Remote Communications

Robot B

George Henckel
Nathan Thomas
Chris Wilcox

EE 382

Junior Design
Dr. Rison and Dr. Wedeward
5 May 2002

Abstract

A wireless communications link and base station were developed to monitor and control an outdoor golfing robot. To implement such a system we used FM transceivers with a serial interface to send the data between the base station and the robot. A graphical user interface (GUI) was created to allow input of the golf ball and hole global positioning satellite (GPS) coordinates. When started, the GUI would send GPS coordinates and a start command to the robot. After the robot had started its task, our system would start updating the GUI with actual coordinates of the robot and its current status.

Table of Contents

List of Figures

List of Tables

Appendices

Introduction

As juniors at New Mexico Tech, we have taken courses that have prepared us to attack a "real world" electrical engineering task. Introduction to Design, EE382, is a class designed to have students use the skills and knowledge acquired from the last few years of study in Electrical Engineering. The junior design project had been the same for quite some time, and the professors decided that it was time for a change. This year's project was to be an autonomous golfing robot.

The robot's primary objective was to use GPS to find a golf ball, pick it up, take it to a hole and drop it in. The robot was broken up into four main subsystems, the chassis, the navigation, the ball/hole location, and the remote communications system.

Our task was to design the remote communication system. When we began this project, we researched many ideas and came up with a solution that we believed would be feasible. Each of us learned a great deal about teamwork and how it is necessary to successfully complete a difficult task. With the experience that we now have gained from this project, it is conceivable that each of us can face a future task similar to this and be successful.

We designed the remote communication station using the software package LabVIEW, commercial transceivers from ABACOM Technologies and the 68HC12 Micro-controller from Motorola to design the GUI, wireless transmission system and to interface with the navigation system of the robot, respectively.

All of the teams worked together to get each subsystem integrated. Given the intensity of the project, we believe that the project was a great success, even though the overall robot failed at completing its ultimate task.

The Solution

We used the DPC-64-RS232 transceiver module from ABACOM Technologies along with a GUI created in LabVIEW for the base station because it was already set up to interface with any PC through the serial port. We also used the DPC-64-CTL transceiver module from ABACOM Technologies for its compatibility with the RS232 module and to interface with the 68HC12. We used the 68HC12's serial communications subsystems to interface with the transceiver and the navigation subsystem of the robot.

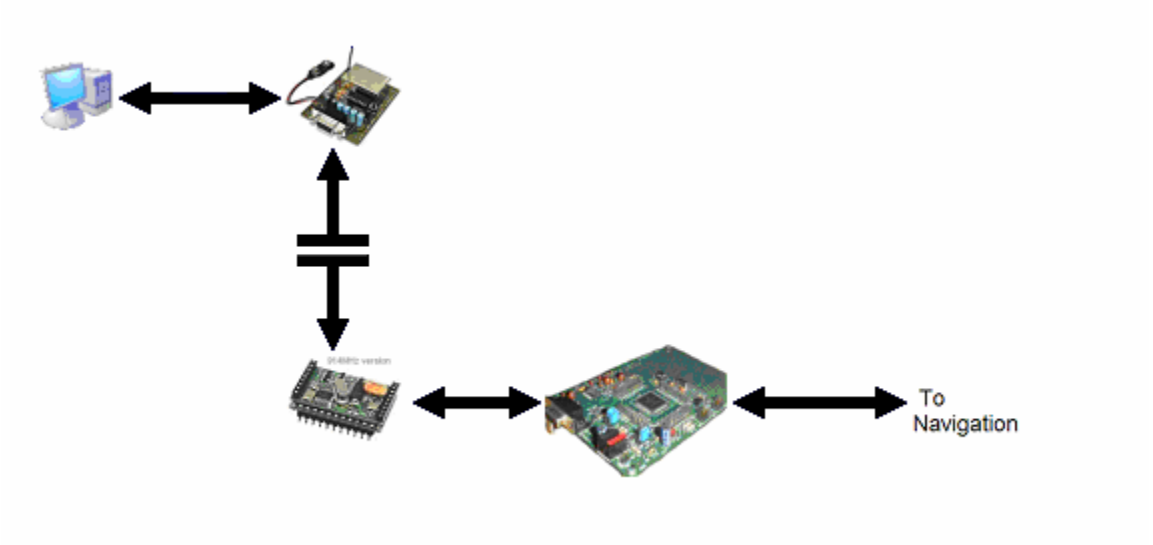


Figure 1: Block Diagram of the Remote Station and Subsystem on Robot.

Board Design and Layout

We designed the circuit board to have the robot transceiver, DPC-64-CTL, connect easily to the 68HC12. We maximized the ground plane on our circuit board, as seen in Figure 3, to minimize voltage spikes and other noise problems that can occur in radio frequency circuitry. A complete wiring diagram can be found in Appendix A.

The "Send Data" line, pin 14, was tied to 5 VDC. With this configuration, the transceiver will automatically output the data that was received from the base station

transceiver, DPC-64-RS232, to "Data Output" line, pin 16. This configuration was set up to be received by the 68HC12's Serial Communications Interface (SCI). The "Data Input" line, pin 18, was automatically configured to receive the data asynchronously from the SCI subsystem on the 68HC12 and transmit it to the DPC-64-RS232, the base station connected to a PC. The "Test" line, pin 9, was connected to a normally open switch connected to the ground plane. This test line was used in the initial stages of our design to test the wireless link between the two transceivers. The "Antenna" line, pin 21, was connected to a simple ¼ wave whip antenna. The "Received Signal Strength Indication" (RSSI), pin 1, outputted a voltage proportional to the signal strength received. Some problems arose when we tried to interface with this. The line would set a proportional voltage out, but we could not read it in time to measure what it was with the A/D converter because our data transmissions took so little time. The reason for this was believed to be because we did not transmit continuously, so an accurate voltage could never really be read. We stopped looking into this issue and put it at the bottom of our priority list and we simply did not have time to get back to it.

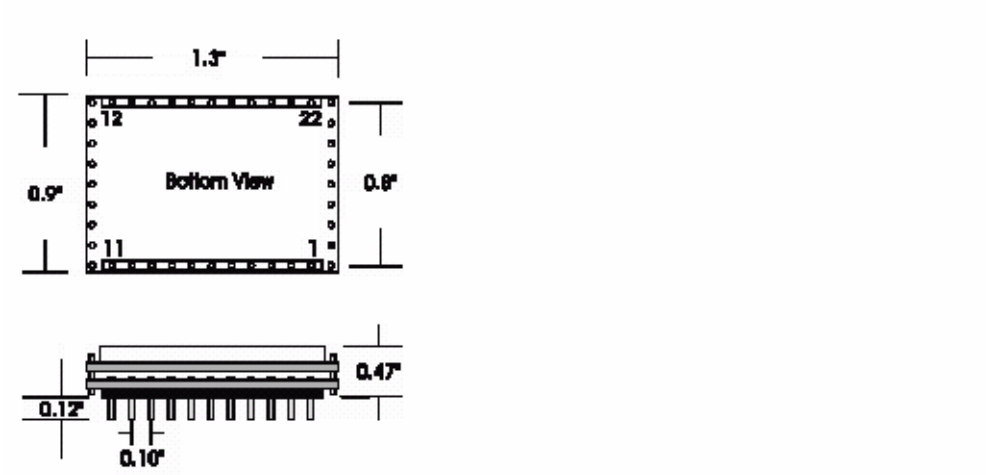


Figure 2: Size and dimensions of the DPC-64-CTL Transceiver Module.

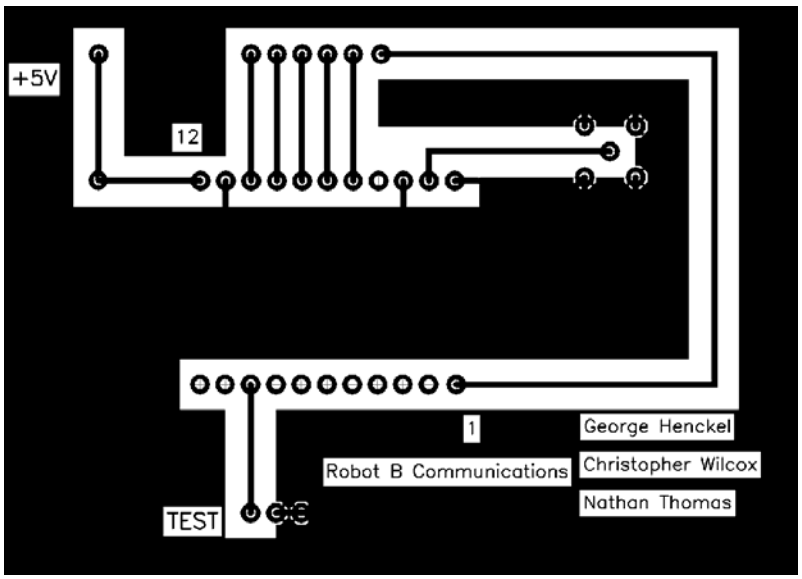


Figure 3: Board layout connecting the Transceiver and 68HC12

Format of Data

The 68HC12 contains several 8-bit data registers. We use these to store the different types of data that we were planning to send and receive. From the base station, we received a packet of twenty-one bytes. The format of this packet can be found in Table 1. The "Menu Byte", the first byte, was the byte we used as a command for the 68HC12 from the remote station. The format and functions of the "Menu Byte" can be found in Figure 4. If it was a 0x01 then the 68HC12 would be instructed to send twenty bytes to the navigation system. Those twenty bytes would be the remaining twenty bytes in the received string. If it was a 0x02, the 68HC12 would continue to send and receive to and from the remote station. If it was a 0x04, then the 68HC12 would set a pin high indicating to the navigation system to have the robot stop.

Byte Number	Data
0	Menu Byte
1	Ball Latitude Degree
2	Ball Latitude Minute
3	Ball Latitude Second
4	Ball Latitude Second 10^{th} and 100^{th} decimal
5	Ball Latitude Second 1000^{th} and 10000^{th} decimal
6	Ball Longitude Degree
7	Ball Longitude Minute
8	Ball Longitude Second
9	Ball Longitude Second 10^{th} and 100^{th} decimal
10	Ball Longitude Second 1000^{th} and 10000^{th} decimal
11	Hole Latitude Degree
12	Hole Latitude Minute
13	Hole Latitude Second
14	Hole Latitude Second 10^{th} and 100^{th} decimal
15	Hole Latitude Second 1000^{th} and 10000^{th} decimal
16	Hole Longitude Degree
17	Hole Longitude Minute
18	Hole Longitude Second
19	Hole Longitude Second 10^{th} and 100^{th} decimal
20	Hole Longitude Second 1000^{th} and 10000^{th} decimal

Table 1: Data sent to the HC12, then the navigation system.

Note: *Initially, Navigation told us they wanted accuracy to 4 decimal places of the Second. We later discovered that we could never really get that accurate from the GPS system.*

Figure 4: The Structure of the Menu Byte is as follows:
INIT – Send the Ball/Hole Position Data to the navigation system
CONT – Do Nothing/Continue Searching
STOP – Stop all Processes

Note: *The Menu Byte is not sent to the navigation system, it tells the HC12 on the robot when to send GPS data, tell the navigation system to continue searching, or to stop.*

Using the 68HC12's Serial Peripheral Interface (SPI) interrupt, a packet of nine bytes were received from the navigation system, which can be found in Table 2. These

nine bytes were the "Status Byte" and the seconds and four decimals places of seconds of the robot's location. The following two bytes were the direction in degrees from North. Once received, this data was resorted in the transmission string and sent back to the remote station. The "Status Byte" from the navigation system told us what the robot was doing. If it was a 0x01, then we were to expect eight more bytes for the robot's location and the direction from North. If it was a 0x02, then the robot was not moving and was searching for the ball/hole with the cameras. If it was a 0x04, then the ball/hole was found. We manipulated the "Progress Byte" with this data as we received it to tell the remote station exactly where in the task the robot was.

Byte Number	Data
0	Status Byte
1	Robot Latitude Second
2	Robot Latitude Second 10^{th} and 100^{th} decimal
3	Robot Latitude Second 1000^{th} and 10000^{th} decimal
4	Robot Longitude Second
5	Robot Longitude Second 10^{th} and 100^{th} decimal
6	Robot Longitude Second 1000^{th} and 10000^{th} decimal
7	Direction 1
8	Direction 2

Table 2: Data received from the navigation system.

Figure 5: The Structure of the Status Byte is as follows:
GPS – Searching for Ball/Hole with GPS.
SEN – Searching for Ball/Hole with Sensors.
CMPT – Task completed.

The format of the packet of fourteen bytes sent back to the remote station can be found in Table 3. The "Progress Byte" was set up as such to let the remote station know exactly where the robot was in its task. The format of the "Progress Byte" can be found

in Figure 6. If it was a 0x01, then the robot was looking for the ball with GPS. If a 0x02, it was looking for the ball with the cameras. If a 0x04, it had the ball. If a 0x08, it would be searching for the Hole with GPS. If a 0x10, it would be searching for the hole with the cameras. If a 0x20, it found the hole. If a 0x40, then the ball was dropped in the hole and the task was completed.

Byte Number	Data
1	Robot Latitude Degree
2	Robot Latitude Minute
3	Robot Latitude Second
4	Robot Latitude Second 10^{th} and 100^{th} decimal
5	Robot Latitude Second 1000^{th} and 10000^{th} decimal
6	Robot Longitude Degree
7	Robot Longitude Minute
8	Robot Longitude Second
9	Robot Longitude Second 10^{th} and 100^{th} decimal
10	Robot Longitude Second 1000^{th} and 10000^{th} decimal
11	Direction 1
12	Direction 2
13	Progress Byte
14	RSSI

Table 3: Data sent to the remote station.

Figure 6: The structure of the Progress Byte is as follows:
BGPS – Looking for the Ball with GPS.
BSEN – Looking for the Ball with the Sensors/Camera.
HAVB – The Robot has the Ball.
HGPS – Looking for the Hole with GPS.
HSEN – Looking for the Hole with Sensors/Camera.
FNDH – The Robot has found the Hole.
BDRP – The Ball has been dropped.

The RSSI was the received signal strength read by the analog-to-digital converter on the 68HC12, which is currently not being used by the remote station.

LabVIEW

LabVIEW proved to be a very efficient way to go. It made the implementation of the base station as simple as possible. The GUI that it provided also proved to be easy to use and quite appealing to most users. The only function it seems to lack is background music of a golfer swinging the club and him yelling "four!"

Like any complex program there are many functions, .vi's, within the LabVIEW GUI. The final program has a total of seven functions and the main program. Each of these functions will be described later in the report. Some LabVIEW experience may be necessary to fully understand the descriptions of the code as well as the code itself. Please keep in mind that like any programming language there may be easier ways to implement the code. However, the only thing that matters is that it works. A hardcopy of the code is in appendix B. However, the electronic copy will be much easier to "thumb through." There is also a specific set of instructions on using the GUI for any user to learn how to use it within minutes in Appendix G.

THE LabVIEW Graphical User Interface

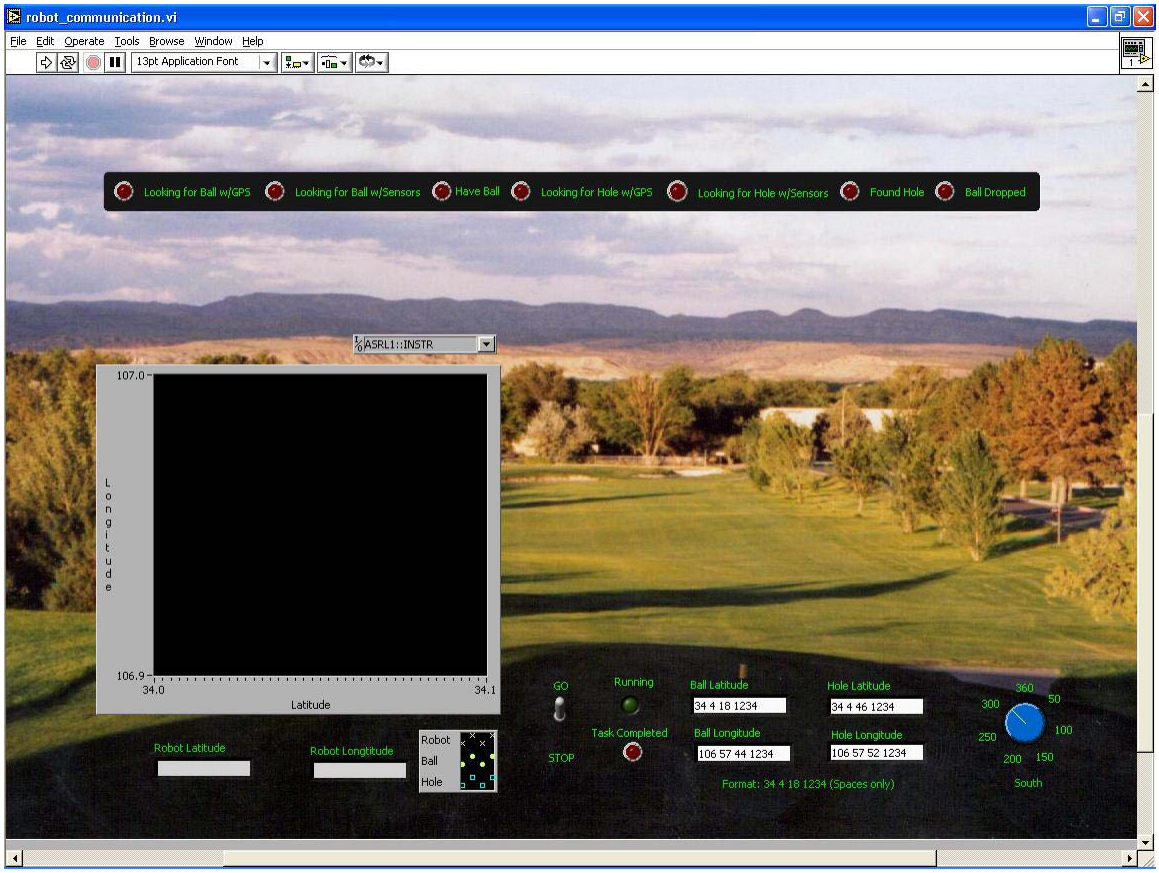


Figure 7: The LabVIEW GUI

Starting with the main function, "robot_communications.vi"; the front panel is the GUI. Open this program to get started. The GUI is where the user can input information and get information back from the robot. First off, the user will be able to decide which port to hook the transceiver up to (i.e. COM1, COM2 …). LabVIEW will list all possible ports in a drop down menu on the GUI. However, the names that LabVIEW supplies for the ports are not labeled very well, but they are manageable; ASRL1::INSTR is equivalent to COM1; ASRL2::INSTR is equivalent to COM2 and so forth. Secondly, the user needs to tell the robot the GPS coordinates of the ball and hole. Default values are displayed in the ball and hole coordinate boxes in their correct format which is also displayed below the input boxes. The format is very important for parsing of the input

strings. The user will need to make sure that the Go/Stop switch is in the "Go" position before running the program. This switch will only be used during the need for an emergency stop when it should be pulled down. After the robot is powered up, the initial state of the robot will be to sit and wait for a command from the LabVIEW code. Starting the program will send this initial command and will initialize the robot to find the ball at the current location indicated in the LabVIEW code. To start the program the user needs to click on the arrow icon at the top left of the LabVIEW windows in the toolbar. The user will be able to quickly view the robot coordinates given in degrees, minutes, seconds, and decimals of seconds. The direction will be displayed on compass with North facing towards the top of the screen. The compass will display the direction in degrees from North (i.e. 90° is East). The progress bar at the top of the screen displays the current activity of the robot. There are seven "lights" that indicate that the robot is looking for the golf ball with GPS, looking for the golf ball with sensors or cameras, retrieved the ball, looking for hole with GPS, looking for hole with sensors or cameras, found hole, and ball dropped. The user will also be able to see where the robot is in reference to the ball and hole on a graph. The graph is very flexible because it is auto adjusting to the range needed to display the ball, hole, and robot. Finally, the "Task Completed" light will light up with the robot has finished its task.

Debugging with the GUI

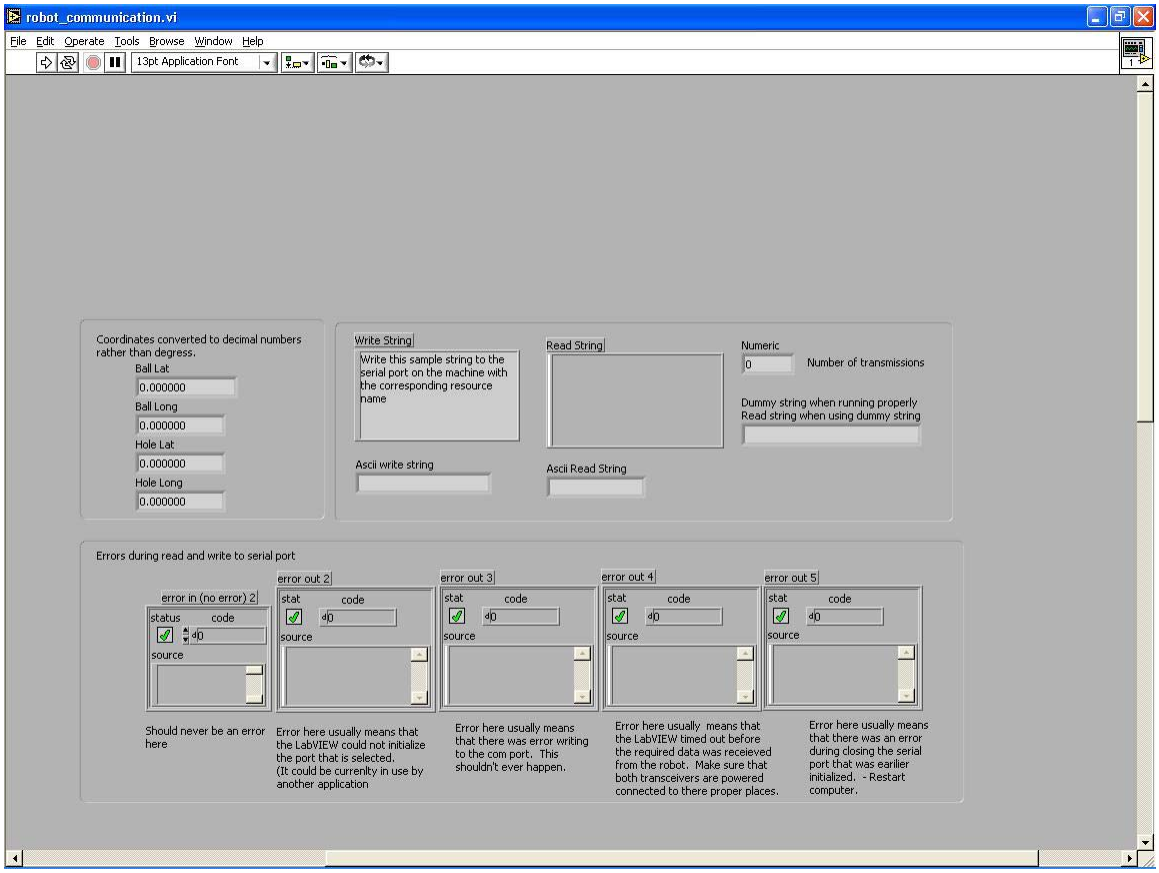


Figure 8: Debugging Portion of the Interface

Out of sight of the main GUI screen, the user can scroll up and see more information. This information was used for debugging and left in the final version of the code in case the user encounters any problems while using the product. There are five displays for errors each checking for errors in different places of the code. A quick way to check for errors is to look at the status box in each of the error displays; a green check means everything is fine and a red x means that there is an error. If there is an error, the user can look up the error code. The error codes are definitions of the error; they are located in the Appendix D as well as in the LabVIEW help. The most common problems have been listed below each of the error displays in the GUI and the codes should never

have to be looked up. The best way to fix the problem has been listed under each error as well.

Hex read and write strings as well as the ASCII read and write strings that are actually sent by the transceivers are also located in the GUI above the main panel. The user can also see how many times the program has sent data to the robot. The last thing that is available is ball and hole latitude and longitude coordinates in degree with decimals for the plotting of the values.

LABVIEW – Behind the Scenes

Behind the scenes gets a little scarier; especially if you are unfamiliar with LabVIEW. However, most users will never have the option to even see the actual code. There are two electronic copies of the code. The first copy is not editable; the diagram (code) is not even accessible to the user. The second copy will include the diagrams so that the "code" can be viewed or edited in LabVIEW by clicking ctrl + E or simply going to the toolbar and clicking on "window" and then "show diagram". The code is pretty well commented and shouldn't need any more explanation for anyone who has some experience with LabVIEW.

A Brief Description of the Functions

Asciitohex.vi: This function converts the string that the transceiver receives from ASCII text to a hexadecimal string. The function will only work the fourteen bytes that we are currently using but can be easily modified to convert more bytes. See the comments in the code to determine how to do this.

Coordtodecandhex.vi: This function converts the input string from the GUI into formatted hex data to be added to the string that is transmitted to the robot. The

function will also convert the coordinates into a fractional number of degrees to be displayed on the graph. This function is used four times in the main function when converting ball and hole, latitude and longitude data.

Numtohex.vi: This function is simple. It takes the Coordtodecandhex.vi function and implements it four times; once for each of the latitude and longitude coordinates for the ball and hole.

Hex String to Binary.vi: This function is only used twice but is very important. This converts one hex digit, four bits, in a binary Boolean array. This is used to decipher the progress/status byte.

Hextoascii.vi: This function changes the hex output into ASCII output that the transceiver can handle. It is only set up to covert the twenty-one bytes that are needed. More bytes can be easily added, see LabVIEW code for details on how to do this.

Hextonum.vi: This is definitely the most complicated and messiest function that was created besides the main function. This function filters through all of the incoming data and translates the hex data into information that the GUI will use. The first ten bytes received by the transceiver are latitude and longitude byes of the robot. The function will convert the incoming hex data into a formatted string to display the coordinates of the robot on the GUI as well as a decimal number of degrees for the graph. The next two bytes are the direction of the robot. In the first byte of the direction we only use one bit. If it is a one we add 255 degrees to the second byte otherwise the heading (degrees from North clockwise) will be the second byte. The next byte is the progress byte which tells us what the robot is currently doing. The

last byte is not currently being used. We had planned on using a signal strength meter, RSSI, which we could not get working due to timing issues with the 68HC12. The byte was left there for easy implementation in the future.

Serial Read with Timeout.vi: This function was a LabVIEW example and is very simple. It simply adds a timeout on the serial read so that there is sufficient time for the base station to wait for the robot to send the data.

Microcontroller Code

The microcontroller code performs three main functions: send/receive with the wireless transceiver, send/receive with the navigation module, and directing information between the wireless transmitter and the navigation module. Experience with programming a 68HC12 microcontroller may be necessary to fully understand the code.

Serial Communications Interface: The 68HC12 SCI subsystem was used to talk with the wireless transceiver. SCI is an asynchronous communications device that works with both TTL and RS232 signals. The SCI port setup registers define baud rate, character format, enable/disable transmitter and/or receiver, and enable interrupts.

The SCI subsystem was setup to run at 9600 baud, normal mode, 8-bit operation, no parity, no interrupts, and enabled the transmitter and receiver. The character format used was one start bit, eight data bits, and one stop bit. Receiving and transmitting data to and from the wireless transceiver involved loading data to the SC0DRL data register, waiting and storing data from the SC0DRL register.

Data to be transmitted was written to the SC0DRL register then the 68HC12 waited for the "Transmitted Data Empty Flag" to be set. This was repeated for every byte of data sent to the wireless transceiver. To receive from the wireless transceiver,

the 68HC12 had to wait for the Received Data Empty Flag to be set and then stored the contents of SC0DRL register in memory.

Serial Peripheral Interface: The SPI subsystem was used to talk to the navigation system. The SPI is a synchronous communications tool. Ball and hole locations would be sent to the navigation system. The navigation system would send back the robots' location and direction

The communications system was setup as a slave for SPI. The SPI was set for normal operation, most significant bit first, slave mode, enable interrupts and enable the SPI subsystem. To send data to the master requires coordination. Data to be sent gets put in the SP0DR data register then a READY line is set high so the master, the navigation system, knows the slave is ready to send. The master then puts data in its SP0DR register and selects the slave to start the data transfer. The slave waits for the slave select to go low and then sets its READY line back low signaling it is not ready to send new data. The slave then waits for the slave select line to go high when the current data has been transmitted. The SP0DR register is read to clear the SPIF flag and is ready for the next byte. This process repeats until all the data has been sent. The READY line is used for "handshaking" so the master and slave can synchronize with each other.

Receiving data does not require "handshaking". Interrupts were used so data could be acquired at anytime to keep the communications system from waiting for the navigation system to send data. The interrupts would occur after a single byte of information had been sent by the master to the slave and the SPIF flag been set. The slave checked the SPIF flag and then read SP0DR register and stored the byte. The

first byte of information sent is checked to see if it is 0x02 or 0x04. If it is either of these values then no more data will be sent by the navigation system. The 0x02 means that the ball/hole location system is looking for the ball. A 0x04 means the ball has been found/captured or the ball has been put in the hole. The meaning is determined on the number of times the value has been received by the communication system. An example would be receiving 0x04 a second time would indicate the ball has been put in the hole.

Additional Functions: The rest of the code deals with data formatting between the navigation system and remote station. Two different formats were used one for the navigation system and another for the remote station. This section of the code just transferred on data format to another data format.

An assumption was made that the golf course would not have a change in minutes. Instead of sending the entire GPS coordinates the navigation system sent only the seconds of the GPS coordinates of the robot. To account for this the communications system records the degrees and minutes of the ball and hole, assuming they are the same, and uses those as the degrees and minutes for the robot.

Simulation of Navigation

In order to complete our design, it was necessary for us to write a simple program that would simulate the navigation system's 68HC12 by sending our 68HC12 sample calculations of the robot's location and direction from North. In the SPI subsystem of the 68HC12, the slave sending data to the master can be tricky. We needed a Ready Line independent of the SPI subsystem to tell the master to select us and start the data transfer. The code that we used to simulate the Navigation's 68HC12 is in Appendix F. A

problem arose between the master and slave in which data would periodically appear out of order or shifted in order. With more time, a solution to this issue would be found.

Power Requirements

The power required from the chassis was 5 VDC for each of our devices with 20 mA for the transceiver and around 50 mA for the 68HC12. This was a total of about 350 mW that we needed to power our devices, which was well within the range of operation that the chassis allotted for us. As for the device connected to the PC, a 9V battery was used to the supplied connection.

Final Budget

The budget turned out to be nearly exactly as expected. It is shown here that we came in under budget. However, in reality, we would have been over budget; we made a rookie mistake and did not take into account the shipping and handling costs for the ordered parts. We were lucky enough to receive a student discount from ABACOM technologies which made up for the difference. The following table contains a list of parts purchased and their expenses.

Description	Estimated Expense's	Actual Expense's
Transceiver for Robot	$160	$143.10
Antenna for Robot	$11	$9.72
Transceiver for Computer	$180	$160.98
Shipping and Handling	Misc.	$30.00
3 * 9 Volt Batteries	Misc.	$5.58
Push Button Switch	Misc.	$4.00
2 * DB9 Connectors	Misc.	$7.04
16 Pin Wire wrap Socket	Misc.	1.50
Risers for Micro-controller	Misc.	2.00
Total costs Total replication costs	$400 $600	$363.92

Table 4: Final Budget

The total replication cost seems quite high. However, personal micro-controllers where used and the pc-board was donated. We also included the student discount costs in the replication budget as well.

Overall, we felt as if our choice in the transceivers that we purchased were worth the money. They allowed for the best implantation that we could expect. If we had it to do all over again we still purchase the same components.

Future Plans

If we had additional time we would include a case for both the base station and the robot. We would also have continued working with RSSI to display the signal

strength on the GUI. Finally and most importantly we would have fixed the interfacing problem between the communication and navigation systems.

Conclusion

We designed a functional subsystem for the robot. Initially the project presented a problem that we did not know how to implement. After researching various possibilities of wireless communications a solution was found. It was a practical solution to the problem which worked quite nicely. We are proud to announce that our task has been completed and are very pleased with the results that we came up with.

References

Abacom Technologies. Spring 2002. <http://www.abacom-tech.com>.

Digi-Key Corporation. Spring 2002. <http://www.digi-key.com>.

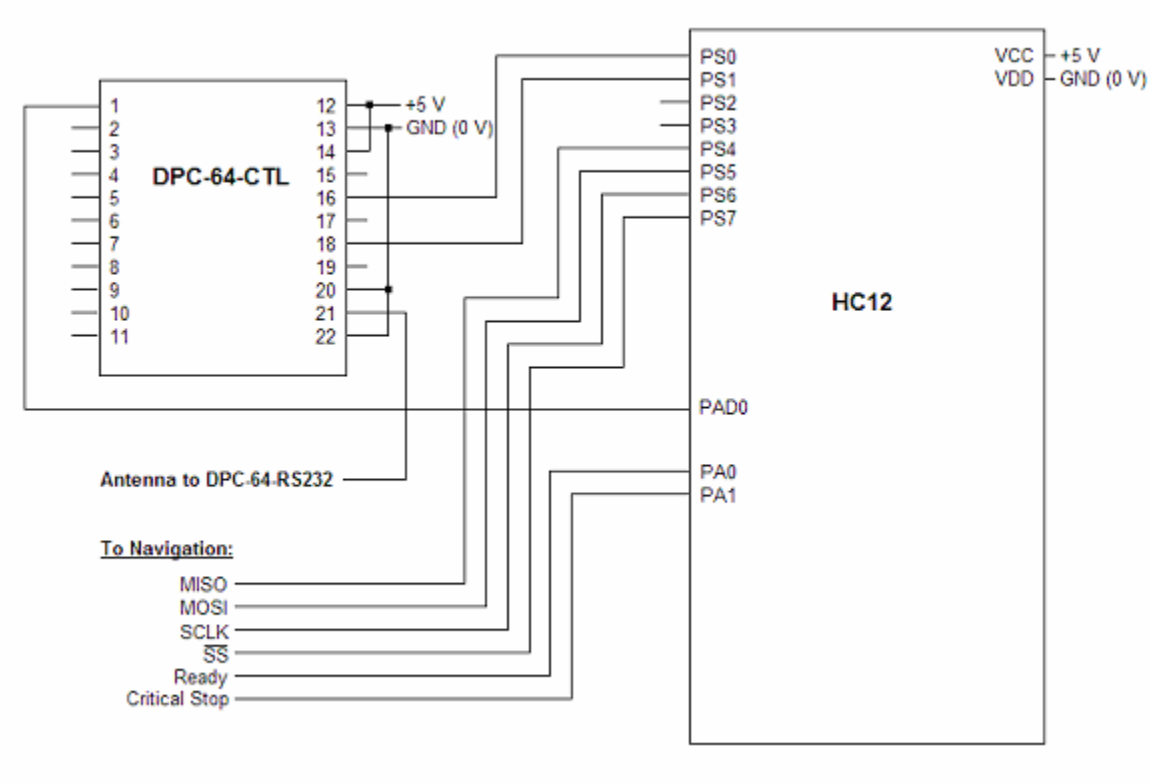
LabVIEW Help File.

Motorola Reference Manual 68HC912B32. Motorola INC. 1997.

Rison, William. Electrical Engineering Dept., New Mexico Institute of Mining and Technology. Spring 2002 <http://www.ee.nmt.edu/~rison>.

Appendix A

Wiring Diagram



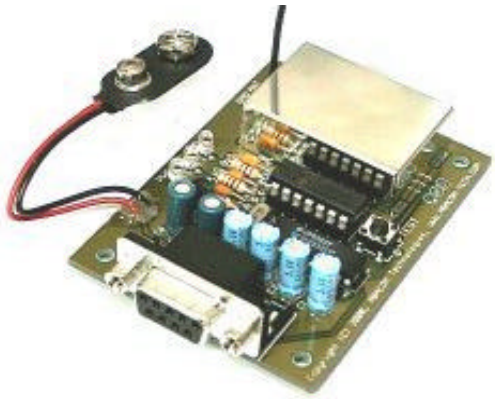
Appendix B

DPC-64-RS232 Transceiver Module

Introduction

The DPC-64-RS232 transceiver module provides a transparent serial link between two host devices, where packets of up to 64 bytes of data may be transceived. All data encoding, decoding and error checking is performed by the DPC-64-RS232 leaving nothing more for the host to do, other than sending data in x.8.N.1 serial format to the DPC. The serial data sent by the host to the DPC is automatically encoded and packetized and then transmitted over the air to the receiving DPC.

Upon receipt of the radio data transmission, the DPC will decode the data, perform an error check and output the original data to the receiving host.



Power Supply

Although the operating supply range is from 7.5Vdc to 15Vdc, 20mA, the DPC is supplied standard with a 9V battery connector – alternative power sources within range may of course be used. Polarity should be observed. As a protective measure for the host equipment, we recommend applying power to the DPC only **after** it has been connected to the host.

Serial interface

The DPC-64-RS232 transceiver implements a standard serial interface and is configured as a DTE.Connection to, for example, a personal computer COM port can be made using a standard DB9 connector serial extension cable having a one male and one female connector.

A four wire interface is used by implementing TxD, RxD, GND and CTS. The CTS flow control line is available for users who wish to transmit more than 64 bytes. Under this condition, when the DPC has received 64 bytes, the DPC will signal the host to pause sending data allowing it to transmit the 64 bytes of data to the receiving DPC. Once the DPC has transmitted the data, the CTS flow control line will toggle, allowing the DPC to receive the next 64 bytes of data from the host.

The host terminal should be set to 9600∗, 8, N, 1 and flow control should be set to ON (∗substitute for other data rates)

The flow control line can be ignored if the total data to be transmitted is less than 64 bytes.

No flow control is used when uploading the received data from the DPC to the receiving host. The relevant flow control lines have been looped back on the DPC.

Test Mode

The DPC features a test mode which is very useful to quickly test the integrity of the RF link. When the test button is pressed, the DPC will transmit an internally generated test message to the receiving DPC. The received test message will be displayed on the receiving host terminal screen when running a terminal program, for example.

Note: For volume applications, the content of the preset message can be customized if required.

Specifications

Operating Temperature	-10 to +55°C	
Supply Voltage	7.5V – 15Vdc	
Supply Current	20mA	transmit or receive
Data Rates A version	1200bps	2MHz crystal
Data Rates B version	2400bps	4MHz crystal
Data Rates C version	4800bps	8MHz crystal
Data Rates D version	9600bps	16MHz crystal
RF Output Power	0dBm	typical
Sensitivity	-107dBm	typical
RF stability	+ - 100KHz	of centre frequency
Deviation	25KHz	typical
RS-232 interface	TxD, RxD, GND , CTS	CTS only for transmit mode
RS-232 protocol	x, 8,N,1	flow control used
Max. bytes per transmission	64	
RF i/o impedance	50Ω	for alternative external antenna
Antenna	¼ wave wire	
Range - open field	up to 500ft	with ¼ wave antenna
Dimensions	83.5 x 53 x 15mm	excluding DB9 connector

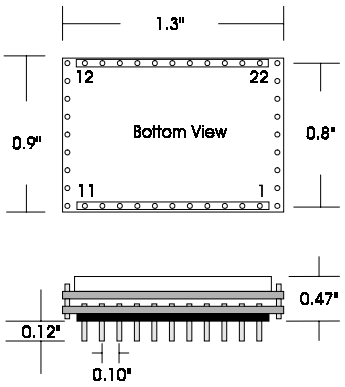
DPC-64-CTL Intelligent RF Transceiver Module

The extremely versatile DPC-64-CTL RF intelligent transceiver modules may be interfaced directly to any CMOS/ TTL serial data hosts such as microcontrollers and microprocessors to create a transparent bi-directional half duplex link. The DPC-64-CTL takes care of the RF communications protocol, eliminating the need for any special data formatting which is required for successful RF data communications. Simply input serial data in the format of 9600, 8,N,1 and the CTL will reproduce your original data at the receiving end. Error checking is automatically performed on the received data. The DPC-64-CTL is capable of transmitting 1-64 byte packets of data at a time. More than 64 bytes may easily be transmitted through simple implementation of the included optional flow control lines.

FEATURES

- **Transparent data formatting and error detection**
- **Processes 1 to 64 byte data packets per transmission**
- **Optional Handshaking lines included to transceive more than 64bytes**
- **Convenient Test Transmission Mode for diagnostic purposes**
- **1200, 2400,4800 or 9600 8,N,1 protocol compatible**
- **Simple to interface to CMOS/TTL hosts**
- **Significantly reduces design time**
- **Automatic TX/RX switching**
- **Automatic data input detection**
- **Available on 433.92, 868 and 914.5MHz**
- **Up to 700ft range**
- **5v operation, <15mA**
- **RSSI output**
- **RF Carrier Detect Output**
- **Compatible with the DPC-64-RS232 modules**

MECHANICAL DIMENSIONS



Pin Number	Description
1	RSSI Output (available on -SS modules)
2-7	No Connection
8	CLK output
9	TEST Link
10	Audio Output
11	Carrier Detect Output
12	+5V Supply
13,20, 22	GND
14	Send Data Control Line- Input
15	Data Ready Control Line- Output
16	Transmit Data– Output
17	Busy Control Line– Output
18	Receive Data– Input
19	Reserved– do not connect
21	Antenna

ABACOM Technologies • 32 Blair Athol Crescent • Etobicoke • ON • M9A 1X5 • Tel +416-236-3858 • Fax +416-236-8866
www.abacom-tech.com

Detailed Pin Description

Pin 1– RSSI Output

The RSSI (received signal strength indication) outputs a DC voltage proportional to the signal strength received. This RSSI signal may be interfaced to external circuits such as an A/D converter or analog displays and serve as an aid to optimize position of the DPC-64-CTLss modules for best performance.

The table below provides typical values of RSSI for varying levels of RF signal strength applied.

RF Input (dBm)	RSSI (V)
-105	0.82
-100	0.88
-90	1.12
-80	1.43
-70	1.75
-60	2.06
-50	2.36
-40	2.57
-30	2.6
-20	2.6

Pins 2-7

No connection.

Pin 8– CLK

This is a 16MHz external clock signal which may be interfaced to external devices if desired. Otherwise this pin may be left unconnected.

Pin 9– Test

This pin is internally pulled high via an internal pull-up resistor. When taken low, typically via a tactile feedback pushbutton switch or any other dry contact, the data input pin 18 will be ignored and a 64 byte internal preset message will be output on pin 16 of the receiving DPC-64-CTL module. The content of this test message will be:

<ABACOM Technologies> DPC-64 www.abacom-tech.com +1(416)236-3858

The test message is intended for diagnostic purposes and serves as a quick test to verify the integrity of the RF link. With a receiving host PC running a simple terminal program such as Hyperterminal configured for xx00,8,N,1 the content of the test message in a correctly configured RF wireless link will be displayed.

Alternatively, an LED connected to the Transmit Data Output pin 17 as shown in the test circuit will flash when a valid test message is received.

Pin 10– Audio Output

The analog signal on pin 10 is the demodulated signal from the receiver and is made available to the designer where it may be used for custom specific design functions. If not required, this pin may be left unconnected.

Pin 11– Carrier Detect

The CD pin is active low in the presence of an RF carrier. The CD may be used as additional control logic for external circuits. If not required, this pin may be left unconnected.

Pin12- +5V Supply

Supply pin 12 should be decoupled to Ground via a 0.1uF ceramic capacitor.

Pins 13,20,22– Supply/RF Ground

The three ground pins are internally connected to the DPC-64-CTL ground plane. We recommend connecting all three ground pins if possible. At a minimum any one of the ground pins must be connect to system ground. Pin 22 is preferred as an RF ground for 50 Ω coaxial cable feeding off board antenna.

Pin 14 – Send Data

The send data control line is active high. When taken high, the data received from the transmitting DPC-64-CTL will be serially output on pin 16. If held low, the data received over-the-air-will be stored in the DPC-64-CTL's buffer until "send data" is taken high. The send data line functions in association with the data ready pin 15.

Pin 15– Data Ready

When the receiving DPC-64 has received valid data, the data ready control line may be used to

signal the receiving host that the DPC-64-CTL has valid data ready to upload. The receiving host then asserts a logic high level on the “send data” line at pin 14 and the data is uploaded.

Many applications require the received data from the DPC-64-CTL to be uploaded to the host without supervision. These applications therefore do not require interfacing the data ready control line and therefore the data ready pin 15 may be left unconnected and the send data pin 14 then simply tied high. In this configuration, the DPC-64-CTL will output the data as it is received.

Pin 16– Data Output

The data that has been transmitted from the DPC-64-CTL’s is checked for errors by the receiving DPC-64CTL. Error free data with output on pin 16 provided that the send data control line pin 14 is at a logic high level.

Pin 17– Busy

The busy control line goes high when the DPC-64-CTL transceiver module has either received its maximum of 64 bytes or when it has detected the end of incoming data (under conditions when < 64 bytes have been received from the host).

The function of the busy line pin 17 is for data flow control with the host. Implementing the busy line is necessary in applications where more than 64 bytes are to be transmitted. Under these circumstances, the DPC-64-CTL transceiver module will receive the first 64 bytes of data, and then use the busy line to signal the host to pause sending further data until it is has completed its data processing functions and transmitted the data.

If the designers application does not require sending more than 64 bytes of data, then the busy control pin 16 may be left unconnected.

Pin 18– Data Input

Data to be transmitted over DPC-64-CTL RF link is fed into pin 18 in standard serial CMOS/TTL level data format of 9600bps, 8 data bits, No parity and one stop bit (9600,8,N,1). One to 64 bytes may be transmitted at a time without the need for the flow control as is implemented with the BUSY pin 17.

The DPC64-CTL transparently formats the data into the correct protocol for RF communications and then transmits the data received from the host.

When the host is not sending data to the DPC-64-CTL transceiver module, it is important that pin 18 is held high. If it is held low, the DPC-64-CTL will see this condition as valid data entering on pin 18 and will begin to transmit this erroneous data.

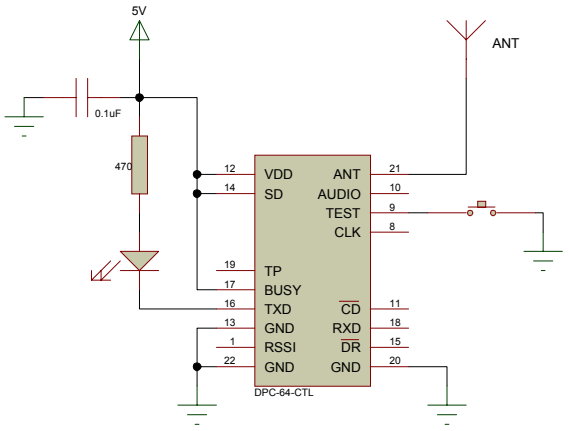
Pin 19– Reserved

Leave pin 19 unconnected. This pin is reserved for manufacturing functions.

Pin 21– Antenna

A simple 1/4 wave whip antenna may be connected close to this pin. If a coaxial cable fed antenna is used, the core of the coax must be connect close to this pin, with the shield connected to the adjacent ground pins 20 or 22.

Test Circuit (Two required to test a link)



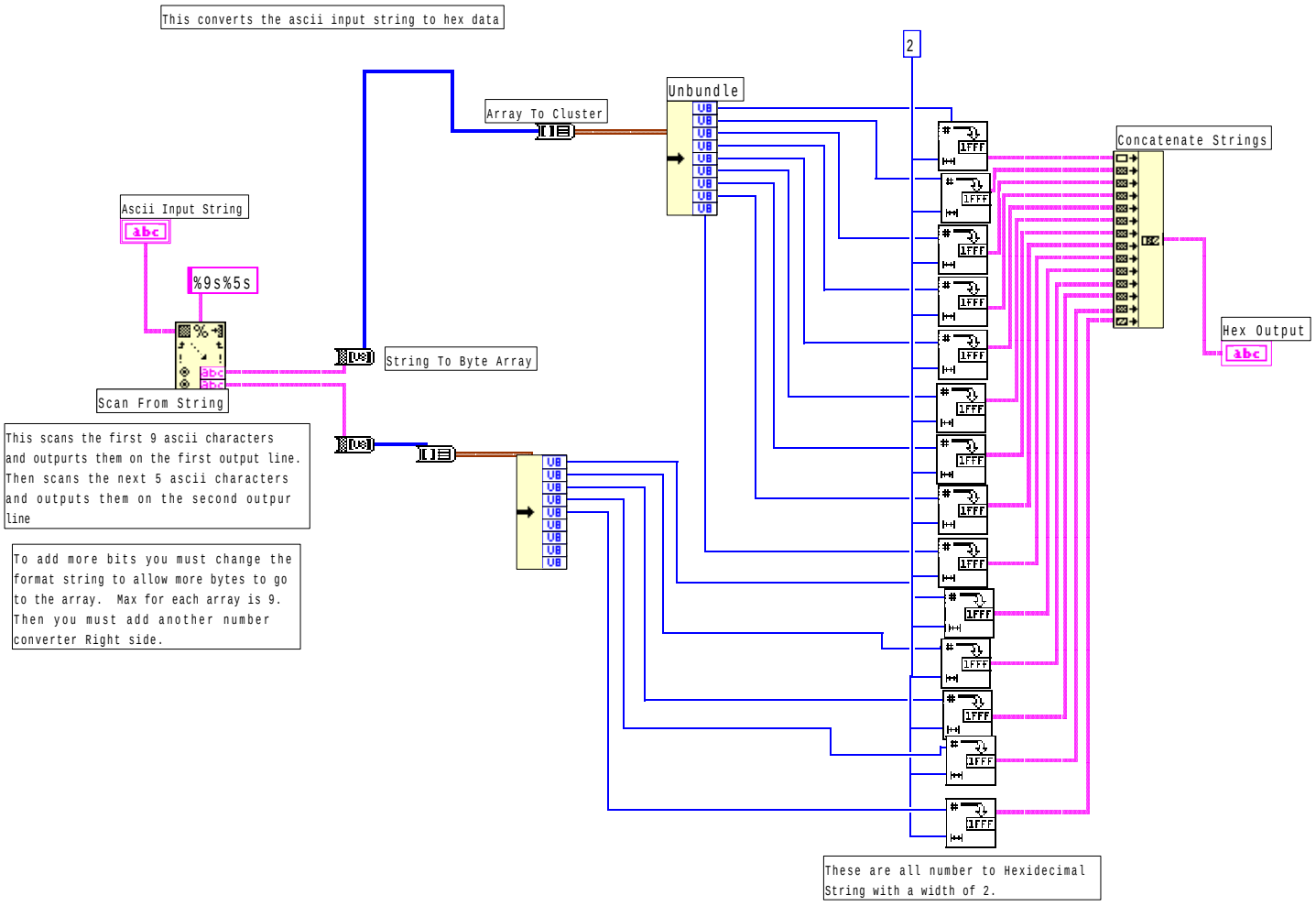
Electrical Characteristics

	Minimum	Typical	Maximum.	Units
DC LEVELS				
Supply Voltage	4.75	5	5.25	V
Supply Current		20		mA
RF				
Receiver Sensitivity		-105dBm		
RF Power Output		1		mW
FM deviation		±10		KHz
Image Rejection		50		dB
Initial Frequency accuracy		±100		Hz
Overall Frequency accuracy		±10		KHz
Max RF input into Receiver		0		dBm
Operating frequency		914.5		MHz
EMC				
Spurious Responses to 1GHz		<-36		dB
LO Leakage, conducted		<60		dBm
LO Leakage, radiated		<60		dBm
Data				
Data rates		9600		bps
Temperature				
Operating	-10		+55	°C
Storage	-40		+100	°C

Appendix C

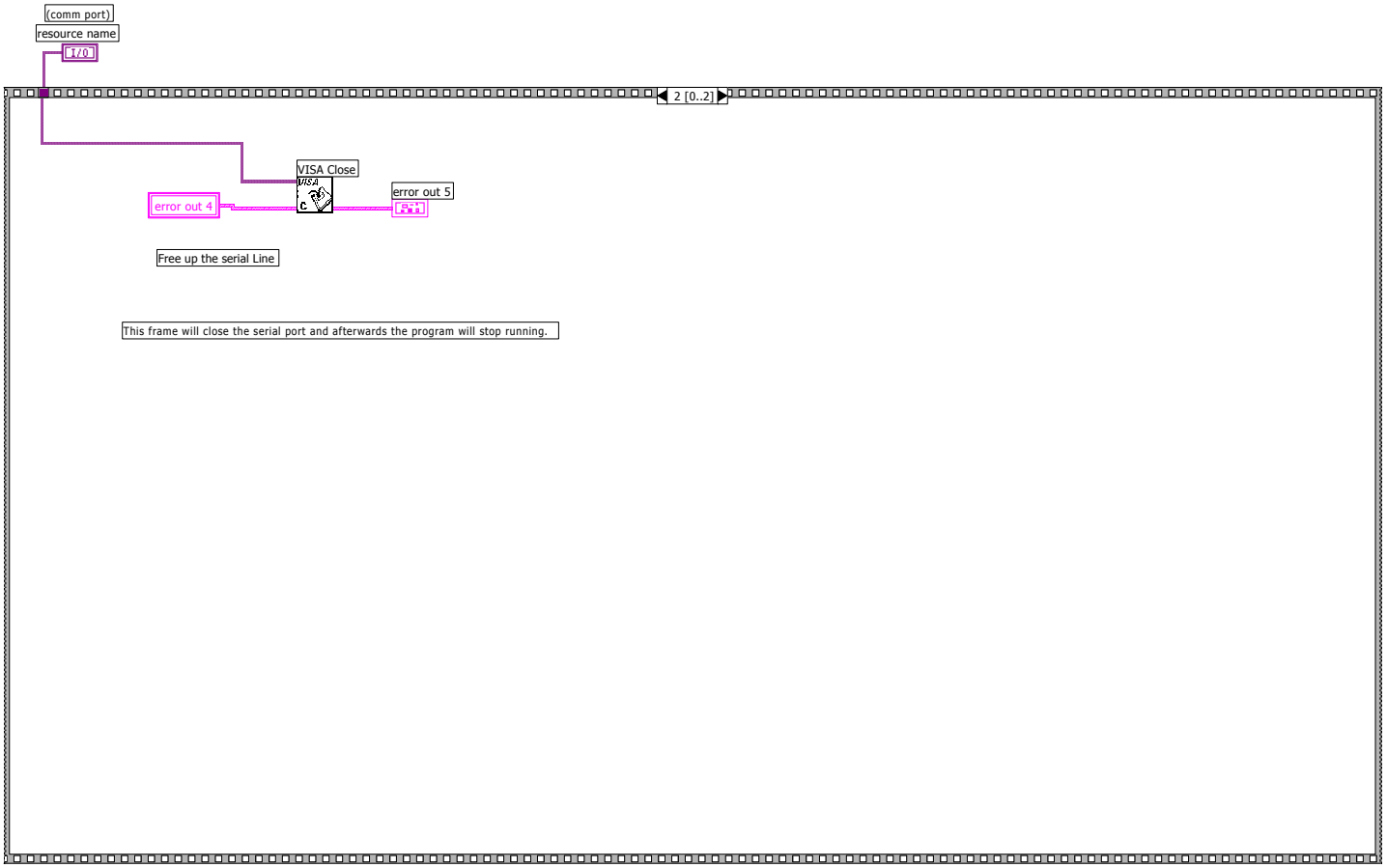
asciitohex.vi
C:\WINDOWS\Desktop\Final LabVIEW Code\asciitohex.vi
Last modified on 4/30/02 at 8:03 PM
Printed on 5/4/02 at 9:45 PM

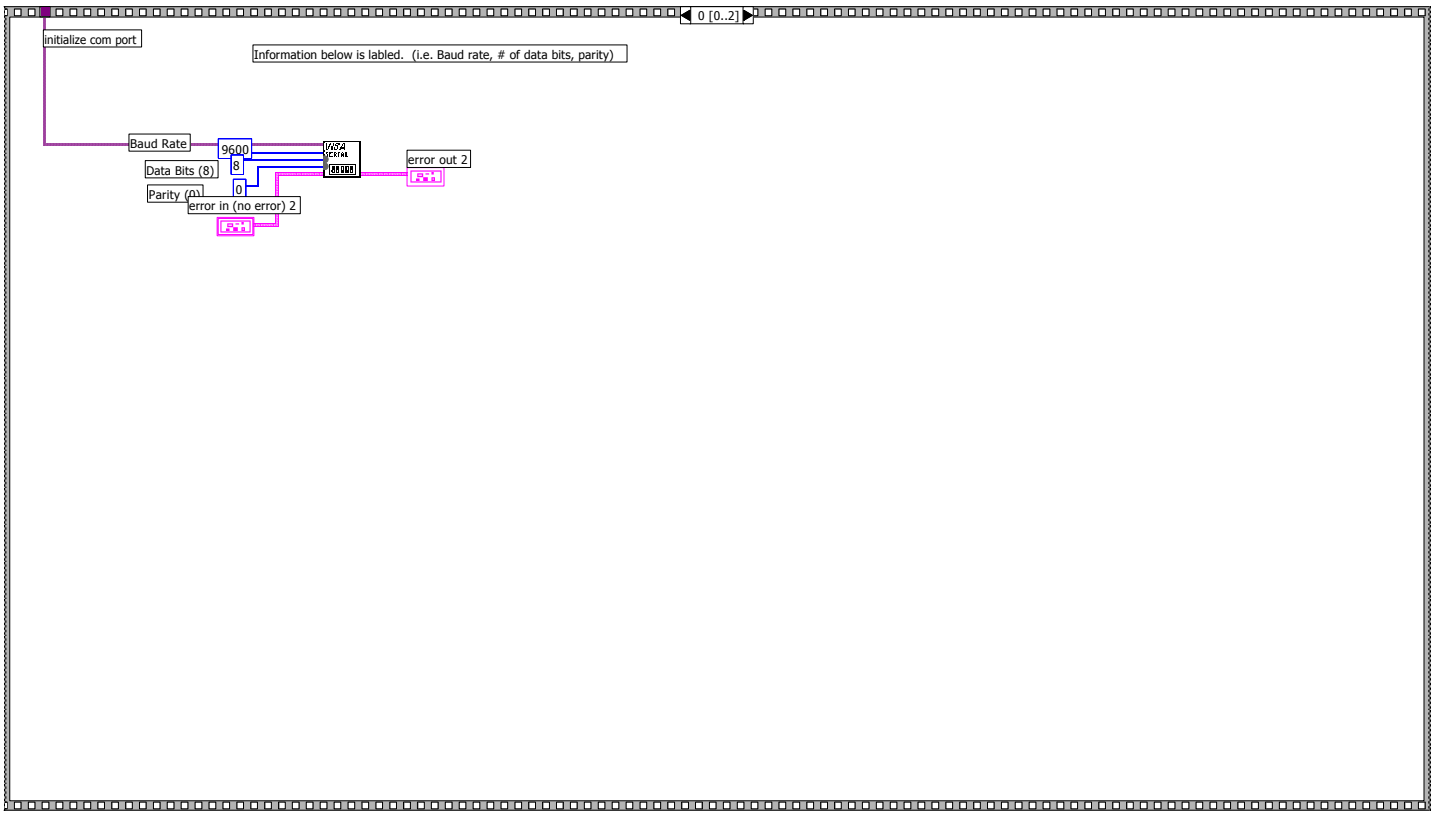
Block Diagram



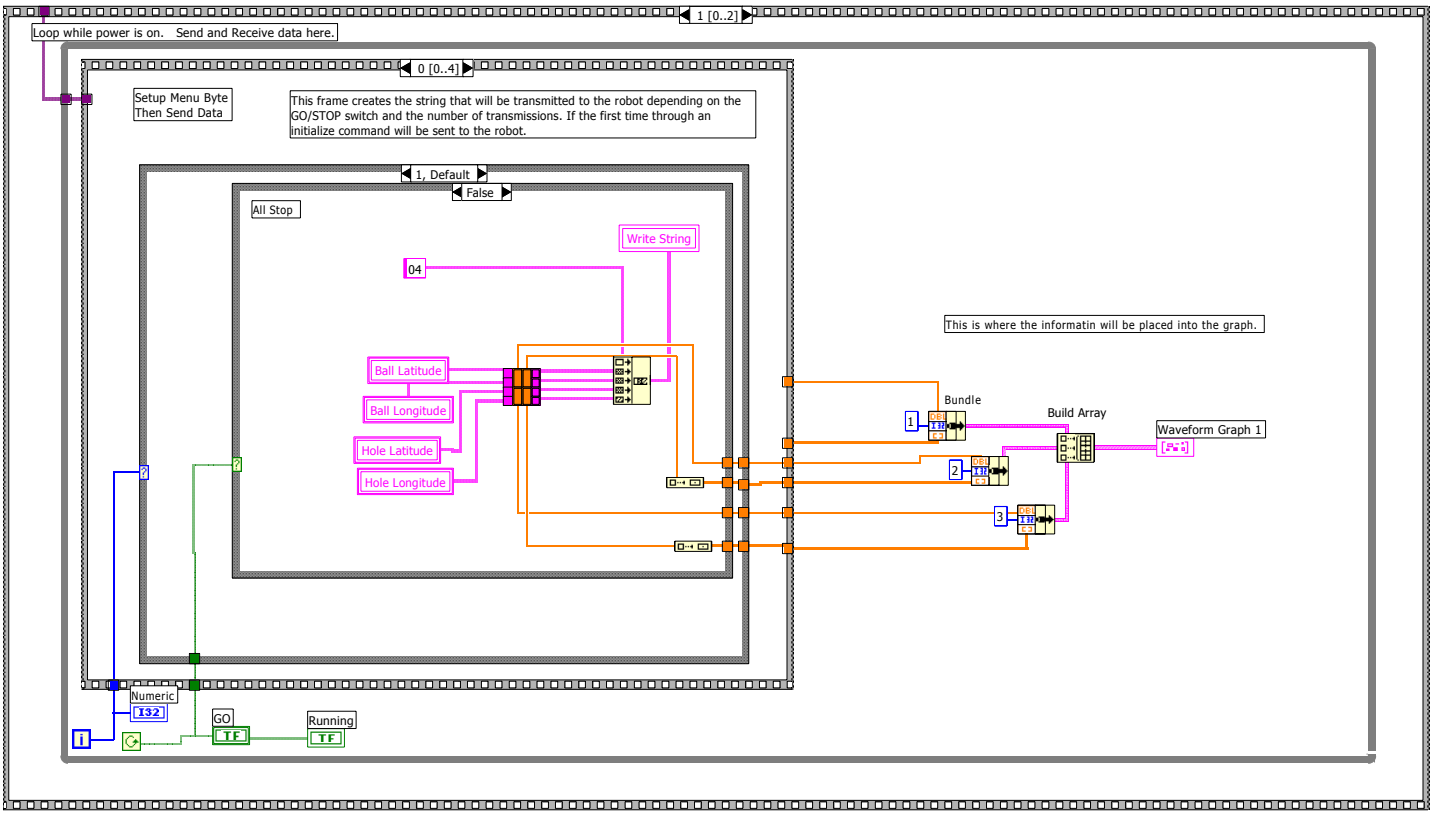
robot_communication.vi
C:\My Documents\College\classes\Spring2002\Junior Design\Software\Final LabVIEW cod\robot_communication.vi
Last modified on 5/3/2002 at 12:52 AM
Printed on 5/4/2002 at 10:15 PM

Block Diagram

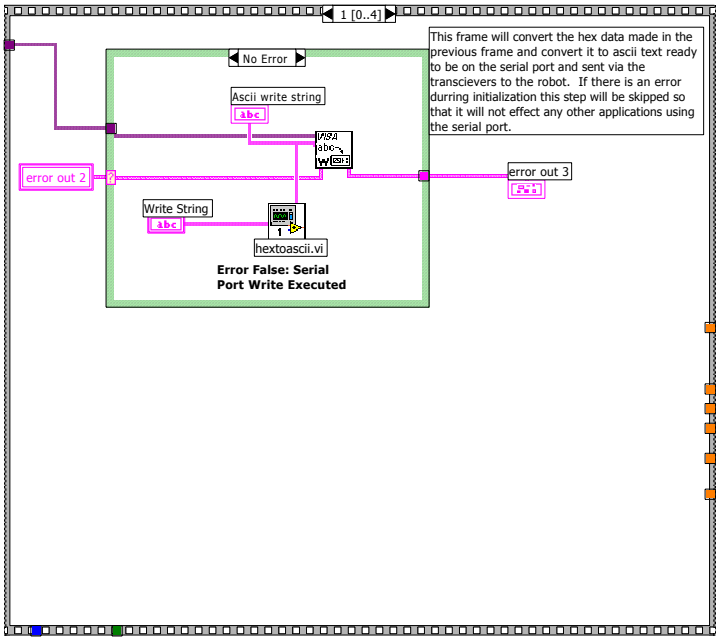


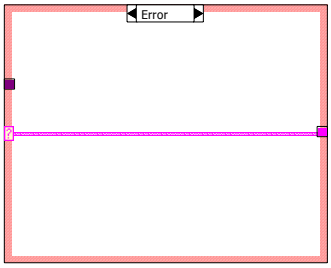


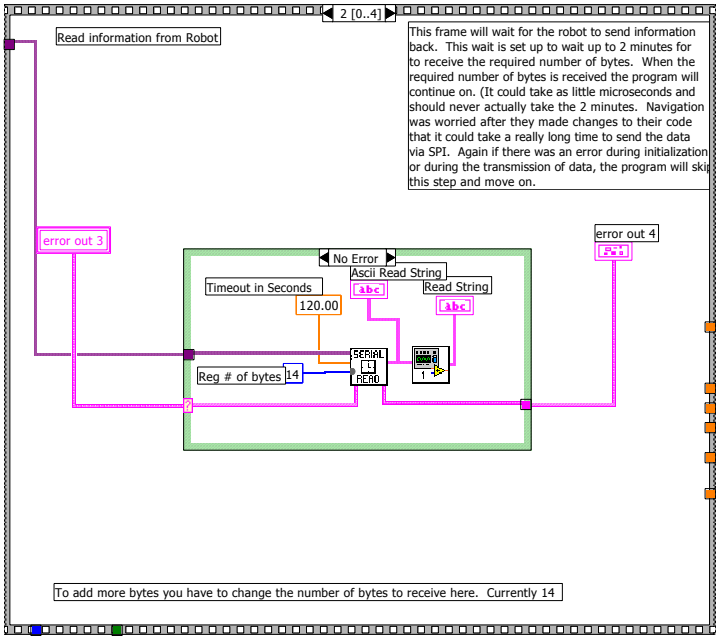
robot_communication.vi
C:\My Documents\College\classes\Spring2002\Junior Design\Software\Final LabVIEW cod\
robot_communication.vi
Last modified on 5/3/2002 at 12:52 AM
Printed on 5/4/2002 at 10:15 PM

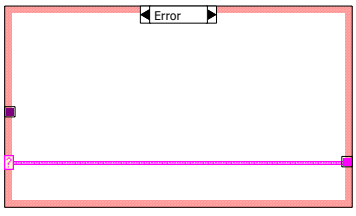


robot_communication.vi
C:\My Documents\College\classes\Spring2002\Junior Design\Software\Final LabVIEW cod\robot_communication.vi
Last modified on 5/3/2002 at 12:52 AM
Printed on 5/4/2002 at 10:15 PM

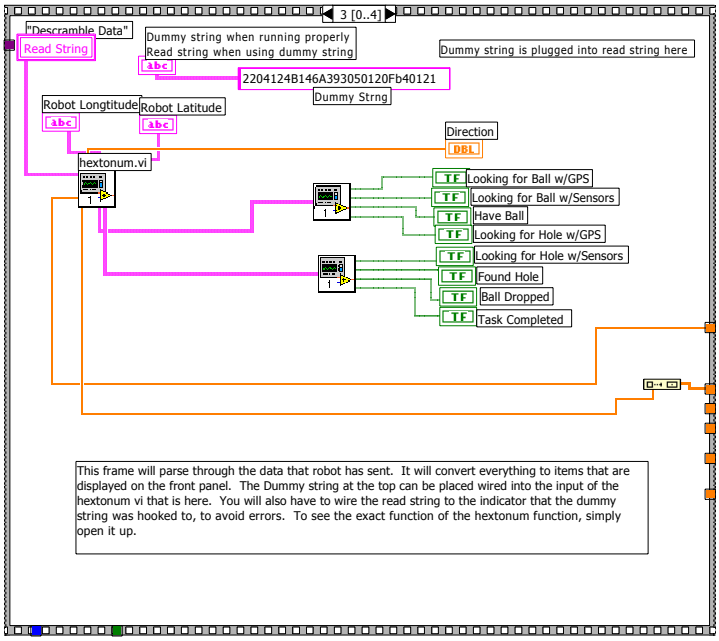


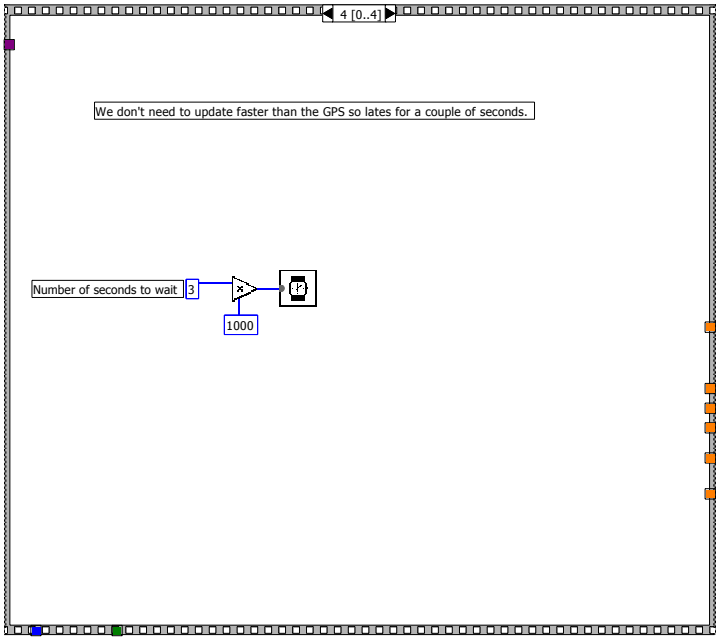






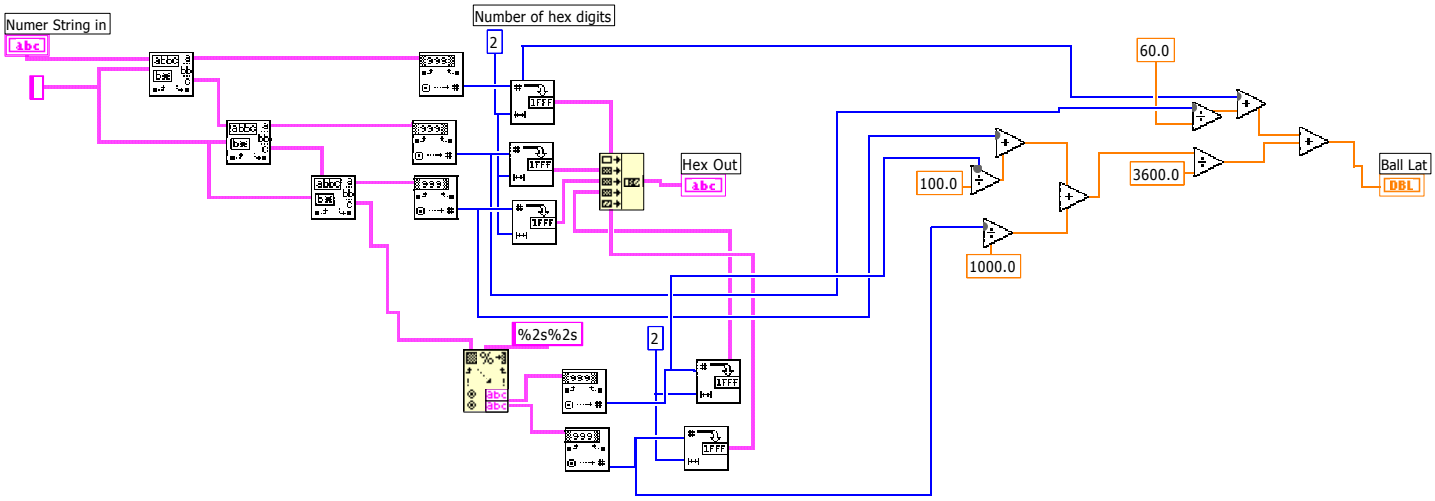
robot_communication.vi
C:\My Documents\College\classes\Spring2002\Junior Design\Software\Final LabVIEW cod\robot_communication.vi
Last modified on 5/3/2002 at 12:52 AM
Printed on 5/4/2002 at 10:15 PM





Coordtodecandhex.vi
C:\My Documents\College\classes\Spring2002\Junior Design\Software\Final LabVIEW cod\
Coordtodecandhex.vi
Last modified on 5/3/2002 at 12:53 AM
Printed on 5/4/2002 at 10:13 PM

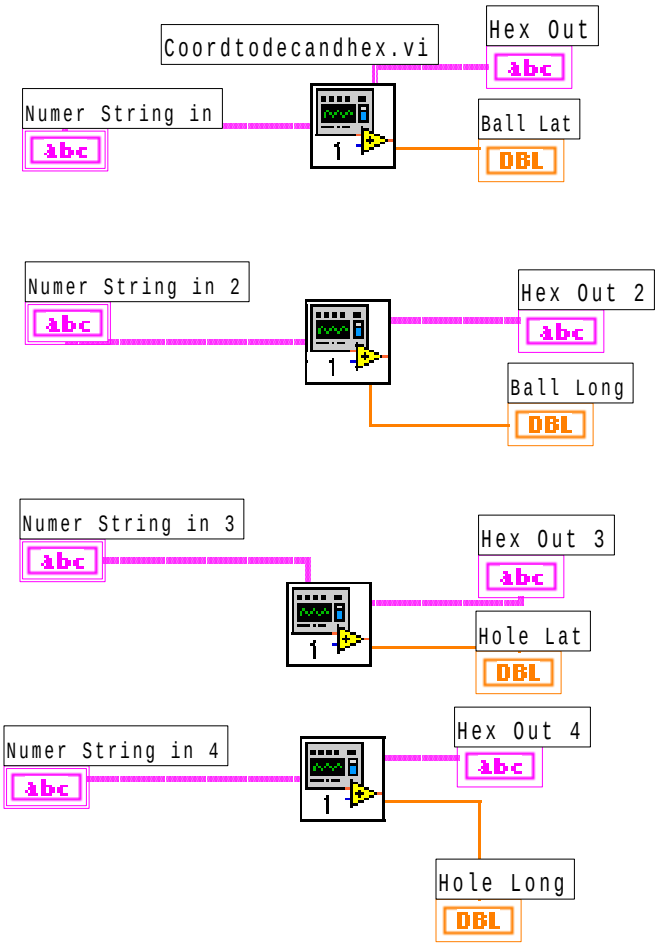
Block Diagram



This will convert the the input string from the GUI into fomratted hex data to be added to the string that is transmitted to the rbot. It will also convert the coordinates into a fractional number of degrees to be displayed on the graph in the main functin of the robot communications vi.

numtohex.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\numtohex.vi
Last modified on 5/2/02 at 11:57 PM
Printed on 5/4/02 at 10:02 PM

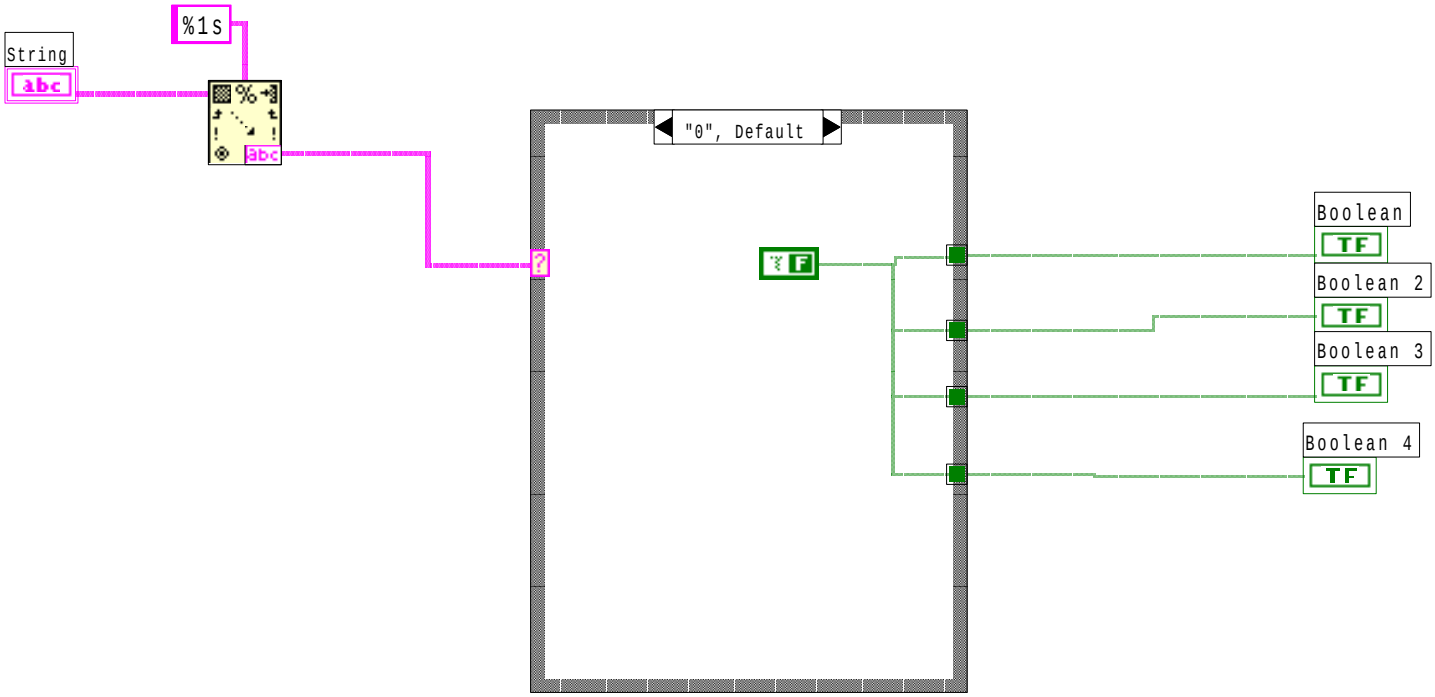
Block Diagram



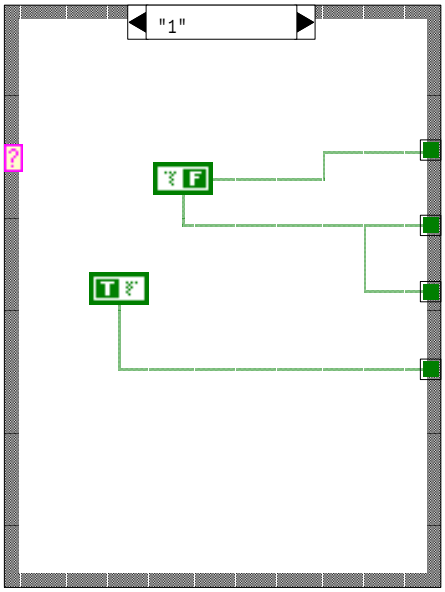
This function uses another function four times to convert the input strings into dec strings and formatted "packets" to be added to the final string to send to the transceiver

Hex string to binary.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Hex string to binary.vi
Last modified on 5/2/02 at 11:50 PM
Printed on 5/4/02 at 9:58 PM

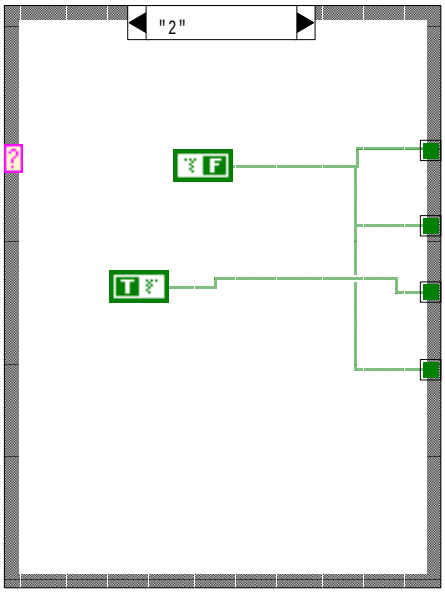
Block Diagram

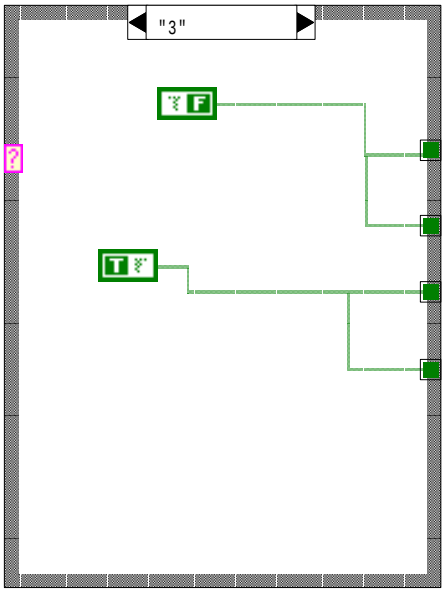


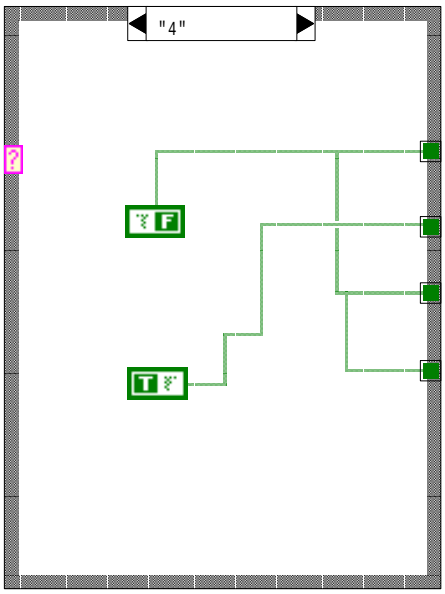
This function converts a hex digit to four binary (true/false) bits.



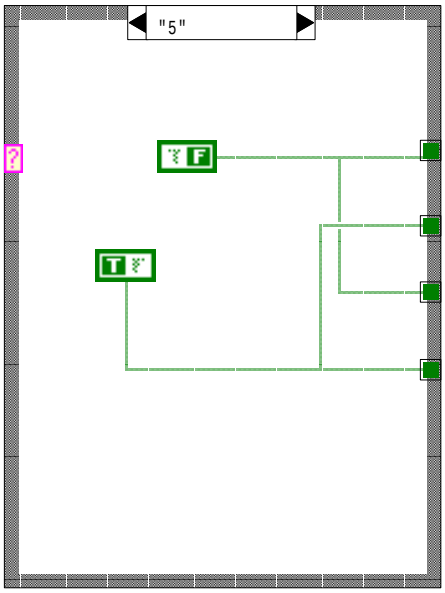
Hex string to binary.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Hex string to binary.vi
Last modified on 5/2/02 at 11:50 PM
Printed on 5/4/02 at 9:58 PM

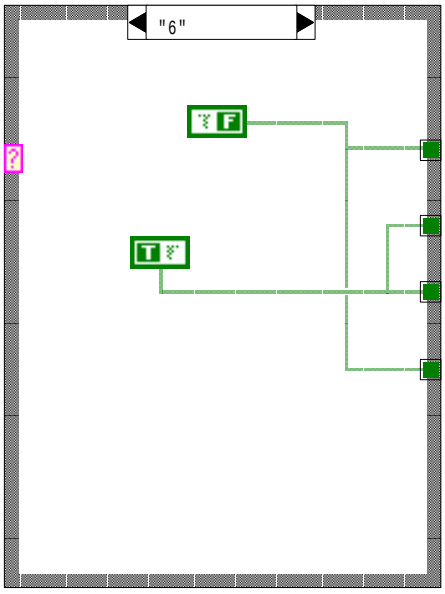


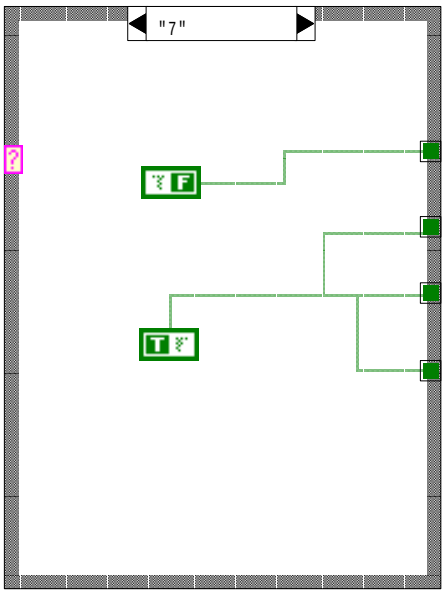




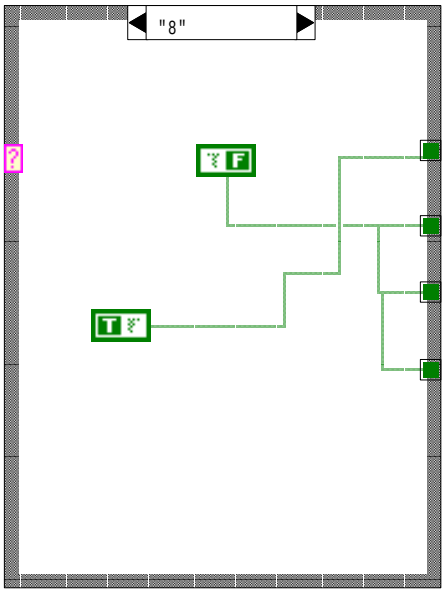
Hex string to binary.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Hex string to binary.vi
Last modified on 5/2/02 at 11:50 PM
Printed on 5/4/02 at 9:58 PM

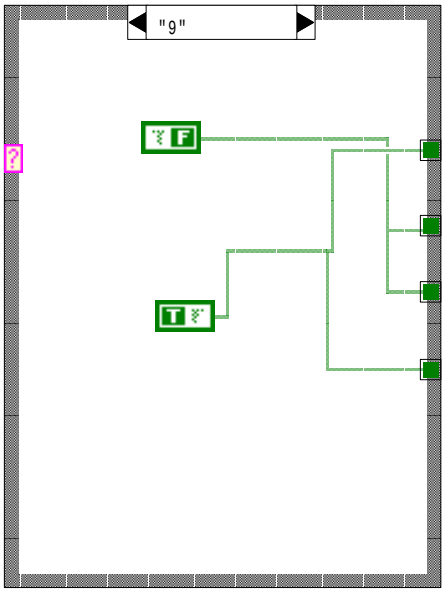


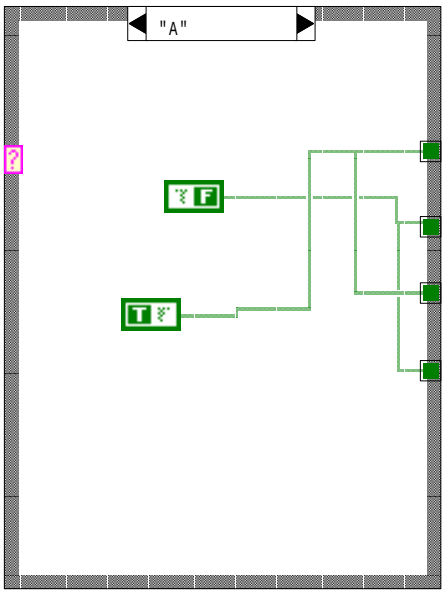




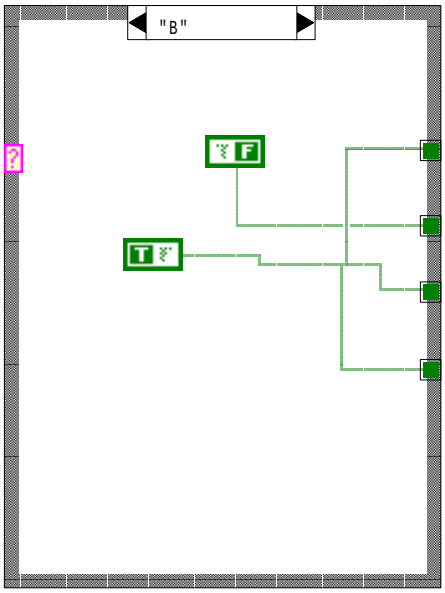
Hex string to binary.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Hex string to binary.vi
Last modified on 5/2/02 at 11:50 PM
Printed on 5/4/02 at 9:58 PM

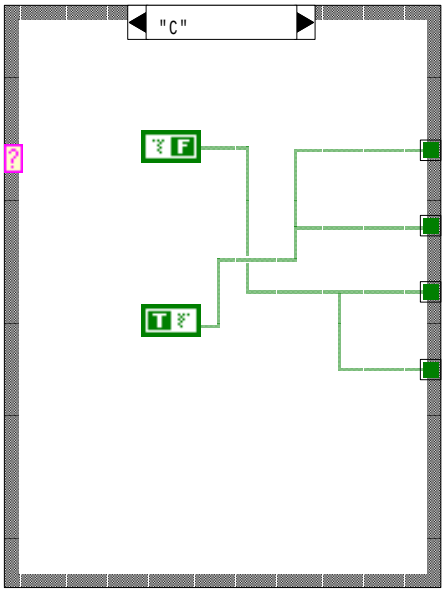


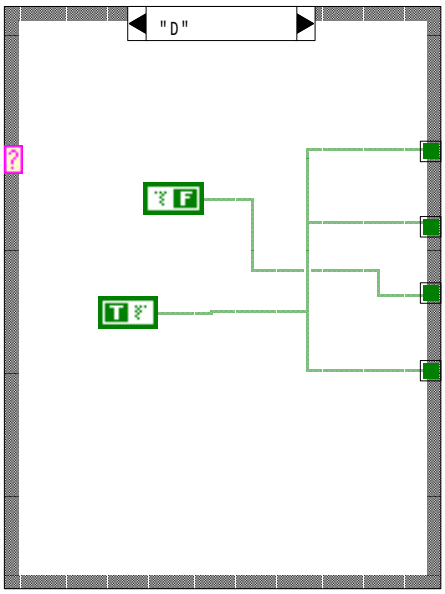




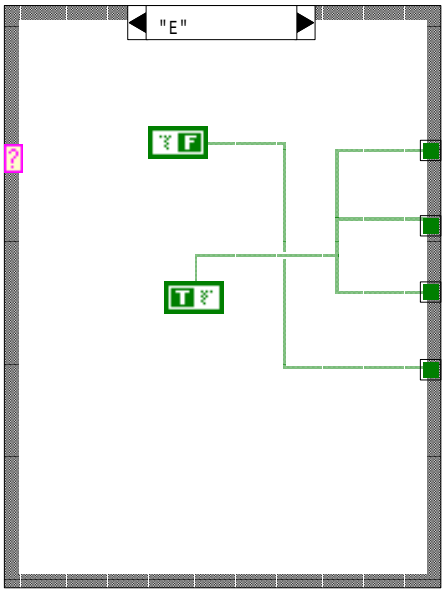
Hex string to binary.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Hex string to binary.vi
Last modified on 5/2/02 at 11:50 PM
Printed on 5/4/02 at 9:58 PM

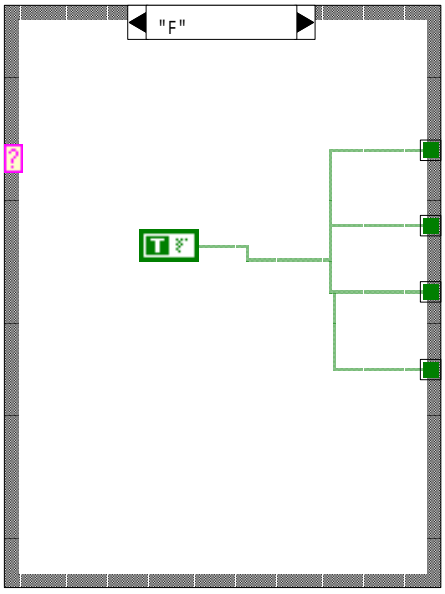






Hex string to binary.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Hex string to binary.vi
Last modified on 5/2/02 at 11:50 PM
Printed on 5/4/02 at 9:58 PM

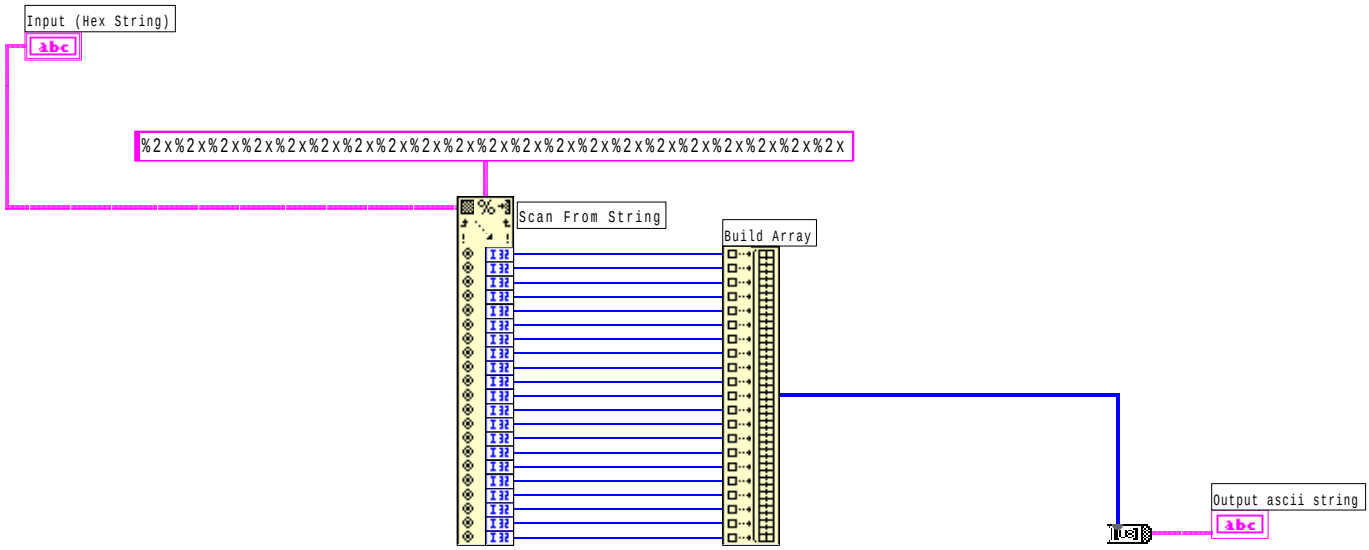




hextoascii.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\hextoascii.vi
Last modified on 5/2/02 at 11:54 PM
Printed on 5/4/02 at 10:00 PM

Block Diagram

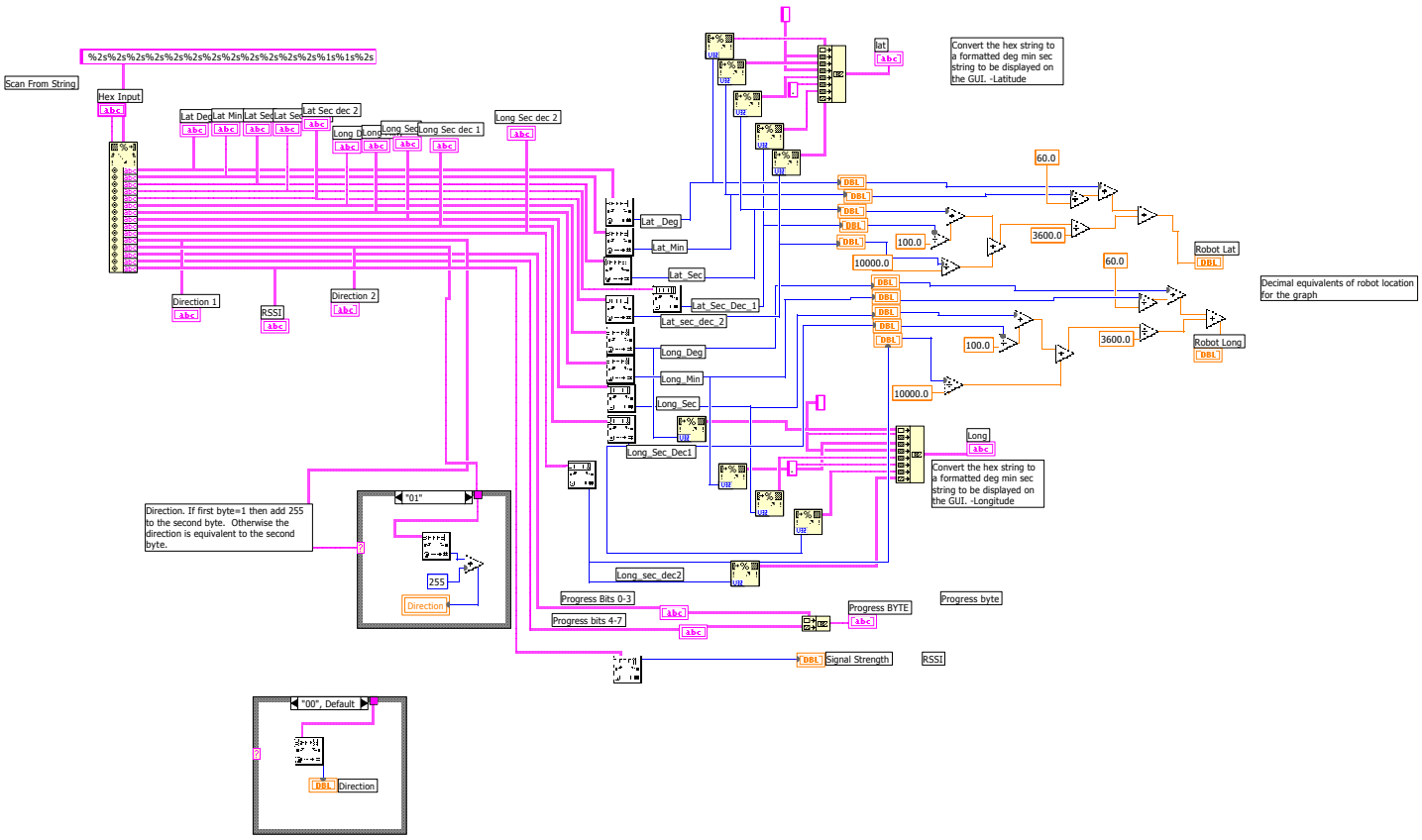
This converts the hexidecimal output string to ascii string for transmission



This program is only set up to convert 21 hex bytes to ascii character. To convert more bytes make the "scan from string" diagram and the "build array" diagram larger. You will also have to add more o"%2s" to the format string. (1 more %2s will constitute for 1 more byte of data and one more block/arrary element is equivalent to one more byte.

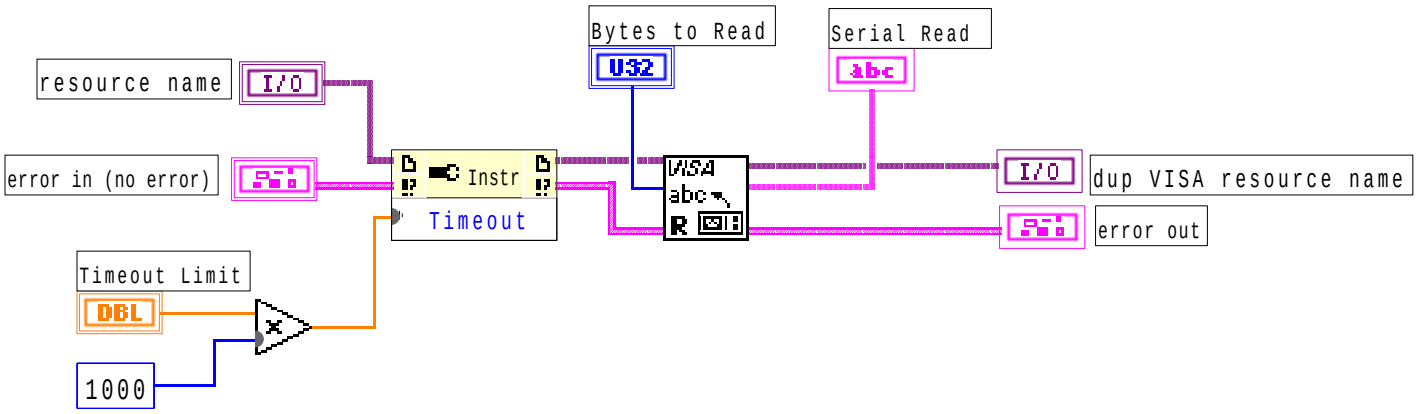
hextonum.vi
C:\My Documents\College\classes\Spring2002\Junior Design\Software\Final LabVIEW cod\hextonum.vi
Last modified on 5/3/2002 at 12:50 AM
Printed on 5/4/2002 at 10:16 PM

Block Diagram



Serial Read with Timeout.vi
C:\WINDOWS\Desktop\Final LabVIEW cod\Serial Read with Timeout.vi
Last modified on 5/2/02 at 11:58 PM
Printed on 5/4/02 at 10:04 PM

Block Diagram



Appendix D

VISA Error Codes

Code	Name	Description
-1073807360	VI_ERROR_SYSTEM_ERROR	Unknown system error (miscellaneous error).
-1073807346	VI_ERROR_INV_OBJECT VI_ERROR_INV_SESSION	The given session or object reference is invalid.
-1073807345	VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained or specified operation cannot be performed because the resource is locked.
-1073807344	VI_ERROR_INV_EXPR	Invalid expression specified for search.
-1073807343	VI_ERROR_RSRC_NFOUND	Insufficient location information or the device or resource is not present in the system.
-1073807342	VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
-1073807341	VI_ERROR_INV_ACC_MODE	Invalid access mode.
-1073807339	VI_ERROR_TMO	Timeout expired before operation completed.
-1073807338	VI_ERROR_CLOSING_FAILED	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.
-1073807333	VI_ERROR_INV_DEGREE	Specified degree is invalid.
-1073807332	VI_ERROR_INV_JOB_ID	Specified job identifier is invalid.
-1073807331	VI_ERROR_NSUP_ATTR	The specified attribute is not defined or supported by the referenced resource.
-1073807330	VI_ERROR_NSUP_ATTR_STATE	The specified state of the attribute is not valid, or is not supported as defined by the resource.
-1073807329	VI_ERROR_ATTR_READONLY	The specified attribute is read-only.
-1073807328	VI_ERROR_INV_LOCK_TYPE	The specified type of lock is not supported by this resource.
-1073807327	VI_ERROR_INV_ACCESS_KEY	The access key to the specified resource is invalid.
-1073807322	VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
-1073807321	VI_ERROR_INV_MECH	Invalid mechanism specified.
-1073807320	VI_ERROR_HNDLR_NINSTALLED	A handler was not installed.

-1073807319	VI_ERROR_INV_HNDLR_REF	The given handler reference is invalid.
-1073807318	VI_ERROR_INV_CONTEXT	Specified event context is invalid.
-1073807313	VI_ERROR_NENABLED	You must be enabled for events of the specified type in order to receive them.
-1073807312	VI_ERROR_ABORT	User abort occurred during transfer.
-1073807308	VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
-1073807307	VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
-1073807306	VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
-1073807305	VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
-1073807304	VI_ERROR_BERR	Bus error occurred during transfer.
-1073807302	VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
-1073807301	VI_ERROR_QUEUE_ERROR	Unable to queue the asynchronous operation.
-1073807300	VI_ERROR_ALLOC	Insufficient system resources to perform necessary memory allocation.
-1073807299	VI_ERROR_INV_MASK	Invalid buffer mask specified.
-1073807298	VI_ERROR_IO	Could not perform read/write operation because of I/O error.
-1073807297	VI_ERROR_INV_FMT	A format specifier in the format string is invalid.
-1073807295	VI_ERROR_NSUP_FMT	A format specifier in the format string is not supported.
-1073807294	VI_ERROR_LINE_IN_USE	The specified trigger line is currently in use.
-1073807286	VI_ERROR_SRQ_NOCCURRED	Service request has not been received for the session.
-1073807282	VI_ERROR_INV_SPACE	Invalid address space specified.
-1073807279	VI_ERROR_INV_OFFSET	Invalid offset specified.
-1073807278	VI_ERROR_INV_WIDTH	Invalid access width

		specified.
-1073807276	VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
-1073807275	VI_ERROR_NSUP_VAR_WIDTH	Cannot support source and destination widths that are different.
-1073807273	VI_ERROR_WINDOW_NMAPPED	The specified session is not currently mapped.
-1073807271	VI_ERROR_RESP_PENDING	A previous response is still pending, causing a multiple query error.
-1073807265	VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
-1073807264	VI_ERROR_NCIC	The interface associated with this session is not currently the controller in charge.
-1073807263	VI_ERROR_NSYS_CNTLR	The interface associated with this session is not the system controller.
-1073807257	VI_ERROR_NSUP_OPER	The given session or object reference does not support this operation.
-1073807254	VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
-1073807253	VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.
-1073807252	VI_ERROR_ASRL_OVERRUN	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
-1073807248	VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
-1073807247	VI_ERROR_USER_BUF	A specified user buffer is not valid or cannot be accessed for the required size.
-1073807246	VI_ERROR_RSRC_BUSY	The resource is valid, but VISA cannot currently access it.
-1073807242	VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
-1073807240	VI_ERROR_INV_PARAMETER	The value of some parameter (which parameter is not known) is invalid.
-1073807239	VI_ERROR_INV_PROT	The protocol specified is invalid.
-1073807237	VI_ERROR_INV_SIZE	Invalid size of window

		specified.
-1073807232	VI_ERROR_WINDOW_MAPPED	The specified session already contains a mapped window.
-1073807231	VI_ERROR_NIMPL_OPER	The given operation is not implemented.
-1073807229	VI_ERROR_INV_LENGTH	Invalid length specified.
-1073807204	VI_ERROR_SESN_NLOCKED	The current session did not have a lock on the resource.
-1073807202	VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
1073676290	VI_SUCCESS_EVENT_EN	Specified event is already enabled for at least one of the specified mechanisms.
1073676291	VI_SUCCESS_EVENT_DIS	Specified event is already disabled for at least one of the specified mechanisms.
1073676292	VI_SUCCESS_QUEUE_EMPTY	Operation completed successfully, but queue was already empty.
1073676293	VI_SUCCESS_TERM_CHAR	The specified termination character was read.
1073676294	VI_SUCCESS_MAX_CNT	The number of bytes transferred is equal to the input count.
1073676407	VI_WARN_CONFIG_NLOADED	The specified configuration either does not exist or could not be loaded. VISA-specified defaults will be used.
1073676413	VI_SUCCESS_DEV_NPRESENT	Session opened successfully, but the device at the specified address is not responding.
1073676418	VI_WARN_NULL_OBJECT	The specified object reference is uninitialized.
1073676416	VI_SUCCESS_QUEUE_NEMPTY	Wait terminated successfully on receipt of an event notification. There is at least one more event occurrence of the type specified by inEventType available for this session.
1073676420	VI_WARN_NSUP_ATTR_STATE	Although the specified state of the attribute is valid, it is not supported by this resource implementation.
1073676421	VI_WARN_UNKNOWN_STATUS	The status code passed to the operation could not be interpreted.
1073676424	VI_WARN_NSUP_BUF	The specified I/O buffer is

		not supported.
1073676440	VI_SUCCESS_NCHAIN	Event handled successfully. Do not invoke any other handlers on this session for this event.
1073676441	VI_SUCCESS_NESTED_SHARED	Operation completed successfully, and this session has nested shared locks.
1073676442	VI_SUCCESS_NESTED_EXCLUSIVE	Operation completed successfully, and this session has nested exclusive locks.
1073676443	VI_SUCCESS_SYNC	Operation completed successfully, but the operation was actually synchronous rather than asynchronous.

Appendix E

```
/*********************************************************************/
/* Robot B                                                           */
/* Communications                                                    */
/* Spring 2002                                                       */
/*********************************************************************/

#include "hc12.h"
#include "DBug12.h"
#define SPI_VEC (* (int *)0x0b18)

volatile char RECEIVE_BUFF[21];
volatile char TRANSMIT_BUFF[14];
volatile char NAV_RECEIVE[9];
volatile char z, count, xx;


void spi_isr(void);

/**********FUNCTIONS*****************************************************/

int RECEIVE()                       // receive 21 bytes from remote station
{
  unsigned int y=0;
  while (y < 21)
    {
      while ((SC0SR1 & 0x20) == 0);  // wait for transmit complete
      RECEIVE_BUFF[y] = SC0DRL;
      TRANSMIT_BUFF[13] = ADR0H;     //Get RSSI value
      y++;
    }
}

int TRANSMIT()                        // transmit 14 bytes to comp.
{
  unsigned int y=0;
  while (y < 14)
    {
      SC0DRL = TRANSMIT_BUFF[y];
      while ((SC0SR1 & 0x80) == 0);
      y++;
    }
}

int ORDER()
{
  // move bytes from nav to trans in proper order
  TRANSMIT_BUFF[0] = RECEIVE_BUFF[1];  //Bounce Back Robot Lat Deg
  TRANSMIT_BUFF[1] = RECEIVE_BUFF[2];  //Bounce Back Robot Lat Min
  TRANSMIT_BUFF[2] = NAV_RECEIVE[1];   //Robot Lat Seconds
  TRANSMIT_BUFF[3] = NAV_RECEIVE[2];   //Robot Lat Seconds Dec1
  TRANSMIT_BUFF[4] = NAV_RECEIVE[3];   //Robot Lat Seconds Dec2
  TRANSMIT_BUFF[5] = RECEIVE_BUFF[6];  //Bounce Back Robot Long Deg
  TRANSMIT_BUFF[6] = RECEIVE_BUFF[7];  //Bounce Back Robot Long Min
  TRANSMIT_BUFF[7] = NAV_RECEIVE[4];   //Robot Long Seconds
  TRANSMIT_BUFF[8] = NAV_RECEIVE[5];   //Robot Long Seconds Dec1
  TRANSMIT_BUFF[9] = NAV_RECEIVE[6];   //Robot Long Seconds Dec2
  TRANSMIT_BUFF[10] = NAV_RECEIVE[7];  //Direction1
  TRANSMIT_BUFF[11] = NAV_RECEIVE[8];  //Direction2


  mod_progress_byte();
}

int INITIALIZE()
{
  //send to nav
  TO_NAV();
  enable();
```

```
}

int WAIT(int waittime)
{
  int i;
  for (i=0; i<waittime;i++);
}

int STOP()
{
  //Set stop bit
  PORTA = PORTA | 0x02;
  WAIT(5000000000);
}

int mod_progress_byte()
{
  if(NAV_RECEIVE[1] == 0x01)
    {
      if(z==0) TRANSMIT_BUFF[12] = 0x01;  // ball w/gps
      if(z==1) TRANSMIT_BUFF[12] = 0x08;  // hole w/gps
    }
  else if(NAV_RECEIVE[1] == 0x02)
    {
      // ball-hole has control
      if(z==0) TRANSMIT_BUFF[12] = 0x02;  // ball w/sensors
      if(z==1) TRANSMIT_BUFF[12] = 0x10;  // hole w/sensors
    }
  else if(NAV_RECEIVE[1] == 0x04)
    {
      // objective comp
      if(z==0) TRANSMIT_BUFF[12] = 0x04;  // have ball
      if(z==1) TRANSMIT_BUFF[12] = 0x20;  // found hole - ball in hole
      z++;
      if(z==2) z=0;
    }
}

int TO_NAV()
{
  char y = 0;
  while(y < 20)                     // send # of bytes to NAV group
    {
      SP0DR = RECEIVE_BUFF[y+1];    // load register: send data to master
      PORTA = 0x01;                 // tell NAV. ready to send
      while((PORTS & 0x80) == 0x80);// Wait for SS to go low
      PORTA = 0;                    // cleary ready line
      while((PORTS & 0x80) != 0x80);// Wait for SS to go high
      y++;                          // increment for next byte
      xx = SP0DR;                   // read spi data register, clears spif
    }
}

/**********MAIN**************************************************************/

int main()
{
  int i;
  COPCTL = 0;          //turns off COP control
  i = 0;
  z = 0;
  count = 0;

/**********PORT SETUP********************************************************/
  DDRS = 0x11;         //set PORTS[0] for output, ss, sclk, mosi inputs
  DDRA = 0x01;         //set PORTA[0] output, PORTA[1-7] input
  PORTA = 0;           //clear PORTA
```

```
/**********SCI SETUP*****************************************************/
  SC0BDL = 0x34;       //set BAUD 9600
  SC0CR1 = 0x00;       //Initialize 8-bit, loop mode, Non-parity
  SC0CR2 = 0x0C;       //Transmit/Receive enable
  SC0DRH = SC0SR1;

/**********A_D SETUP*****************************************************/
  ATDCTL2 = 0x80;      //power up a-d
  ATDCTL4 = 0x01;      //8-bit operation, 4 atd clock periods
  ATDCTL5 = 0x60;      //8 conversions, continuous conversion, channel ADR0

/**********SPI SETUP*****************************************************/
  SP0CR1 = 0xCC;
  SP0CR2 = 0x00;
  SP0BR = 0x07;              // sets clock speed, clock is set by master
  SPI_VEC = (int) spi_isr;  // remaps spi interrupt vector

  while(1)
    {
    begin:
    RECEIVE();                          // receive data from comp.
    switch (RECEIVE_BUFF[0])            // if menu byte is:
      {
      case 0x01:  INITIALIZE();         // 0x01 then perform initail stuff
        break;
      case 0x02:  WAIT(1);              // 0x02 then wait for a time
        break;
      case 0x04:  STOP();               // 0x04 then stop everything
        goto begin;                     // if reached reset HC12
        break;
      default:    goto begin;
        break;
      }
   ORDER();                        // reorders data for remote station
   TRANSMIT();                     // transmit data to remote station
   }
}

/**********INTERRUPTS****************************************************/
@interrupt void spi_isr(void)
{
  if (count > 8) count = 0;          // increment counter and reset counter
  while((SP0SR & 0x80) == 0);        // wait for transmit complete
  NAV_RECEIVE[count] = SP0DR;        // store received data in buffer
  //xx = SP0DR;
  if((NAV_RECEIVE[0] == 0x02) || (NAV_RECEIVE[0] == 0x04)) count = 9;
  count++;
}
```

Appendix F

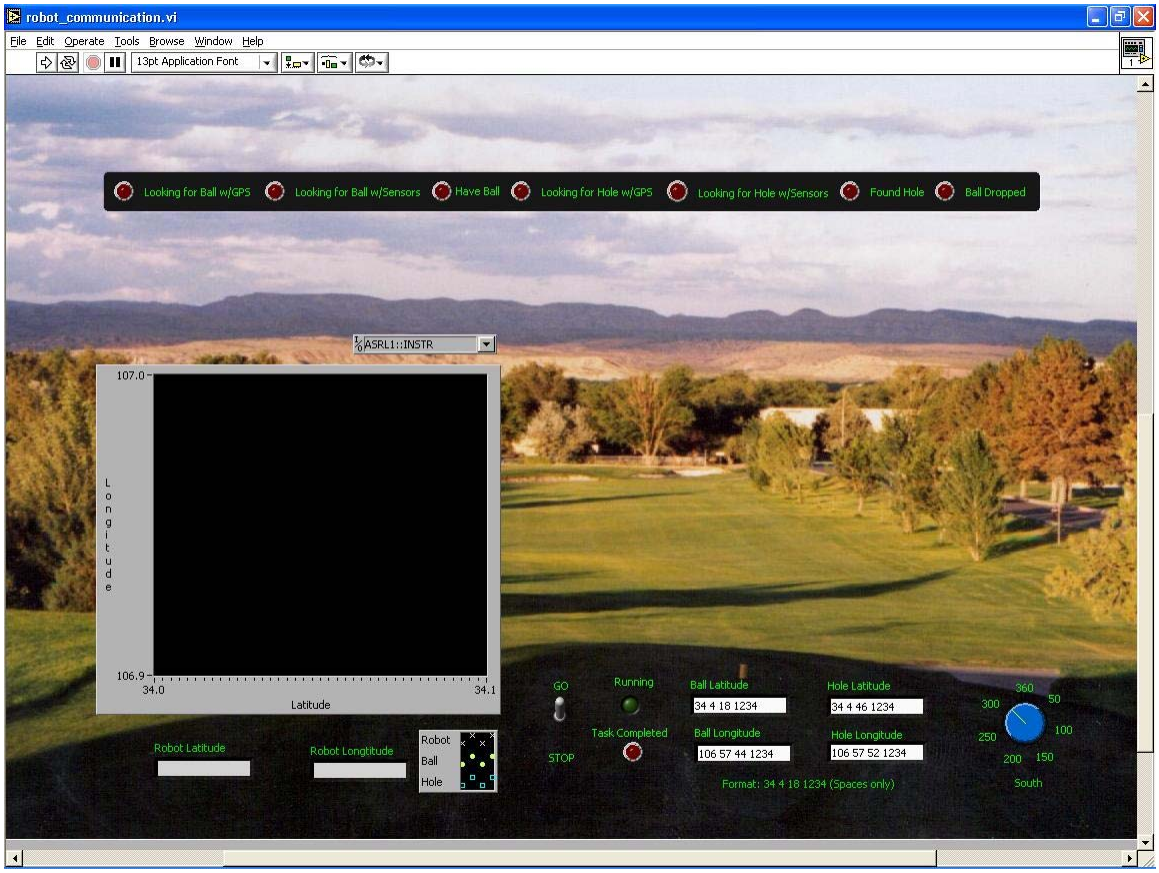
```
// Comm SPI test
// EE 382 Team B
// Preliminary Version
// 4-25-02
// Rev. alpha 2.4.2
/***************************/

#include <hc12.h>
#include <DBug12.h>

#define D_1MS (8000/4)
#define comm_SS (0x01)


void delay(unsigned int ms);
void receive(int SS, int x);
void send(int SS, int x);


char receive_array[22] = {0};
char send_array[9] = {0};



main()
{

   //COPCTL = 0;
  DDRA = 0x01;        //Set PORTA


  //Setup SPI

  DDRS = 0xE0;           //SS, SCLK, MOSI OUTPUTS
  PORTS = PORTS | 0x80;     //BRING SS HIGH TO DESELECT SLAVE;
  SP0CR1 = 0x5C;
  SP0CR2 = 0x00;      //Not bidirectional, normal mode
  SP0BR = 0x07;          //SPI clock set to 1Mhz

  delay(300);

  while ((PORTA & 0x02) == 0);   // wait for directions

  receive(comm_SS, 20);     // Receive directions from communications


  send_array[0] = 0x01; // progress byte
  send_array[1] = 0x10; // lat sec
  send_array[2] = 0x11; // lat sec dec1
  send_array[3] = 0x12; // lat sec dec2
  send_array[4] = 0x13; // long sec
  send_array[5] = 0x14; // long sec dec1
  send_array[6] = 0x15; // long sec dec2
  send_array[7] = 0x00; // direction1
  send_array[8] = 0xB4; // direction2


  delay(300);

  send(comm_SS, 9);

}

//***Receive Function***
void receive(SS, x)
{
  int i;
```

```
  //DBug12FNP->printf("entered receive function\n\r");

  for (i=0; i < x; i++)
    {
      PORTS = PORTS & ~0x80;      //BRING SS low TO SELECT SLAVE;

      SP0DR = 0xac;         //start SCLK

      while((SP0SR & 0x80) == 0);    //Wait for transfer to complete

      receive_array[i] = SP0DR;          // store data
      PORTS = PORTS | 0x80;       //BRING SS high TO DESELECT SLAVE;

      DBug12FNP->printf(" %x ",receive_array[i]);
      delay(1);
      if (i < (x-1))
   {while ((PORTA & 0x02) == 0);}    // wait for comm to be ready
    }
  DBug12FNP->printf(" \n\r");
  return;
}



//***Send Function***
void send(SS, x)
{
  int i;
  PORTS = PORTS & ~0x80;          //Set slave select low
  for (i=0; i < x; i++)
    {
      SP0DR = send_array[i];         //Send data to slave
      while((SP0SR & 0x80) == 0);       //Wait for transfer to complete
      DBug12FNP->printf(" %x ",send_array[i]);
    }
  DBug12FNP->printf("\n\rtransfer complete\n\r");
  PORTS = PORTS | 0x80;
  return;
}



/*****Delay Function*******/
void delay(unsigned int ms)
{
  int i;

  while (ms>0)
    {
      i=D_1MS;
      while (i>0)
   {
     i--;
   }
      ms--;
    }
}
```

Appendix G

Instructions Manual

THE LabVIEW Graphical User Interface



Step 1: Open the robot_communications.vi file.

Step 2: Choose which serial port to use.

LabVIEW will list all possible ports in a drop down menu right above the graph.

ASRL1::INSTR is equivalent to COM1

ASRL2::INSTR is equivalent to COM2

Step 3: Give GPS coordinates of the ball and hole.

Enter the coordinates in the form 34 12 18 1234. Spaces only.

Step 5: Make sure that the Go/Stop switch is in the “Go” position.

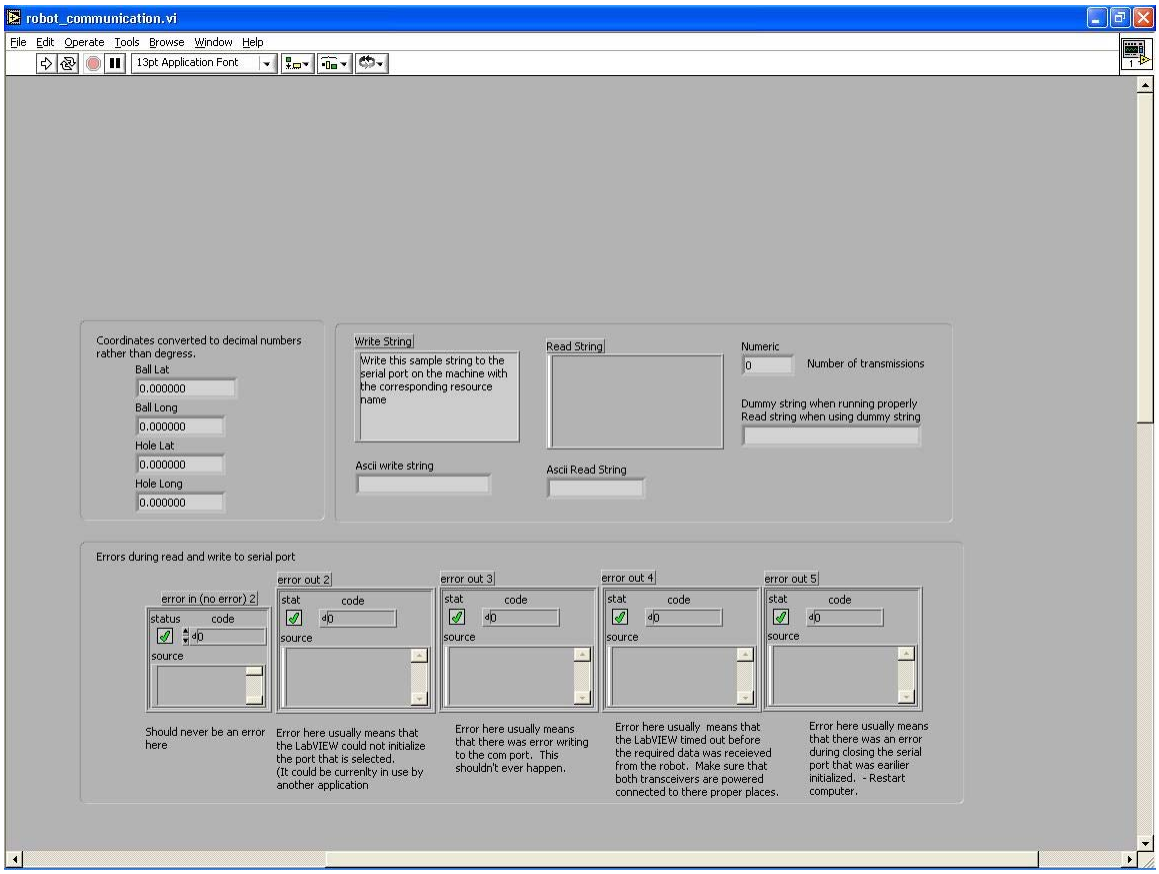
Step 6: Power up the robot. The robot should wait for LabVIEW to start running.

Step 7: Start the LabVIEW code by clicking the arrow icon in the top left of the toolbar.

Step 8: Sit back and watch.

The user will also be able to quickly view the robot coordinates given in degrees, minutes, seconds, and decimals of seconds. The compass will display the direction in degrees from North (i.e. 90° is East). The progress bar at the top of the screen displays the current activity of the robot. The user will also be able to see where the robot is in reference to the ball and hole on a graph. The graph is auto adjusting to the range needed to display the ball, hole, and robot. Finally, the "Task Completed" light will light up with the robot has finished it task.

Troubleshooting: Debugging the GUI



If the robot coordinates are all zeros

When the robot coordinates are all zeros it usually means that LabVIEW is not receiving any information from the robot. Scroll up to debugging portion of the GUI. There are five displays for errors each checking for errors in different places of the code. A quick way to check for errors is to look at status box in each of the error displays; a green check means everything is fine and a red x means that there is an error. If there is an error, the user can look up the error code. The error codes are definitions of the error; they are located in the appendix D as well as in the LabVIEW help. The most common problems have been listed by each of the error displays in the GUI and the codes should never have to be looked up. The best way to fix the problem has been listed under each error as well.

Other information displayed on GUI

Hex read and write strings

ASCII read and write strings

Number of times the program has sent data to the robot

Ball and hole latitude and longitude coordinates in degrees only