INITIAL CHARACTERIZATION OF OBIC IMU

AND

DESIGN OF A DATAC BUS INTERFACE

A Thesis presented to the
Faculty of the College of Engineering and Technology
Ohio University

In partial fulfillment of the requirements for
the degree
Master of Science

by,
Navin G. Mathur,

November 1993
| TABLE OF CONTENTS |
|-------------------|----------------|
| List of figures   | iii            |
| 1. INTRODUCTION   | 1              |
| 2. INERTIAL MEASUREMENT UNIT | 4 |
| 2.1. Inertial Navigation | 4 |
| 2.2 Displacement Computations | 5 |
| 2.3. THE Q-BIC Inertial Measurement Unit | 7 |
| 2.3.1. Overview of the Q-BIC IMU sensor | 7 |
| 2.3.2. Operation of the Q-BIC IMU sensor | 11 |
| 2.3.3. Q-BIC Model | 16 |
| 2.4. Interface to the Q-BIC IMU | 18 |
| 2.4.1. A/D Conversion | 18 |
| 2.4.2. Data Collection | 19 |
| 2.4.2.1. 8751 program algorithm | 19 |
| 2.4.2.2. C program for data collection | 20 |
| 2.4.2.3. Initial Q-BIC performance results | 21 |
| 3. DATAC TERMINAL INTERFACE | 26 |
| 3.1. Introduction | 26 |
| 3.2. DATAC operation and capabilities | 27 |
| 3.3. DATAC Terminal user Interface | 31 |
| 3.3.1. Address Strobe | 32 |
| 3.3.2. Data Strobe | 33 |
| 3.3.3. Read - Write | 33 |
| 3.4. Electrical Interface to the DATAC terminal | 33 |
| 3.5. Interface between the DATAC Terminal and the PC | 34 |
parallel port

3.6. Algorithm Used in 8751 for DATAC Terminal Interface 37

4. SUMMARY AND CONCLUSIONS

REFERENCES 43

APPENDICES

A. ASSEMBLY CODE FOR 8751 IN THE A/D CONVERTER CIRCUIT 46
B. ASSEMBLY CODE FOR THE IBM PS/2 FOR SERIAL AND PARALLEL INTERFACING 51
C. C PROGRAM USED TO COLLECT 8 CHANNEL DATA SERIALLY VIA A/D CONVERTER CIRCUITRY 61
D. CONNECTOR CONFIGURATION FOR THE DATAC TERMINAL INTERFACE 65
E. ASSEMBLY CODE FOR 8751 IN DATAC TERMINAL INTERFACE 67
F. C PROGRAM TO TEST THE DATAC TERMINAL INTERFACE 73
G. GPS DATA FORMAT 76

Acknowledgements 85
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Block diagram of the system</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Q-BIC IMU Sensor Unit</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Q-BIC IMU proof mass[4]</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Exploded view of the Q-BIC IMU sensor[4]</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Cross section of the Q-BIC IMU sensor[4]</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Q-BIC IMU sensor electronics</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Effect of linear acceleration</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.7a</td>
<td>Arrangement of the Q-BIC IMU undergoing angular rotation</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.7b</td>
<td>Q-BIC IMU circuit arrangement for gyroscope servo</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>Change in the cross section of the x and y capacitor due to a z-direction acceleration</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>Circuit Diagram for 12 bit, 8 channel A/D converter</td>
<td>21</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>Q-BIC IMU Sensor data output</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2.11</td>
<td>Histogram of the signal outputs from the Q-BIC IMU Sensor</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Overview of the DATAC bus</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Block diagram of a DATAC terminal</td>
<td>28</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Configuration of the Shared Interface RAM</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Timing of Signals on DATAC Terminal User Interface</td>
<td>32</td>
</tr>
</tbody>
</table>
Figure 3.5  Circuit diagram for the Interface between 36 DATAC Terminal and the PC parallel port
1. INTRODUCTION

Inertial measurements such as linear and angular accelerations or angular rates can be used to maintain the attitude and position of a vehicle with respect to some initial attitude and position. The inertial measurements are integrated once or twice over time to find the change in attitude and position. Because of this, inertially derived information is only as accurate as the initial conditions and the accuracy will degrade as a function of time. Typical position degradations are on the order of 1-2 nautical miles per hour.

When inertial sensors were first introduced for positioning purposes, they were inaccurate, bulky, expensive and relied heavily on precision mechanical devices such as gimbals. Inertial navigation technology improved significantly in the 1960's with the use of digital computers and strapdown sensors.

Strapdown sensors are directly mounted on to a vehicle, rather than trying to maintain a level platform using gimbals. All necessary coordinate rotations are performed by the computer. Toward the later part of 1980's, the development of low cost, medium accuracy inertial sensors begun. At the same time, a new satellite-based positioning system, the Global Positioning System (GPS), started to approach its initial operational capability. The GPS provides position data with an accuracy on the order of 100m (95%). The integration of Inertial measurements with GPS-derived position data is very beneficial for the following reasons:
1) inertial measurements are autonomous and provide good short-term stability

2) the GPS provides long-term stability.

In addition, almost all current GPS receivers provide position updates only once per second, while the latency of the position data typically exceeds 0.5 seconds. To use GPS for high-accuracy guidance purposes, it is necessary to increase the number of measurements to at least 5 times per second. Also, GPS can experience temporary outages due to radio frequency interference or satellite shielding and shadowing. A relatively simple inertial measurement unit could be used to overcome these problems.

This thesis presents the theory of operation and the initial characterization of a low cost Inertial Measurement Unit (IMU). Next, the interface between the IMU and a Personal Computer (PC) is described. For flight evaluation purposes, the position data must be made available to the flight control computer. For this project, the target aircraft is the Transport Systems Research Vehicle (TSRV) Boeing 737-100 operated by the National Aeronautics and Space Administration (NASA) Langley Research Center. The TSRV uses a Digital Autonomous Terminal Access Communication (DATAC) bus for all data transfers. A general purpose interface was designed and tested to transfer position data from a PC to a terminal connected to the DATAC bus. Figure 1.1 shows the block diagram of the overall system.

The analog navigational data from the IMU is in the form of accelerations. These data are converted to a serial, digital data
stream using a microcontroller and an Analog-to-Digital (A/D) converter. The digital data is processed by the PC and the resulting position information is sent to the DATAC Terminal Interface (DTI). The DTI is implemented using a microcontroller to buffer the data and to satisfy the timing requirements for the DATAC terminal.

The actual position calculations based on the inertial measurement data are outside the scope of this thesis, but GPS-derived position data was used to test the DTI onboard the TSRV.
2. INERTIAL MEASUREMENT UNIT

2.1. INERTIAL NAVIGATION

"Neither position nor velocity can be sensed Inertially"[14]. However, a few principals of Inertia can be used to determine the linear and angular accelerations of a moving body with respect to a reference frame. These accelerations can then be used to find the velocity and position of a moving body. Inertial Navigation is the navigation of a body using the principles of Inertia. The fundamentals of Inertial Navigation lie in three laws of motion suggested by Sir Isaac Newton.

Newton’s First law of motion: Every body remains at rest or in a constant velocity along a straight path unless it is acted upon by an external force.

Newton’s Second law of motion: A force acting on the body causes it to accelerate in the direction of the force, and the acceleration is directly proportional to the mass of the body.

Newton’s Third law of motion: Every action has an equal and opposite reaction. These laws apply to both linear as well as angular motion.

The unit used to measure linear and angular accelerations or angular rates is referred to as an IMU (Inertial Measurement Unit). An IMU has two major components which are the specific force transducer and some type of gyroscope. The specific force transducer measures the magnitude of the force acting upon the transducer in a particular direction. A specific force transducer is usually referred to as an accelerometer; however, it should be
recognized that an accelerometer measures specific force rather
than acceleration (e.g. a free falling accelerometer will indicate
zero acceleration although it is subjected to the earth's
gravitational field).

The conventional accelerometer includes a mass/spring system
which responds to the external forces. Either capacitive,
inductive, optical or mechanical pickoffs can be used to sense the
displacement of the mass due to an external force.
The traditional gyroscope consists of a spinning wheel supported by
a gimbal, which provides the spin axis of the wheel one degree of
freedom to tilt relative to the base. Due to the rapid rotation of
the wheel, it provides a resistance against external angular
rotation such that the wheel maintains its orientation in space.
This process generates a torque in the direction perpendicular to
the wheel's angular rotation. The magnitude of this torque depends
upon the external angular rate. Thus the gyro provides the angular
rate of the body.

Instead of measuring angular rates using a gyroscope, an
angular accelerometer could be used as well. Integration of the
angular accelerations would also provide angular rates. An angular
accelerometer consist of two closely spaced linear accelerometers.

2.2. DISPLACEMENT COMPUTATIONS

In free space, let \( a_x \) be the acceleration of an object in the
\( x \) direction, and let \( x_0 \) be the initial \( x \)-coordinate. Then the
displacement, \( x - x_0 \), is given by equation 1.
\[ x-x_0 = \int_0^t (a_x \, dt) \, dt \tag{1} \]

Similar expressions exist for the displacements in the y and z directions.

In the presence of a non-rotating earth, the measured acceleration is given by:

\[ A = a + g \tag{2} \]

where \( a \) is the spatial radial acceleration of the object and \( g \) is the earth's gravitational force.

Adding the rotation of the earth, and accounting for the movement of the object with respect to the earth, the spatial radial acceleration can be given as [18]:

\[ a = A - (\omega + \omega_0) \times V - g \tag{3} \]

The first term on the right hand side of equation (3) is the measured acceleration, the second term is the CORRIOLIS CORRECTION where \( \omega_0 \) is the angular velocity of the earth, and \( \omega \) is the angular velocity of the object with respect to the earth. \( V \) is the linear velocity of the object with respect to the earth. The third term in the equation is the gravitational field correction.

The gravity at sea level is given by:

\[ g = 978.049(1 + 0.00529 \sin^2(\phi)) \text{ cm/sec}^2 \tag{4} \]

Where \( \phi \) is the geodetic latitude. The equation holds to within 0.02 cm/sec\(^2\). The gravity decreases approximately \( 10^{-6} \) \( g \) per 10 ft increase in altitude above sea level [16].
The gyro output is used to determine the orientation of the platform the Inertial Measurement Unit is mounted on. Once the platform is aligned, changes in velocity will be sensed by the appropriate accelerometers and would be integrated twice as given in equation 1 to compute the vehicle’s current position[14].

2.3. THE Q-BIC INERTIAL MEASUREMENT UNIT

Advances in technology since 1980 have resulted in many types of low cost IMU’s. One such sensor is the Q-BIC IMU developed by M. Morrison. The Q-BIC IMU provides three angular accelerations and three linear accelerations. The operation of this sensor involves the concepts of Inertia and magnetic levitation (MAGLEV). The levitation is provided by a simple Law "Opposite poles attract and similar poles repel". The sensor weighs approximately 50 grams and, according to the inventor, is capable of micro-g accuracy in three orthogonal linear directions and 0.01 degrees per hour stability in three orthogonal angular degrees of freedom[3].

The actual arrangement of the Q-BIC IMU sensor in terms of its axes, electronics and connector is shown in figure 2.1.

2.3.1. OVERVIEW OF THE Q-BIC IMU SENSOR

Figure 2.2 shows the Q-BIC IMU proof mass. Each of the six faces of the proof mass has six capacitance plates numbered 54 and three permanent magnets with magnet caps numbered 56. Internally, i.e inside the support numbered 52, there are connections between m and n, p and o; and q and r on each face.
Figure 2.1 Q-BIC IMU Sensor unit

Figure 2.2 Q-BIC IMU proof mass
[from: M.morrison U.S. Patent # 4711125]

Figure 2.3 shows the exploded view of the complete sensor. The proof mass in figure 2.2 is contained in the frame numbered 32 and the proof mass is positioned using the supports numbered 38 on each face. The supports numbered 38 have the second plates (40) of each capacitor on the sides of the proof mass (50). It also has coils
numbered 42 and provides the pins numbered 39 for the capacitive pickoffs and to provide current to the coils. The direction of the currents through the coils provide the direction of the magnetic field which either attracts or repells the magnet on the proof mass. Once in operation, there is no physical contact between the outer assembly and the proof mass. The cube is sealed properly to avoid any external electrical interference.

Additional details about the cube are shown in figure 2.4
which shows a cross section of the cube. Here the function of each part of the cube can be seen. Element 70 is a permanent magnet and element 72 is a magnet cap. Coils are wrapped around element 72 from element 42 and the current flowing through the coil is provided by the external pins 39g and 39h. There is no direct connection between the coil and the magnet cap. A current through the coil will give rise to a magnetic field which in turn exerts a force on the magnetic cap. This force results in the displacement of the proof mass. Parts 40a, 54m, 40d and 54n form a pair of capacitors with an air gap (73) in between them. Part 62 connects the two capacitors. These capacitors are used to sense the displacement of the proof mass. Next, currents are generated in the coils to provide repulsion or attraction to the permanent magnet to keep the proof mass in the centralized position.

Figure 2.5 shows the sensor electronics of the Q-BIC IMU sensor. There are 3 accelerometer servo circuits labeled 100 and 3 gyro servo circuits labeled 90. These circuits observe the change in impedance between two input capacitors and provide proper current outputs to the coils. Part 110 is a sine wave generator circuitry.

The capacitances between b and e on each face are responsible for picking up the linear displacement of the proof mass and the corresponding coils responsible for keeping the cube in the centralized position are between contacts i and j on each of the faces. In a similar way, capacitances between a-d and between c-f are responsible for sensing the angular displacement of the proof
mass and the coils between g-h and k-l maintain the cube centralized. The connections between the d and f points on opposite faces of the proof mass are discussed in the next section.

2.3.2. OPERATION OF THE Q-BIC IMU SENSOR

The operation of the Q-BIC is based on capacitive sensing and magnetic levitation. A change in capacitance value gives rise to
the proper amount of current in a proper direction through the coils via the sensor electronics.

According to Newton’s Laws of motion a body remains at rest or in constant velocity unless acted upon by an external force and an external force applied to a body gives it acceleration. When the IMU assembly is subjected to an acceleration, because of the Inertia the proof mass will try to remain in place while the outer
assembly moves ahead. In figure 2.3, assume that the cube is accelerating in the positive z direction. In this case the distance between two capacitor plates on side 3 would decrease while on side 1 the distance would increase. Based on the fact that for a parallel plate capacitor, capacitance is inversely proportional to the distance between the plates, the capacitance on side 1 would decrease while the capacitance on side 3 it would increase. From figure 2.6 it can be seen that there is a significant difference of impedance on the input to the accelerometer circuitry.

![Diagram](image)

Figure 2.6 Effect of Linear acceleration
This change in impedance can be utilized to vary the gain of an operational amplifier, which in turn would provide an estimate of the acceleration the cube is undergoing. The accelerometer circuitry provides two outputs, one is for the gain variation and determines the amount and direction of the current through the coils while the second output provides the estimate of the linear acceleration to the user.

Next assume that the cube is rotated in the z-y direction. The operation of the gyro is different from the accelerometer since here the capacitances are connected differently on opposite faces.

Figure 2.7a Arrangement of the Q-BIC IMU undergoing angular rotation

Figure 2.7a shows how the plates and the IMU assembly are arranged when subjected to angular acceleration. The gyro electronics senses the angular displacement via the capacitors as shown in figure 2.7b. When the angle between the proof mass and the
outer assembly is equal to $\Theta$, the capacitance between $a$, $d$ on face 1 decreases as well as the capacitance between $c$ and $f$ on side 3. Similarly, the capacitance between $c$ and $f$ on side 1 increases as well as the capacitance between $a$ and $d$ on side 3.

![DIAGRAM](image)

Figure 2.7(b) 0-BIC IMU circuit arrangement for gyroscope servo

The capacitance for each of the square plate capacitors is given by [17]:

$$C = \varepsilon_0 \frac{a^2}{d} \left(1 + a \frac{\Theta}{2d}\right)$$

(5)

Where $d$ is the distance between the plates, $a$ is length of one side of the square plate, $\varepsilon_0$ is the permittivity of the free space and $\Theta$ is the angle between the plates of the capacitor.

Because of the cross coupling of the capacitors between side 1 and side 3, the gyro circuitry is most sensitive to angular changes. When the cube moves in the positive $z$ direction, the angular acceleration in the $x$-$y$ direction is not affected since
capacitances on side 1 decrease while on side 3 the capacitance increase and therefore the gyro sees no difference in the impedance. On faces 2, 4, 5 and 6 a small difference in the impedance is observed. As shown in figure 2.8, there could be a small change in the area of the capacitors, but this will not result in a change in the linear x and y accelerations.

Similarly, if the cube is constructed symmetrically, there will be no cross-coupling between the angular accelerations either. However, in practice some cross-coupling will exist between all outputs.

2.3.3. Q-BIC MODEL

As seen in the previous section, each output from the cube is dependent on other signals. This dependence of the signals on one another can be modelled in the form of a set of equations as shown
A = accelerations

o = output signal from the Q-BIC

B = bias

C = coefficient

NL = nonlinearity

GC = gravity correction

SF = scale factor

**LINEAR ACCELERATION EQUATIONS**

\[
A_x = o_x SF_x + NL_x o_x^2 + B_x + GC_x + o_y C_{x-y} + o_z C_{x-z} + o_{xy} C_{x-xy} + o_{xz} C_{x-xz} + o_{yz} C_{x-yz}
\]

\[
A_y = o_y SF_y + NL_y o_y^2 + B_y + GC_y + o_x C_{y-x} + o_z C_{y-z} + o_{yx} C_{y-xy} + o_{yz} C_{y-yz}
\]

\[
A_z = o_z SF_z + NL_z o_z^2 + B_z + GC_z + o_y C_{z-y} + o_x C_{z-x} + o_{yz} C_{z-yz} + o_{xz} C_{z-xz} + o_{yz} C_{z-yz}
\]

**ANGULAR ACCELERATION EQUATIONS**

\[
A_{xy} = o_{xy} SF_{xy} + NL_{xy} o_{xy}^2 + B_{xy} + o_y C_{xy-y} + o_z C_{xy-z} + o_x C_{xy-x} + o_{xz} C_{xy-xz} + o_{yz} C_{xy-yz}
\]

\[
A_{xz} = o_{xz} SF_{xz} + NL_{xz} o_{xz}^2 + B_{xz} + o_y C_{xz-y} + o_z C_{xz-z} + o_x C_{xz-x} + o_{xy} C_{xz-xy} + o_{yz} C_{xz-yz}
\]

\[
A_{yz} = o_{yz} SF_{yz} + NL_{yz} o_{yz}^2 + B_{yz} + o_y C_{yz-y} + o_z C_{yz-z} + o_x C_{yz-x} + o_{xz} C_{yz-xz} + o_{xy} C_{yz-xy}
\]

Also included are nonlinearity coefficients, which are due to the fringing effects in the capacitor and due to the nonlinearity of the resistances used in the sensor circuitry. The gravity correction is known a priori. The biases and scale factors for the
linear accelerations can be found by obtaining the outputs due to gravity for the positive and negative X, Y and Z directions. All other coefficients and biases should be estimated using a truth reference system.

2.4. INTERFACE TO THE Q-BIC IMU

The Q-BIC IMU provides six analog output signals which correspond to the three linear and the three angular accelerations. These signals are output by the amplifiers in the sensor electronics. To analyze these signals, an A/D converter circuit was designed and implemented which interfaces the Q-BIC to a personal computer using a serial communication interface.

2.4.1. A/D CONVERSION

To interface the A/D converter to the computer, an 8-bit microcontroller 8751H was selected (more about this integrated circuit (IC) is explained in chapter three). For the A/D conversion MAXIM's MAX180 was selected, which is a 12 bit A/D converter with 8 multiplexed input channels. This IC takes one selected analog signal at a time and gives out a 'busy' signal while the conversion is in process. It has the capability of providing the complete 12 bit output in parallel. A signal RD (active low) tells the IC to start the conversion. The IC can operate in either an unipolar or differential input mode. For this circuit the differential input mode is used. The full scale is -2.5V to 2.5V which results in a quantization size of $5/2^{12} = 1.221$ mV. More about the IC can be
found in the data sheet of the MAX 180 [18].

The complete circuit for the A/D converter is given in figure 2.9. Port 0 of the 8751 is used to select one of the eight input channels, Port 1 is used to collect the lower 8 output bits; Port 2 is used to collect the higher 4 output bits and to read the 'busy' signal; Port 3 is used to provide the serial data output. A quartz crystal oscillating at 11.0592 MHz is used to obtain a desired baud rate of 9600 bits per second. An IC 1232 is used to convert the serial output from the 8751 to a voltage level compatible with the PC serial communications input.

A crystal of 1.2288 MHz is used for the MAX180. Out of 8 input signals, 6 are the signals from the Q-BIC IMU, one is ground and the last is a generated reference voltage.

Along with the A/D converter output data collection, the 8751 also performs averaging of the input signals. It is this averaged signal which is given out to the PC. The algorithm used in the 8751 is discussed next.

2.4.2. DATA COLLECTION

2.4.2.1. 8751 program algorithm

Table 1 provides the algorithm for the 8751 microcontroller used for the A/D conversion and interface with a PC. See appendix A for a complete listing of the assembly language code.

Provided that the input signal is noisy and has a sufficient bandwidth, the digital averaging increases the resolution and accuracy of the analog signal by a factor of
Initialization of 8751 for 9600 baud
Enable the serial output for 1 start and 1 stop bit
select channel 0
Repeat the following forever
  Initialize the memory from 68H to 7FH to zero
  repeat 256 times
    Repeat for all 8 channels
      Start the A/D conversion
      Read the output from the MAX180 when ready
      Add the results from all 8 channels to each of their sums
      Divide the 8 outputs by 256
      Output the data to the serial interface

Algorithm 1: Algorithm for A/D conversion

\[ \sqrt{256} = 16 \]  \hspace{1cm} (6)

This in effect increases the A/D conversion from 12 bits to 16 bits. After the averaging, the circuitry provides 8 updates per second for each of the eight channels.

2.4.2.2. C program for data collection

A C-language program was written to receive the data from the A/D converter and to process the results. This program simply receives and decodes the data stream from the A/D converter, and writes the data to a file.

The program reads from the serial communication port using assembly language routines (See appendix B for a listing of the
Figure 2.9 Circuit diagram for 12 bit, 8 channel A/D converter
Assembly language routines). Receipt of the data is interrupt driven[15]. Every time a byte arrives at the PC serial port, an interrupt is generated and the byte is stored in an input data buffer. The C program checks the data buffer for new data. If new data is received, the program checks if the data is a preamble or the converted digital data from the Q-BIC IMU. Then it collects 16 bytes of the data, 2 bytes of data for each channel. Next it checks if the input is negative or positive based on the most significant bit of the input data. Then it scales the data properly and writes the data to disk. See Appendix C for a complete listing of the C program.

2.4.2.3 Initial Q-BIC performance results

Figure 2.10 shows static data collected from the Q-BIC IMU using the above algorithms and circuits. The data clearly shows the settling time of each channel and the effect of the acceleration when gravity is acting in the x direction and no other linear or angular accelerations are acting on the Q-BIC IMU. The graph shows the settling time to be on the order of 20 minutes. This time depends on the channel and the output from the channel.

The readings were taken three times and the results were similar and hence can be concluded that the repeatability of the sensor is good. The histogram of the signals is shown in figure 2.11, the graph reveals that, after stabilization, the noise on the signal is Gaussian and it also shows that quantization noise in the ground and reference voltage signal is exactly \( \frac{5}{2^{16}} \), which is the 16 bit
accuracy obtained by digital averaging.

The noise level on both the ground signal and the voltage reference is on the order of 0.1 mV rms. The noise on the Q-BIC outputs is on the order of 1 mV rms. Therefore, the A/D converter is adequate to allow an initial assessment of the Q-BIC performance. Table 2.1 summarizes the noise levels on the Q-BIC IMU outputs. Based on a 1-g (9.8 m/s²) output of approximately 0.5 volts, it can be concluded that the noise level on the acceleration is of the order of

\[
\left(\frac{0.001}{0.5}\right) \times 9.8 = 2 \text{cm/s}^2
\]

The resulting positioning accuracy depends on the accuracy and stability of the angular accelerations, and the algorithm used. The complete characterization of the Q-BIC IMU is fairly involved and is outside the scope of this thesis.

<table>
<thead>
<tr>
<th>OUTPUT SIGNAL</th>
<th>Std. Deviation(mV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X LINEAR ACCELERATION</td>
<td>0.0010</td>
</tr>
<tr>
<td>Y LINEAR ACCELERATION</td>
<td>0.0003</td>
</tr>
<tr>
<td>Z LINEAR ACCELERATION</td>
<td>0.0007</td>
</tr>
<tr>
<td>XY ANGULAR ACCELERATION</td>
<td>0.0013</td>
</tr>
<tr>
<td>XZ ANGULAR ACCELERATION</td>
<td>0.0006</td>
</tr>
<tr>
<td>YZ ANGULAR ACCELERATION</td>
<td>0.0014</td>
</tr>
</tbody>
</table>

Table 2.1 RMS noise levels for the Q-BIC IMU output signals
Figure 2.10 Q-BIC IMU Sensor Data output
Figure 2.11 Histogram of the signal outputs from the Q-BIC IMU Sensor
3. DATA TERMINAL INTERFACE

3.1. INTRODUCTION

The need for a communication bus on aircraft was recognized when aircraft were computerized and a great deal of data was to be communicated between the computers and the systems on board the aircraft. A node to node connection is one of the options, but it would be bulky and also would add complexity to the overall aircraft system. At the same time, the connections should provide compatibility for several different kinds of avionics equipments.

DATA TERMINAL INTERFACE (DATAC) is one of the answers to the above problems. DATAC stands for Digital Autonomous Terminal Access Communication. It is a digital data bus designed for communications between aircraft computers and aircraft systems. DATA TERMINAL INTERFACE (DATAC) evolved from ARINC (Aeronautical Radio Incorporated) specification 429, and is also referred to as ARINC 629. It is a single bus which runs throughout the aircraft, which provides weight savings as compared to the node to node connections. The operation of the bus is multiplexed, which also reduces the complexity of the system. The bus has DATA TERMINAL INTERFACE (DATAC) terminals at several places which provide the interfacing between the avionics equipment and the bus. More details about the DATA TERMINAL INTERFACE (DATAC) terminals is provided in the next section. Due to large vibration on board the aircraft, in ordinary connections there is a fair amount of chances for wire breakage during operation. In case of the DATA TERMINAL INTERFACE (DATAC) bus, instead of using stubs to connect equipment to the bus, a simple inductive coupler is used, so there is no physical
splicing[6]. The bus allows for adding or removing the DATA terminals with a minimal change in the software. DATA operates very similar to a Local Area Network like ETHERNET. The topology of DATA bus is linear, hence it has plug-in capability almost anywhere along the length of the cable. Another feature of DATA is that it functions 'memory-to-memory'. As data moves out of various terminals, addresses are generated which directs data into the memory of one or more subsystems[6]. The bus itself usually consists of a twisted pair of wires, Fiber Optics can be used in environments with a high level of interference.

![Figure 3.1 Overview of the DATA bus](image)

3.2. DATA OPERATION AND CAPABILITIES

Figure 3.1 shows a block diagram of the DATA data bus. The bus runs linearly and it has DATA terminals at various places, which provide user equipment with access to the bus. The DATA terminals shown in the figure 3.1 are the interfaces used
to translate the input signal from the user into the DATAC format. Figure 3.2 shows the block diagram of a DATAC terminal. The terminal contains a receiver and a transmitter which enables the user to access the bus. These interfaces work as the terminal manager and as bus couplers. The terminal monitors the correct formatting of the data sent and also ensures that only one system uses the DATAC bus at a particular time. In the terminal, the receiver section collects the data from the demodulator and decodes it to produce 16-bit parallel data words. The transmitter section takes 16-bit parallel data words, adds a synchronization waveform and a parity bit and sends it to the modulator section. The data is transmitted serially in Manchester II format[7].
The DATAC protocol is a CSMA/CD, Carrier Sense Multiple Access with Collision Detection which is similar to the technique used in a LAN. Along with CSMA/CD the DATAC also uses the concept of Time Division Multiplexing (TDM), where each user gets to use the complete bandwidth of the bus for a small amount of time. Before sending the data to the bus, the terminal monitors the activity on the bus and in particular listens for a very specific interval of silence. This interval is sometimes called the 'bus quiet time'. The silent period is programmed into the terminal and is unique for each bus terminal[7]. When the terminal detects this interval it sends the data out. The bus access is gained by the terminal during the bus quiet time, but in order to remove the ambiguity as to which terminal should send data, each terminal is equipped with a timer and a unique value of the quiet time, also called a Terminal Gap (TG).

When the start of the bus quiet time is detected, each terminal counter starts counting to its TG value and whichever terminal reaches its TG value first assumes its right to send the data over the bus. When one of the terminals starts transmitting data, other terminals reset their counters and wait for the next 'bus quiet time'. Once the terminal has completed its transmission, its own timer is inhibited from counting until at least one other timing qualifier is satisfied[7]. This ensures that the terminal having the shortest TG doesn't monopolize the bus.

Once gaining access to the bus, the terminal starts sending the data. Each message sent across the bus has a 16-bit preamble
label, which identifies the type of data in the message. If a terminal is programmed to receive a certain preamble, it will process the data following the preamble and pass it on to the attached system. The terminal provides an interrupt signal to the user system when a label is received. It also generates a processing vector from its memory, and passes this vector to the connected system[7].

To ensure that all the terminals can send their data without collisions, the operation of the bus can be divided into four major modes: Mode A, Mode B, Mode C, and Scat-C. The DATAC is usually operated in mode A. One complete set of data transmissions from all the terminals is called a FRAME. In mode A, the transmission from a terminal is periodic and all the frames are of similar length and duration. The period of transmission is chosen long enough for all the terminals to send data. This period is called a Major Frame and typically lasts for 50 ms. This Major Frame can be subdivided into Minor Frames, which allow a terminal to access the bus more often. Other modes are defined with respect to the periodicity of the frames, see reference [6].

Technically, the characteristics of DATAC can be summarized as follows:
1) Data rate of DATAC is 1 Mbits per second.
2) Maximum number of terminals that can be connected to DATAC = 63.
   Maximum message length = 31 Strings.
3) Word length = 16 bits.
4) Address length = 11 bits.
5) String length = label + n data words.
   \[ n = 2^p - 1 \quad [p = 1 \text{ to } 16] \]

3.3. DATAC TERMINAL USER INTERFACE

The purpose of the user interface to the DATAC terminal is to write data in to the DATAC terminal memory at particular locations. A two-port memory called Shared Interface RAM (SIR) is used. The configuration of the SIR is shown in figure 3.3. The user system sends data and addresses to the SIR. The data is then transmitted to a certain target system depending on which address the user writes the data to. The SIR has 2048 data locations (11-
bit address) for 16 bit data words. In addition to the address and data the user also has to provide certain control signals with proper timing.

These control signals are the Address strobe (AS), the Data strobe (DS) and the Read-write (R/W). A typical timing of the signals is shown in figure 3.4. The following sections discuss these signals in detail.

3.3.1. Address Strobe

The Address Strobe is a simple square wave signal. The positive half-cycle of the address strobe provides the memory access to the DATAC port while the negative half-cycle initiates the memory access for the user port. The timing on this waveform is not very critical, but the period of the wave should be within 2 to 8 ms and it should run continuously irrespective of the data being transferred or not. During the negative transition of the AS,
the DATAC terminal receives the address data from the user.

3.3.2. Data Strobe

As shown in figure 3.4, the data strobe is a rectangular wave with a minimum of 500 ns negative transition. The negative transition of the wave should be completely contained within the negative transition of the Address Strobe. Like the Address Strobe, this wave should also run continuously. For this reason, the Data strobe is usually derived from the Address Strobe. Both these waves are used by the internal logic of the DATAC terminal and are required even in the absence of a data transfer. For data being written to DATAC, the negative transition of the Data Strobe signals DATAC that the data is stable on the databus. The DATAC terminal latches the data at this point and terminates the memory access.

3.3.3. Read-Write

This signal decides the direction of the data flow to or from the user. When high, data is read from the DATAC terminal and when low, data is written into the DATAC terminal. Since the Address Strobe and the Data Strobe are continuous, this signal provides a method of defining reading or writing for the DATAC terminal.

3.4. ELECTRICAL INTERFACE TO THE DATAC TERMINAL

As discussed in section 3.3, an interface to the DATAC terminal has 11 address bits, 16 data bits and three control lines.
The interface is provided through a 63 pin AMP connector (number 205842-1). The definition of the connector pins are provided in appendix D. Along with Address bits (AB00 - AB10), data bits (DB00 - DB15), Address strobe (AS), Data Strobe (DS), read-write (R/W), the DATAC terminal provides the user with a special 10 ms clock signal. This signal lets the user interface know that the DATAC terminal bus is ready to accept new addresses and data on the lines. Upon receipt of a 10ms clock pulse, all data must be transmitted to the DATAC terminal within the next 2 ms. All signals are TTL compatible. Address and data buses are active high while address strobe, data strobe and Write are active low.

3.5 INTERFACE BETWEEN THE DATAC TERMINAL AND THE PC PARALLEL PORT

The heart of the interface is a 8751H 8-bit microcontroller. This microcontroller has 256 byte of on-chip data Random Access Memory (RAM) and 4K bytes of programmable memory. It has 4 I/O ports, each individually programmable with 8 I/O lines each. The chip can handle 5 interrupt sources and is capable of performing serial communication with variable baud rate. The chip can operate at frequencies from 3.5 MHz to 16 MHz. Figure 3.5 shows the circuit diagram for the interface between the DATAC terminal and the PC. Port 0 of the 8751H is used as a 8-bit input port from the PC. Port 1 is used as the output port for the DATAC terminal. Port 2 is used to select the data or the addresses for the DATAC terminal and is also used to signal the computer when the 8751H cannot accept data. In addition, one pin of port 2 is used as the read-write output
control for the DATAC terminal. External interrupt 0 is used to handle the 10 ms synchronization from the DATAC terminal and external interrupt 1 is used to handle interrupts from the PC. During normal operation of the 8751H, the ALE pin provides a rectangular wave with a frequency 1/6 of the connected parallel oscillator. This signal is used to generate the address strobe and data strobe signals. IBM PS/2’s parallel port was used for communication with the circuit. Appendix B gives the assembly code for the IBM PS/2 for the parallel communication. Two subroutines called par_data and par_cont are used for the data word and the control signals, respectively. Par_data puts 8 bits of data on the data lines while par_cont provides an interrupt signal on the STB (active low) line. Three J-K flip flops were used to divide the ALE signal to obtain a square wave of 250 KHz which is used as the AS signal. This signal is buffered and is provided to the DATAC terminal. Similarly two single-shot circuits were used to give proper timing for the DS signal using the AS signal. This signal is also buffered before it is sent to the DATAC terminal.

The constraint on the read-write line is that it should be low for at least one negative transition of the address strobe while writing to the DATAC. A NAND gate, a J-K flip-flop and two single-shot circuit are used to satisfy this requirement as shown in figure 3.5.

The latches (74LS374) for the address bus are always enabled and are given addresses through port 1 and a clock pulse from port 2 to latch the address data. The operation of the data latches is
Figure 3.5 Circuit Diagram for the Interface between the DATAC Terminal and the PC parallel port
similar except that they are enabled only when the read-write signal goes low.

3.6. ALGORITHM USED IN THE 8751H FOR THE DATA AC TERMINAL INTERFACE

The message format from the PC to the DATA AC Terminal Interface (DTI) is as follows:

\[ \$, \text{ LSB}_1, \text{ MSB}_1, \text{ LSB}_2, \text{ MSB}_2, \ldots, \text{ LSB}_{11}, \text{ MSB}_{11}, <\text{CR}> \]

where:

\$
\quad \text{is the synchronization character: hexadecimal 24}

\text{LSB}_n \text{ is the least significant byte for data word } n \ (n = 1, 11)

\text{MSB}_n \text{ is the most significant byte for data word } n \ (n = 1, 11)

<\text{CR}> \text{ is the carriage return character: hexadecimal 0D}

The length of one message is 24 bytes. The DTI maintains two 24-byte data buffers.

The double buffering is necessary to ensure that always a complete set of data is send to the DATA AC Terminal. In the case of a single buffer, it could happen that halve the bytes are from a new message and halve the byte are from an old message.

Initially, the DTI starts filling the first data buffer. Upon receipt of 24 bytes, the DTI checks for the existence of the \$
\text{ at the beginning of the buffer and the } <\text{CR}> \text{ at the end of the buffer. If either the } \$ \text{ or the } <\text{CR}> \text{ is not present, the buffer is shifted left, such that the oldest byte is discarded. Upon receipt of the next byte, the DTI will again check for the } \$ \text{ and } <\text{CR}> \text{ and shift left if the check is negative. This process continues until a } \$ \text{ is found in the first byte and a } <\text{CR}> \text{ is found in the 24th
byte. At that time, the DTI sets a flag that buf1 is available and starts filling the second data buffer (buf2). If buf2 is filled, the DTI sets a flag that buf2 is available and starts filling the first data buffer.

There are eight registers available in the 8751H (R0-R7) but only registers R0 and R1 can be used for addressing the internal RAM. In the internal RAM, only locations 30H to 7FH can be used for general purpose. PSW.1 and PSW.5 are user definable flags. The complete algorithm can be divided into three parts

1) Initialization
2) Interrupt service routine for the PC parallel port.
3) Interrupt service routine for the DATAC terminal.

These parts of the algorithm are shown in algorithm 2.

| R0 = 40H (start of buffer 1) |
| R1 = 60H (start of buffer 2) |
| R6 = 00H (number of bytes in buffer initially) |
| psw.1 = 0 (select buffer 1) |
| psw.5 = 0 (set no data flag) |

Algorithm 2a Initialization

The circuit was implemented and tested successfully with the C program given in appendix F. Recently, the interface has also been successfully tested in flight on a Boeing 737 operated by the National Aeronautics and Space Administration (NASA) Langley Research Center. For this test, GPS position data was transmitted
from the PC to the DATAC terminal using the DATAC terminal

store the byte in active buffer  
increment byte count (R6)  
if complete message is received (R6 = 18)  
  Check if it is the first buffer or the second buffer  
    if the first buffer  
      if there is a $ in the beginning and <CR> at the end  
        shift buffer 1 to the left  
        decrement byte count  
      else  
        select second buffer  
        set the data present flag  
        reset the byte count  
    else  
      select second buffer  
    end  
  else  
    point to the second buffer  
    If there is a $ in the beginning and <CR> at the end  
      shift second buffer to left  
      decrement the byte count  
    else  
      select buffer 1  
      set the data present flag  
      reset the byte counter  
    end  
end

return from interrupt

Algorithm 2b Interrupt from PC parallel port

interface. Appendix G shows the format for the GPS data. The transfer of a set of data (11 words) takes approximately 0.2 ms which satisfy the 2 ms requirement (see section 3.4)
save the accumulator
save memory pointer (R0)
R2 = 01H  (MSB of DATAC address)
R3 = 60H  (LSB of DATAC address)
if there is no data then return from Interrupt
  if we are filing buffer 1
    select buffer 2
    set read write high (i.e read only)
    increment R0 (not to send $)
  else
    select buffer 1
    set the read-write high
    increment R0 (not to send $)
end
for R3 = 60H to 6AH in step of 1
  (Send 11 data words)
    pl = R2 ; clock the first latch with p2.3
    (High address byte)
    pl = R3 ; clock the second latch with p2.2
    (Low address byte)
    pl = @R0 ; clock the third latch with p2.1
    (Low data byte)
    increment R0
    pl = @R0 ; clock the fourth latch with p2.0
    (High data byte)
    increment R0
    clock on p2.7  (Give out the read/write pulse)
end
retrieved R0 and accumulator contents

return from interrupt

Algorithm 2c Interrupt from DATAC terminal
SUMMARY AND CONCLUSION

The theory of operation and an initial characterization of a low cost inertial measurement unit (IMU) is presented. An interface was designed to convert the analog signals from the Q-BIC IMU to a serial data stream compatible with a Personal Computer (PC). The interface is based on a microcontroller and an 8-channel 12-bit Analog-to-digital (A/D) converter. Using digital conversion, the effective number of bits for the A/D conversion was increased from 12 to 16 bits. It was found that the warm-up time for the Q-BIC is approximately 20 minutes, after which stable readings are available with a noise level on the order of 1 mV root-mean-square (rms) on a +2.5 to - 2.5 Volts scale. Linear acceleration noise for the Q-BIC IMU is approximately 2 cm/sec² on a per sample basis at a rate of 8 samples per second. The resolution of the A/D converter is approximately an order of magnitude better than the noise level on the Q-BIC IMU measurement. This justifies the use of the interface presented in this thesis for the characterization of the Q-BIC IMU.

A second interface was designed and implemented to transfer position data from a PC parallel port to a DATAC bus terminal. This DATAC Terminal Interface (DTI) is also configured around a microcontroller, which buffers the data and satisfies the DATAC terminal timing requirements. The DTI was successfully flight tested on the TSRV Boeing 737 operated by NASA Langley Research Center.

A fully functional version of the Q-BIC IMU can be integrated with
the Global Positioning System (GPS) to provide an inexpensive, compact and accurate position computing device for navigational purposes.
REFERENCES


APPENDIX A

ASSEMBLY CODE FOR 8751H IN THE A/D CONVERTER CIRCUIT
The following 8751 assembly code is used in the EEPROM of the 8751 used in the A/D conversion circuitry.

; IO1:  Program for 8-channel "voltmeter"
; August 25, 1992

init:  mov  psw,#00H          ; program status word = 0
       mov  sp,#30H          ; stack points to internal ram
       mov  scon,#0101110B   ; 8 data bits, 1 start, 1 stop
       mov  tmod,#0000000B   ; timer1 is baud rate clock (only use high four bits)

mov  tcon,#00H          ; set timer control port
mov  pcon,#00H          ; set smod bit to zero
mov  ie,#00H            ; disable interrupts
mov  ip,#00H            ; set priority level to lowest
mov  thl,-3             ; set baud rate to 9600
setb  trl               ; turn on baud rate clock
ljmp  loop0             ; jump to main program

sum:  mov  r7,#0ffh        ; add ffH as the last byte
       orl  a,#11110000B     ; make the most sig bits to 1

ljmp  sumdo             ; copy character into r7
chrout: mov  r7,a         ; wait unitl the last one is done
jnb   ti,$
clr   ti
mov   a,r7
mov   sbuf,a
ret

; ----- Start of program

loop0: mov  r1,#68h        ; point to mem loc 68H

loop8: mov  @r1,#00h       ; clean all the mem loc from 68h-7fh
inc   r1
cjne  r1,#80h,loop8       ; inc reg count
mov   r3,#0ffh            ; clean the complete memory
         ; count for # of bits
loop1: mov r0,#55h ; point to temp storage
    mov r2,#00001000B ; mem loc
    ; common to keep RD' high
    ; & ch select
loop2: mov p0,r2 ; set RD' high & select
    nop
    anl p0,#11110111B ; desired ch
    inc r2 ; start conversion (set
    inc r0 ; RD' low)
    mov a,r4 ; inc to select next ch
    mov @r0,a ; inc to select next mem
    mov a,r6 ; location
    anl a,#00001111B ; mov the data stored to
    mov @r0,a ; memory
    inc r0 ; inc for next memory
    mov a,r6 ; location
    inc r0 ; taking the next 4
    mov a,#00001111B ; digital bits
    mov @r0,a ; clean them
    loop3: jnb p2.7,loop3 ; store them in the mem
    mov r4,p1 ; location
    mov r6,p2 ; repeat till all ch
    cjne r2,#00001000B,loop2 ; scanned
    inc r0 ;
    mov a,r4 ;
    mov @r0,a ;
    inc r0 ;
    mov a,r6 ;
    anl a,#00001111b ;
    mov @r0,a ;

    ;--- Addition of the numbers
    ;
    inc r3 ; inc the # of data points
    ; added
    mov r1,#67H
    mov r0,#57h ; point to mem loc 58h
loop4: inc r1
    inc r0
    mov r7,#00h
    mov a,@r0
    add a,@r1 ; data for the ms byte
    ; get the data in acc
    ; add it to the gross sum
mov @r1,a
inc r0
inc r1
mov a,@r0
jb acc.3,sum
sumdo: addc a,@r1
mov @r1,a
inc r1
mov a,@r1
addc a,r7
mov @r1,a
cjne r1,#7fh,loop4
cjne r3,#0ffh,loop1

; add it with gross along
; w/carry gen
; store it again
; inc for the next mem loc
; last of the gross sum
; add the last one with 00
; or ff
; store it again
; check if all the 8 data
; are added
; check if the loop is
; done 256 times

; transmit the data
; send a preamble FFFF (cannot occur in the 12-bit data)
; mov a,#11111111B
lcall chrout
lcall chrout

; followed by the division of data by 16 and it’s transmission
; mov r1,#67h
loop:
inc r1
mov a,@r1
swap a
anl a,#00001111b
mov @r1,a
inc r1
mov a,@r1
swap a
anl a,#11110000b
dec r1
orl a,@r1
lcall chrout
inc r1
mov a,@r1
swap a
anl a,#00001111b
mov @r1,a
inc    rl
mov    a,@rl
swap   a
anl    a,#11110000b
dec    rl
orl    a,@rl
lcall  chrout
inc    rl
cjne   rl,#7fh,loop
      ; jump to loop if not
      ; equals 7fH
      ;
ljmp   loop0
      ; repeat with first
      ; channel

END
APPENDIX B
ASSEMBLY CODE FOR IBM PS/2 FOR SERIAL AND PARALLEL INTERFACING
This section of 286 assembly code handles serial and parallel communication via IBM PS/2

.286
.MODEL SMALL

BUFFSIZE = 2000

SEGMENT dseg1 PARA PUBLIC 'DATA'

intsav1 dw 4 dup(?); storage for saved interrupt vectors
intSav dw 4 dup(?); storage for saved interrupt vectors
bufIn dw 2 dup(?); Buffer input pointers
bufOut dw 2 dup(?); Buffer output pointers
buff0 db BUFSIZE dup(?); The COM1 circular buffer

ENDS

GROUP DGROUP dseg1
.CODE

ASSUME ds:DGROUP

PUBLIC _serInit ; Routine is available to other modules

bufAddr dw OFFSET DGROUP:buff0
bufLim dw OFFSET DGROUP:buff0+BUFFSIZE

; serInit(port id)
;==========================================
; Function: Initializes the serial port for the proper baud rate,
; Start and stop bits and where is the interrupt service routine
;==========================================

_serInit PROC NEAR ; NEAR type subroutine

ARG port:WORD,config:WORD
push bp
mov bp,sp
push di
push si
mov cx,[port]
mov ax,[config]
mov dx,cx
mov ah,0 ; Call ROM BIOS port init function
int 14H ; Disable interrupts
push es
xor ax,ax ; zero ax
mov es,ax ; ES = 0 (to access low memory)
mov    si, cx
shl    si, 1
mov    ax, [si+bufAddr]
mov    [si+bufIn], ax
mov    [si+bufOut], ax
mov    bx, es:[si+400H]
mov    si, 1
sub    si, cx
shl    si, 1
shl    si, 1
mov    ax, es:[si+2CH]
mov    [si+intSav], ax
mov    ax, es:[si+intSav+2]
mov    ax, OFFSET intServ
mov    es:[si+2CH], ax
mov    ax, SEG intServ
mov    es:[si+2CH+2], ax
mov    dx, bx
in     al, dx
add    dx, 4
in     al, dx
or     al, 8
out    dx, al
sub    dx, 3
mov    al, 1
out    dx, al
in     al, 21H
mov    ah, 0EFH
ror     ah, cl
and    al, ah
out    21H, al
sti
pop    es
pop    si
pop    di
pop    bp
ret
Function: Serial receive interrupt service routine. Handles an interrupt from a serial port and returns (as fast as possible).

Function: Serial receive interrupt service routine. Handles an interrupt from a serial port and returns (as fast as possible).

intServ PROC NEAR
push ax
push bx
push cx
push dx
push si
push di
push ds
push es
mov ax, DGROUP
; set DS to DGROUP for access to variables
mov ds, ax
xor ax, ax
mov es, ax
; ES = 0 (access to low memory)

; Find which port the interrupt is from
xor si, si
; SI = 0 (assume COM1)
mov dx, es:400H
; DX = base address for COM1
add dx, 2
; Get IIR address
in al, dx
; get IIR contents
mov ah, al
; make a copy of the input
and ah, 1
; Any interrupts here?
jz intC1
; If there is an interrupt
; go process it
mov dx, es:402H
; Otherwise DX = base port
add dx, 2
; Get IIR address for COM2
in al, dx
; Get IIR contents
mov ah, al
; Make a copy of it
and ah, 1
; Are there interrupts?
jnz nodat
; If not, forget this interrupt
add si, 2
; If yes, increment SI for use with COM2

; Find out which type of interrupt it is, and handle it:

intC1: cmp al, 0
; Is it a modem status change?
jne notMSt
; If not, try the next interrupt type
add     dx,4          ; If it is, read modem;
in     al,dx          ; status to clear it
notMSt: cmp   al,2  ; And exit
            jne   notTRE
            jmp   endInt
notTRE: cmp   al,4  ; Is it a transmit
            jne   notTRE
            sub   dx,2
            in     al,dx
            call   NEAR PTR putb
            jmp   endint
notRRF: cmp   al,6  ; Is it a receiver reg
            jne   endInt
            add   dx,3
            in     al,dx
            ; Send end-of-interrupt to the 8259
endInt:  mov    al,20H,al
            ; Send EOI
            ; Restore registers and exit
nodat:  pop     es
            pop     ds
            pop     di
            pop     si
            pop     dx
            pop     cx
            pop     bx
            pop     ax
            iret   ; Return from interrupt
intServ  endp       ; End of routine

_serClose(port)
;==============================
; Function: Close the port; restore it to the state it was in;
; before serInit was called.
;==============================
public _serClose
; Routine is available to
; other modules
_serClose PROC NEAR
ARG port:WORD
push bp
mov bp,sp
push di
push si
push es
xor ax,ax
mov es,ax
cli
mov cx,[port]
mov si,cx
shl si,1
xor ax,ax
mov es,ax
mov bx,es:[si+400H]
;
mov si,1
sub si,cx
shl si,1
shl si,1
mov ax,[si+intSav]
;
mov es:[si+2CH],ax
mov ax,[si+intSav+2]
mov es:[si+2CH+2],ax
mov dx,bx
add dx,4
in al,dx
and al,0F7H
out dx,al
sub dx,3
xor al,al
out dx,al
in al,21H
mov ah,10H
ror ah,cl
or al,ah
out 21H,al
mov al,20H
out 20H,al
sti
pop es
pop si

; FAR type subroutine

; Disable interrupts
; SI = 2*port #
; BX = I/O port base address
; SI = 4*(1 - port #)

; Restore the interrupt from intSav

; Get current value
; Turn off OUT2 bit (interrupt enable)
; Set it

; Get interrupt enable register address
; Clear all interrupt enable bits
; Set them

; Get 8259 enable byte
; Get a mask for the enable for this port

; Turn off the enable bit for this port
; Set it

; Write an EOI just in case

; Re-enable interrupts at the processor
getb(SI = offset); returns AL = char
;=============================================================================
; Function: Get a byte out of the circular buffer, SI contains a word offset within the buffer pointers (0 for the COM1 buffer, 2 for the COM2 buffer). On return, AX will contain a character from the buffer, or -1 if there were no bytes left in the buffer.
;=============================================================================
PUBLIC getb

getb PROC NEAR
mov  ax,-1 ; Assume nothing left
mov  bx,[si+bufOut] ; BX = bufOut
cmp  bx,[si+bufIn] ; bufOut = bufIn ?
je   getb2 ; If yes, go exit
mov  al,[bx] ; Otherwise, get byte from buffer
xor  ah,ah ; Clear top byte of AX
inc  bx ; Increment bufOut
cmp  bx,[si+bufLim]
jne  getb3 ; If not, continue
mov  bx,[si+bufAddr]

getb3: mov  [si+bufOut],bx ; Update bufOut
getb2: ret
getb endp ; End of subroutine

_serRecv(port)
;=============================================================================
; Function: If a character is available from the serial port (port), return it in char; otherwise, return -1 in char.
;=============================================================================
PUBLIC _serRecv

_serRecv PROC NEAR
ARG    port:WORD
push   bp
mov    bp,sp
push   di
push   si
mov    si,[port]
shl    si,1
cli
call   NEAR PTR getb ; Get a character from the buffer
PUBLIC putb

putb PROC NEAR
    push bp
    mov bp, sp
    push si
    push di
    mov dx, [port]
    push bp
    mov bp, sp
    push si
    push di
    mov ah, 3
    int 14h

_serStat PROC NEAR
    ARG port:WORD
    push bp
    mov bp, sp
    push si
    push di
    mov dx, [port]
    mov ah, 3
    int 14h
POP
bp
ret
_serStat endp

; par_data(char)
;=================================================================================
; Function: sends the char on the parallel port 0
;=================================================================================

PUBLIC _par_data
_par_data PROC NEAR

ARG char:WORD
push bp
mov bp, sp
push si
push di
push ds
push es
xor ax, ax
mov es, ax
xor si, si
mov dx, es+408H
inc dx

kgbfbi:
in al, dx
and al, 80H
jnz kgbfbi
mov ax, [char]
xor ah, ah
dec dx
out dx, al

pop es
pop ds
pop di
pop si
pop bp
ret
_par_data endp

; par_cont(data)
;=================================================================================
; Function: sends the data as the control word for the
; printer port
;=================================================================================

PUBLIC _par_cont
_par_cont PROC NEAR
ARG data:WORD
push bp
mov bp, sp
push si
push di
push ds
push es
xor ax, ax
mov es, ax
xor si, si
mov dx, es: 408H
inc dx
kgbfbil: in al, dx
and al, 80H
jnz kgbfbil
mov dx, es: 408H
add dx, 2
mov ax, [data]
xor ah, ah
out dx, al
pop es
pop ds
pop di
pop si
pop bp
ret
_par_cont endp
END

; ES = 0 to access low memory
; SI = 0 assuming LPT1
; Read the printer status

; Check if BUSY is high
; If high then wait for it to become low
; Port address for the LPT1
; Port address for the parallel port control
; Take data in AX
; Send the control word

; End of assembly code
APPENDIX C
C PROGRAM USED TO COLLECT EIGHT CHANNEL DATA SERIALLY VIA A/D CONVERTER CIRCUITRY
The following C program was used with appendix D assembly code to collect data coming out of the A/D convertor.

```c
#include <bios.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <time.h>

extern "C"
extern serInit( int port,int config);
extern serRecv(int port);
extern serStat(int port);
extern serClose(int port);

#define true 1
#define false 0
#define port 0
#define config 0xF3

int main (void)
{
    long ch[20],icha = 0,sav[30],ipnt = 0,itst1 = 0xFF,l = 0;
    double rch[20];
    char flag = false;
    int stat1,isave = 0x00,i , flagg = 0;
    clock_t start = 0,end = 0,nomax = 5;
    FILE  *file;  //file2;

    if ((file = fopen("a:\zmin~s.m","w")) == NULL)    // Cannot open output file
        return 1;

    // Start the file in the matlab compatible format
    fprintf(file,"x = [ \n");
    clrscr();                      // clear the screen
    start = clock();              // start taking time
    //measurements
    serInit(port , config);       // Initialize the
    //serial port

    while(((end - start)/CLK_TCK) <= nomax)  // do while time is 30
```
icha = serRecv(port);  // do till no 0xFF
    // received from
if(icha == -1)
goto jmm;
if((isave == itstl) && (icha == itstl) && (ipnt == 1))
{
    // If first character is FF second
    // is FF and if it is first char
    ipnt = 2;
    sav[ipnt] = iha;
    isave = iha;
    flag = true;
}
else
{
    // Ignore first FF & FF
    // characters
    ipnt++;
    // adn store next 16 characters
    sav[ipnt] = iha;
    isave = iha;
}
if((flag == false) && (ipnt > 1))
{
    ipnt = 1;
    sav[ipnt] = isave;
}
if(ipnt == 18)
{
    for(i=1 ; i <= 8;i++)
    {
        ch[i] = sav[1+2*i] + 256 * sav[2+2*i];
        if((ch[i] & 0x8000) != 0)
            rch[i] =(float)(ch[i] - 65536);
        else
            rch[i] =(float)(ch[i]);
        rch[i] = rch[i] * 5.000/65536.000;
    }
    ipnt = 0;
    flag = false;
    for(i = 1 ; i <= 8 ; i++)
    {
        fprintf(file,"%2.8f ",rch[i]);
        if( i == 8)
        {
            l++;
            fprintf(file,";\n");
        }
    }
end = clock();
printf("current time is %f seconds \n", (end -
start)/CLK_TCK);

// If the file becomes more than line 9000 then open second file
if(l >= 9000 && flagg == 0)
{
  fclose(file);
  file = fopen("c:\qdata1.m","w");
  flagg = 1;
}

fprintf(file," ];\n");
fclose(file);
serClose(port);
return 0
}
APPENDIX D
CONNECTOR CONFIGURATION FOR THE DATAC TERMINAL INTERFACE.
<table>
<thead>
<tr>
<th>PIN</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address bit 00</td>
</tr>
<tr>
<td>3</td>
<td>Address bit 01</td>
</tr>
<tr>
<td>5</td>
<td>Address bit 02</td>
</tr>
<tr>
<td>7</td>
<td>Address bit 03</td>
</tr>
<tr>
<td>9</td>
<td>Address bit 04</td>
</tr>
<tr>
<td>11</td>
<td>Address bit 05</td>
</tr>
<tr>
<td>13</td>
<td>Address bit 06</td>
</tr>
<tr>
<td>15</td>
<td>Address bit 07</td>
</tr>
<tr>
<td>17</td>
<td>Address bit 08</td>
</tr>
<tr>
<td>19</td>
<td>Address bit 09</td>
</tr>
<tr>
<td>21</td>
<td>Address bit 10</td>
</tr>
<tr>
<td>23</td>
<td>Data bit 00</td>
</tr>
<tr>
<td>25</td>
<td>Data bit 01</td>
</tr>
<tr>
<td>27</td>
<td>Data bit 02</td>
</tr>
<tr>
<td>29</td>
<td>Data bit 03</td>
</tr>
<tr>
<td>31</td>
<td>Data bit 04</td>
</tr>
<tr>
<td>33</td>
<td>Data bit 05</td>
</tr>
<tr>
<td>35</td>
<td>Data bit 06</td>
</tr>
<tr>
<td>37</td>
<td>Data bit 07</td>
</tr>
<tr>
<td>39</td>
<td>Data bit 08</td>
</tr>
<tr>
<td>41</td>
<td>Data bit 09</td>
</tr>
<tr>
<td>43</td>
<td>Data bit 10</td>
</tr>
<tr>
<td>45</td>
<td>Data bit 11</td>
</tr>
<tr>
<td>47</td>
<td>Data bit 12</td>
</tr>
<tr>
<td>49</td>
<td>Data bit 13</td>
</tr>
<tr>
<td>51</td>
<td>Data bit 14</td>
</tr>
<tr>
<td>53</td>
<td>Data bit 15</td>
</tr>
<tr>
<td>55</td>
<td>AS (ADDRESS STROBE - ACTIVE LOW)</td>
</tr>
<tr>
<td>57</td>
<td>DS (DATA STROBE - ACTIVE LOW)</td>
</tr>
<tr>
<td>59</td>
<td>R/W (READ-WRITE - WRITE ACTIVE LOW)</td>
</tr>
<tr>
<td>61</td>
<td>CLK (SPECIAL DATAC 10 ms CLOCK)</td>
</tr>
</tbody>
</table>

**EVEN NUMBERED PINS - RETURN LOW FOR EACH SIGNAL**
APPENDIX E
ASSEMBLY CODE FOR 8751 IN DATA TERMINAL INTERFACE
The following 8751 assembly routine was used in the EEPROM of the 8751 used in the DATAC interface.

; FEB 22nd, 1993

;******************************************************************************************
; INTERRUPT LOCATIONS: Here the interrupt finds out where the corresponding service routine is.
;******************************************************************************************
ljmp init ; Go for the program

; This is the vector address for INTO
; Data being sent to the DATAC
; 0003H
org 0003H
ljmp intserv ; If interrupt then jump to intserv.

; This is the vector address for INT1
; Data coming out of the computer
; 0013H
org 0013H ; If interrupt then jump to interr
ljmp interr

;INITIALIZATION ROUTINE
;******************************************************************************************

init: mov psw,#00H ; program status word = 0
      mov sp,#30H ; stack points to internal
      mov tmod,#00100000B ; timer1 is baud rate clock
                         ; (only use high four bits)
      setb p3.2 ; Enable interrupt 0
      setb p3.3 ; Enable interrupt 1
      mov tcon,#00000101B ; Set interrupt 0 type
                         ; control bit...
                         ; .interrupt on falling
                         ; edge i.e...
                         ; negative edge triggered
      mov pcon,#00H ; set smod bit to zero
      mov ie,#10000101B ; enable ext interrupt
                         ; INTO and INT1
      mov ip,#00000101B ; priority level of INTO
                         ; is highest
      mov r1,#40H ; Initialize for interrupt
      mov r0,#40H ; pointer
      mov r6,#00H ; let reg.0 point to mem
                  ; loc 40H
      mov r6,#00H ; Counter Initialized to
mov     p2,#80H ; zero
        ; Make sure the rd/wr~ pin is high
clr     psw.1  ; Make user definable flag = 0
clr     psw.5  ; Flag to show Data coming initially
setb    p2.4   ; Enable computer to write(BUSY HIGH)
setb    p2.5   ; Indicate that first data is coming in

lala:   jnb     psw.5,fafi  ; If psw.5 is not 1 then jump to fafi
        ; Else reset pin 2.5
kala:   jnb     psw.1,rafi  ; If psw.1 is not 1 then jump to rafi
clr     p2.6
ljmp    lala
fafi:   setb    p2.5  ; Set pin p2.5
        ;
        ;
rafi:   setb    p2.6  ; Set pin p2.6
        ;
ljmp    lala

;******************************************************************************
; Loop that takes care of the data coming from computer
;******************************************************************************

aool:   clr     p2.4  ; Disable computer from writing
        ;
mov     r0,#61H ; Put one pointer on 61H location
mov     r1,#60H ; And another one on 60H

coop:   mov     a,@r0 ;
mov     @r1,a
        ;
inc     r0      ; Rotate data once
        ;
inc     r1
        ;
cjne    r0,#78H,coop
        ;
dec     r6      ; Decrement the counter
setb    p2.4    ; Enable computer again
ljmp    tea     ; Goto end

cool:   clr     p2.4  ; Disable computer from writing
        ;
mov     r0,#41H ; Put one pointer on 41H location
mov     r1,#40H ; Put another on 40H

loop:   mov     a,@r0 ;
mov     @r1,a ;
inc r0
inc r1
cjne r0,#58H,loop
dec r6
setb p2.4
ljmp tea

interr: mov @rl,p0
inc r1
inc r6
cjne r6,#18H,tea
mov a,rl
xrl a,#78H
jz cann

; Check if starts with $ and ends with <CR>
mov r0,#40H
mov r1,#57H
cjne @r0,#24H,cool
mov a,r1
xrl a,#78H
jz cann

; If second then goto ;:cann:

; Check if starts with $ and ends with <CR>
cjne @rl,#00H,cool

;Enable flags
setb psw.1
setb psw.5
mov rl,#60H
mov r6,#00H
ljmp tea
mov r0,#60H
mov r1,#77H
cjne @r0,#24H,aool

; If any one of these is not true then shift

cjne @rl,#00H,aool

;Enable flags
clr psw.1
mov rl,#40H
mov r6,#00H

; If all is right Enable buffer one
;...and enable transmission of data
; Then point to next buffer
; and reset the pointer
; End of the interrupt
; Put one pointer to 60H
; Put second pointer to 77H
; Check to see if first byte is $ and last byte is <CR>
; | Check to see if first byte is $ and last byte is <CR>
; | If any of these is not true then shift

; Enable second buffer and disable 1st
; Then point to next buffer
; And reset the counter
ljmp tea

; If correct then enable transmission of data and buffer

tea: reti

;******************************************************************************

; Loop that takes care of the data transfer to the DATAC terminal

;******************************************************************************

kaka: mov r0,#60H

setb p2.7

inc r0

ljmp loop8

intserv:clr p2.4

mov r5,a

mov a,r0

mov r7,a

mov r2,#01H

mov r3,#60H

jnb psw.5,what

jnb psw.1,kaka

mov r0,#40H

setb p2.7

inc r0

loop8: mov p1,r3

setb p2.3

clr p2.3

mov p1,r2

setb p2.2

clr p2.2

mov p1,@r0

setb p2.1

clr p2.1

; Point to memory location
; 60H

; Make sure RD/WR- is high

; Not to send ' $ '

; Disable the computer to
; write

; Save the accumulator

; Save value of R0

; Higher address of the
; DATA

; Lower address of the
; DATA

; At first when no data in
; any buffer

; If the psw.1 = 0 send
; the first
; buffer

; Point to memory location
; 40H

; Make the rd/wr- high

; Not to send ' $ '

; give port0 ,LOWER
; address stored

; Give a clock pulse to
; first chip

; take clock back

; give port0 ,HIGHER
; address stored

; Give a clock pulse to
; 2 nd chip

; take clock back

; give port0 ,data stored

; Give out a clock to the
; 3rd IC

; take clock back
inc r0 ; inc mem location
mov p1,@r0 ; give port0, data stored
setb p2.0 ; Give out a clock to the
clr p2.0 ; 4th IC
inc r0 ; Take it back
clr p2.7 ; Inc memory location
nop p2.7 ; Give a RD/WR~ pulse
nop ; Take it back
take p2.7 ; to lengthen pulse to get at ...
nop ; ... least one AS~ negative
nop ; transition
nop setb p2.7
inc r3 ; Take it back
inc r3 ; Increment the DATAC
cjne r3,#6bH, loop8 ; address
what mov a,r7 ; To check if all 11 data
mov r0,a ; words are sent
mov a,r5 ; Restore the value of
setb p2.4 ; accumulator
reti ; Enable the computer to
END ; write again
; Return from the
; interrupt
APPENDIX F

C PROGRAM TO TEST THE DATA TERMINAL INTERFACE
The following C program was used to test the DATAC interface board.

```c
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<dos.h>
#include<bios.h>

extern "C"{
 extern par_data(int chr);
 extern par_cont(int data);
}

int main(void)
{
    int i,j = 0,chr,chrara = 0x01,chraral = 0x00;
    char r;
    clrscr();
    //Send STB pin high first
    par_cont(chrara);
    //while(j <= 2)
    //{  
    //Send the data '$_'
    par_data(0x24);
    //Make the STB pin low to give INTO and clock for the latch
    par_cont(chraral);
    //Make STB pin high again
    par_cont(chrara);
    // Send equal amount of characters required
    for (i = 0; i <= 21; i++)
    {
        //Send the data
        par_data(0x01);
        //Make the STB pin low to give INTO and clock for the latch
        par_cont(chraral);
        //Make STB pin high again
        par_cont(chrara);
    }
    //Send the data '<CR>'
    par_data(0x0d);
    //Make the STB pin low to give INTO and clock for the latch
    par_cont(chraral);
    //Make STB pin high again
    par_cont(chrara);
    //j++;
    getch();
    //Send the data '$_'
```
par_data(0x24);
//Make the STB pin low to give INTO and clock for the latch
par_cont(chrara1);
//Make STB pin high again
par_cont(chrara);

//Send more than required characters to the DATAC
for (i = 0; i <= 24; i++)
{
    //Send the data
    par_data(0x04);
    //Make the STB pin low to give INTO and clock for the latch
    par_cont(chrara1);
    //Make STB pin high again
    par_cont(chrara);
    //Send the data
    par_cont(chrara);
    getch();
    //Send the data <CR>
    par_data(0x0d);
    //Make the STB pin low to give INTO and clock for the latch
    par_cont(chrara1);
    //Make STB pin high again
    par_cont(chrara);
    //Send exact number of characters required
    for (i = 0; i <= 21; i++)
    {
        //Send the data
        par_data(i);
        //Make the STB pin low to give INTO and clock for the latch
        par_cont(chrara1);
        //Make STB pin high again
        par_cont(chrara);
    }
    //Send the data <CR>
    par_data(0x0d);
    //Make the STB pin low to give INTO and clock for the latch
    par_cont(chrara1);
    //Make STB pin high again
    par_cont(chrara);
    //}
    return 0;
}
APPENDIX G

GPS DATA FORMAT
## OHIO UNIVERSITY GPS DATA FORMAT
### DRAFT

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>SIR ADDRESS</th>
<th>RESOLUTION (Exact)</th>
<th>RANGE (Approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>160</td>
<td>1e-4 seconds</td>
<td>+/- 2.5 days</td>
</tr>
<tr>
<td></td>
<td>161</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latitude</td>
<td>162</td>
<td>5e-8 degrees</td>
<td>+/- 107 deg.</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Longitude</td>
<td>164</td>
<td>1e-7 degrees</td>
<td>+/- 214 deg.</td>
</tr>
<tr>
<td></td>
<td>165</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ellip. Height</td>
<td>166</td>
<td>1e-3 meters</td>
<td>+/- 2148 km</td>
</tr>
<tr>
<td></td>
<td>167</td>
<td></td>
<td></td>
</tr>
<tr>
<td>East Vel.</td>
<td>168</td>
<td>0.01 m/s</td>
<td>+/- 327 m/s</td>
</tr>
<tr>
<td>North Vel.</td>
<td>169</td>
<td>0.01 m/s</td>
<td>+/- 327 m/s</td>
</tr>
<tr>
<td>Vert. Vel.</td>
<td>16A</td>
<td>0.01 m/s</td>
<td>+/- 327 m/s</td>
</tr>
</tbody>
</table>
PARAMETER: GPS Time

SOURCE: OHIO UNIV. GPS

---

SIR ADDRESSES

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
<th>DW1 = 160</th>
<th>Bit #</th>
<th>Significance</th>
<th>DW2 = 161</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-4</td>
<td>Seconds</td>
<td>0</td>
<td>1e-4 *2**16</td>
<td>Seconds</td>
</tr>
<tr>
<td>1</td>
<td>1e-4 *2**1</td>
<td>Seconds</td>
<td>1</td>
<td>1e-4 *2**17</td>
<td>Seconds</td>
</tr>
<tr>
<td>2</td>
<td>1e-4 *2**2</td>
<td>Seconds</td>
<td>2</td>
<td>1e-4 *2**18</td>
<td>Seconds</td>
</tr>
<tr>
<td>3</td>
<td>1e-4 *2**3</td>
<td>Seconds</td>
<td>3</td>
<td>1e-4 *2**19</td>
<td>Seconds</td>
</tr>
<tr>
<td>4</td>
<td>1e-4 *2**4</td>
<td>Seconds</td>
<td>4</td>
<td>1e-4 *2**20</td>
<td>Seconds</td>
</tr>
<tr>
<td>5</td>
<td>1e-4 *2**5</td>
<td>Seconds</td>
<td>5</td>
<td>1e-4 *2**21</td>
<td>Seconds</td>
</tr>
<tr>
<td>6</td>
<td>1e-4 *2**6</td>
<td>Seconds</td>
<td>6</td>
<td>1e-4 *2**22</td>
<td>Seconds</td>
</tr>
<tr>
<td>7</td>
<td>1e-4 *2**7</td>
<td>Seconds</td>
<td>7</td>
<td>1e-4 *2**23</td>
<td>Seconds</td>
</tr>
<tr>
<td>8</td>
<td>1e-4 *2**8</td>
<td>Seconds</td>
<td>8</td>
<td>1e-4 *2**24</td>
<td>Seconds</td>
</tr>
<tr>
<td>9</td>
<td>1e-4 *2**9</td>
<td>Seconds</td>
<td>9</td>
<td>1e-4 *2**25</td>
<td>Seconds</td>
</tr>
<tr>
<td>10</td>
<td>1e-4 *2**10</td>
<td>Seconds</td>
<td>10</td>
<td>1e-4 *2**26</td>
<td>Seconds</td>
</tr>
<tr>
<td>11</td>
<td>1e-4 *2**11</td>
<td>Seconds</td>
<td>11</td>
<td>1e-4 *2**27</td>
<td>Seconds</td>
</tr>
<tr>
<td>12</td>
<td>1e-4 *2**12</td>
<td>Seconds</td>
<td>12</td>
<td>1e-4 *2**28</td>
<td>Seconds</td>
</tr>
<tr>
<td>13</td>
<td>1e-4 *2**13</td>
<td>Seconds</td>
<td>13</td>
<td>1e-4 *2**29</td>
<td>Seconds</td>
</tr>
<tr>
<td>14</td>
<td>1e-4 *2**14</td>
<td>Seconds</td>
<td>14</td>
<td>1e-4 *2**30</td>
<td>Seconds</td>
</tr>
<tr>
<td>15</td>
<td>1e-4 *2**15</td>
<td>Seconds</td>
<td>15</td>
<td>S</td>
<td>Sign</td>
</tr>
</tbody>
</table>

Note 1. Data = 2’s complement binary.
Note 2. Time of day: rolls over after one day (86400 seconds).
Note 3. Time is UTC (GPS time corrected for leap seconds).
PARAMETER: Latitude

SOURCE: OHIO UNIV. GPS

SIR ADDRESSES

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5e-8 Degrees</td>
<td>0</td>
<td>5e-8 *2**16 Degrees</td>
</tr>
<tr>
<td>1</td>
<td>5e-8 *2**1 Degrees</td>
<td>1</td>
<td>5e-8 *2**17 Degrees</td>
</tr>
<tr>
<td>2</td>
<td>5e-8 *2**2 Degrees</td>
<td>2</td>
<td>5e-8 *2**18 Degrees</td>
</tr>
<tr>
<td>3</td>
<td>5e-8 *2**3 Degrees</td>
<td>3</td>
<td>5e-8 *2**19 Degrees</td>
</tr>
<tr>
<td>4</td>
<td>5e-8 *2**4 Degrees</td>
<td>4</td>
<td>5e-8 *2**20 Degrees</td>
</tr>
<tr>
<td>5</td>
<td>5e-8 *2**5 Degrees</td>
<td>5</td>
<td>5e-8 *2**21 Degrees</td>
</tr>
<tr>
<td>6</td>
<td>5e-8 *2**6 Degrees</td>
<td>6</td>
<td>5e-8 *2**22 Degrees</td>
</tr>
<tr>
<td>7</td>
<td>5e-8 *2**7 Degrees</td>
<td>7</td>
<td>5e-8 *2**23 Degrees</td>
</tr>
<tr>
<td>8</td>
<td>5e-8 *2**8 Degrees</td>
<td>8</td>
<td>5e-8 *2**24 Degrees</td>
</tr>
<tr>
<td>9</td>
<td>5e-8 *2**9 Degrees</td>
<td>9</td>
<td>5e-8 *2**25 Degrees</td>
</tr>
<tr>
<td>10</td>
<td>5e-8 *2**10 Degrees</td>
<td>10</td>
<td>5e-8 *2**26 Degrees</td>
</tr>
<tr>
<td>11</td>
<td>5e-8 *2**11 Degrees</td>
<td>11</td>
<td>5e-8 *2**27 Degrees</td>
</tr>
<tr>
<td>12</td>
<td>5e-8 *2**12 Degrees</td>
<td>12</td>
<td>5e-8 *2**28 Degrees</td>
</tr>
<tr>
<td>13</td>
<td>5e-8 *2**13 Degrees</td>
<td>13</td>
<td>5e-8 *2**29 Degrees</td>
</tr>
<tr>
<td>14</td>
<td>5e-8 *2**14 Degrees</td>
<td>14</td>
<td>5e-8 *2**30 Degrees</td>
</tr>
<tr>
<td>15</td>
<td>5e-8 *2**15 Degrees</td>
<td>15</td>
<td>S Sign</td>
</tr>
</tbody>
</table>

Note 1. Data = 2's complement binary.
Note 2. 5e-8 is approx. 5e-8*60*1852 = 0.005 meters. 5e-8*2**30 = 53.6870912 degrees.
PARAMETER: Longitude

SOURCE: OHIO UNIV. GPS

SIR ADDRESSES

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-7</td>
</tr>
<tr>
<td>1</td>
<td>1e-7 *2**1</td>
</tr>
<tr>
<td>2</td>
<td>1e-7 *2**2</td>
</tr>
<tr>
<td>3</td>
<td>1e-7 *2**3</td>
</tr>
<tr>
<td>4</td>
<td>1e-7 *2**4</td>
</tr>
<tr>
<td>5</td>
<td>1e-7 *2**5</td>
</tr>
<tr>
<td>6</td>
<td>1e-7 *2**6</td>
</tr>
<tr>
<td>7</td>
<td>1e-7 *2**7</td>
</tr>
<tr>
<td>8</td>
<td>1e-7 *2**8</td>
</tr>
<tr>
<td>9</td>
<td>1e-7 *2**9</td>
</tr>
<tr>
<td>10</td>
<td>1e-7 *2**10</td>
</tr>
<tr>
<td>11</td>
<td>1e-7 *2**11</td>
</tr>
<tr>
<td>12</td>
<td>1e-7 *2**12</td>
</tr>
<tr>
<td>13</td>
<td>1e-7 *2**13</td>
</tr>
<tr>
<td>14</td>
<td>1e-7 *2**14</td>
</tr>
<tr>
<td>15</td>
<td>1e-7 *2**15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-7</td>
</tr>
<tr>
<td>1</td>
<td>1e-7 *2**16</td>
</tr>
<tr>
<td>2</td>
<td>1e-7 *2**17</td>
</tr>
<tr>
<td>3</td>
<td>1e-7 *2**18</td>
</tr>
<tr>
<td>4</td>
<td>1e-7 *2**19</td>
</tr>
<tr>
<td>5</td>
<td>1e-7 *2**20</td>
</tr>
<tr>
<td>6</td>
<td>1e-7 *2**21</td>
</tr>
<tr>
<td>7</td>
<td>1e-7 *2**22</td>
</tr>
<tr>
<td>8</td>
<td>1e-7 *2**23</td>
</tr>
<tr>
<td>9</td>
<td>1e-7 *2**24</td>
</tr>
<tr>
<td>10</td>
<td>1e-7 *2**25</td>
</tr>
<tr>
<td>11</td>
<td>1e-7 *2**26</td>
</tr>
<tr>
<td>12</td>
<td>1e-7 *2**27</td>
</tr>
<tr>
<td>13</td>
<td>1e-7 *2**28</td>
</tr>
<tr>
<td>14</td>
<td>1e-7 *2**29</td>
</tr>
<tr>
<td>15</td>
<td>1e-7 *2**30</td>
</tr>
<tr>
<td>S</td>
<td>Sign</td>
</tr>
</tbody>
</table>

Note 1. Data = 2's complement binary.
Note 2. 1e-7 is approx. 1e-7*60*1852 = 0.01 meters.
       1e-7*2**30 = 107.3741824 degrees.
PARAMETER: Ellipsoidal Height

SOURCE: OHIO UNIV. GPS

---

SIR ADDRESSES

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
<th></th>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-3</td>
<td>Meters</td>
<td>0</td>
<td>1e-3 *2**16 Meters</td>
</tr>
<tr>
<td>1</td>
<td>1e-3 *2**1</td>
<td>Meters</td>
<td>1</td>
<td>1e-3 *2**17 Meters</td>
</tr>
<tr>
<td>2</td>
<td>1e-3 *2**2</td>
<td>Meters</td>
<td>2</td>
<td>1e-3 *2**18 Meters</td>
</tr>
<tr>
<td>3</td>
<td>1e-3 *2**3</td>
<td>Meters</td>
<td>3</td>
<td>1e-3 *2**19 Meters</td>
</tr>
<tr>
<td>4</td>
<td>1e-3 *2**4</td>
<td>Meters</td>
<td>4</td>
<td>1e-3 *2**20 Meters</td>
</tr>
<tr>
<td>5</td>
<td>1e-3 *2**5</td>
<td>Meters</td>
<td>5</td>
<td>1e-3 *2**21 Meters</td>
</tr>
<tr>
<td>6</td>
<td>1e-3 *2**6</td>
<td>Meters</td>
<td>6</td>
<td>1e-3 *2**22 Meters</td>
</tr>
<tr>
<td>7</td>
<td>1e-3 *2**7</td>
<td>Meters</td>
<td>7</td>
<td>1e-3 *2**23 Meters</td>
</tr>
<tr>
<td>8</td>
<td>1e-3 *2**8</td>
<td>Meters</td>
<td>8</td>
<td>1e-3 *2**24 Meters</td>
</tr>
<tr>
<td>9</td>
<td>1e-3 *2**9</td>
<td>Meters</td>
<td>9</td>
<td>1e-3 *2**25 Meters</td>
</tr>
<tr>
<td>10</td>
<td>1e-3 *2**10</td>
<td>Meters</td>
<td>10</td>
<td>1e-3 *2**26 Meters</td>
</tr>
<tr>
<td>11</td>
<td>1e-3 *2**11</td>
<td>Meters</td>
<td>11</td>
<td>1e-3 *2**27 Meters</td>
</tr>
<tr>
<td>12</td>
<td>1e-3 *2**12</td>
<td>Meters</td>
<td>12</td>
<td>1e-3 *2**28 Meters</td>
</tr>
<tr>
<td>13</td>
<td>1e-3 *2**13</td>
<td>Meters</td>
<td>13</td>
<td>1e-3 *2**29 Meters</td>
</tr>
<tr>
<td>14</td>
<td>1e-3 *2**14</td>
<td>Meters</td>
<td>14</td>
<td>1e-3 *2**30 Meters</td>
</tr>
<tr>
<td>15</td>
<td>1e-3 *2**15</td>
<td>Meters</td>
<td>15</td>
<td>S Sign</td>
</tr>
</tbody>
</table>

Note 1. Data = 2's complement binary.

Note 2. Height is with respect to the WGS-84 ellipsoid.
PARAMETER: East Velocity

SOURCE: OHIO UNIV. GPS

SIR ADDRESS

DW1 = 168

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-2</td>
</tr>
<tr>
<td>1</td>
<td>1e-2 *2**1</td>
</tr>
<tr>
<td>2</td>
<td>1e-2 *2**2</td>
</tr>
<tr>
<td>3</td>
<td>1e-2 *2**3</td>
</tr>
<tr>
<td>4</td>
<td>1e-2 *2**4</td>
</tr>
<tr>
<td>5</td>
<td>1e-2 *2**5</td>
</tr>
<tr>
<td>6</td>
<td>1e-2 *2**6</td>
</tr>
<tr>
<td>7</td>
<td>1e-2 *2**7</td>
</tr>
<tr>
<td>8</td>
<td>1e-2 *2**8</td>
</tr>
<tr>
<td>9</td>
<td>1e-2 *2**9</td>
</tr>
<tr>
<td>10</td>
<td>1e-2 *2**10</td>
</tr>
<tr>
<td>11</td>
<td>1e-2 *2**11</td>
</tr>
<tr>
<td>12</td>
<td>1e-2 *2**12</td>
</tr>
<tr>
<td>13</td>
<td>1e-2 *2**13</td>
</tr>
<tr>
<td>14</td>
<td>1e-2 *2**14</td>
</tr>
<tr>
<td>15</td>
<td>S</td>
</tr>
</tbody>
</table>

Note 1. Data = 2’s complement binary.
Note 2. Maximum velocity = 1e-2*2**15 = 327 m/s (637 knots).
Note 3. 1 m/s = 1*3600/1852 knots (= approx. 1.943 knots).
PARAMETER: North Velocity

SOURCE: OHIO UNIV. GPS

SIR ADDRESS

DW1 = 169

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-2 Meters/sec</td>
</tr>
<tr>
<td>1</td>
<td>1e-2 *2**1 Meters/sec</td>
</tr>
<tr>
<td>2</td>
<td>1e-2 *2**2 Meters/sec</td>
</tr>
<tr>
<td>3</td>
<td>1e-2 *2**3 Meters/sec</td>
</tr>
<tr>
<td>4</td>
<td>1e-2 *2**4 Meters/sec</td>
</tr>
<tr>
<td>5</td>
<td>1e-2 *2**5 Meters/sec</td>
</tr>
<tr>
<td>6</td>
<td>1e-2 *2**6 Meters/sec</td>
</tr>
<tr>
<td>7</td>
<td>1e-2 *2**7 Meters/sec</td>
</tr>
<tr>
<td>8</td>
<td>1e-2 *2**8 Meters/sec</td>
</tr>
<tr>
<td>9</td>
<td>1e-2 *2**9 Meters/sec</td>
</tr>
<tr>
<td>10</td>
<td>1e-2 *2**10 Meters/sec</td>
</tr>
<tr>
<td>11</td>
<td>1e-2 *2**11 Meters/sec</td>
</tr>
<tr>
<td>12</td>
<td>1e-2 *2**12 Meters/sec</td>
</tr>
<tr>
<td>13</td>
<td>1e-2 *2**13 Meters/sec</td>
</tr>
<tr>
<td>14</td>
<td>1e-2 *2**14 Meters/sec</td>
</tr>
<tr>
<td>15</td>
<td>S Sign</td>
</tr>
</tbody>
</table>

Note 1. Data = 2's complement binary.
Note 2. Maximum velocity = 1e-2*2**15 = 327 m/s (637 knots).
Note 3. 1 m/s = 1*3600/1852 knots (= approx. 1.943 knots).
PARAMETER: Vertical Velocity

SOURCE: OHIO UNIV. GPS

SIR ADDRESS

DW1 = 16A

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1e-2 Meters/sec</td>
</tr>
<tr>
<td>1</td>
<td>1e-2 *2**1 Meters/sec</td>
</tr>
<tr>
<td>2</td>
<td>1e-2 *2**2 Meters/sec</td>
</tr>
<tr>
<td>3</td>
<td>1e-2 *2**3 Meters/sec</td>
</tr>
<tr>
<td>4</td>
<td>1e-2 *2**4 Meters/sec</td>
</tr>
<tr>
<td>5</td>
<td>1e-2 *2**5 Meters/sec</td>
</tr>
<tr>
<td>6</td>
<td>1e-2 *2**6 Meters/sec</td>
</tr>
<tr>
<td>7</td>
<td>1e-2 *2**7 Meters/sec</td>
</tr>
<tr>
<td>8</td>
<td>1e-2 *2**8 Meters/sec</td>
</tr>
<tr>
<td>9</td>
<td>1e-2 *2**9 Meters/sec</td>
</tr>
<tr>
<td>10</td>
<td>1e-2 *2**10 Meters/sec</td>
</tr>
<tr>
<td>11</td>
<td>1e-2 *2**11 Meters/sec</td>
</tr>
<tr>
<td>12</td>
<td>1e-2 *2**12 Meters/sec</td>
</tr>
<tr>
<td>13</td>
<td>1e-2 *2**13 Meters/sec</td>
</tr>
<tr>
<td>14</td>
<td>1e-2 *2**14 Meters/sec</td>
</tr>
<tr>
<td>15</td>
<td>S Sign</td>
</tr>
</tbody>
</table>

Note 1. Data = 2's complement binary.
Note 2. Maximum velocity = 1e-2*2**15 = 327 m/s (637 knots).
Note 3. 1 m/s = 1*3600/1852 knots (= approx. 1.943 knots).
Note 4. Velocity is positive when climbing.
Acknowledgements

This work was supported, in part, by the National Aeronautics and Space Administration Langley Research Center under Grant NAG 1 1423.

I would sincerely like to thank Dr. Frank Van Graas, Associate professor of Electrical and Computer Engineering, for being the chairman of the thesis committee. Without his constant encouragement, advice and recommendation, this work would not have been possible.

I would also like to thank Dr. J. Dill, Associate Professor of Electrical and Computer Engineering, Dr. C. Vassiliadis, Assistant Professor of Electrical and Computer Engineering and Dr. Sunil Agrawal, Associate Professor of Mechanical Engineering for being on the thesis committee.

Finally, I would like to dedicate this thesis to my parents, whose blessings have made my dream come true.