots (A - H) of 62.5 ms ± 95.2 μs (one symbol, 3 bits duration with the nominal bit rate of 31,500 bits/sec forenamed in 2.2.1). Hence, a GPS time slot establishes the incremental capacity resource that can be assigned to an individual DGPS NABS ground station.

Within each time slot, a message is transmitted. This message can be of variable length, up to the maximum allowed per slot of 222 bytes (8-bit word). As shown on Figure 2.5, the requirement of timing tolerance (early and late), training sequence, FEC, ramp down, and propagation guard will be 190 bits in length. At a rate of 10,500 symbols/sec, each time slot is able to have a duration of 656.25 symbols (1968 bits).
Ideally, application layer should be 1778 bits long (maximum). However, only 1776 bits maximum can been applied for 222 bytes data in application layer. The redundant two bits are filled by "Fill Bits" and are not used for data transmission.

Figure 2.5 VDL TDMA timing structure
3. Implementation of the VDL System

The hardware and software parts of a prototype VHF data link system has been completed and installed at the Ohio University. This system is based on SCAT-I equipment developed by the E-Systems Company. The stability and performance of the system has been tested. Section 3.1 will describe the hardware set up of the VDL system which includes the VDL transmitter and receiver, GPS receivers, control components, and interfaces. Section 3.2 depicts the algorithm and describes the software development of this system.

3.1 System Set Up and Interfacing

The implementation of VDL system (Figure 3.1) includes a VDL transmitter for broadcasting messages, a VDL receiver for receiving messages, two GPS receivers for time synchronization, a computers for controlling and communicating with the VDL transmitter, and another computer for controlling and communicating with the VDL receiver. Two voltage converter Printed Circuits Boards (PCBs) are built to convert the TTL level GPS 1 Pulse Per Second (1PPS) signal to a RS-232 level and a RS-422 level 1 PPS signal [16].
Collect GPS Data

VHF VHF LIIL2

~

VHF Data Link 112 - 117.95 MHz

Collect GPS Data

L1/L2

1PPS in TTL level

Ashtech GPS receiver

Voltage Converter

Generate Random Messages to Fill in the Application Data Layer

1PPS in RS-232 level

1PPS in RS-422 level

QNX CPU

VDL Transmitter

RS-232

Ground Reference Station

User Station

QNX CPU

VDL Transmitter

RS-232

Ashtech GPS receiver

Voltage Converter

NovAtel GPS receiver

VDL Receiver

DOS CPU

Figure 3.1 VHF data link set up
Random messages were generated by software in the ground reference station to fill in the application layer and replace the Type-1 GPS correction data. The VDL transmitter and receiver accomplish the message encoding, modulation, demodulation and decoding. A major task in developing the interface was to configure the messages according to the TDMA protocol.

In Figure 3.1,

- **VDL transmitter**: E-systems VDL transmitter [17].
  The VDL transmitter encodes, modulates, and broadcast the data messages which were received from the transmitter control program. The transmitter receives control commands and application data messages from the QNX [18] Central Processing Unit (CPU). The transmitter respond to the QNX CPU by sending back acknowledgment messages.

- **QNX CPU**: Intel 80486DX CPU with QNX operation system.
  The QNX CPU is the host computer of the transmitter control program. The duties of the QNX CPU are:
  1. Establish serial port communication with the transmitter using one serial port (COM 2). The QNX CPU sends control commands to transmitter, initializes the transmitter, and receives the transmitter's acknowledgment messages.
2. Receive GPS 1PPS from the other serial port (COM 1). The QNX CPU receives
the 1 PPS to trigger the generated random data messages in according with the
TDMA protocol.

3. Generate random data messages and send those messages to the transmitter via
COM 2 at a specific time slot in reference to the 1PPS.

- VDL receiver: E-systems VDL receiver [19].
  The VDL receiver receives data messages from the VDL channel, demodulates, and
  presents these data messages to the receiver control program. The VDL receiver
  also receives control commands from the DOS CPU and responds to it by sending
  back acknowledgment messages.

- DOS CPU: Intel 80486DX CPU with MS-DOS operating system.
  The DOS CPU is the host computer for the VDL receiver control program. This
  DOS CPU is able to:
  1. Establish communication with VDL receiver using one serial port (COM 2). The
     DOS CPU sends control commands to receiver and accepts the receiver’s
     acknowledgment messages.
  2. Receive GPS position data logs from the other serial port (COM 1). The DOS
     CPU uses these data logs to obtain the user position in Earth-Centered-Earth-
     Fixed (ECEF) coordinates and also a time reference.
3. Receive VDL application data messages from COM 2 and decode.

4. Log VDL messages and user position to the screen or a file.

- Ashtech GPS receiver: Ashtech Z-12 L1/L2 band 12 channel GPS receiver [20]. The Ashtech GPS receiver is used because of a convenient 1 PPS output option. The 1PPS output option provides the 1PPS signal (through the voltage converter PCBs) to both VDL transmitter and receiver for system synchronization

- Converter PCBs: Hardware layout include Maxim MAX202E, MAX485E transceiver chips (Figure 3.2). These PCBs convert the GPS 1PPS from the 1PPS output of Ashtech GPS receiver at TTL voltage levels (0V ~ 5V) to RS-232 voltage levels (±V_o, 3V ≤ V_o ≤ 25V) and RS-422 voltage levels (±V_o, 0.2V ≤ V_o ≤ 25V) respectively [21]. The output drivers are triggered by the trailing edges of 1PPS pulses. The transient time at the output of RS-422 (MAX485E) last for 10 µs to 30µs and will not affect the precision for the time reference for the VDL transmitter/receiver (Figure 3.3). The transient time of RS-232 output (MAX202E) does not affect the precision for the time reference at the QNX CPU either. The 1PPS provides the QNX CPU with a time reference in GPS time with a one-second rollover. The 1PPS also synchronizes the VDL transmitter and receiver in order to obtain time synchronization for the TDMA.
NovAtel GPS receiver: NovAtel OEM-1 3151 L1 band 10 channel GPS receiver.

The NovAtel GPS receiver [22] is used to obtain the user's position with a GPS time reference.

![PCB layout of voltage converter](image)

**Figure 3.2 PCB layout of voltage converter**

### 3.1.1 Interfacing by RS-232

The RS-232 serial port communication interface protocol [23] is used in the VDL system. Null MODEM cable connection (Figure 3.4) were assigned to build a 2-way communication between: 1) the QNX CPU and the VDL transmitter, 2) the DOS CPU and the VDL receiver, and 3) the DOS CPU and the GPS receiver. The transmitter control program “listens” to 1PPS, thus, the communication between the transmission program and 1PPS is a one-way communication.
20

Ashtech
1PPS output
TTL level

+5V
0V

command 990 is issued,
Trailing edge is
synchronized
with GPS time

-2 ms-

1PPS @
RS-232 level
MAX202E
output

+25V
0V

\[ t_r = t_f = 250\text{ns} - 2000\text{ns} \]

-2 ms-

1PPS @
RS-422 level
MAX485E
output

+10.5V
0V

\[ t_r = t_f = 10\text{us} - 30\text{us} \]

-2 ms-

Reference: product characteristic manual, Maxim Integrated Products

Figure 3.3 Characteristic of transceiver chips response to 1PPS

Computer

Null modem cable

Terminal

Figure 3.4 Null MODEM Cable Assignment for Computer to Terminal
Figure 3.5 shows the details of the interfacing of RS-232, in both the VDL transmitter and the VDL receiver with their control program host computers.

![Diagram of RS-232 interface assignments in the VDL system]

**Figure 3.5** RS-232 interface assignments in the VDL system

### 3.1.2 Synchronization in VDL -- 1 PPS

The 1 Pulse Per Second is converted to RS-422 voltage levels in order to be used at the VDL transmitter and receiver. This 1PPS will help the data link system to synchronize their time reference. As forenamed in section 2.2.1, the symbol rate of the data link is 10,500 symbol/sec. If both VDL transmitter and VDL receiver are synchronized, the Time Of Arrival (TOA) of each message at the user station will be indicated by the VDL receiver (0 ≤ TOA ≤ 10,499). This TOA is a delta value in
reference to the 1PPS which indicate the “arrived symbols number” at the end of training sequence (85 bits + 3 bits of timing tolerance). The TOA can help the user to decide the time of arrival of each message in the TDMA, thus determining in which slot the data was transmitted.

3.2 Software implementation

The development of software for this system included the VDL transmitter control program and the VDL receiver control program using the C programming language. The main tasks of these programs are serial port communications and VDL transmitter/receiver accessing.

3.2.1 Transmitter Control Program

Originally, three serial communication ports were considered for the VDL transmitter control program. There are: 1) Collect DGPS correction data, 2) Obtain 1PPS from voltage level converter PCB, and 3) Communicate with VDL transmitter. The QNX operating system was chosen for the transmitter control program to access more communication ports at a time. Moreover, the QNX OS does better time resolution than the MS-DOS system. Practically, only two communication ports are accessed by the transmitter control program. The transmitter control program generates
random messages itself to filled in the application layer and send those messages to the VDL transmitter.

The objectives of the VDL transmitter program are:

- Initialize serial communication ports

  After initialization, the transmitter control program should be able to listen to 1PPS from COM 1, communicate with VDL transmitter via COM 2, send commands to transmitter, and listen to the acknowledgment messages sent by the transmitter.

- Send serial data input to VDL transmitter

  The transmitter control program sends VDL transmitter commands [17] and data, these are: 1) Tasking command, 2) Tune command, 3) Initiated BIT command 4) Idle command, 5) Exit idle command, 6) Shutdown command, 7) Transmitter power control command, and 8) Application data (generated random messages).

- Receive serial data output from VDL transmitter

  The VDL transmitter will respond to any command/data received within 100 ms by acknowledgment messages (if not, the VDL transmitter is deemed failed). This feature will allow the control program to make sure that the messages sent to the VDL transmitter are received correctly.

- Generate random messages to fill in the application data layer to be sent to VDL transmitter. By fixing the random seed, the generated random messages will be the same every time they are transmitted. To embed a sequential number in each
message, the first 3 bytes of the application data were filled by a sequential number (known as message counter). The fourth byte of the application data was filled with the slot identification number. This slot id number indicates the time at which this message has been sent through the serial communication port to the VDL transmitter (this needs to be one time slot prior to the valid time slot used for transmitting this message to VDL channel - to be discussed in next group) (Figure 3.6).

- Decide the time to send a data message using the 1PPS and fit the requirement of TDMA.

<table>
<thead>
<tr>
<th>Common Header (5 bytes)</th>
<th>Application Data (222 bytes maximum)</th>
<th>Check sum (1 byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync byte 1</td>
<td>slot id</td>
<td>calculated by sum</td>
</tr>
<tr>
<td>sync byte 2</td>
<td>seq # byte 1</td>
<td>all previous bytes</td>
</tr>
<tr>
<td>message id</td>
<td>seq # byte 2</td>
<td>and with FF</td>
</tr>
<tr>
<td>sequence number</td>
<td>seq # byte 3</td>
<td></td>
</tr>
<tr>
<td>number of data bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 3.6 VDL application data format for VDL system bench test*

Figure 3.7 is the transmitter event chart. In QNX, the time resolution was set to 0.5 ms/tick in order to generate a tick counter interrupt at 2 kHz. This tick counter is reset every time the 1PPS interrupts and keeps the counter centered around 2 kHz (± 1 tick). The VDL transmitter requires the QNX CPU to send an application data message one time slot prior to an intended transmitting time slot. As such, the generated
sequential random application data messages should be transmitted to the VDL channel through the next available intended time slot. A intended time slot of the VDL transmitter is initiated by the transmitter control program.

For instance, in Figure 3.7, assume that the VDL transmitter is initialized to have one time slot transmission at time slot “C”. The QNX CPU needs to catch the timing on the serial communication port and send a random application message to the VDL transmitter at the beginning of time slot “B” and wait for the VDL transmitter to respond to the control program by an acknowledgment message to make sure that this message has been accepted by the VDL transmitter and also to get ready to transmit on the channel at the time slot “C” (Figure 3.8.b). The serial communication port between the VDL transmitter and QNX CPU communicates at 115,200 bps (factory set, as does
the VDL receiver with the MS-DOS CPU). During maximum length application data transmission, 228 bytes (header + data + check sum) will need to be sent from the QNX CPU to the VDL transmitter. At the forenamed baud rate, the nominal delay is $\leq 20$ ms which does not exceed the one time slot epoch (62.5 ms) and the data message will be transmitted through the data link during the next available time slot (Figure 3.8.a) with a maximum of 16 messages/sec (all time slots).

![Diagram](image)

**Figure 3.8 TDMA timing structure and data send time in QNX CPU**

Figure 3.9 is the flow chart of the VDL transmitter control program. In the future, this transmitter control program should be modified to access one more serial communication port and act like a interface in transmitting the real correction GPS data.
Build serial ports communication and set the VDL transmitter (Tune frequency, set time slot and power attenuation)

Counter < Number of messages or key not hit? 

Is tick at the send time prior to the desired time slot?

Generate random message and send to VDL transmitter by serial com port

Check if the ACK message respond to program

Report error

Figure 3.9 Flow chart of the VDL transmitter control program
3.2.2 Receiver Control Program

In order to obtain GPS time, position, and the VDL messages, the VDL receiver program was designed to access two serial communication ports under the MS-DOS environment. The program is able to: 1) communicate with NovAtel GPS receiver to obtain a GPS position fix with time reference, and 2) communicate with VDL receiver to obtain acknowledgment/application data messages.

The objectives of receiver control program are:

- Initialize serial communication ports
  
  After initialization, the receiver control program should be able to: 1) listen to NovAtel GPS receiver for GPS time and position logs from COM 1, 2) Communicate with VDL transmitter via COM 2, send commands to VDL receiver and listen to the response of VDL receiver by acknowledgment messages, and 3) stand by to receive VDL application data messages.

- Receive serial data output from NovAtel GPS receiver, obtain time and position fix.
  
  Decode the logs from GPS messages to get the user position with time reference. A 38,400 bps serial communication was built between the MS-DOS CPU and NovAtel GPS receiver which was able to handle the GPS data logs rollover in every second

- Send serial data input to VDL receiver
The VDL receiver control program sends VDL receiver commands [19]. These are:

1) Tasking command, 2) Tune command, 3) Initiated BIT command, 4) Idle command, 5) Exit idle command, and 6) Shutdown command.

- Receive serial data output from VDL receiver

The VDL receiver will automatically provide acknowledgment messages to the control program at the rate of 16 Hz, no matter whether a command is sent to the receiver or not. When the application data messages are received, the receiver will still provide acknowledgment messages to fill in “empty” time slots (time slots not been used for transmission). However, the VDL receiver is not dependent on the TDMA structure.

The VDL receiver control program event chart is depicted in Figure 3.10. The receiver control program is a big loop which keeps checking both new message flags from the VDL buffer and new message flags from the GPS buffers. Hardware interrupt routines were generated to access two serial communication ports by different usage of those message flags. Data output from the GPS receiver passes through COM 1, generates interrupt request (IRQ) EF₉, and stores it in a temporary buffer to match the pattern of GPS message synchronization. Not until a whole GPS message is received is the new message flag raised for the main routine to read. After the main routine reads out that message, the new message flag will be erased. Similar to COM 1, data output from the VDL receiver passes through COM 2, generates IRQ F7₉, and is stored in
another temporary buffer until the whole VDL message is received. Once the main routine reads a new message from either the GPS buffer or the VDL buffer, the new message is sent to decode and depending on user’s request, is logged on to the screen or a file.

![Diagram](image)

**Figure 3.10 Event Chart of the receiver control program host in MS-DOS CPU**

Figure 3.11 is the flow chart of the VDL receiver control program. The Intel 80486-DX2 CPU is fast enough to handle the main loop processing with one complete cycle of processing done in a few micro seconds. The interrupts generated by COM 1 (38,400 bps) and COM 2 (115,200 bps) will never be missed. The receiver is initialized to stand on a specialized frequency and “open” to receive any combination of time slot message transmission at this frequency. The power level of the received message and
data FEC value will be calculated by the VDL receiver and reported to the receiver control program as output parameters.

Figure 3.11 Flow chart of the VDL receiver control program
4. VDL Performance Evaluation

The VDL was tested in both the laboratory and flight test environments. From the laboratory bench test, the general performance of this VDL system has been critically reviewed and finalized in section 4.1. Section 4.2 shows the results of the flight tests. Parameters and characteristics are investigated and discussed in this section.

4.1 VDL System Laboratory Performance

Figure 4.1 shows the physical connection of the VDL system implementation. All of the connections were made using coaxial cables to reduce power loss. A programmable attenuator was set between the VDL transmitter and receiver to obtain different power level VDL signals. A spectrum analyzer was connected to the output of the programmable attenuator for measuring the received signal power. The Ashtech GPS receiver and the NovAtel GPS receiver used different antennas. Tests followed a procedure of three steps: 1) Program the attenuator to a specified power level, 2) Tune the VDL receiver to a specified frequency (112.95 MHz) [24] and stand by for receiving VDL messages, and 3) Tune the VDL transmitter to a specified frequency and time slot and start sending sequential random data messages.
Figure 4.1 VDL system physical connections for laboratory bench test
Table 4.1 is the actual measurement of system power budget for the bench test. By controlling the programmable attenuator, the received signal power can be adjusted to a particular value.

### Table 4.1 VDL system power budget

<table>
<thead>
<tr>
<th>Description</th>
<th>Power Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal VDL Transmit Power</td>
<td>+43 dBm</td>
</tr>
<tr>
<td>VDL Transmitter Programmable Power Attenuation (0 dB to 15 dB)</td>
<td>-10 dB</td>
</tr>
<tr>
<td>Fixed 23 dB attenuator</td>
<td>-23 dB</td>
</tr>
<tr>
<td>Cable loss</td>
<td>-5 dB</td>
</tr>
<tr>
<td>net VDL Power</td>
<td>+5 dBm</td>
</tr>
<tr>
<td>Programmable Attenuator</td>
<td>TBD dB</td>
</tr>
<tr>
<td>VDL received power</td>
<td>TBD dBm</td>
</tr>
</tbody>
</table>

TBD: to be determined

Power supplies with different output voltage levels are required in the VDL transmitter and receiver respectively. Table 4.2 is a summary of all power inputs of the VDL system. The maximum power dissipation for the transmitter is less than 100 Watts and 10 watts for the receiver.

### Table 4.2 VDL power Inputs

<table>
<thead>
<tr>
<th>VDC ± 5%</th>
<th>Used by Transmitter (Max. Amps)</th>
<th>Used by Receiver (Max. Amps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-12 or -15</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>+15</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>+28</td>
<td>3</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Test results are discussed in the following sections.

### 4.1.1 VDL Receiver Indicated Power

The VDL receiver indicates the peak power during receiving each message. The exact power received is reported by the spectrum analyzer which receives the same VDL signal as the VDL receiver through a splitter. Table 4.3 shows that the average received message power as output by the receiver saturated at -76 dBm.

<table>
<thead>
<tr>
<th>Programmable Attenuator (dB)</th>
<th>Measured receiver Power (dBm)</th>
<th>Average Power (dBm) Indicated by the VDL receiver (one time slot)</th>
<th>Average Power (dBm) Indicated by the VDL receiver (all time slot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>-40</td>
<td>-40</td>
<td>-40</td>
</tr>
<tr>
<td>55</td>
<td>-50</td>
<td>-50</td>
<td>-50.13</td>
</tr>
<tr>
<td>65</td>
<td>-60</td>
<td>-60</td>
<td>-60</td>
</tr>
<tr>
<td>75</td>
<td>-70</td>
<td>-71</td>
<td>-70.99</td>
</tr>
<tr>
<td>85</td>
<td>-80</td>
<td>-75.19</td>
<td>-75.02</td>
</tr>
<tr>
<td>95</td>
<td>-90</td>
<td>-76</td>
<td>-75.67</td>
</tr>
</tbody>
</table>
4.1.2 Message Loss Rate and Receiver Sensitivity

The prototype VDL receiver manufactured by E-Systems with a sensitivity range of -65 dBm to -70 dBm. However, the VDL receiver was specified by SCAT-I to have a minimum sensitivity of -87 dBm at the input [19]. The sequential random messages were transmitted, hence, contiguous application data were anticipated to be received by the VDL receiver. The "lost message" state can be decided by comparing a sequence number with the sequence number embedded in the received message (known as "message counter"). However, in receiving sequential data of 1,600 messages, -65 dBm was inferred to be the sensitivity of the VDL receiver (Figure 4.2 and Figure 4.3). The application data FEC can be used to decide message loss rate as well. In Figure 4.4 and Figure 4.5, the percentage of messages lost rate was plotted versus the received signal power level. As mentioned in section 2.2.1, the FEC function in the VDL receiver will detect up to 4 symbol errors and correct 3 symbols as well. A received message data FEC equal to 4 indicates a "lost message" state in the statistics.
Figure 4.2 Message received by different power level (one time slot)

Figure 4.3 Message received by different power level (all time slots)
Figure 4.4 Message loss rate using application data FEC = 4 (one time slot)

Figure 4.5 Message loss rate using Application Data FEC = 4 (all time slots)
4.1.3 Received Message FEC Function

Figure 4.6 and Figure 4.7 depict the header FEC distribution histogram for one time slot and for all time slots data transmission, respectively. At different received power levels, the header FEC were kept at 0 for most of the time. These histograms indicate that even if the received power is lesser than the VDL receiver sensitivity, the header FEC will still work. In Table 4.4 and Table 4.5, there are no header FEC equal to 2 (non-corrected errors) in 1,600 messages transmissions. This indicates that all errors in the header were corrected by this header FEC function without “lost message” state being introduced by header FEC = 2.

Figure 4.8 and Figure 4.9 are the application data FEC distribution histograms for one time slot and all time slots data transmission. Refer also to Table 4.6 and Table 4.7. The calculation for data loss rate was determined by received message counter and application data FEC. The “lost message” state was decided by: 1) when received message counter does not match the sequence number, or 2) when the received message had an FEC = 4 indication. In application data, the message loss rate is reflected by FEC distribution.
Figure 4.6 Header FEC distribution by one time slot transmission

Figure 4.7 Header FEC distribution by all time slots transmission
Figure 4.8 Application data FEC distribution by one time slot transmission

Figure 4.9 Application data FEC distribution by all time slots transmission
Table 4.4 Indicated Header FEC by one time slot transmission

<table>
<thead>
<tr>
<th>Received Power</th>
<th>Header FEC =0</th>
<th>Header FEC = 1</th>
<th>Header FEC = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-50 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-60 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-70 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-80 dBm</td>
<td>99.9375%</td>
<td>0.0625%</td>
<td>0%</td>
</tr>
<tr>
<td>-90 dBm</td>
<td>98.4375%</td>
<td>1.5625%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 4.5 Indicated Header FEC by all time slots transmission

<table>
<thead>
<tr>
<th>Received Power</th>
<th>Header FEC =0</th>
<th>Header FEC = 1</th>
<th>Header FEC = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-50 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-60 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-70 dBm</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>-80 dBm</td>
<td>99.8125%</td>
<td>0.1875%</td>
<td>0%</td>
</tr>
<tr>
<td>-90 dBm</td>
<td>98.5625%</td>
<td>1.4375%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 4.6 Indicated application data FEC by one time slot transmission

<table>
<thead>
<tr>
<th>Received Power</th>
<th>Application Data FEC =0</th>
<th>Application Data FEC =1</th>
<th>Application Data FEC =2</th>
<th>Application Data FEC =3</th>
<th>Application Data FEC =4</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40 dBm</td>
<td>89.1875%</td>
<td>8.1875%</td>
<td>2%</td>
<td>0.375%</td>
<td>0.25%</td>
</tr>
<tr>
<td>-50 dBm</td>
<td>89.125%</td>
<td>9.375%</td>
<td>1.625%</td>
<td>0.3125%</td>
<td>0.3125%</td>
</tr>
<tr>
<td>-60 dBm</td>
<td>89.125%</td>
<td>8.3125%</td>
<td>1.9375%</td>
<td>0.375%</td>
<td>0.25%</td>
</tr>
<tr>
<td>-70 dBm</td>
<td>74.1875%</td>
<td>14%</td>
<td>5.25%</td>
<td>2.8125%</td>
<td>3%</td>
</tr>
<tr>
<td>-80 dBm</td>
<td>44.6875%</td>
<td>19.375%</td>
<td>10.75%</td>
<td>9.9375%</td>
<td>15.25%</td>
</tr>
<tr>
<td>-90 dBm</td>
<td>37.9375%</td>
<td>14.5625%</td>
<td>9.125%</td>
<td>7.5625%</td>
<td>30.8125%</td>
</tr>
</tbody>
</table>
Table 4.7 Indicated application FEC by all time slots transmission

<table>
<thead>
<tr>
<th>Received Power</th>
<th>Application Data FEC =0</th>
<th>Application Data FEC =1</th>
<th>Application Data FEC =2</th>
<th>Application Data FEC =3</th>
<th>Application Data FEC =4</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40 dBm</td>
<td>90.125%</td>
<td>7.5625%</td>
<td>1.75%</td>
<td>0.25%</td>
<td>0.3125%</td>
</tr>
<tr>
<td>-50 dBm</td>
<td>88%</td>
<td>10.125%</td>
<td>1.375%</td>
<td>0.25%</td>
<td>0.25%</td>
</tr>
<tr>
<td>-60 dBm</td>
<td>89.1875%</td>
<td>8.8125%</td>
<td>1.5625%</td>
<td>0.1875%</td>
<td>0.25%</td>
</tr>
<tr>
<td>-70 dBm</td>
<td>81.125%</td>
<td>12.25%</td>
<td>3%</td>
<td>1.625%</td>
<td>2%</td>
</tr>
<tr>
<td>-80 dBm</td>
<td>51.75%</td>
<td>18.3125%</td>
<td>11.6875%</td>
<td>7.8125%</td>
<td>10.4375%</td>
</tr>
<tr>
<td>-90 dBm</td>
<td>27.875%</td>
<td>11.9375%</td>
<td>9.75%</td>
<td>10.3125%</td>
<td>40.125%</td>
</tr>
</tbody>
</table>

4.1.4 The “Lost Message” State in Terms of FEC = 4

Plots depicted in Figure 4.10 and Figure 4.11 are lost messages followed by the indicated FEC in one time slot and all time slots, at a received message power of -50 dBm. The data FEC count for one time slot (Figure 4.10) is spread more than the all time slots case (Figure 4.11) which implies that the Digital Signal Processing (DSP) CPU in the VDL receiver needs to continually re-synchronize due to the lower transmission rate of two messages per second (one time slot). In the all time slots transmission, received application data FECS are distributed with approximately 100 messages at FEC = 0 because the data stream is being fed to the receiver from the channel continuously. The DSP CPU will not need to adjust the Automatic Gain Control (AGC) or re-synchronize by messages.
Message Lost using 1600 packets (one slot at -50 dBm)

Application data FEC distribution (one slot at -50 dBm)

Figure 4.10 Received data FEC performance at -50 dBm (One Time Slot)

Messages lost using 1600 packets (all slots at -50 dBm)

Application data FEC distribution (all slots at -50 dBm)

Figure 4.11 Received data FEC performance at -50 dBm (All Time Slots)
4.2 VDL System Flight Test

The VDL flight test implementation is depicted on Figure 4.12. The VDL transmitter (ground station) antenna was mounted on the roof of Avionics Engineering Center hangar located at Ohio University Airport with an ECEF coordinate \([X, Y, Z] = [669511.5345, -4903263.8638, 4010628.3153]\). The VDL transmitter was broadcasting one time slot messages with a peak power of +43 dBm (20 W). The VDL receiver was mounted on an Piper Saratoga aircraft. The signal received by the VDL receiver was obtained by splitting the incoming signal at the exist VOR horizontally polarized navigation antenna.

Flight patterns for the VDL performance tests (Figure 4.13) included: 1) Orbital flight path to determine horizontal antenna pattern at 1,500 ft Above Ground Level (AGL) and 10,000 AGL. 2) Radial flight path to determine vertical antenna pattern for north, south, east and west radials in the radius from 0 nmi to 20 nmi (VDL minimum service range) at 1,500 ft AGL and 10,000 AGL. Due to time and receiver limitations, only the orbital paths with a radius of 10 nmi (18,520 m) and a height of 1,500 ft (457.2 m) were obtained. The aircraft was navigated using Loran-C receiver [25]. Data logs for all analysis processing in this flight test start at GPS time 332,162 sec (Wednesday, April 2, 15:15:02 EST) until GPS time 333,974 sec. The VDL receiver program recorded both GPS positioning data and VDL messages. The analysis of this flight test is reported in the following sections to verify the performance of the VDL system.
Figure 4.12 VDL implementation for flight test
4.2.1 Position of the User

The orbital flight paths (Figure 4.14) depict the flight test route. The range (Figure 4.15), altitude (Figure 4.16) and bearing (Figure 4.17) of the aircraft are derived by the GPS logs obtained from the NovAtel GPS receiver mounted on the test aircraft. Since an open-loop GPS positioning technique was applied by these logs, a 100 m horizontal and a 156 m vertical positioning errors are expected in Figure 4.14 and Figure 4.15. The time axis in these charts indicate normalized GPS time logs during the VDL messages reception.
Figure 4.14 Orbital flight path

Figure 4.15 Range from VDL transmitter antenna
Figure 4.16 Altitude above VDL transmitter antenna

Figure 4.17 Bearing from VDL transmitter antenna
4.2.2 VDL Message Reception

The normalized GPS time in Figure 4.18 indicates the time logs during the reception of VDL data. The slope of this chart should be 0.5 with respect to one time slot transmission (1800 sec × 2 messages/sec). However, 1872 sec were logged to collect 3600 valid VDL messages, 72 sec more than the anticipated time epoch (1800 sec). The VDL messages have not been logged sequentially, the last log message counter is 21,637 and the first message counter is 17,893 which means 3744 VDL messages should have been transmitted. In reality, 144 of the VDL messages have not been transmitted from the VDL transmitter [26]. This is thought to be due to a firmware error in the transmitter. Due to this reason, the received message packets mentioned above cannot be included in the calculation of message loss rate. The data index was calculated by actual messages accumulated.

Figure 4.19 shows the performance of received VDL messages. Around data index 2500 to 3000, an increase in “lost message” states was observed. This can be assumed to be caused by terrain obstructions.
Figure 4.18 Normalized GPS time

Figure 4.19 Received message counter at orbital path
4.2.3 Indicated Power

In calculating the free space loss of the signal transmission, the carrier frequency was 112.95 MHz, the average distance from transmitter to receiver was 18,531 m.

\[
\lambda = \frac{c}{f} = \frac{299792458 \text{ m/s}}{112.95 \times 10^6 \text{ /s}} = 2.652 \text{ m}
\]

\[
R = \sqrt{18520^2 + 4572^2} = 18531 \text{ m}
\]

\[
\frac{P_r}{P_t} = \left( \frac{\lambda}{4\pi R} \right)^2 = \left( \frac{2.652}{4 \times \pi \times 18531} \right)^2 = 1.3 \times 10^{-10} = -98.86 \text{ dB}
\]

Where:

\( \lambda \) = wave length
\( c \) = speed of light
\( R \) = distance between transmitter and receiver
\( P_t \) = transmitted power
\( P_r \) = received power

Therefore, -98.86 dB of free space loss is expected in the broadcast. With the transmission power at +43 dBm in the transmitter antenna, a -55.86 dBm power was expected to be indicated by the VDL receiver. However, a -76.52 dBm average power was experienced in the test. In Figure 4.20 the receiver indicated power are plotted. Approximately 20 dB power loss was incurred due to cable loss, the VDL transmitter antenna and the VDL receiver antenna.
4.2.4 Application Data FEC Distribution

The application data FEC distribution obtained from the flight test (Figure 4.21) follows the application data FEC distribution obtained from the bench test (Figure 4.10). The reason the FEC = 4 in Figure 4.21 is more than in Figure 4.10 is due to the fact that the power level is less. The histogram in Figure 4.22 indicates the application data FEC distribution during the flight test. The histogram plot obtained from the flight test (Figure 4.22) follows the plot that was obtained using the values from the bench test (Figure 4.8) and is consistent with the data shown in Table 4.6.
Figure 4.21 Application data FEC distribution (flight test)

Figure 4.22 Application data FEC histogram (flight test)
5. Conclusion and Recommendation

A VHF Data Link system was implemented using an E-Systems prototype VDL transmitter and receiver for LAAS/SCAT-I DGPS Navigation Augmentation Broadcast System. This system utilized D8PSK modulation and a TDMA structure. Performance of this system of the E-Systems equipment was evaluated and the results were summarized.

The firmware error that causes messages to be rejected by the prototype VDL transmitter should be corrected by the manufacturer. The VDL receiver sensitivity needs to be improved to achieve the required -87 dBm signal level within the specified service volume.

Instead of using two GPS receivers, only one receiver is needed at the airborne unit. The reason for using two GPS receivers is because of the convenience of 1PPS output from the Ashtech GPS receiver. The reference station in this system is designed to be used as an interface for broadcasting GPS correction data. Thus, one more serial port should be accessed by the QNX CPU to collect the correction data.

The recommendation for future work are: 1) derive the Bit Error Rate (BER) using this system implementation, 2) use different VDL transmitters to access different
time slots and test the performance of TDMA structure, and 3) test ground installation
and aircraft-specific antenna patterns.
REFERENCES:


[16] Hardware layout was designed and built by the author.


Appendix

1. VDL Transmitter Control Program

/* ------------------------------------------------------------------------
VDL Transmitter Interface Program Ver. 1.0
VDL.c, using QNX Watcom C.
OU-AEC-EECS
March, 1997
by MUdH/ FvG/ SY
-------------------------------------------------------------------------- */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <conio.h>
#include <time.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/dev.h>
#include <sys/irqinfo.h>
#include <sys/proxy.h>
#include <sys/kernel.h>
#include <math.h>
#include <i86.h>
#include "vdl.h"

double round(double);
void GenerateData(FILE *fp, int, int);

/* ------------------------------------------------------------------------
Define global variables
-------------------------------------------------------------------------- *

pid_t tick_proxy;
pid_t pps_proxy;
volatile UINT32 tick_counter;
volatile UINT32 pps_counter;
volatile UINT32 delta_counter;

UINT8 data[BUFFER_SIZE];
UINT8 ackdata[BUFFER_SIZE];
UINT8 outputdata[BUFFER_SIZE];

/* ------------------------------------------------------------------------
The hardware interrupt handler
-------------------------------------------------------------------------- */
#include <check_stack.h>

#define off( check_stack )

pid_t far tick_handler()
{
++tick_counter;

return tick_proxy;
}

pid_t far pps_handler()
{
    /* --------------------------------------------------
    Reset the tick_counter if a PPS comes in
    --------------------------------------------------- */
    delta_counter = tick_counter;
    ++pps_counter;
    tick_counter = 0;

    return pps_proxy;
}

/* ---------------------------------------------------------
Main program
----------------------------------------------------------- */
void main()
{
    /* ---------------------------------------------------------------------
    Local variables
    --------------------------------------------------------------------- */
    int id, i, ack, j;
    int n, armed;
    div_t a;
    int fd1, fd2, fd4, fd5, fd6;
    int n1, n2, n3, n4, n5, n6;
    int m1, m2, m3, m4, m5, m6;
    INT16 checksum = 0;
    INT16 dataflag[8] = {0};
    INT16 sendflag;
    unsigned tick, tijd;
    INT16 start, stop;
    INT16 strptr, endptr;
    struct timespec res;
    double frequency;
    char string[20];
    int h,d;
    int atten;
    unsigned char ffreq, freq;
    unsigned long nows, no_of_words;
    unsigned int seed, logflag;
    unsigned char timeslots=0;
    int msglength;
    UINT32 counter = 0;
    FILE *fp, *fd3;

    /* ----------------------------------------------- */
Get parameters from the input file

```c
if( (fp = fopen("init.dat","r")) == NULL)
    printf("Error reading init.dat\n");
for(i=0;i<4;i++) fscanf(fp,"%s",string);

/* Frequencies */
    fscanf(fp,"%lf",&frequency);
    h = (int)floor(frequency);
    freq = (unsigned char)h;
    h = (int)round(((frequency-freq)/0.025));
    ffreq = (unsigned char)h;

/* Time slots */
    for(i=0;i<10;i++) fscanf(fp,"%s", string);
    for(i=0;i<8;i++) { 
        fscanf(fp,"%d", &dataflag[i]);
        timeslots = timeslots | (dataflag[i] << i);
    }

/* Power Attenuation */
    for(i=0;i<2;i++) fscanf(fp,"%s", string);
    scanf(fp,"%d",&atten);
```

```c
/* Bytes in each message */
    for(i=0;i<3;i++) fscanf(fp,"%s", string);
    scanf(fp,"%d",&msglength);

/* number of messages */
    for(i=0;i<6;i++) fscanf(fp,"%s", string);
    scanf(fp,"%d",&nows);

for(i=0;i<6;i++) fscanf(fp,"%s", string);
    scanf(fp,"%d",&seed);

    for(i=0;i<6;i++) fscanf(fp,"%s", string);
    scanf(fp,"%d",&logflag);

    printf("%d %d\n", freq, ffreq);
    for(i=0;i<8;i++) printf("%d ", dataflag[i]);
    printf("%d\n",nows);
    printf("Seed value: %d\n",seed);
```

```c
srand(seed);
fclose(fp);
```

```c
/* --- Set the timer resolution --- */
```

```c
res.tv_nsec = RESOLUTION;
```
if( clock_setres( CLOCK_REALTIME, &res ) == -1 ) {
    printf("Clock set resolution failed\n");
    exit( EXIT_FAILURE );
}

/*@---------------------------------------------------------------------
    Get a proxy for the interrupt handler to kick
    and attach it to a timer
---------------------------------------------------------------------*/
if( (id = qnx_hint_attach( 0, &tick_handler, FP_SEG(&tick_counter) ) ) == -1 ) {
    printf("Unable to attach 'tick' interrupt.");
    return;
}

if( (id = qnx_hint_attach( 5, &pps_handler, FP_SEG(&pps_counter) ) ) == -1 ) {
    printf("Unable to attach 'PPS' interrupt.");
    return;
}

/*@---------------------------------------------------------------------
    Initializing the serial ports
---------------------------------------------------------------------*/
system("stty baud = 115200 < /dev/serl");
system("stty +ihflow +ohflow +lkhf ow < /dev/serl");
system("stty -isflow -osflow -lksflow < /dev/serl");
system("stty -ohpaged < /dev/serl");
system("stty +RTS +DTR < /dev/serl");

/*@---------------------------------------------------------------------
    Open a stream to the serial device connected to
    the VDL, attach a proxy to it and arm the proxy
---------------------------------------------------------------------*/
    fd1 = open("/dev/ser1", O_RDWR);

if(logflag)
    fd3 = fopen("outp.dat", "wb");

/*@---------------------------------------------------------------------
    MAIN LOOP
---------------------------------------------------------------------*/
outputdata[0] = 0x1b;
outputdata[1] = 0x03;
outputdata[2] = 0x0A;
outputdata[3] = 0xAA;
outputdata[4] = 223;

for(i=0;i<222;i++) {
    outputdata[5+i] = 0x55;
checksum = 0;
for(i=0;i<227;i++) checksum += outputdata[i];
outputdata[227] = (UINT8)(checksum & 0x00FF);

printf("\n\nMAIN LOOP: \n\n");
n = 0;
n1 = 0;
n2 = 0;
m1 = 0; m2 = 0; m3 = 0; m4 = 0; m5 = 0; m6 = 0;
strptr = 0;
endptr = 0;

/* Give the Task Command to the VDL Transmitter */
ack = TaskCommand(fd1);
if(ack) printf("Task command given \n\n");
else printf("Task command not acknowledged \n\n");

/* Give the VDL Transmitter tune command */
ack = TxTuneCommand(freq, ffreq, timeslots, fd1);
if(ack) printf("Tune command given \n\n");
else printf("Tune command not acknowledged \n\n");

/* Give the VDL Transmitter Power control command */
ack = TxPowerControl(atten, fd1);
if(ack) printf("Power command given \n\n");
else printf("Power command not acknowledged \n\n");

ack = 1;

no_of_words = 0;

while((no_of_words<=nows) && !kbhit()) {
    /* Extract the current time */
    tick = tick_counter;
    
    /* Check if desired slot is reached */
    a = div(tick % 1000, 125);
    if( (dataflag[(a.quot+1)%8]==1)&&(sendflag==1) &&
        (a.rem >=10) && (a.rem <= 100) )
        { write(fd1, outputdata, 228);
        /* Write to the output file */
        if(logflag)
fwrite(outputdata,sizeof(unsigned char),(msglength+6),fp);
start = tick_counter;
stop = start;
ack = 0;
sendflag = 0;
strptr = 0;
endptr = 0;
counter++;
printf("Time send: %ld Total send: %ld\n",tick,counter);

/ * Create new message (if desired) */

GenerateData(fd3, msglength, a.quot);
no_of_words++;

/ * End of new message */

} else if(a.rem >= 100 ) sendflag = 1;

/ * Read out the serial port number 1 */

n1 = dev_read(fd1,&data[endptr],BUFFER_SIZE,0,0,0,0,0);
endptr +=n1;
if(endptr >= BUFFER_SIZE) {
    strptr = 0;
    endptr = 0;
    printf("Buffer Overflow\n");
}

/ * Check if an Acknowledgement was received */

if( ack == 0 ) {
    stop = tick_counter;
    if((stop-start) < ACKWAIT) {
        if( (endptr-strptr) >= 17 ) {
            if( (data[strptr]==0x1b)&&(data[strptr+1]==0x03) ) {
                ack = 1;
            }
        }
    }
    else {
        printf("No Acknowledgement Received Fellow %ld \n",
                (stop-start));
        ack = 1;
    }
}
}
if(logflag)
    fclose(fd3);

exit( EXIT_SUCCESS );

void GenerateData(FILE *fp, int length, int slotid)
{
    /* Local variables */
    int i,tmp;
    static int mcnt1 = 0, mcnt2 =0, mcnt3 =0;
    unsigned int checksum;

    /* Determine the counter of the messages */
    if(++mcnt3==256) {
        mcnt3 = 0;
        mcnt2++;
    }
    if(mcnt2==256) {
        mcnt2 = 0;
        mcnt1++;
    }

    /* Fill the message */
    outputdata[0] = 0x001b; /* Sync byte # 1 */
    outputdata[1] = 0x0003; /* Sync byte # 2 */
    outputdata[2]= 0x000A; /* Application Data ID */
    outputdata[3]= 0x00aa; /* sequence number */
    outputdata[4]= length+1; /* size bytes */
    outputdata[5] = (unsigned char)mcnt1;
    outputdata[6] = (unsigned char)mcnt2;
    outputdata[7] = (unsigned char)mcnt3;
    outputdata[8] = (unsigned char)slotid;

    /* fill with circular sequence bytes */
    /* if(mcnt3 == 1)
    {
        for(i=0;i<((length-4));i++)
            outputdata[9+i] = i+1;
    }
    else
    {
        tmp = outputdata[length+4];
        for(i=((length+4)); i>=10; i--)
            outputdata[i] = outputdata[i-1];
        outputdata[9] = tmp;
    }*/

    /* Fill with random bytes */
    for(i=0;i<((length-4));i++) {
        tmp = (unsigned char)(rand() % 95);
        /* avoid 1Bh and 03h in the data messages */
        if ( ((tmp == 0x1b) || (tmp == 0x03))}
tmp++;  
outputdata[9+i] = tmp;

/* Compute checksum */
checksum = 0;
for(i=0;i<length+5;i++) checksum += outputdata[i];
outputdata[length+5] = (unsigned char)(checksum & 0x00ff);

int TaskCommand(int fd)
{
    /* Definition of local variables */
    INT16 i, status, n, m = 0;
    UINT8 taskbuf[10];
    INT16 ack = 0, checksum;
    INT16 start, stop;
    UINT8 buf[BUFFER_SIZE];
    UINT8 strptr = 0, endptr = 0;
    ACK *sys_par;

    /* Initialize local variables */
    taskbuf[0] = Ox1b; /* Sync byte # 1 */
    taskbuf[1] = Ox03; /* Sync byte # 2 */
    taskbuf[2] = Ox01; /* Task command ID */
    taskbuf[3] = Ox9b; /* Arbitrarily chosen sequence number */
    taskbuf[4] = Ox03; /* 3 bytes */
    taskbuf[5] = Ox05; /* TX */
    taskbuf[6] = Ox00; /* D8PSK */

    /* Compute the checksum */
    checksum = 0;
    for(i=0;i<7;i++) checksum += taskbuf[i];
taskbuf[7] = checksum & 255;

    /* Transmit the data */
    write(fd, taskbuf, 8*sizeof(unsigned char));

    /* Wait for an acknowledgement from the transmitter */
    start = tick_counter;
stop = start;

while( ((stop-start)<ACKWAIT)&&(ack==0) ) {

    /* Read the serial port */
    n = dev_read(fd, &buf[endptr], BUFFER_SIZE,0,0,0,0,0,0); endptr += n;

    /* Check if an acknowledgement has been found */
    if( (endptr-strptr) >= 17 ) {
        if( (buf[strptr] == Ox1b) &&(buf[strptr+1] == Ox03) ) {
            ack = 1;
            sys_par = (ACK *)&buf[strptr]);
        } else strptr++;
    }

    /* Read the timer again */
    stop = tick_counter;
}

if(ack == 1) {
    for(i=0;i<17;i++) printf( "%x ",buf[strptr+i]);
    printf("\n");
}

if(ack != 1) printf("Task command has never been acknowledged.\n");
else if( (sys_par->bit_taskerrors & Ox01) == 1 )
    printf("Tasking parameter error: %x \n", sys_par->bit_taskerrors);
return ack;
}

int TxPowerControl(int power, int fd)
{
    /*-----------------------------------
    Definition of local variables
    -----------------------------------*/
    INT16 i, status, n;
    UINT8 taskbuf[7];
    INT16 ack=0, checksum=0;
    INT16 start, stop;
    UINT8 buf[BUFFER_SIZE];
    UINT8 strptr = 0, endptr = 0;
    ACK *sys_par;

    /*-------------------------------------
    Initialize local variables
    -------------------------------------*/
    taskbuf[0] = 0x1b; /* Sync byte # 1 */
    taskbuf[1] = 0x03; /* Sync byte # 2 */
taskbuf[2] = 0x10; /* Power control ID */
taskbuf[3] = 0xb7; /* Arbitrarily chosen sequence number */
taskbuf[4] = 0x02; /* 2 bytes */
taskbuf[5] = power;

/* Compute the checksum */
checksum = 0;
for(i=0;i<6;i++) checksum += taskbuf[i];
taskbuf[6] = checksum & 255;

/* Transmit the data */
write(fd, taskbuf, 7*sizeof(unsigned char));

/* Wait for an acknowledgement from the transmitter */
start = tick_counter;
stop = start;

while( ((stop-start)<ACKWAIT)&&(ack==0) ) {
    /* Read the serial port */
    n = dev_read(fd, &buf[endptr], BUFFER_SIZE,0,0,0,0,0,0);
    endptr += n;
    /* Check if an acknowledgement has been found */
    if( (endptr-strptr) >= 17 ) {
        if( (buf[strptr] == Ox 1 b )&&(buf[strptr+1] == Ox03) ) {
            ack = 1;
            sys_par = (ACK *)&buf[strptr];
        } else strptr++;
    }
    /* Read the timer again */
    stop = tick_counter;
}

return ack;

int TxTuneCommand( unsigned char freq, unsigned char fracfreq,
                   unsigned char timeslot, int fd) {
    /* Definition of local variables */
INT16   i, status, n;
UINT8   taskbuf[10];
INT16   checksum = 0;
INT16   ack;
INT16   start, stop;
UINT8   buf[BUFFER_SIZE];
UINT8   strptr = 0, endptr = 0;
ACK     *sys_par;

/*-----------------------------------------------------*/
INT16   i,
/* ----------------------------------------------------
Initialize local variables
-----------------------------------------------------*/
#define 1
/* ----------------------------------------------------
Initialize local variables
-----------------------------------------------------*/

taskbuf[0] = 0x1b;  /* Sync byte # 1 */
taskbuf[1] = 0x03;  /* Sync byte # 2 */
taskbuf[2] = 0x20;  /* Tune control ID */
taskbuf[3] = 0xab;  /* Arbitrarily chosen sequence number */
taskbuf[4] = 0x04;  /* 4 bytes */
taskbuf[5] = freq;  /* Frequency */
taskbuf[6] = fracfreq;  /* Fraction of Frequency */
taskbuf[7] = timeslot;  /* Timeslot */

/*----------------------------------------------------------
Compute the checksum
-----------------------------------------------------------*/
checksum = 0;
for(i=0;i<8;i++) checksum += taskbuf[i];
taskbuf[8] = checksum & 255;

/*----------------------------------------------------------
Transmit the data
-----------------------------------------------------------*/
write(fd, taskbuf, 9*sizeof(unsigned char));

/*----------------------------------------------------------
Wait for an acknowledgement from the transmitter
-----------------------------------------------------------*/
start = tick_counter;
stop  = start;
while( ((stop-start)<ACKWAIT)&&(ack==0) ) {
  /* Read the serial port */
  n = dev_read(fd, &buf[endptr], BUFFER_SIZE,0,0,0,0,0,0);
  endptr += n;

  /* Check if an acknowledgement has been found */
  if( (endptr-strptr) >= 17 ) {
    if( (buf[strptr] == 0x1b)&(buf[strptr+1] == 0x03) ) {
ack = 1;
sys_par = (ACK *)&buf[strptr];
} else strptr++;

/* Read the timer again */

stop = tick_counter;

return ack;

}

double round(double a)
{
    double b,c;
    b = floor(a);
    if( (a-b)<0.5 ) c = b;
    else c = b+1.0;
    return c;
}
#ifndef VDL_H
#define VDL_H

/* Serial buffer sizes */
#define BUFFER_SIZE 2048 /* bytes */

/* Timer issues */
#define RESOLUTION 500000 /* nano-seconds */
#define ACKWAIT 20000000/RESOLUTION /* ticks */

/* Type definitions */
#define ACKSIZE 16

typedef char INT8;
typedef unsigned char UINT8;
typedef short signed int INT16;
typedef short unsigned int UINT16;
typedef long signed int INT32;
typedef long unsigned int UINT32;

/* Acknowledgement structure */

struct ACK {
    UINT8 sync1;
    UINT8 sync2;
    UINT8 messid;
    UINT8 seqno;
    UINT8 numbytes;
    UINT8 status;
    UINT8 id;
    UINT8 power;
    UINT8 type;
    UINT8 freq;
    UINT8 frac;
    UINT8 timeslot;
    UINT8 systemstate;
    UINT8 bit_poweron;
    UINT8 bit_continuous;
    UINT8 bit_taskerrors;
};
typedef struct ACK

#endif

ACK;
2 VDL Receiver Control Program

Programming by Borland C++ 3.1 (DOS version)
Project name: N4.prj
Execute File: N4.EXE
by SY, AEC-EECS-OU
March 1997

Include the following files:
2. COMSET4.C: all settings of the com port communication.
4. SERVICE.C: SCAT-1 receiver commands.

Program: B8.C
Functions: receiving data from VDL receiver and decoding.
           receiving data from NovAtel receiver and decoding.
Date: January 8, 1997
Update: 03/20/97

#include "b8.h"

in_data_t in_data; // received data structure
void getvd1(void);
void GetNov4(void);

int msgflag = 0,
msg_read = 0,
cksflag,
logflag = FALSE,
byte = 0,
getgps = FALSE,
gpsflag = 0,
gps_read = 0,
newgpsflag = 0,
vlddone = 0,
gpsdone = 0;
unsigned long message_num;
unsigned cks_errors;
float frequency;
double out[4];
unsigned long data_count = 0;

Main Function

void main(void)
tune_t t;
int i,h,key;
char help[40];
char com_mand[50];
int vect1(); // set interrupt vector for com1
int vect2(); // set interrupt vector for com2

clrscr();
gotoxy(5,10);
printf("Initializing system, please wait 5 seconds ...
");

init_com1(); // initialize com1
sprintf(com_mand,"COM1 38400,N,8,1,N,OFF");
for(i=0;i<strlen(com_mand);i++)
    sendcmd1(com_mand[i]);
sendcmd1(0x0d);
sendcmd1(0x0a);
delay(1500);

// Initialize com1 to 38400 baud rate
init_384();
// Give the LOG commands to the NovAtelligers
// Track the PXYB format data only!
sprintf(help,"%s","log com1,pxyb,ontime,1,0");
for(i=0;i<strlen(help);i++)
    sendcmd1(help[i]);
sendcmd1(0x0d);
sendcmd1(0x0a);
delay(1500);

init_com2(); /* initialize com2 */

t.time_slot=255;
t.frequency=112;
t.freq_fraction=38;
tune_receiver(&t);
_setcursortype(_NOCURSOR);
clrscr();
menu();
do{

    if(newmsgflag)
        getvdl();

    if(newgpsflag)
        GetNov4();

    if(logflag&&vlddone)
    {
        data_output(out_file);
        vlddone=FALSE;
    }
gpsdone = FALSE;
}

screen_print();

if(kbhit() != 0)
{
    key = getch();
    if(key == 27)
    {
        if(logflag)
        {
            logflag = FALSE;
            fclose(out_file);
        }
        update();
        clrscr();
        menu();
        if(exitflag)
            break;
    }
    if(key == 117)
    {
        update();
        clrscr();
    }
}
} while(!exitflag);

//re-init NovAtel receiver
sprintf(help, "%s", "unlogall");
for(i=0;i<10;i++)
{
    for(h=0;h<strlen(help);h++)
        sendcmd1(help[h]);
    sendcmd1(0x0d);
    sendcmd1(0x0a);
}

sprintf(help, "%s", "com1 9600,N,8,1,N,OFF");
for(i=0;i<strlen(help);i++)
    sendcmd1(help[i]);
sendcmd1(0x0d);
sendcmd1(0x0a);
delay(1500);
restore_vect1();
restore_vect2(); // restore irq vector table

="/****************************************************************************/

function to get the vdl data"
```c
void getvdl(void)
{
    for(byte=0;byte<rcv_data[msg_read][4]+5;byte++)
        store_data2(byte, rcv_data[msg_read][byte]);

    in_data.toa_total = in_data.toa[0]*256+in_data.toa[1];

    if (in_data.header[2]==0xa4) //a data message was received
        //check out the redundant first message from channel and discard
        if( (in_data.data[0] == 0x55) && (in_data.data[1] == 0x55)
            && (in_data.data[2] == 0x55) && (in_data.data[3] == 0x55))
        {
            in_data.total_hea_stat=0;
            in_data.total_app_stat=0;
            data_messages = 0;
            msg_read = (msg_read+1)%100;
            newmsgflag--; //erase one newmsgflag when read one msg out
        }

    in_data.total_hea_stat += in_data.hea_fec_stat;
    in_data.total_app_stat += in_data.app_fec_stat;
    message_num=in_data.data[2];
    message_num+=(in_data.data[1]*256)+(in_data.data[0]*256*256);
    cksflag = checksum_func();

    if(msg_counter>16)
        no_dataflag=TRUE;
    msg_read = (msg_read+1)%100;
    newmsgflag--; //erase one newmsgflag when read one msg out
    data_count++;
    vdldone=TRUE;
    return;
}

/*------------------------------------------------------------------------------------------
function to control program inputs
------------------------------------------------------------------------------------------*/
void menu(void)
{
    tune_t t;
    int answer=0;
    _setcursortypet( NORMALCURSOR);
    draw_box(17, 24, CLEAR);
gotoxy(14,1);
printf(" VDL RECEIVER PROGRAM\n");
gotoxy(14,2);
```
printf(" Avionics Engineering Center\n");
gotoxy(14,3);
printf(" Ohio University\n");
gotoxy(14,4);
printf(" version 1.0\n");
gotoxy(10,18);
printf("1) to exit program");
gotoxy(10,19);
printf("2) to tune receiver");
gotoxy(10,20);
printf("3) to view");
gotoxy(10,21);
printf("4) to open a file for logging record");
gotoxy(10,22);
printf("enter: ");

while((answer<49)||(answer>52)) // Wait until answer is 1-4
{
    gotoxy(17,22);
    printf(" ");
    gotoxy(17,22);
    answer = getch();
    printf("%c", answer);
}

if(answer == '1') // '1' exits the program
{
    gotoxy(10,23);
    printf("Exit--are you sure(y/n)? ");
    while(!kbhit);
    if((getch()=='y')||(getch()=='Y'))
    {
        exitflag=TRUE;
        restore_vect2(); /* restore irq vector table */
        _setcursortype(_NORMALCORSOR);
        clrscr();
        exit(0);
    }
    else
    {
    }
}
if(answer == '2') // tune the receiver
{
    if(newmsgflag)
    {
        for(byte=0;byte<rcv_data[msg_read][4]+5;byte++)
            store_data2(byte, rcv_data[msg_read][byte]);
        frequency=in_data.frequency+(.025*in_data.freq_fraction);
    }
msg_read = (msg_read+1)%100;
newmsgflag--;
}
draw_box(17, 24, CLEAR); //get the original frequency before tune
gotoxy(10,18);
printf("current frequency : %.3f", frequency);
tune_data(&t);
tune_receiver(&t);
do
{
    if(newmsgflag)
    {
        for(byte=0;byte<rcv_data[msg_read][4]+5; byte++)
            store_data2(byte, rcv_data[msg_read][byte]);
        frequency=in_data.frequency+(.025*in_data.freq_fraction);
        msg_read = (msg_read+1)%100;
        newmsgflag--;
    }
} while(frequency != t.freq_total); //wait until the receiver response
clrscr(); //the tune data.
return;
}
if(answer == '3') // return to program
{
    clrscr();
    return;
} if(answer == '4')
{
    clrscr();
    open_output();
    return;
}

/*------------------------------------------------------------------------
open an output logging file
------------------------------------------------------------------------*/
void open_output(void)
{
    char key,filename[13],dummy[13];
    FILE *stream;
draw_box(17, 24, CLEAR);
gotoxy(5,18);
printf("This will get logs from VDL receiver");
gotoxy(5,19);
printf("Enter name for logging file\"********\":(*.out )");
fflush(stdin);
gets(dummy);
sprintf(filename,"%s.out",dummy);
while((stream = fopen(filename,"r"):!= NULL)
{
    draw_box(17,24,CLEAR);
gotoxy(5,18);
printf("FILE ALREADY EXISTS!!\n");
gotoxy(5,22);
printf("Overwrite it? (Y/N) ");
do{
    while(!kbhit());
    key=getch();
}while(key != 'y' && key != 'Y' && key != 'n' && key != 'N');

if(key == 'y' || key == 'Y')
{
    fclose(stream);
    break;
}
if(key == 'n' || key == 'N')
{
    fclose(stream);
    draw_box(17,24,CLEAR);
gotoxy(5,18);
printf("Enter filename for output logging again: ");
fflush(stdin);
gets(dummy);
sprintf(filename,"%s.out",dummy);
}
if ((out_file = fopen(filename, "w+t")) == NULL)
{
    draw_box(17,24,CLEAR);
gotoxy(5,18);
printf("Cannot open output file.");
gotoxy(5,22);
printf("press a key to continue...");
while(!kbhit());
logflag = FALSE;
return;
}
else
{
    draw_box(17,24,CLEAR);
gotoxy(5,18);
printf("%s has been opened for logs output.", filename);
gotoxy(5,23);
printf("press a key to start logging...");
while(!kbhit());
crscr();
in_data.total_hear_stat=0;
in_data.total_app_stat=0;
logflag = TRUE;
}
void data_output(FILE *out_file)
{
    int i, j, k;
    char bar[1];

    if((in_data.header[2]==0xa4)&&(cksflag == 0))
    {
        if(getgps)
            fprintf(out_file, "%-11.2lf %-11.2lf %-11.2lf %-11.2lf\n",
                    out[0], out[1], out[2], out[3]);
        else
            fprintf(out_file, "00000000.00 00000000.00 00000000.00 00000000.00\n");

        fprintf(out_file, "%lu %i %i %s%x %s%x %i %i ",
                    message_num,
                    in_data.total_he_a_stat,
                    in_data.total_app_stat,
                    HEX(in_data.received_cks),
                    in_data.received_cks,
                    HEX(in_data.calculated_cks),
                    in_data.calculated_cks,
                    cks_errors);

        fprintf(out_file, "%s%d %d %d %i %i ",
                    SIGN(in_data.data_power),
                    ~((char)ina_data.data_power+1),
                    in_data.wordcount,
                    in_data.toa_total,
                    in_data.he_a_fec_stat,
                    in_data.app_fec_stat);

        for(i=0; i< 4; i++)
            fprintf(out_file,"%2X ", in_data.data[i]);

        fprintf(out_file,"\n\n");
    }
    else
    { // print acknowledgement
        if(gpsdone)
        {
            fprintf(out_file, "%-11.2lf %-11.2lf %-11.2lf %-11.2lf\n"
/* ---------------------------------------------------------------------------  
function to print the incoming message to the screen  
--------------------------------------------------------------------------- */

void screen_print(void) {
    short i, j, k;
    _setcursortype(_NOCURSOR);
    if(!logflag) {
        gotoxy(2,13);
        printf("NovAtel: GPS TIME     X     Y     Z");
        if(getgps) {
            gotoxy(11,14);
            printf("%-10.2lf %-11.2lf %-11.2lf %-11.2lf", 
                out[0],out[1],out[2],out[3]);
        }
    }

    frequency=in_data.frequency+(.025*in_data.freq_fraction);
    fprintf(out_file,"%.3f%si %3i %s%i %s%n"
            , frequency
            , stat_code()
            , in_data.id_command
            , SIGN(in_data.stat_power)
            , ~(char)in_data.stat_power+1
            , sys_state());

    fprintf(out_file,"%s%x	 %s%x	 %s%x
" 
            , HEX(in_data.bool_bit[0]), in_data.bool_bit[0]
            , HEX(in_data.bool_bit[1]), in_data.bool_bit[1]
            , HEX(in_data.bool_bit[2]), in_data.bool_bit[2]);*/

    fprintf(out_file, "%i %i %s%x %s%x %i ",&
                , in_data.total_hea_stat
                , in_data.total_app_stat
                , HEX(in_data.received_cks)
                , in_data.received_cks
                , HEX(in_data.calculated_cks)
                , in_data.calculated_cks
                , cks_errors);

    sprintf(bar, "0");
    fprintf(out_file, "%u %i %i %s%x %s%x %i ", 0, 0,bar, 0,bar, 0,0);
    for(i=0; i<4; i++)
        fprintf(out_file,"%2X ",0);
    fprintf(out_file,"\n\n");
}
gotoxy(5, 2);
printf("Messages received: \%lu", data_count);
gotoxy(39,2);
printf("Checksum errors: %d", cks_errors);
gotoxy(5, 3);
printf("Data messages: \%u", data_messages);
gotoxy(39,3);
printf("Status messages: \%lu", status_messages);
gotoxy(5,4);
printf("Received Checksum: \%s\%X Calculated Checksum: \%s\%X"
      , HEX(in_data.received_cks), in_data.received_cks
      , HEX(in_data.calculated_cks), in_data.calculated_cks);

frequency = in_data.frequency + (0.025 * in_data.freq_fraction);
gotoxy(5,5);
printf("Receiver Frequency: \%.3f MHz  Status code: \%s"
      , frequency, stat_code());
gotoxy(5,6);
printf("BIT Booleans: \%s\%X \%s\%X \%s\%X System State: \%s"
      , HEX(in_data.bool_bit[0]), in_data.bool_bit[0]
      , HEX(in_data.bool_bit[1]), in_data.bool_bit[1]
      , HEX(in_data.bool_bit[2]), in_data.bool_bit[2]
      , sys_state());
gotoxy(5,7);
printf("ID of Command: \%3i", in_data.id_command);
gotoxy(39,7);
printf("Status power: \%s\%i dBm", SIGN(in_data.stat_power)
      , ~(char)in_data.stat_power+1);

draw_box(1,12, NO_CLEAR);

if(no_dataflag)
{
    gotoxy(2,15);
    printf("Data messages:");
    for(i=16;i<26;i++)
    {
        gotoxy(1,i);
        printf(" ");
    }
    gotoxy(2,17);
    printf("Not receiving data messages!");
}
else
{
    gotoxy(5,8);
    printf("Total Header FEC: \%-5d  Total Data FEC: \%-5d",
            in_data.total_hea_stat, in_data.total_app_stat);
gotoxy(5,9);
printf("Header FEC: %3d Data FEC: %3d\n", in_data.hea_fec_stat, in_data.app_fec_stat);
gotoxy(5,10);
printf("Word Count: %-3d", in_data.wordcount);
gotoxy(39,10);
printf("ToA Delta: %-5d", in_data.toa_total);
gotoxy(5,11);
printf("Data messages:\n");
for(i=16;i<26;i++)
{
   printf("\n");
   printf("\n");
}
   //for print first 4 bytes only!!
gotoxy(2,16);
for(i=0; i<4; i++)
   printf("%-2X ",in_data.data[i]);
for(i=0; i<9; i++)
{
   gotoxy(2,i+16);
   for(j=0; j<26; j++)
   {
      if((i*26+j) < in_data.wordcount)
         printf("%-2X ", in_data.data[i*26+j]);
   }
} */
}
else
{
   gotoxy(2,13);
   printf("NovAtel: GPS TIME X Y Z\n");
   if(getgps)
   {
      gotoxy(11,14);
      printf("%-10.2lf %-11.2lf %-11.2lf %-11.2lf", out[0],out[1],out[2],out[3]);
   }
   gotoxy(5, 2);
   printf("Messages received: %lu", data_count);
gotoxy(39,2);
   printf("Checksum errors: %d", cks_errors);
gotoxy(5, 3);
   printf("Data messages: %u", data_messages);
gotoxy(5,4);
   printf("Received Checksum: %s%X Calculated Checksum: %s%X\n", HEX(in_data.received_cks), in_data.received_cks
, HEX(in_data.calculated_cks), in_data.calculated_cks);
gotoxy(39,3);
printf("Status messages: %lu", status_messages);
*/
}

/* ---------------------------------------------------------------------------
Function to store and decode one message into struct type in_data_t
--------------------------------------------------------------------------- */

void store_data2(int byte, int data)
{
    if ((byte>=0)&&(byte<=4))
        in_data.header[byte]=data;       // stores the header data
    else
    {
        //a data message was received
        if (in_data.header[2]==0xa4)
            message_case(byte, data);
        //a status message was received
        else if(in_data.header[2]==0xa3)
            status_case(byte, data);
    }
}

/* ---------------------------------------------------------------------------
function to store incoming status/ACK message
--------------------------------------------------------------------------- */

void status_case(int byte, int data)
{
    switch(byte)
    {
    case 5:
        in_data.status_code=data;       //stores the status code
        break;
    case 6:
        in_data.id_command=data;        //message sequence number
        break;
    case 7:
        in_data.stat_power=data;        //power in dBm
        break;
    case 8:
        in_data.signal_type=data;       //signal type, 0=D8PSK
        break;
    case 9:
        in_data.frequency=data;         //receiver integer frequency
        break;
    case 10:
in_data.freq_fraction=data;          //fractional part of the frequency
break;

case 11:
in_data.time_slot=data;             //ignored by receiver
break;

case 12:
in_data.system_state=data;
    //0=power on, 1=operational, 2=idle
break;

case 13: case 14: case 15:
in_data.bool_bit[byte-13]=data;
break;                            //see header file for details

case 16:
in_data.received_cks=data;         //get message checksum
  status_messages++;
  msg_counter++;
break;

}
case 9: case 10:
    in_data.toa[byte-9]=data;  //TOA delta of GPS 1 Hz. sync
    break;

case 11:
    in_data.hea_fec_stat=data;  //header FEC status
    break;

case 12:
    in_data.app_fec_stat=data;  //application data fec status
    break;

}/*
*--------------------------------------------------------------------------
* function to calculate the checksum
*--------------------------------------------------------------------------*/

int checksum_func(void)
{
    int cks, i;
    cks=0;
    for(i=0; i<5; i++)
        cks += in_data.header[i];
    if(in_data.header[2]==0xa4)  //a data message was received
    {
        cks += in_data.data_power;
        cks += in_data.wordcount;
        cks += in_data.spare[0];
        cks += in_data.spare[1];
        cks += in_data.toa[0];
        cks += in_data.toa[1];
        cks += in_data.hea_fec_stat;
        cks += in_data.app_fec_stat;
        for(i=0; i<in_data.wordcount; i++)
            cks += in_data.data[i];
    }
    else
    {
        cks += in_data.status_code;
        cks += in_data.id_command;
        cks += in_data.stat_power;
        cks += in_data.signal_type;
        cks += in_data.frequency;
        cks += in_data.freq_fraction;
        cks += in_data.time_slot;
        cks += in_data.system_state;
        cks += in_data.bool_bit[0];
        cks += in_data.bool_bit[1];
        cks += in_data.bool_bit[2];
in_data.calculated_cks = cks&0xFF;
if((in_data.calculated_cks-in_data.received_cks)!=0)
{
    cks_errors++;
    return TRUE;
}
else
    return FALSE;

/* --------------------------------------------------------------------------------
function to interpret the system state as a string
--------------------------------------------------------------------------------*/
char *sys_state(void)
{
    switch(in_data.system_state)
    {
    case 0:
        return ("Power on");
    case 1:
        return ("Operational");
    case 2:
        return ("Idle");
    default :
        return ("ERROR");
    }
}

/* --------------------------------------------------------------------------------
function to interpret the status code as a string
--------------------------------------------------------------------------------*/
char *stat_code(void)
{
    switch(in_data.status_code)
    {
    case 0:
        return ("Critical ERROR");
    case 1:
        return ("ACK");
    case 2:
        return ("Power on BIT");
    case 3:
        return ("Background BIT");
    default :
        return ("ERROR");
    }
/*---------------------------------------------------------------------------
function to round a number
---------------------------------------------------------------------------*/

int round(double number)
{
    if((number-floor(number))<0.5)
        return(floor(number));
    else
        return(ceil(number));
}

/*---------------------------------------------------------------------------
function to reset all data values
---------------------------------------------------------------------------*/

void update(void)
{
    int i;

    _setcursortype(_NOCURSOR);
    clrscr();
    if(viewmessageflag)
    {
        for(i=0; i<222; i++)
            in_data.data[i]=0;
        gotoxy(1,3);
        printf("Data message: ");
    }
    for(i=0; i<5; i++)
        in_data.header[i]=0;
    cks_errors =0;
    data_count =0;
    data_messages =0;
    status_messages =0;
    in_data.wordcount =0;
    in_data.spare[0] =0,
    in_data.spare[1] =0;
    in_data.toa[0] =0,
    in_data.toa[1] =0;
    in_data.toa_total =0;
    in_data.total_hea_stat =0;
    in_data.total_app_stat =0;
    in_data.hea_fec_stat =0;
    in_data.app_fec_stat =0;
    in_data.received_cks =0;
    in_data.calculated_cks =0;
    in_data.id_command =0;
    in_data.signal_type =0;
    in_data.freq_fraction =0;
in_data.time_slot = 0;
for(i=0; i<3; i++)
    in_data.bool_bit[i] = 0;
}

/*
 function to print a box
*/

void draw_box(int top_y, int bottom_y, int clear_stat)
{
    int i;

gotoxy(1, top_y);
printf("Eiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii"");
gotoxy(1, bottom_y);
printf("ttEiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii"");
for(i = top_y+1; i < bottom_y; i++)
{
    gotoxy(1, i);
    printf("ttEiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii"");
    gotoxy(80, i);
    printf("ttEiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii"");
}
for(i = top_y+1; i < bottom_y; i++)
{
    gotoxy(2, i);
    printf("ttEiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii"");
}
}
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include "comset4.h"
#include "service.h"

#define NO_CLEAR 0
#define TRUE 1
#define FALSE 0
#define CLEAR 1
#define OFFSET (long)sizeof(short)*228
#define HEX(x) ((x)<16 ? ("0x0") : ("0x"))
#define HEXSIZE(x) ((x)<16 ? ("0") : ("") )
#define SIGN(x) ((x)&(1 << 15) >= 1)?(x) == 0 ? ("") : ("-"))
#define TIMEX(x) ((x)<10 ? ("0") : ("")) /*macro for the time to print*/

typedef struct {
  short header[5]; //header[0] is sync byte 1 (always 27, 1bh)
  //header[1] is sync byte 2 (always 3, 03h)
  //header[2] is message ID (range 00h-fth)
  //header[3] is sequence # (range 0-255)
  //header[4] is number of data bytes
  int data_power; //measured power (range -127 to +128 dBm )
  int stat_power; //measured power (range -127 to +128 dBm )
  short wordcount; //message length (range 0-222)
  short spare[2]; //two unused bytes
  short toa[2]; //TOA Delta of GPS 1 Hz. sync (.5 symbol)
  short toa_total; //combination of toa_bytes (range 0-10499)
  short hea_stat; //header bits in error (range 0-2)
  short app_stat; //application data words in error (range 0-4)
  short total_heafstat; //total header bits in error
  short total_app_stat; //total application words in error
  short data[222]; //application data max 222 bytes
  short calculated_cks; //calculated checksum
  short received_cks; //received checksum
short status_code; //critical error=0, ACK=1, power on bit=2, background bit=3
short id_command; //sequence number (range 0-255)
short signal_type; //D8PSK=0, others for future growth
short frequency; //1 MHz steps (range 112-117)
short freq_fraction; //25 kHz each (range 0-39)
short time_slot; //echos tune command input data (range 00h-ffh)
short system_state; //power on=0, operational=1, idle=2
short bool_bit[3]; //bool_bit[0]=power on bit, 0=code integrity, //1=memory integrity, 2=processor integrity, //3-7=reserved;
//bool_bit[1]=continuous/initiated bit, //0-2=not used,3=PLL out of lock, 4=no receipt of //1 Hz pulse, 5=C51 not functioning, 6-7=reserved;
//bool_bit[2]=tasking errors, 0=invalid tasking parameters, 1=invalid tuning parameters, 2=not used, 3-7=reserved;

}

in_data_t;

 /*******************************************************************************************/
/* Function Prototypes */
 /*******************************************************************************************/

void store_data2(int, int); //stores input data
void screen_print(void); //prints output to screen
void message_case(int, int); //function to store incoming data message
void status_case(int, int); //function to store incoming status message
int round(double number); //function to round a number
void draw_box(int, int, int); //prints a box to the screen
void clearscreen(void); //clears screen and leaves options
char *sys_state(void); //interprets the system state as a string
char *stat_code(void); //interprets the status code as a string
void file_input(char *); //opens file for input comparisons
void menu(void); //menu function
int checksum_func(void); //calculates the checksum
void update(void); //resets all data values
void data_output(FILE *); //prints to output file every time data message is received
void open_output(void);

 /*******************************************************************************************/
/* global variables */
 /*******************************************************************************************/

FILE *out_file;
int rcv_data[100][INBUFFER]; // Buffer for rcv_data
int msg_counter=0; // initial the message counter to 0
int gps_data[10][GPSBUFFER];
unsigned data_byte=0;
unsigned data_messages=0;
unsigned long status_messages=0;
int readyflag=FALSE, exitflag=FALSE, // flag to exit program
no_dataflag=TRUE, // if no data is being received
viewmessageflag=FALSE, // used to view the data messages
messageflag=FALSE, // flag to show if message received was data
newmsgflag=0; // assume a new message received

#endif
```c
/* -------------------------------------------------------------
COMSET4.C
AEC-EECS-OU
DEC, 1996
--------------------------------------------------------------- */
#pragma inline
#include "eomset4.h"

/* global variables
--------------------------------------------------------------- */
extern int msgflag,
    msg_read,
    newmsgflag,
    gpsflag,
    gps_read,
    newgpsflag,
    rcv_data[10][INBUFFER],    /* Buffer for rcv_data */
    gps_data[10][GPSBUFFER];

unsigned in_byte,gps_byte;

void interrupt (*old_irq1)();
void interrupt (*old_irq2)();    //holds old irq interrupt vect

/* function to initialize the com port 1
--------------------------------------------------------------- */
void init_com1(void)
{
    asm cli            /* disable interrupts */
    in_port(COM1);     /* read tx register */
    out_port(DFR1, 0x80);  /* enable DLAB for Baud Selection */
    out_port(IER1, BL);
    out_port(COM1, BH96);
    out_port(DFR1, 0x03);    /* set baud rate for 9600 */
    out_port(OCR1, in_port(OCR1)|0x08); /* enable ser irq */
    out_port(IER1,0x01);    /* enable RxRDY irq */
    out_port(IEP1, in_port(IEP1)&IRQX_BIT1); /* enable serial irq x */
    asm sti            /* enable interrupts */
}

/* function to initialize the com port 2
--------------------------------------------------------------- */
void init_com2(void)
{
    asm cli            /* disable interrupts */
```

in_port(COM2);     /* read tx register */
out_port(DFR2, 0x80); /* enable DLAB for Baud Selection */
out_port(IER2, BL);
out_port(COM2, BH115);    // set baud rate to 115200
out_port(DFR2, 0x03);     /* 8-bit, no parity, 1 stop */
out_port(OCR2, in_port(OCR2)|0x08);/* enable ser irq */
out_port(IER2,0x01);     /* enable RxRDY irq */
out_port(IEP2, in_port(IEP2)&IRQX_BIT2);    /* enable serial irq x */
asm sti            /* enable interrupts */
*/
void init_384(void)
{
    asm cli          /* disable interrupts */
in_port(COM1);     /* read tx register */
out_port(DFR1, 0x80); /* enable DLAB for Baud Selection */
out_port(IER1, BL);
out_port(COM1, BH384);    /* set baud rate for 38400 */
out_port(DFR1, 0x03);     /* 8-bit, no parity, 1 stop */
out_port(OCR1, in_port(OCR1)|0x08);    /* enable ser irq */
out_port(IER1,0x01);     /* enable RxRDY irq */
out_port(IEP1, in_port(IEP1)&IRQX_BIT1);    /* enable serial irq x */
asm sti            /* enable interrupts */
}

void int_vect1(void)
{
    old_irq1=getvect(SER_INTNO1);  /* save old vect addr */
    setvect(SER_INTNO1,sio_int1);  /* SIO address in table */
}

void int_vect2(void)
{
    old_irq2=getvect(SER_INTNO2);  /* save old vect addr */
    setvect(SER_INTNO2,sio_int2);  /* SIO address in table */
}

void restore_vect1(void)
{
    setvect(SER_INTNO1,old_irq1);
}

void restore_vect2(void)
{
    setvect(SER_INTNO2,old_irq2);
}
void out_port(int port_id, int out_byte)
{
    asm mov ax, port_id
    asm mov dx, ax
    asm mov ax, out_byte
    asm out dx, al
    return;
}

int in_port(int port_id)
{
    asm mov ax, port_id
    asm mov dx, ax
    asm in al, dx
    asm mov ah, 0
    return(_AX);
}

 void interrupt sio_int2()
{
int status, tmp;
static int Has1b=FALSE, Hassyn=FALSE;

do 
{
    status= in_port(IIR2);   /* get irq pending */
    if(status == 4)
    {
        /* rcv data rdy */
        tmp= in_port(RxD2);   /* get data and store */
        if(Hassyn)
        {
            rcv_data[msgflag][in_byte]= tmp;
            in_byte++;
            if((in_byte>5)&&(in_byte>rcv_data[msgflag][4]+5))
            {
                Hassyn=FALSE;
                msgflag = (msgflag+1)%100;
                newmsgflag++;
                // how many msgs are ready in buffer
                if(newmsgflag > 99)
                {
                    newmsgflag = 99;
                    msg_read = (msg_read+1)%100;
                }
            }
        }
    }

}
else if(tmp == 27)
    Haslb = TRUE;
else if((tmp == 3)&&(Haslb != 0))
{
    Hassyn = TRUE;
    rcv_data[msgflag][0] = 27;
    rcv_data[msgflag][1] = 3;
    in_byte = 2;
}
else
    Haslb = FALSE;
}
while(in_port(IIR2) != 1); /* see no irqs pending */
asm mov al,ICP2 /* specific EOI irq */
asm out ICP2,al
/* ---------------------------------------------------------------------
self calling function to get one byte of data from the
initialized com1 
--------------------------------------------------------------------- */
void interrupt sio_int1(){
    int status,tmp;
    static int Hasaa = FALSE,
              Has44 = FALSE,
              Novsync = FALSE;
    do
    {
        status=in_port(IIR1); /* get irq pending */
        if(status == 4)
        {
            tmp = in_port(RxD1); /* rcv data rdy */
            if(Novsync)
            {
                gps_data[gpsflag][gps_byte] = tmp;
                gps_byte++;
                if(gps_byte >= 88)
                {
                    Novsync = FALSE;
                    gpsflag = (gpsflag+1)%10;
                    newgpsflag++;
                    if(newgpsflag > 9)
                    {
                        newgpsflag = 9;
                        gps_read = (gps_read+1)%10;
                    }
                }
            }
        }
    }
} else if(tmp == 0xAA)
    Hasaa = TRUE;
else if(Hasaa && (tmp == 0x44))
{
    Has44 = TRUE;
    Hasaa = FALSE;
}
else if((tmp == 0x11) && Has44)
{
    Novsync = TRUE;
    Has44 = FALSE;
    gps_data[gpsflag][0] = 0xAA;
    gps_data[gpsflag][1] = 0x44;
    gps_data[gpsflag][2] = 0x11;
    gps_byte = 3;
}
else
{
    Has44 = FALSE;
    Hasaa = FALSE;
}
} while(in_port(IIRl) != 1); /* see no irqs pending */

asm mov al,ICPI /* specific EOI irq */
asm out ICP1,al
}
/* -------------------------------------------------------------
COMSET4.h
-------------------------------------------------------------*/

#ifndef _COMSET4_H_
defme _COMSET4_H_

/* -------------------------------------------------------------
included files
-------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#include <time.h>

/* Rev data register */
/* line status register */
/* transmit data register */
/* data format register */
/* output control register */

(Continued on next page)

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#include <time.h>

#define INBUFFER 240
#define GPSBUFFER 100
#define TRUE 1
#define FALSE 0

/* -------------------------------------------------------------
com port define statements
-------------------------------------------------------------*/
#define ICP1 0x20 /* interrupt control port */
#define ICP2 0x20
#define IEP1 0x21 /* interrupt enable port */
#define IEP2 0x21
#define IRQX_BIT1 0x0ef /* 0x0f7 COM2, 0x0ef COM1 */
#define IRQX_BIT2 0x0f7 /* 0x0f7 COM2, 0x0ef COM1 */
#define SER_INTNO1 0x0c /* 0x0b COM2, 0x0c COM1 */
#define SER_INTNO2 0x0b /* 0x0b COM2, 0x0c COM1 */
#define COM1 0x03f8
#define COM2 0x02f8 /* 0x03f8 COM1, 0x02f8 COM2 */
#define RxD1 COM1
#define RxD2 COM2 /* Rcv data register */
#define TxD1 COM1
#define TxD2 COM2 /* transmit data register */
#define IER1 COM1+1 /* interrupt enable register */
#define IER2 COM2+1
#define IIR1 COM1+2 /* interrupt identification register */
#define IIR2 COM2+2
#define DFR1 COM1+3 /* data format register */
#define DFR2 COM2+3
#define OCR1 COM1+4 /* output control register */
#define OCR2 COM2+4
#define LSR1 COM1+5 /* line status register */
#define LSR2 COM2+5

#define BH24 0x30 /* Baud High 2400 */
#define BH48 0x18 /* Baud High 4800 */
#define BH96 0x0c  /* Baud High 9600 */
#define BH192 0x06  /* Baud High 19.2k */
#define BH384 0x03  /* Baud High 38.4k */
#define BH576 0x02  /* Baud High 57.6k */
#define BH115 0x01  /* Baud High 115k */
#define BL 0x00    /* Baud Low all rates 2400 - 38.4k */

void out_port(int, int); // sends byte out specified port
int in_port(int);         // returns byte from port
void int_vect1(void);     // adj interrupt vect table for new ISR
void int_vect2(void);     // restores interrupt vect table
void restore_vect1(void); // initializes com port
void restore_vect2(void); // interrupt service routine for serial port
void init_com1(void);
void init_com2(void);
void init_384(void);
void interrupt_sio_int1();
void interrupt_sio_int2();
#endif
/* *******************************************************
GETNOV4.C
AEC-EECS-OU
November, 1996
******************************************************* */

#include <stdlib.h>
#include <alloc.h>
#include <stdio.h>
#include <conio.h>
#define TIME_OFFSET 16
#define X_OFFSET 24
#define Y_OFFSET 32
#define Z_OFFSET 40
#define GPSBUFFER 100

extern int
gps_data[10][GPSBUFFER],
getgps,
gpsflag,
gps_read,
newgpsflag;
extern double
out[4];
unsigned char
buffer[88];

void GetNov4(void)
{
    int i, abyte;
    char check;
    double gpstime, ecef[3];
    char *char_buff;

    for(abyte = 0; abyte < 88; abyte++)
        buffer[abyte] = gps_data[gps_read][abyte];

    gps_read = (gps_read+1)%10;
    newgpsflag--;

    // Calculate the check sum
    check = buffer[0];
    for(i=1;i<88;i++)
        check = check ^ buffer[i];
    check = check ^ buffer[3];

    if(check != buffer[3])
    {
        gotoxy(11,14);
        printf("Check sum failed! ");
        getgps = 0;
        return;
    }
    else
char_buff = (char *) calloc(8, sizeof(char));

for(i=0;i<8;i++)
    char_buff[i] = buffer[TIME_OFFSET+i];
gpstime = *((double *) char_buff);

for(i=0;i<8;i++)
    char_buff[i] = buffer[X.Offset+i];
ecef[0] = *((double *) char_buff);

for(i=0;i<8;i++)
    char_buff[i] = buffer[Y.Offset+i];
ecef[1] = *((double *) char_buff);

for(i=0;i<8;i++)
    char_buff[i] = buffer[Z.Offset+i];
ecef[2] = *((double *) char_buff);

// set the output sign, and write the output
getgps = 1;
out[0] = gpstime;
out[1] = ecef[0];
out[2] = ecef[1];
out[3] = ecef[2];

// Reset counter and synchronization flag
free(char_buff);
/*------------------------------------------------------------
SERVICE.C
AEC-EECS-OU
OCT, 1996
------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include "comset4.h"

typedef
struct
{
    float freq_total;       // the complete frequency
    short frequency;        // 1 MHz steps (range 112-117)
    short freq_fraction;    // 25 kHz each (range 0-39)
    short time_slot;        // time slot assignment (range 00h-ffh)
}
tune_t;

.prototype of functions
--------------------------------------------------------------*/

void sync(void);           // synchronize 1st byte(27) and 2nd byte(03)
void tune_data(tune_t *);  // function to get tuning data from user
void tune_receiver(tune_t *); // called to tune the receiver
void task_cmd(void);       // tasking command for receiver
void init_bit_cmd(void);   // tells receiver to run initiated bit
void idle_cmd(void);       // puts receiver into idle state
void exit_idle_cmd(void);  // returns receiver from idle state
                          // to normal state
void shutdown_cmd(void);   // shuts down the receiver
void sendcmd(int, int *);  // function to round a number
void sendcmd1(int);
int round(double number);

/*--------------------------------------------------------------
function to give tasking command to receiver or transmitter
--------------------------------------------------------------*/

void task_cmd(void)       // tasking command for VDL receiver
{
    int i, status;
    int taskbuf[8] = {0x1b, 0x03, 0x01, 0, 3, 0, 0, 34&0xff};
    sendcmd(8,taskbuf);
}
void idle_cmd(void)  // idle command for VDL receiver
{
    int i, status;
    int taskbuf[6] = {0x1b, 0x03, 0x02, 0, 1, 33&0xff};
    sendcmd(6, taskbuf);
}

void exit_idle_cmd(void)  // exit idle command for VDL receiver
{
    int i, status;
    int taskbuf[6] = {0x1b, 0x03, 0x03, 0, 1, 34&0xff};
    sendcmd(6, taskbuf);
}

void shutdown_cmd(void)
{
    int i, status;
    int taskbuf[6] = {0x1b, 0x03, 0x04, 0, 1, 35&0xff};
    sendcmd(6, taskbuf);
}

void init_bit_cmd(void)
{
    int i, status;
    int taskbuf[6] = {0x1b, 0x03, 0x09, 0, 1, 40&0xff};
    sendcmd(6, taskbuf);
}

void tune_receiver(tune_t *t)
{
    int i, status, cks=0;
int taskbuf[10];
t->time_slot=255; // set time slots even the receiver doesn’t use it

taskbuf[0] = 27;
taskbuf[1] = 3;
taskbuf[2] = 0x20;
taskbuf[3] = 0;
taskbuf[4] = 4;
taskbuf[5] = t->frequency;
taskbuf[6] = t->freq_fraction;
taskbuf[7] = t->time_slot;

for(i=0;i<8;i++)
{
    cks+=taskbuf[i];
}
taskbuf[8]=cks&255;

sendcmd(9,taskbuf);
return;

/***************************************************************************/

void tune_data(tune_t *tx)
{
    float freqval=0;
    char freqstrg[10]=""
    while((freqval<112.0)&&(freqval>117.975))
    {
        gotoxy(10,20);
        printf("in .025 MHz increments");
        gotoxy(10,19);
        printf("Enter transmitter frequency(112.000 - 117.975 MHz): ");
        fgets(freqstrg, sizeof(freqstrg), stdin);
        sscanf(freqstrg, "%f", &freqval);
    }

    tx->frequency=floor(freqval);
    tx->freq_fraction=round((freqval-(float)floor(freqval))/0.025));
    tx->freq_total=(double)tx->frequency+(double)tx->freq_fraction*0.025;
}

/***************************************************************************/

function to send A command to VDL receiver

/******************************************************************************/
void sendcmd(int j, int *taskbuf)
{
    int i=0, status=0;
    for(i=0; i<j; i++)
    {
        status = 0;
        while((status&0x20) != 0x20)
        {
            asm mov ax, LSR2
            asm mov dx, ax
            asm in al, dx
            asm xor ah, ah
            status = AX;
        }
        out_port(TxD2, *(taskbuf+i));  // Transmit data.
    }
    delay(100);
}

/*============================================================================
 function to send A command to NovAtel receiver
============================================================================*/

void sendcmd1(int i)
{
    int status=0;
    while((status&0x20) != 0x20)
    {
        asm mov ax, LSR1
        asm mov dx, ax
        asm in al, dx
        asm xor ah, ah
        status = AX;
    }
    out_port(TxD1, i);  // Transmit data.
/*--------------------------------------------------------------
SERVICE.H
--------------------------------------------------------------*/

#ifndef SERVICE_H
#define SERVICE_H

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#include <time.h>

typedef struct
{
    float freq_total;   // the complete frequency
    short frequency;    // 1 MHz steps (range 112-117)
    short freq_fraction; // 25 kHz each (range 0-39)
    short time_slot;    // time slot assignment (range 00-ffh)
} tune_t;

tunet;

/*--------------------------------------------------------------
prototype of functions
--------------------------------------------------------------*/

void sync(void);        // synchronize 1st byte(27) and 2nd byte(03)
void tune_data(tune_t *); // function to get tuning data from user
void tune_receiver(tune_t *); // called to tune the receiver
void task_cmd(void);    // tasking command for receiver
void init_bit_cmd(void); // tells receiver to run initiated bit
void idle_cmd(void);    // puts receiver into idle state
void exit_idlecmd(void); // returns receiver from idle state
    // to normal state
void shutdown_cmd(void); // shuts down the receiver
void sendcmd(int, int *);    
void sendcmd1(int);
#endif