REMOTE DATA ACQUISITION USING LABVIEW

A Design Project Report

Presented to the Engineering Division of the Graduate School

of Cornell University

in Partial fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

by

Andre Harrison

Project Advisor: Andy Ruina

Degree Date: August 2006

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Remote Data Acquisition Program using LabView

Author: Andre Harrison

Abstract:

This report presents the design and testing of a LabView program that receives, plots, and saves data over a wireless RS-232 connection. The purpose of the program is to view the various changing parameter values of a walking robot in real-time in order to test and evaluate its walking cycle. The program has been tested for both functionality and speed, which will determine how many parameters can be viewed at one time.

Report Approved by

Project Advisor:_____Date:_____

ECE Project Advisor:_____Date:_____

Executive Summary

The purpose of this report is to present the design and implementation of a real-time LabView Data Acquisition program. The program is designed to be used by the Marathon Walking Robot project in order to view real-time plots of various position, velocity, acceleration, control signal, and other miscellaneous parameters in order to debug problems, study the dynamics of the system, and to potentially compare against simulation results. This report is expected to be viewed by future users of the program and those who wish to improve upon and expand it capabilities.

The LabView program was able to successfully receive data from the micro-controller unit. The program was able to plot and save multiple parameters that were received in real-time. The program was able to change what data was being saved, what parameters were plotted and how many graphs were displayed not only at initialization, but while the program was running without losing data at a transmission rate of 1kb/s. The current limitation of the program is that it must have this same functionality at 25kb/s, which it currently does not. Future improvements such as the ones mentioned in this report may reduce the program loop time so that it can accept higher data rates even, while displaying four graphs at once.

Table of Contents

List of Figures

Introduction

The goal of the Marathon Walking Robot project is to develop a bipedal robot able to set new records in walking distance and efficiency. To maximize efficiency the robots developed by the lab utilize the idea of passive dynamic walking, which allows a biped to walk stably down a shallow slope without any motors or control. In order to get the robot to walk on any surface all that is needed is the energy added due to the slope. Trying to use passive dynamics to get the robot to walk requires knowing something about the dynamics of the robot in order to understand and control the robot's motion. In order to collect this information, while the robot is walking the lab needed a way to view and collect the value of several parameters as they changed over time, such as leg position, velocity, and acceleration. In order to do this a Data Acquisition program was made.

Background

Previous Work

In August of 2005 a program was designed by Ko Ihara to plot and save data output over an RS-232 port. It is very useful for debugging code and plotting data. It is a very simple program that works, but due to its simplicity it has some limitations.

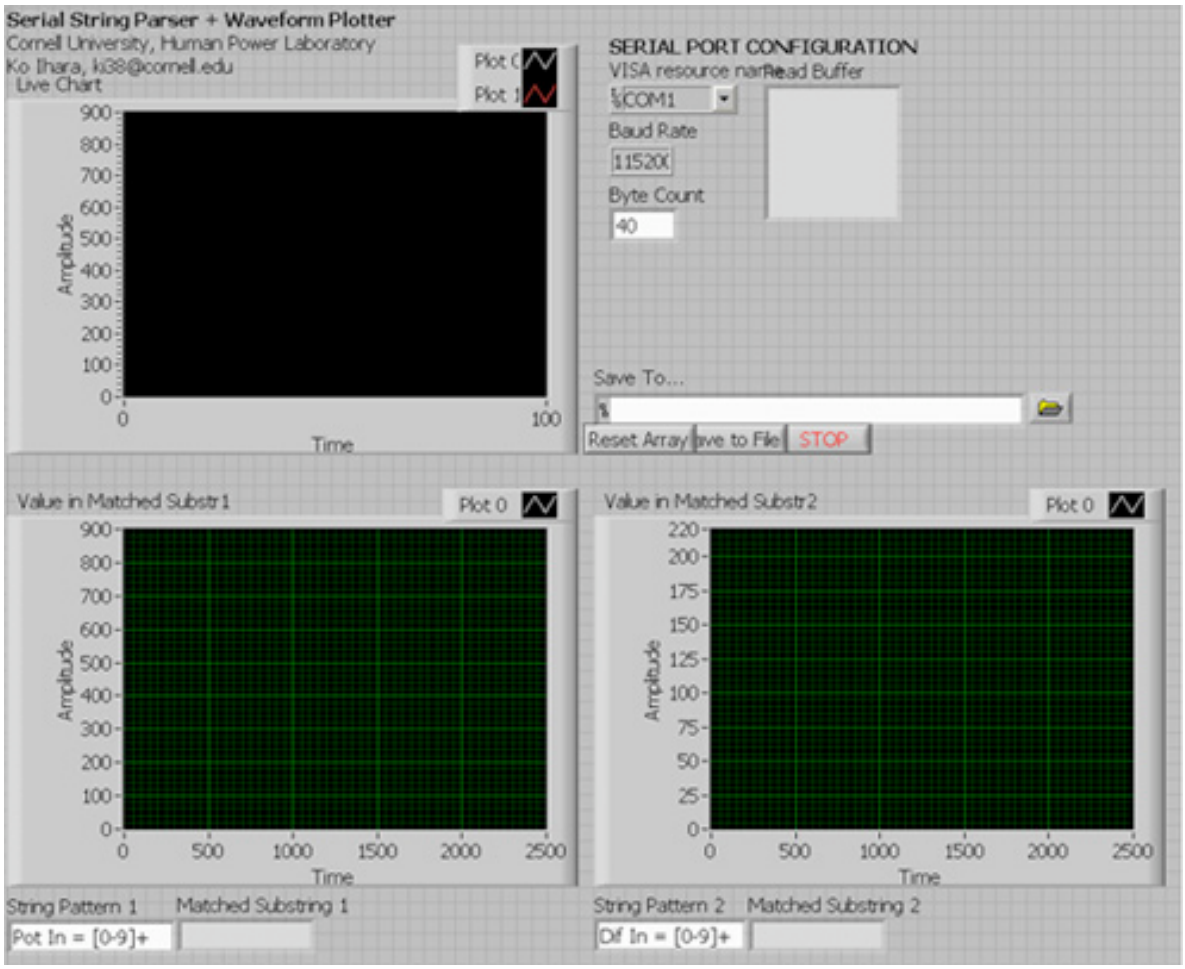


Figure 1. Front Panel of Serial DAQ

Limitations

- The code on the robot micro-controller was developed to be used with HyperTerminal as an early debugging technique and not specifically for the LabView program. Thus the Data parsing and communication is done using strings. In order to send one data point the numerical value is converted to a string and a string header is attached and then sent to the LabView program, which searches for the header and must convert the string number back into an actual number. This number is then added to an array and plotted. This requires a lot of bytes and reduces how much data can be collected in a given period of time.

- Only two graphs or sets of data can be saved or viewed at any one time.
- In order to change the data being plotted the program must be stopped and the new header string must be typed in. This means the user must know or remember the different header strings for each parameter to be plotted.

Project Description

My project serves as a major update to this in order to address the above limitations and to add some more functionality. To increase the amount of data that can be sent per second my program uses a more efficient encoding scheme for communication. The encoding details are abstracted so that the user interface is more "user friendly". The requirements for my program are 1.)Receive and save 1 – 4(minimum), 8(maximum) different parameters at 100 data points per second, (the original goal was 16); 2.) Display 1 – 4(minimum), 6(maximum) different plots in real-time; 3.) Save the last few seconds of data.

It receives data over the RS-232 port with the following port settings 8 data bits, no parity, 1 stop bit, no flow control and a baud rate of 115K. These are the standard settings for the Innovation First micro-controller, the MCU used by the project group at the start of the project and for most of this year. To remove the need of a long wire to be connected between the robot and computer, a Bluetooth RS-232 module was purchased to replace the wire. This module supported 115Kbaud, but late in the year we discovered it could only actually communicate in bursts and so could only transmit a maximum of 25Kbps. This severely cut down how much data could be received and significantly scaled back the project goals.

The purpose of the program is not to only help debug the robot, but to also allow the robot to be studied and possibly simulated based on the measured parameter values. The saved

parameters could be used to check against a simulated robot's expected parameter values as a function of time, if a constructed robot is successfully simulated.

Program Design

User Interface

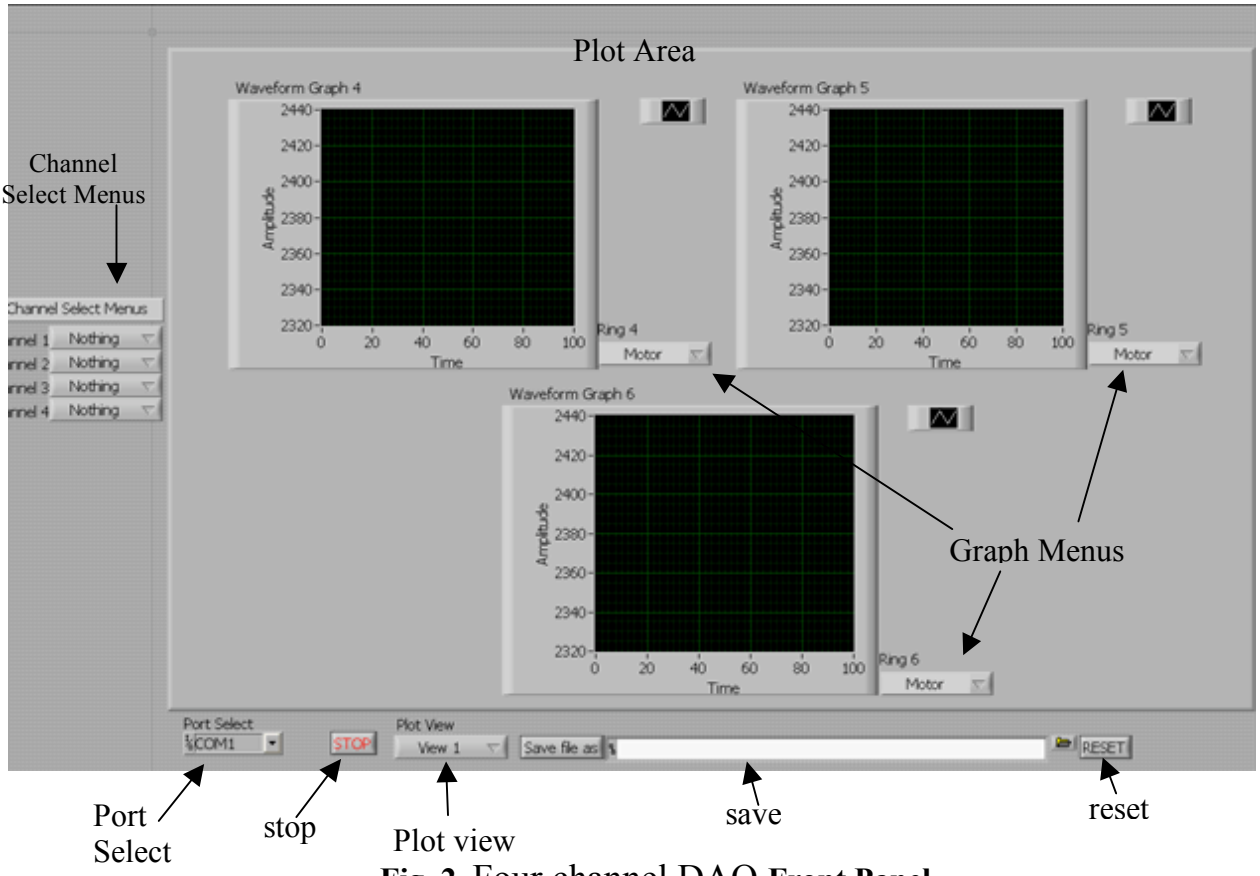


Fig. 2 Four channel DAQ **Front Panel**

The user interface is made up of the following parts:

- Plot area where 1 – 4 plots are displayed based upon the plot view selection menu.
- Graph menu – controls what parameter is plotted on the graph
- Channel select menus - control what parameter(s) are saved or can be plotted.
- Plot view menu – selects how many plots are shown in the plot area.

- Save button and file path – controls the name and location of where the data is saved and when to save the data.
- Reset button – clears all arrays storing the parameter data points.
- Port select – selects which port to read RS-232 data from.

High level view

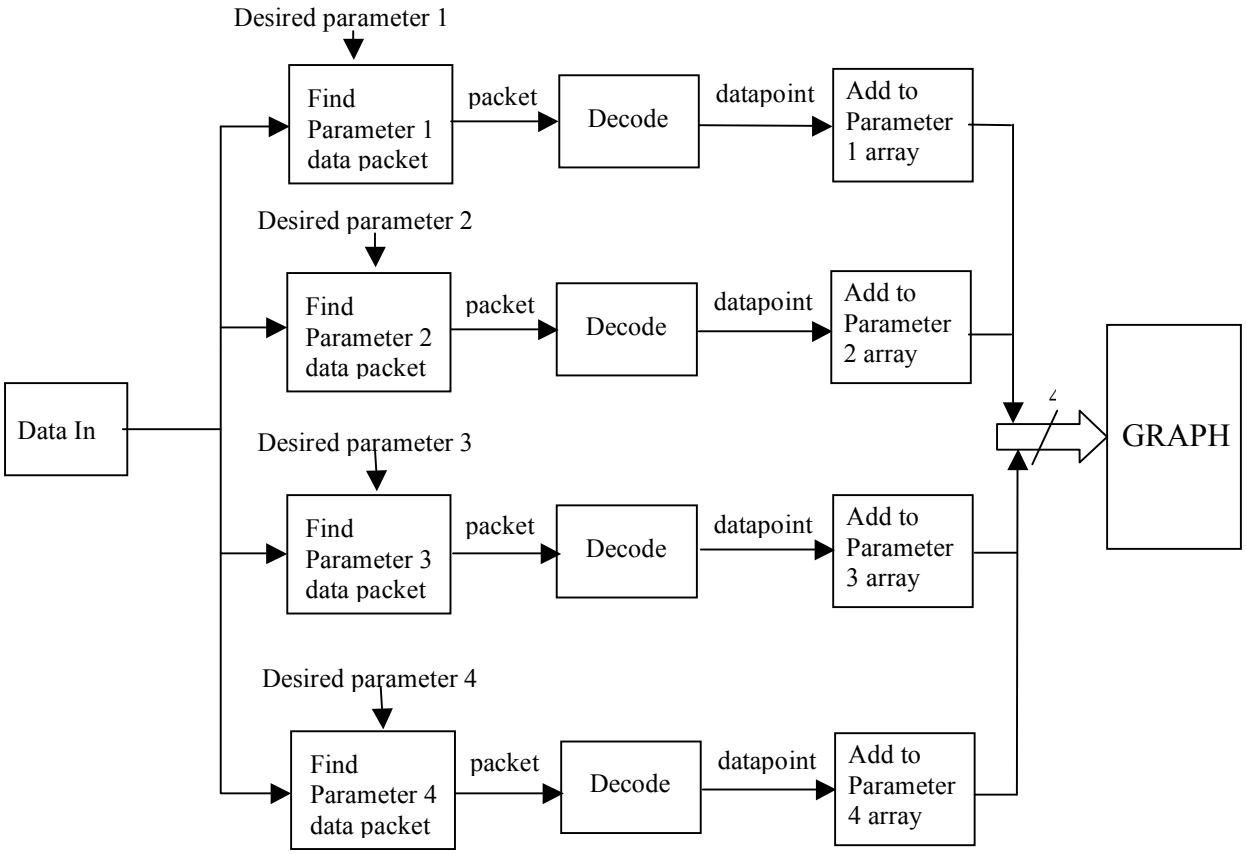


Figure 3. Program Overview

Overview

Data received over the RS-232 port is encoded into packets and each packet is made up of a header and a data point. The header not only indicates the start of the data point, but it also tells what parameter the data point is for. Based on the parameters chosen in the front panel

channel select menus the headers that correspond to those parameters are searched for in the received data. When one of the headers is found the corresponding information is decoded into a data point and added tot he end of the array of data points for that parameter. The arrays for the parameters are then sent to the graphs(s) to be plotted if that parameter was selected on the graphs menu.

Encoding scheme

The micro-controller on the robot transmits a stream of packets through the RS-232 port. Each packet consists of a header byte and five data bytes. In order to distinguish a header byte from a data byte the header is only allowed to take on values from 128 to 255 (the most significant bit is always 1), while the data bytes can only have values from 0 to 127, (the MSB is always 0). This limits the maximum number of possible parameters that can be used to 128, but this is much higher than the number of independent parameters the robot has. It also means that each data byte only holds 7 of the 32 bits for each data point. Fig. 3 below shows the encoding scheme that is being used. This encoding scheme is preferred due to its simplicity of implementation and the low overhead the scheme requires.

By setting the encoding scheme the maximum amount of data that can be saved is also fixed. Since communication is done over the RS-232 port at 115,200 baud with no parity and one stop bit every byte transmitted over the RS-232 port requires 10 bits: 8 bits for the data, plus the stop and start bits. Every data point transmitted is only four bytes, but using the encoding scheme each packet is six bytes. For each parameter there should be 100 data points received every second. Thus data for 19 different variables can be transmitted and thereby saved using this scheme. Unfortunately the wireless RS-232 modules we are using cannot support such high

data rates. It is estimated they can only transmit at ~ 25Kbps. This reduces the maximum number of variables from 19 to 4.

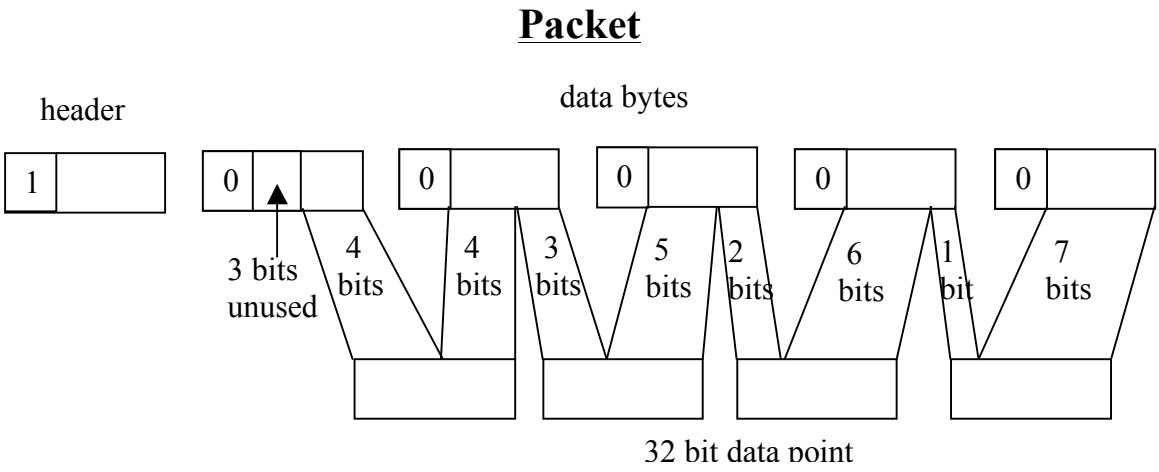


Figure 4. Data encoding diagram. The top 3 bits of the seven available bits in the most significant data byte are not used.

As the figure above shows there is no error detection used in this scheme, there are several reasons for this. The main reason is that error detection would require the micro-controller to remember old values long enough for an error to be detected in the LabView program and send a command telling the MCU to resend the data. The amount of memory and processing time this would require on the MCU were simply impractical. Also, because 100 data points per second are sent for each parameter and the robot does not move extremely quickly the parameters should not change extremely quickly so a single erroneous data point should not matter. Finally, the environment the program will be used is expected to have low data corruption so error detection is largely not needed.

Design choices

There are many possible programs that I could have chosen to create the Data Acquisition program task such C, Perl, Matlab, or Java. This version was done in LabView for several reasons. LabView is a well known program used to interface with external devices. It has many

pre-built modules able to do complex tasks very simply such as opening and closing various I/O ports, numeric, string searches, and especially real-time graph displays. In most other languages such as C, Perl, Java, and even Matlab generating a GUI would be a project in itself. In the end LabView was the best possible program to create a relatively nice GUI fairly quickly. The difficulty that comes with using LabView is that it is a completely graphical language and while everything is transferable as programs get bigger adding and modifying parts becomes more difficult. Modifying an existing LabView program is very much like rewiring a circuit schematic implemented using discrete components.

Functions

These describe the functional blocks of the program. Most of these were separate VI's, but were combined with their calling VI's in order to increase the program speed.

- DAQ_4c_v1.vi – Contains the main while loop for the entire program. Displays and plots 1-4 graphs of data based on the users choice. Also saves data in a file specified by the user.
- master ring select.vi – Called by DAQ_4c_v1.vi. Contains all the parameter names that can be saved. By altering this all the four channels select rings in DAQ_4c_v1.vi are changed at run-time.
- case structure_v3.vi – Called by DAQ_4c_v1.vi. Disables graphs not being used in order to speed up execution time. It works by making one tab view enabled and disabling the others. The tab that is enabled has the desired number of graphs on it. The VI takes in a reference to the tab control container, which controls how many plots are displayed. It also takes in the value of the plot view select in order to choose how many graphs are

displayed.

- Big parse with sync_4c.vi – Called by DAQ_4c_v1.vi. Reads in data packets, searches for the header byte for each parameter chosen by the channel select menus, decodes the data bytes into a data point and adds it to the array for that parameter. Takes in Reset command, the data read in from the RS-232 port, the values of the channel select menus, the new array for all the parameters being saved, and the new number of bytes to be read in from the RS-232 port before a search is done. The VI outputs the current array for all the parameters that are being saved, the current number of bytes to read in from the RS-232 port, and the information about the parameters being plotted.
- Sync – A subsection of Big parse with sync_4c.vi. Aligns the parse block with the incoming data packets by adjusting the number of bytes read in from the RS-232 port so that the 1st byte, of the block of bytes being searched, is the packet header. Allows the program to wait for 6*n bytes, where is n an integer, to be received before searching for the header types. The optimum number for n is the number of parameters being sent out by the M.C.U. This means that each search should find one data point per loop as long as the parameters being looked for are being sent.
- Parsing block – A subsection of Big parse with sync_4c.vi. Associates a chosen parameter to a header value and resets the array on reset. There are four parsing blocks in Big parse with sync_4c.vi one for each parameter being saved. If synchronization has been established then based on the parameter chosen in one of the channel selection menus the corresponding header is searched for in the received data packets. The block also allows the plot on the front panel to display the name of the parameter being graphed by outputting the name of the parameter. It is also bundled with a Boolean that controls

whether the plot should be displayed, if no parameter is chosen.

- comm_decoder_4c.vi – Called by parsing block section. Looks for parameter header and decodes data point into a value and adds that to the end of the array for that parameter. The VI takes in value of the header being search for, the array for that parameter, and the block of packets to be searched. It outputs the modified array.
- channel select and format_v1.2.vi – Called by DAQ_4c_v1.vi. Chooses one of the four parameters being saved, based on one of the graph menu selections, and formats the data to be plotted. There are four VI's used in DAQ_4c_v1.vi one for each graph that may be plotted. The VI takes in a reference to the graph to be plotted, the value of the graph menu, a 2-D array of all the parameters being saved, an array of the parameter information, and controls turning the graph on or off. The VI outputs the formatted graph with the name of the plot in the legend.

Test Results

A simple program was designed by Jason Cortell (the Lab manager) to test the communication protocol and the program using a newly set up micro-controller, the Isopod. The program I designed plotted and saved data successfully. It could plot on multiple graphs, change parameters of a channel, change what was being plotted, and also change the number of graphs shown without stopping the program. Although the program worked on a functional level, it was unable to run fast enough to handle all the incoming data at the rate it was being sent. At the time this report was written the program is too slow to plot 100 data points per second for four different parameters.

Future Improvements

In order to increase its speed all of the VI's would have to be combined into one. Thus the program would not have to open, run, and close multiple subVI's every loop, which is what it is doing now and is a very time consuming process. The sub VI's were created in order to make the development process easier to and to allow the overall program to be edited more easily. Now that the program is functionally working, there is little need for the subVI's.

The next step, if the program were operating at the desired speed, is to add a control to the program to allow commands to be sent to the MCU. This is a relatively simple thing to do and would allow the robot to take commands from the user; it could allow variable values and control constants to be changed during robot operation speeding up the debugging process. The communication protocol described above could still be used for transmitting data to the micro controller. Once a simple control is added a nice feature would be to get the program to pick and change what data the micro-controller is actually sending out. By telling the MCU what parameters have been selected on the channel select menus the MCU will only need to send data points for those parameters.

Right now, in order to change what header corresponds to a parameter value or to add more header values (since all 128 possible values were not programmed in) it is necessary to edit every parsing block section and change the case statements. This is a problem that arose when changing from a bunch of sub VI's to one VI. It would be better if there was one master case block that all the other case blocks inherited their cases from.

Conclusion

I feel that the work I have done on this project is largely a success. I successfully

designed a Data Acquisition program that could plot and save data in real time and tested that it functionally works. The program was unable to meet the speed requirement, but if all the VI's were combined into one file the program should be able to meet the speed requirements. Conceptually the alterations needed to meet the speed requirements are simple, but would take several hours to do and test. I hope this program find use within the Marathon Waling Robot project team and help in the goal of building an efficient walking biped.

Appendix I - Test Results

Test code used to test program

```
/* A framework for Isopod data acquisition and control.
Code which is not time-critical goes in the marked space in function
main.
Time-critical code, which must execute at specific (1 mS) time
intervals,
goes in function timedcode */

/* Freescale 56F805 peripheral base address definitions (Isopod V2) */
#define SYS_BASE 0x0c00 //; System
#define TMRA_BASE 0x0d00 //; Timer A
#define TMRB_BASE 0x0d20 //; Timer B
#define TMRC_BASE 0x0d40 //; Timer C
#define TMRD_BASE 0x0d60 //; Timer D
#define CAN_BASE 0x0d80 //; CAN network
#define PWMA_BASE 0x0e00 //; PWM A
#define PWMB_BASE 0x0e20 //; PWM B
#define DEC0_BASE 0x0e40 //; Quadrature decoder 0
#define DEC1_BASE 0x0e50 //; Quadrature decoder 1
#define ITCN_BASE 0x0e60 //; Interrupt controller
#define ADCA_BASE 0x0e80 //; Analog to digital A
#define ADCB_BASE 0x0ec0 //; Analog to digital B
#define SCI0_BASE 0x0f00 //; Serial communications interface 0
#define SCI1_BASE 0x0f10 //; Serial communications interface 1
#define SPI_BASE 0x0f20 //; Serial peripheral interface
#define COP_BASE 0x0f30 //; Computer operating properly timer
#define PFIU_BASE 0x0f40 //; Program flash interface unit
#define DFIU_BASE 0x0f60 //; Data flash interface unit
#define BFIU_BASE 0x0f80 //; Boot flash interface unit
#define CLKGEN_BASE 0x0fa0 //; Clock generator
#define GPIOA_BASE 0x0fb0 //; General-purpose i/o A
#define GPIOB_BASE 0x0fc0 //; General-purpose i/o B
#define GPIOD_BASE 0x0fe0 //; General-purpose i/o C
#define GPIOE_BASE 0x0ff0 //; General-purpose i/o D

/* Core interrupt register */
#define CORE_IPR (short *)0xfffb //; Core interrupt register

/* ITCN Interrupt controller */
#define ITCN_GRP0 (short *)(ITCN_BASE+0x00) //; Priority Level Reg. (3)
#define ITCN_GRP1 (short *)(ITCN_BASE+0x01) //; Priority Level Reg. (3)
#define ITCN_GRP2 (short *)(ITCN_BASE+0x02) //; Priority Level Reg. (3)
#define ITCN_GRP3 (short *)(ITCN_BASE+0x03) //; Priority Level Reg. (3)
#define ITCN_GRP4 (short *)(ITCN_BASE+0x04) //; Priority Level Reg. (3)
#define ITCN_GRP5 (short *)(ITCN_BASE+0x05) //; Priority Level Reg. (3)
#define ITCN_GRP6 (short *)(ITCN_BASE+0x06) //; Priority Level Reg. (3)
#define ITCN_GRP7 (short *)(ITCN_BASE+0x07) //; Priority Level Reg. (3)
#define ITCN_GRP8 (short *)(ITCN_BASE+0x08) //; Priority Level Reg. (3)
#define ITCN_GRP9 (short *)(ITCN_BASE+0x09) //; Priority Level Reg. (3)
#define ITCN_GRP10 (short *)(ITCN_BASE+0x0a) //; Priority Level Reg. (3)
#define ITCN_GRP11 (short *)(ITCN_BASE+0x0b) //; Priority Level Reg. (3)
#define ITCN_GRP12 (short *)(ITCN_BASE+0x0c) //; Priority Level Reg. (3)
#define ITCN_GRP13 (short *)(ITCN_BASE+0x0d) //; Priority Level Reg. (3)
#define ITCN_GRP14 (short *)(ITCN_BASE+0x0e) //; Priority Level Reg. (3)
#define ITCN_GRP15 (short *)(ITCN_BASE+0x0f) //; Priority Level Reg. (3)

/* PWM controller A */
#define PWMA_PMCTL (short volatile *)(PWMA_BASE+0x00) //;Control
register
#define PWMA_PMFCTL (short *)(PWMA_BASE+0x01) //;Fault
control register
#define PWMA_PMFSA (short volatile *)(PWMA_BASE+0x02) //;Fault
status & acknowledge register
#define PWMA_PMOUT (short *)(PWMA_BASE+0x03) //;Output
```

```
control register
#define PWMA_PMCNT (short volatile *)(PWMA_BASE+0x04) //;Counter
register
#define PWMA_PWMCM (short *)(PWMA_BASE+0x05) //;Counter
modulo register
#define PWMA_PWMVAL0 (short *)(PWMA_BASE+0x06) //;Value
register 0
#define PWMA_PWMVAL1 (short *)(PWMA_BASE+0x07) //;Value
register 1
#define PWMA_PWMVAL2 (short *)(PWMA_BASE+0x08) //;Value
register 2
#define PWMA_PWMVAL3 (short *)(PWMA_BASE+0x09) //;Value
register 3
#define PWMA_PWMVAL4 (short *)(PWMA_BASE+0x0a) //;Value
register 4
#define PWMA_PWMVAL5 (short *)(PWMA_BASE+0x0b) //;Value
register 5
#define PWMA_PMDEADTM (short *)(PWMA_BASE+0x0c) //;Deadtime
register
#define PWMA_PMDISMAP1 (short *)(PWMA_BASE+0x0d) //;Disable
mapping register 1
#define PWMA_PMDISMAP2 (short *)(PWMA_BASE+0x0e) //;Disable
mapping register 2
#define PWMA_PMCFG (short *)(PWMA_BASE+0x0f) //;Configure
register
#define PWMA_PMCCR (short *)(PWMA_BASE+0x10) //;Channel
control register
#define PWMA_PMPORT (short volatile *)(PWMA_BASE+0x11) //;Port
register

/* PWM controller B */
#define PWMB_PMCTL (short volatile *)(PWMB_BASE+0x00) //;Control
register
#define PWMB_PMFCTL (short *)(PWMB_BASE+0x01) //;Fault
control register
#define PWMB_PMFSA (short volatile *)(PWMB_BASE+0x02) //;Fault
status & acknowledge register
#define PWMB_PMOUT (short *)(PWMB_BASE+0x03) //;Output
control register
#define PWMB_PMCNT (short volatile *)(PWMB_BASE+0x04) //;Counter
register
#define PWMB_PWMCM (short *)(PWMB_BASE+0x05) //;Counter
modulo register
#define PWMB_PWMVAL0 (short *)(PWMB_BASE+0x06) //;Value
register 0
#define PWMB_PWMVAL1 (short *)(PWMB_BASE+0x07) //;Value
register 1
#define PWMB_PWMVAL2 (short *)(PWMB_BASE+0x08) //;Value
register 2
#define PWMB_PWMVAL3 (short *)(PWMB_BASE+0x09) //;Value
register 3
#define PWMB_PWMVAL4 (short *)(PWMB_BASE+0x0a) //;Value
register 4
#define PWMB_PWMVAL5 (short *)(PWMB_BASE+0x0b) //;Value
register 5
#define PWMB_PMDEADTM (short *)(PWMB_BASE+0x0c) //;Deadtime
register
#define PWMB_PMDISMAP1 (short *)(PWMB_BASE+0x0d) //;Disable
mapping register 1
#define PWMB_PMDISMAP2 (short *)(PWMB_BASE+0x0e) //;Disable
mapping register 2
#define PWMB_PMCFG (short *)(PWMB_BASE+0x0f) //;Configure
register
#define PWMB_PMCCR (short *)(PWMB_BASE+0x10) //;Channel
control register
#define PWMB_PMPORT (short volatile *)(PWMB_BASE+0x11) //;Port
register


/* ADCA Analog-to-digital converter A*/
```

```
#define ADCA_ADCR1 (short *)(ADCA_BASE+0x00) //; ADC A
control register 1
#define ADCA_ADCR2 (short *)(ADCA_BASE+0x01) //; ADC A
control register 2
#define ADCA_ADZCC (short *)(ADCA_BASE+0x02) //; ADC A
zero-crossing control register
#define ADCA_ADLST1 (short *)(ADCA_BASE+0x03) //; ADC A
channel list register 1
#define ADCA_ADLST2 (short *)(ADCA_BASE+0x04) //; ADC A
channel list register 2
#define ADCA_ADSDIS (short *)(ADCA_BASE+0x05) //; ADC A
sample disable register
#define ADCA_ADSTAT (short volatile *)(ADCA_BASE+0x06) //; ADC A
status register
#define ADCA_ADLSTAT (short volatile *)(ADCA_BASE+0x07) //; ADC A limit
status register
#define ADCA_ADZCSTAT (short volatile *)(ADCA_BASE+0x08) //; ADC A
zero-crossing status register
#define ADCA_ADRSLT0 (short volatile *)(ADCA_BASE+0x09) //; ADC A ch 0
result register
#define ADCA_ADRSLT1 (short volatile *)(ADCA_BASE+0x0a) //; ADC A ch 1
result register
#define ADCA_ADRSLT2 (short volatile *)(ADCA_BASE+0x0b) //; ADC A ch 2
result register
#define ADCA_ADRSLT3 (short volatile *)(ADCA_BASE+0x0c) //; ADC A ch 3
result register
#define ADCA_ADRSLT4 (short volatile *)(ADCA_BASE+0x0d) //; ADC A ch 4
result register
#define ADCA_ADRSLT5 (short volatile *)(ADCA_BASE+0x0e) //; ADC A ch 5
result register
#define ADCA_ADRSLT6 (short volatile *)(ADCA_BASE+0x0f) //; ADC A ch 6
result register
#define ADCA_ADRSLT7 (short volatile *)(ADCA_BASE+0x10) //; ADC A ch 7
result register
#define ADCA_ADLLMT0 (short *)(ADCA_BASE+0x11) //; ADC A ch 0
low limit register
#define ADCA_ADLLMT1 (short *)(ADCA_BASE+0x12) //; ADC A ch 1
low limit register
#define ADCA_ADLLMT2 (short *)(ADCA_BASE+0x13) //; ADC A ch 2
low limit register
#define ADCA_ADLLMT3 (short *)(ADCA_BASE+0x14) //; ADC A ch 3
low limit register
#define ADCA_ADLLMT4 (short *)(ADCA_BASE+0x15) //; ADC A ch 4
low limit register
#define ADCA_ADLLMT5 (short *)(ADCA_BASE+0x16) //; ADC A ch 5
low limit register
#define ADCA_ADLLMT6 (short *)(ADCA_BASE+0x17) //; ADC A ch 6
low limit register
#define ADCA_ADLLMT7 (short *)(ADCA_BASE+0x18) //; ADC A ch 7
low limit register
#define ADCA_ADHLMT0 (short *)(ADCA_BASE+0x19) //; ADC A ch 0
high limit register
#define ADCA_ADHLMT1 (short *)(ADCA_BASE+0x1a) //; ADC A ch 1
high limit register
#define ADCA_ADHLMT2 (short *)(ADCA_BASE+0x1b) //; ADC A ch 2
high limit register
#define ADCA_ADHLMT3 (short *)(ADCA_BASE+0x1c) //; ADC A ch 3
high limit register
#define ADCA_ADHLMT4 (short *)(ADCA_BASE+0x1d) //; ADC A ch 4
high limit register
#define ADCA_ADHLMT5 (short *)(ADCA_BASE+0x1e) //; ADC A ch 5
high limit register
#define ADCA_ADHLMT6 (short *)(ADCA_BASE+0x1f) //; ADC A ch 6
high limit register
#define ADCA_ADHLMT7 (short *)(ADCA_BASE+0x20) //; ADC A ch 7
high limit register
#define ADCA_ADOFS0 (short *)(ADCA_BASE+0x21) //; ADC A ch 0
offset register
#define ADCA_ADOFS1 (short *)(ADCA_BASE+0x22) //; ADC A ch 1
offset register
```

```
#define ADCA_ADOFS2 (short *)(ADCA_BASE+0x23) //; ADC A ch 2
offset register
#define ADCA_ADOFS3 (short *)(ADCA_BASE+0x24) //; ADC A ch 3
offset register
#define ADCA_ADOFS4 (short *)(ADCA_BASE+0x25) //; ADC A ch 4
offset register
#define ADCA_ADOFS5 (short *)(ADCA_BASE+0x26) //; ADC A ch 5
offset register
#define ADCA_ADOFS6 (short *)(ADCA_BASE+0x27) //; ADC A ch 6
offset register
#define ADCA_ADOFS7 (short *)(ADCA_BASE+0x28) //; ADC A ch 7
offset register

/* TMRA0 Quad counter-timer A timer 0 */
#define TMRA0_CMP1 (short *)(TMRA_BASE+0x00) //; Counter A0
compare 1 reg.
#define TMRA0_CMP2 (short *)(TMRA_BASE+0x01) //; Counter A0
compare 2 reg.
#define TMRA0_CAP (volatile short *)(TMRA_BASE+0x02) //; Counter A0
capture reg.
#define TMRA0_LOAD (short *)(TMRA_BASE+0x03) //; Counter A0
load reg.
#define TMRA0_HOLD (volatile short *)(TMRA_BASE+0x04) //; Counter A0
hold reg.
#define TMRA0_CNTR (volatile short *)(TMRA_BASE+0x05) //; Counter A0
value reg.
#define TMRA0_CTRL (short *)(TMRA_BASE+0x06) //; Counter A0
control reg.
#define TMRA0_SCR (volatile short *)(TMRA_BASE+0x07) //; Counter A0
status control reg.

/* TMRA1 Quad counter-timer A timer 1 */
#define TMRA1_CMP1 (short *)(TMRA_BASE+0x08) //; Counter A1
compare 1 reg.
#define TMRA1_CMP2 (short *)(TMRA_BASE+0x09) //; Counter A1
compare 2 reg.
#define TMRA1_CAP (volatile short *)(TMRA_BASE+0x0a) //; Counter A1
capture reg.
#define TMRA1_LOAD (short *)(TMRA_BASE+0x0b) //; Counter A1
load reg.
#define TMRA1_HOLD (volatile short *)(TMRA_BASE+0x0c) //; Counter A1
hold reg.
#define TMRA1_CNTR (volatile short *)(TMRA_BASE+0x0d) //; Counter A1
value reg.
#define TMRA1_CTRL (short *)(TMRA_BASE+0x0e) //; Counter A1
control reg.
#define TMRA1_SCR (volatile short *)(TMRA_BASE+0x0f) //; Counter A1
status control

/* TMRA2 Quad counter-timer A timer 2 */
#define TMRA2_CMP1 (short *)(TMRA_BASE+0x10) //; Counter A2
compare 1 reg.
#define TMRA2_CMP2 (short *)(TMRA_BASE+0x11) //; Counter A2
compare 2 reg.
#define TMRA2_CAP (volatile short *)(TMRA_BASE+0x12) //; Counter A2
capture reg.
#define TMRA2_LOAD (short *)(TMRA_BASE+0x13) //; Counter A2
load reg.
#define TMRA2_HOLD (volatile short *)(TMRA_BASE+0x14) //; Counter A2
hold reg.
#define TMRA2_CNTR (volatile short *)(TMRA_BASE+0x15) //; Counter A2
value reg.
#define TMRA2_CTRL (short *)(TMRA_BASE+0x16) //; Counter A2
control reg.
#define TMRA2_SCR (volatile short *)(TMRA_BASE+0x17) //; Counter A2
status control

/* TMRA3 Quad counter-timer A timer 3 */
#define TMRA3_CMP1 (short *)(TMRA_BASE+0x18) //; Counter A3
compare 1 reg.
```

```
#define TMRA3_CMP2 (short *)(TMRA_BASE+0x19) //; Counter A3
compare 2 reg.
#define TMRA3_CAP (volatile short *)(TMRA_BASE+0x1a) //; Counter A3
capture reg.
#define TMRA3_LOAD (short *)(TMRA_BASE+0x1b) //; Counter A3
load reg.
#define TMRA3_HOLD (volatile short *)(TMRA_BASE+0x1c) //; Counter A3
hold reg.
#define TMRA3_CNTR (volatile short *)(TMRA_BASE+0x1d) //; Counter A3
value reg.
#define TMRA3_CTRL (short *)(TMRA_BASE+0x1e) //; Counter A3
control reg.
#define TMRA3_SCR (volatile short *)(TMRA_BASE+0x1f) //; Counter A3
status control

/* TMRB0 Quad counter-timer B timer 0 */
#define TMRB0_CMP1 (short *)(TMRB_BASE+0x00) //; Counter B0
compare 1 reg.
#define TMRB0_CMP2 (short *)(TMRB_BASE+0x01) //; Counter B0
compare 2 reg.
#define TMRB0_CAP (volatile short *)(TMRB_BASE+0x02) //; Counter B0
capture reg.
#define TMRB0_LOAD (short *)(TMRB_BASE+0x03) //; Counter B0
load reg.
#define TMRB0_HOLD (volatile short *)(TMRB_BASE+0x04) //; Counter B0
hold reg.
#define TMRB0_CNTR (volatile short *)(TMRB_BASE+0x05) //; Counter B0
value reg.
#define TMRB0_CTRL (short *)(TMRB_BASE+0x06) //; Counter B0
control reg.
#define TMRB0_SCR (volatile short *)(TMRB_BASE+0x07) //; Counter B0
status control reg.

/* TMRB1 Quad counter-timer B timer 1 */
#define TMRB1_CMP1 (short *)(TMRB_BASE+0x08) //; Counter B1
compare 1 reg.
#define TMRB1_CMP2 (short *)(TMRB_BASE+0x09) //; Counter B1
compare 2 reg.
#define TMRB1_CAP (volatile short *)(TMRB_BASE+0x0a) //; Counter B1
capture reg.
#define TMRB1_LOAD (short *)(TMRB_BASE+0x0b) //; Counter B1
load reg.
#define TMRB1_HOLD (volatile short *)(TMRB_BASE+0x0c) //; Counter B1
hold reg.
#define TMRB1_CNTR (volatile short *)(TMRB_BASE+0x0d) //; Counter B1
value reg.
#define TMRB1_CTRL (short *)(TMRB_BASE+0x0e) //; Counter B1
control reg.
#define TMRB1_SCR (volatile short *)(TMRB_BASE+0x0f) //; Counter B1
status control

/* TMRB2 Quad counter-timer B timer 2 */
#define TMRB2_CMP1 (short *)(TMRB_BASE+0x10) //; Counter B2
compare 1 reg.
#define TMRB2_CMP2 (short *)(TMRB_BASE+0x11) //; Counter B2
compare 2 reg.
#define TMRB2_CAP (volatile short *)(TMRB_BASE+0x12) //; Counter B2
capture reg.
#define TMRB2_LOAD (short *)(TMRB_BASE+0x13) //; Counter B2
load reg.
#define TMRB2_HOLD (volatile short *)(TMRB_BASE+0x14) //; Counter B2
hold reg.
#define TMRB2_CNTR (volatile short *)(TMRB_BASE+0x15) //; Counter B2
value reg.
#define TMRB2_CTRL (short *)(TMRB_BASE+0x16) //; Counter B2
control reg.
#define TMRB2_SCR (volatile short *)(TMRB_BASE+0x17) //; Counter B2
status control

/* TMRB3 Quad counter-timer B timer 3 */
```

```
#define TMRB3_CMP1 (short *)(TMRB_BASE+0x18) //; Counter B3
compare 1 reg.
#define TMRB3_CMP2 (short *)(TMRB_BASE+0x19) //; Counter B3
compare 2 reg.
#define TMRB3_CAP (volatile short *)(TMRB_BASE+0x1a) //; Counter B3
capture reg.
#define TMRB3_LOAD (short *)(TMRB_BASE+0x1b) //; Counter B3
load reg.
#define TMRB3_HOLD (volatile short *)(TMRB_BASE+0x1c) //; Counter B3
hold reg.
#define TMRB3_CNTR (volatile short *)(TMRB_BASE+0x1d) //; Counter B3
value reg.
#define TMRB3_CTRL (short *)(TMRB_BASE+0x1e) //; Counter B3
control reg.
#define TMRB3_SCR (volatile short *)(TMRB_BASE+0x1f) //; Counter B3
status control

/* TMRC0 Quad counter-timer C timer 0 */
#define TMRC0_CMP1 (short *)(TMRC_BASE+0x00) //; Counter C0
compare 1 reg.
#define TMRC0_CMP2 (short *)(TMRC_BASE+0x01) //; Counter C0
compare 2 reg.
#define TMRC0_CAP (volatile short *)(TMRC_BASE+0x02) //; Counter C0
capture reg.
#define TMRC0_LOAD (short *)(TMRC_BASE+0x03) //; Counter C0
load reg.
#define TMRC0_HOLD (volatile short *)(TMRC_BASE+0x04) //; Counter C0
hold reg.
#define TMRC0_CNTR (volatile short *)(TMRC_BASE+0x05) //; Counter C0
value reg.
#define TMRC0_CTRL (short *)(TMRC_BASE+0x06) //; Counter C0
control reg.
#define TMRC0_SCR (volatile short *)(TMRC_BASE+0x07) //; Counter C0
status control reg.

/* TMRC1 Quad counter-timer C timer 1 */
#define TMRC1_CMP1 (short *)(TMRC_BASE+0x08) //; Counter C1
compare 1 reg.
#define TMRC1_CMP2 (short *)(TMRC_BASE+0x09) //; Counter C1
compare 2 reg.
#define TMRC1_CAP (volatile short *)(TMRC_BASE+0x0a) //; Counter C1
capture reg.
#define TMRC1_LOAD (short *)(TMRC_BASE+0x0b) //; Counter C1
load reg.
#define TMRC1_HOLD (volatile short *)(TMRC_BASE+0x0c) //; Counter C1
hold reg.
#define TMRC1_CNTR (volatile short *)(TMRC_BASE+0x0d) //; Counter C1
value reg.
#define TMRC1_CTRL (short *)(TMRC_BASE+0x0e) //; Counter C1
control reg.
#define TMRC1_SCR (volatile short *)(TMRC_BASE+0x0f) //; Counter C1
status control

/* TMRC2 Quad counter-timer C timer 2 */
#define TMRC2_CMP1 (short *)(TMRC_BASE+0x10) //; Counter C2
compare 1 reg.
#define TMRC2_CMP2 (short *)(TMRC_BASE+0x11) //; Counter C2
compare 2 reg.
#define TMRC2_CAP (volatile short *)(TMRC_BASE+0x12) //; Counter C2
capture reg.
#define TMRC2_LOAD (short *)(TMRC_BASE+0x13) //; Counter C2
load reg.
#define TMRC2_HOLD (volatile short *)(TMRC_BASE+0x14) //; Counter C2
hold reg.
#define TMRC2_CNTR (volatile short *)(TMRC_BASE+0x15) //; Counter C2
value reg.
#define TMRC2_CTRL (short *)(TMRC_BASE+0x16) //; Counter C2
control reg.
#define TMRC2_SCR (volatile short *)(TMRC_BASE+0x17) //; Counter C2
status control
```

```
/* TMRC3 Quad counter-timer C timer 3 */
#define TMRC3_CMP1 (short *)(TMRC_BASE+0x18) //; Counter C3
compare 1 reg.
#define TMRC3_CMP2 (short *)(TMRC_BASE+0x19) //; Counter C3
compare 2 reg.
#define TMRC3_CAP (volatile short *)(TMRC_BASE+0x1a) //; Counter C3
capture reg.
#define TMRC3_LOAD (short *)(TMRC_BASE+0x1b) //; Counter C3
load reg.
#define TMRC3_HOLD (volatile short *)(TMRC_BASE+0x1c) //; Counter C3
hold reg.
#define TMRC3_CNTR (volatile short *)(TMRC_BASE+0x1d) //; Counter C3
value reg.
#define TMRC3_CTRL (short *)(TMRC_BASE+0x1e) //; Counter C3
control reg.
#define TMRC3_SCR (volatile short *)(TMRC_BASE+0x1f) //; Counter C3
status control

/* TMRD0 Quad counter-timer D timer 0 */
#define TMRD0_CMP1 (short *)(TMRD_BASE+0x00) //; Counter D0
compare 1 reg.
#define TMRD0_CMP2 (short *)(TMRD_BASE+0x01) //; Counter D0
compare 2 reg.
#define TMRD0_CAP (volatile short *)(TMRD_BASE+0x02) //; Counter D0
capture reg.
#define TMRD0_LOAD (short *)(TMRD_BASE+0x03) //; Counter D0
load reg.
#define TMRD0_HOLD (volatile short *)(TMRD_BASE+0x04) //; Counter D0
hold reg.
#define TMRD0_CNTR (volatile short *)(TMRD_BASE+0x05) //; Counter D0
value reg.
#define TMRD0_CTRL (short *)(TMRD_BASE+0x06) //; Counter D0
control reg.
#define TMRD0_SCR (volatile short *)(TMRD_BASE+0x07) //; Counter D0
status control reg.

/* TMRD1 Quad counter-timer D timer 1 */
#define TMRD1_CMP1 (short *)(TMRD_BASE+0x08) //; Counter D1
compare 1 reg.
#define TMRD1_CMP2 (short *)(TMRD_BASE+0x09) //; Counter D1
compare 2 reg.
#define TMRD1_CAP (volatile short *)(TMRD_BASE+0x0a) //; Counter D1
capture reg.
#define TMRD1_LOAD (short *)(TMRD_BASE+0x0b) //; Counter D1
load reg.
#define TMRD1_HOLD (volatile short *)(TMRD_BASE+0x0c) //; Counter D1
hold reg.
#define TMRD1_CNTR (volatile short *)(TMRD_BASE+0x0d) //; Counter D1
value reg.
#define TMRD1_CTRL (short *)(TMRD_BASE+0x0e) //; Counter D1
control reg.
#define TMRD1_SCR (volatile short *)(TMRD_BASE+0x0f) //; Counter D1
status control

/* TMRD2 Quad counter-timer D timer 2 */
#define TMRD2_CMP1 (short *)(TMRD_BASE+0x10) //; Counter D2
compare 1 reg.
#define TMRD2_CMP2 (short *)(TMRD_BASE+0x11) //; Counter D2
compare 2 reg.
#define TMRD2_CAP (volatile short *)(TMRD_BASE+0x12) //; Counter D2
capture reg.
#define TMRD2_LOAD (short *)(TMRD_BASE+0x13) //; Counter D2
load reg.
#define TMRD2_HOLD (volatile short *)(TMRD_BASE+0x14) //; Counter D2
hold reg.
#define TMRD2_CNTR (volatile short *)(TMRD_BASE+0x15) //; Counter D2
value reg.
#define TMRD2_CTRL (short *)(TMRD_BASE+0x16) //; Counter D2
control reg.
```

```
#define TMRD2_SCR (volatile short *)(TMRD_BASE+0x17) //; Counter D2
status control

/* TMRD3 Quad counter-timer D timer 3 */
#define TMRD3_CMP1 (short *)(TMRD_BASE+0x18) //; Counter D3
compare 1 reg.
#define TMRD3_CMP2 (short *)(TMRD_BASE+0x19) //; Counter D3
compare 2 reg.
#define TMRD3_CAP (volatile short *)(TMRD_BASE+0x1a) //; Counter D3
capture reg.
#define TMRD3_LOAD (short *)(TMRD_BASE+0x1b) //; Counter D3
load reg.
#define TMRD3_HOLD (volatile short *)(TMRD_BASE+0x1c) //; Counter D3
hold reg.
#define TMRD3_CNTR (volatile short *)(TMRD_BASE+0x1d) //; Counter D3
value reg.
#define TMRD3_CTRL (short *)(TMRD_BASE+0x1e) //; Counter D3
control reg.
#define TMRD3_SCR (volatile short *)(TMRD_BASE+0x1f) //; Counter D3
status control


/* Serial Communications Interface 0 */

//Baud rate register
#define SCI0_SCIBR ( short *)(SCI0_BASE+0x00)
//Control register
#define SCI0_SCICR ( short *)(SCI0_BASE+0x01)
//Status register
#define SCI0_SCISR (volatile short *)(SCI0_BASE+0x02)
//Data register
#define SCI0_SCIDR (volatile short *)(SCI0_BASE+0x03)

/* Serial Communications Interface 1 */

//Baud rate register
#define SCI1_SCIBR ( short *)(SCI1_BASE+0x00)
//Control register
#define SCI1_SCICR ( short *)(SCI1_BASE+0x01)
//Status register
#define SCI1_SCISR (volatile short *)(SCI1_BASE+0x02)
//Data register
#define SCI1_SCIDR (volatile short *)(SCI1_BASE+0x03)

/* GPIOA General Purpose Input/Output Port A */
#define GPIOA_PUR (short *)(GPIOA_BASE+0x00) //; Port A
pull-up reg.
#define GPIOA_DR (volatile short *)(GPIOA_BASE+0x01) //; Port A data
reg.
#define GPIOA_DDR (short *)(GPIOA_BASE+0x02) //; Port A data
direction reg.
#define GPIOA_PER (short *)(GPIOA_BASE+0x03) //; Port A
peripheral reg.
#define GPIOA_IAR (short *)(GPIOA_BASE+0x04) //; Port A
interrupt assert reg.
#define GPIOA_IENR (short *)(GPIOA_BASE+0x05) //; Port A
interrupt enable reg.
#define GPIOA_IPOLAR (short *)(GPIOA_BASE+0x06) //; Port A
interrupt polarity reg.
#define GPIOA_IPR (volatile short *)(GPIOA_BASE+0x07) //; Port A
interrupt pending reg.
#define GPIOA_IESR (short *)(GPIOA_BASE+0x08) //; Port A
interrupt edge sensitive reg.

/* GPIOB General Purpose Input/Output Port B */
#define GPIOB_PUR (short *)(GPIOB_BASE+0x00) //; Port B
pull-up reg.
#define GPIOB_DR (volatile short *)(GPIOB_BASE+0x01) //; Port B data
reg.
#define GPIOB_DDR (short *)(GPIOB_BASE+0x02) //; Port B data
```

```
direction reg.
#define GPIOB_PER (short *)(GPIOB_BASE+0x03) //; Port B
peripheral reg.
#define GPIOB_IAR (short *)(GPIOB_BASE+0x04) //; Port B
interrupt assert reg.
#define GPIOB_IENR (short *)(GPIOB_BASE+0x05) //; Port B
interrupt enable reg.
#define GPIOB_IPOLAR (short *)(GPIOB_BASE+0x06) //; Port B
interrupt polarity reg.
#define GPIOB_IPR (volatile short *)(GPIOB_BASE+0x07) //; Port B
interrupt pending reg.
#define GPIOB_IESR (short *)(GPIOB_BASE+0x08) //; Port B
interrupt edge sensitive reg.

/* GPIOD General Purpose Input/Output Port D */
#define GPIOD_PUR (short *)(GPIOD_BASE+0x00) //; Port D
pull-up reg.
#define GPIOD_DR (volatile short *)(GPIOD_BASE+0x01) //; Port D data
reg.
#define GPIOD_DDR (short *)(GPIOD_BASE+0x02) //; Port D data
direction reg.
#define GPIOD_PER (short *)(GPIOD_BASE+0x03) //; Port D
peripheral reg.
#define GPIOD_IAR (short *)(GPIOD_BASE+0x04) //; Port D
interrupt assert reg.
#define GPIOD_IENR (short *)(GPIOD_BASE+0x05) //; Port D
interrupt enable reg.
#define GPIOD_IPOLAR (short *)(GPIOD_BASE+0x06) //; Port D
interrupt polarity reg.
#define GPIOD_IPR (volatile short *)(GPIOD_BASE+0x07) //; Port D
interrupt pending reg.
#define GPIOD_IESR (short *)(GPIOD_BASE+0x08) //; Port D
interrupt edge sensitive reg.

/* On-board LED setup - the Isopod V2 uses PORTD bits 0 for red,
1 for yellow, and 2 for green.*/
#define REDLED_ON *GPIOD_DR = *GPIOD_DR|0x0001
#define REDLED_OFF *GPIOD_DR = *GPIOD_DR&0x00FE
#define YELLED_ON *GPIOD_DR = *GPIOD_DR|0x0002
#define YELLED_OFF *GPIOD_DR = *GPIOD_DR&0x00FD
#define GRNLED_ON *GPIOD_DR = *GPIOD_DR|0x0004
#define GRNLED_OFF *GPIOD_DR = *GPIOD_DR&0x00FB


/* ------ Function declarations ------ */
void init_tmra0(void);
void init_tmra1(void);
void init_tmrb0(void);
void init_tmrb1(void);
void init_tmrc0(void);
void init_gpiod(void);
void timedcode(void);
//void send(char t);
//void sendArr(char ts[]);
void init_SCI0(void);
void init_SCI1(void);
void init_ADCA(void);
void init_PWMA(void);
void PWMA_update(void);
void PWMB_update(void);
short int delta_TMRB0(void);
void rec_lint(void);
void send_lint(void);

/* ------ Global variable definitions ------ */

/* Global variable def. for ROM-RAM copy. Note location - before
function definitions.
A default value from the definition is stored in flash memory, but
copied to RAM at
```

```
start. This allows the value to be easily modified during operation.
*/

long int Rec_Data[16];

/* ---------- Function definitions ---------- */

/* Subtract previous TMRB0 reading from current reading */
short int delta_TMRB0(void)
{
static short int prevcount = 0;
short int temp;

temp = prevcount;
prevcount = *TMRB0_CNTR;

return(*TMRB0_CNTR - temp);
}

/* Parse one incoming byte, if available, on serial port 1, and build up
a long (32 bit)
word at 7 bits per incoming byte, most significant to least. Discard the
three most
significant bits. Valid data bytes are in the range of 0 to 127 decimal
(0b00000000 to 0b01111111); 128 to 143 (0b10000000 to 0b10001111) are
considered address
values. Upon receiving a valid address byte, the routine begins looking
for valid
data bytes. If 5 successive valid data bytes are received, Rec_Data at
that address
is updated with the new value. */
void rec_lint(void)
{
static char rbytenum = 0x0, rlintlabel = 0x0;
static long int rcurrentdata = 0x0;
unsigned long int new_data;
char temp1, temp2;

if (*SCI1_SCISR&0x2000) //Receive buffer full - process new
byte
{
new_data = *SCI1_SCISR&0x2000; //Read status register
buffer full flag
new_data = *SCI1_SCIDR; //Read data register,
resetting flag

if ((new_data>=0x80)&&(new_data<=0x8f)&&(rbytenum==0x0))
//New byte is a label
{
rlintlabel = new_data - 0x80;
rcurrentdata = 0x0;
rbytenum++; //Start looking for data
bytes
}
else if
((new_data>=0x0)&&(new_data<=0x7f)&&(rbytenum>=0x1)&&(rbytenum<=0x5))
//new byte is valid data
{
if (rbytenum == 0x1) //trim off all but
lowest 4 bits
{
new_data&=0xf;
}
new_data<<=((0x5 - rbytenum)*0x7);
rcurrentdata|=new_data;
rbytenum++;
if (rbytenum == 0x6) //parsing of new long
word is complete
{
Rec_Data[rlintlabel] = rcurrentdata;
```

```
//Store new word in Rec_Data
rcurrentdata = 0x0;
//and start looking for new label byte
rbytenum = 0x0;
}
}
else //Data is not valid - start
over, looking for new label byte
{
rcurrentdata = 0x0;
rbytenum = 0x0;
}
}
}


/* Convert 32-bit long words in array Rec_Data into strings of 6 bytes,
and send them out serial port 1. The first byte tranmitted is an
identifier bit, with
valid values between 128 and 143 (for 16 element Rec_Data). Next is a
byte containing the
most significant 4 bits of the long word in its lower four bits.
Finally, the least
significant 28 bits of the long word are sent out 7 bits at a time, in
four transmit
bytes. Valid values for the 5 data-containing bytes are 0 to 127.

Send each value in the array Rec_Data and repeat; one byte is
transmitted each time this routine
is called.*/
void send_lint(void)
{
static char sbytenum = 0x0, slintlabel = 0x0;
static long int scurrentdata = 0x0;
long int ltemp;
short temp;

if (*SCI1_SCISR&0x8000) //transmit buffer empty, need to send
data
{
if (sbytenum == 0x0) //need to send address byte
{
temp = *SCI1_SCISR; //read status register
*SCI1_SCIDR = slintlabel + 0x80; //send
address byte
scurrentdata = Rec_Data[slintlabel]; //load
long int for sending
sbytenum++; //ready to send data
bytes
}

else if ((sbytenum >= 0x1)&&(sbytenum <= 0x5)) //need
to send data byte
{
ltemp = (unsigned long int)scurrentdata>>((0x5 -
sbytenum)*0x7);
temp = *SCI1_SCISR; //read status
register
*SCI1_SCIDR = ltemp; //send data byte
if (sbytenum == 0x1)
{
scurrentdata &= 0x0fffffff;
}
else
{
scurrentdata &= ((unsigned long
int)0x0fffffff>>((sbytenum-0x1)*0x7));
}
sbytenum++;
if (sbytenum == 0x6) //word is sent, send
```

```
new word
{
sbytenum = 0x0;
slintlabel++;
if (slintlabel == 0x10) //Rec_Data fully
sent, start again
{
slintlabel = 0x0;
}
}
}
}
}

void init_SCI0(void)
{
//Baud rate register
*SCI0_SCIBR = 260; //9600 baud

//Control register
*SCI0_SCICR = 0x0008; //Transmit enable, 8 bit, NP, 1 stop

//Data register
*SCI0_SCIDR = 0x0000;
}

void init_SCI1(void)
{
//Baud rate register
*SCI1_SCIBR = 22; //115 Kbaud (use decimal 260 for 9600) at 40
MHz clock

//Control register
*SCI1_SCICR = 0x000C; //Transmit enable, receive enable, 8 bit,
NP, 1 stop

//Data register
*SCI1_SCIDR = 0x0000;
}


void init_PWMA()
{
short temp;
*PWMA_PWMCM = 0x03E8; //0x03E8 for 20KHz; 0x7FFF for 76 Hz
with 1/8 prescale
*PWMA_PMDEADTM = 0x0000; //No deadtime
*PWMA_PWMVAL0 = 0x000A; //Initial PWM duty cycle = 1% at 20 KHz
*PWMA_PWMVAL1 = 0x000A; //Initial PWM duty cycle = 1% at 20 KHz
*PWMA_PWMVAL2 = 0x000A; //Initial PWM duty cycle = 1% at 20 KHz
*PWMA_PWMVAL3 = 0x000A; //Initial PWM duty cycle = 1% at 20 KHz
*PWMA_PWMVAL4 = 0x000A; //Initial PWM duty cycle = 1% at 20 KHz
*PWMA_PWMVAL5 = 0x000A; //Initial PWM duty cycle = 1% at 20 KHz
*PWMA_PMDISMAP1 = 0b0000000000000000; //Disable all fault inputs
*PWMA_PMDISMAP2 = 0b0000000000000000; //Disable all fault inputs
*PWMA_PMCFG = 0b0000000000001110; //Center-aligned, positive
polarity, independent
*PWMA_PMCCR = 0b0000000000000000; //non-masked, independent
PWM values, no channel swaps
*PWMA_PMFCTL = 0b0000000000000000; //No interrupts, manual
fault clear
*PWMA_PMOUT = 0b0000000000000000; //Disable output pads for
now; PWM control of outputs
*PWMA_PMCTL = 0b0000100000000000; //Disable PWM outputs for
now, update on half cycles,
temp = *PWMA_PMCTL|0x0002; //Read LDOK
*PWMA_PMCTL = *PWMA_PMCTL|0x0002; //Set LDOK
*PWMA_PMCTL = *PWMA_PMCTL|0x0001; //Enable PWM outputs
*PWMA_PMOUT = *PWMA_PMOUT|0x8000; //Enable PWM output pads
}
```

```
void PWMA_update(void)
{
short temp;

temp = *PWMA_PMCTL|0x0002; //Read LDOK
*PWMA_PMCTL = *PWMA_PMCTL|0x0002; //Set LDOK
}

void PWMB_update(void)
{
short temp;

temp = *PWMB_PMCTL|0x0002; //Read LDOK
*PWMB_PMCTL = *PWMB_PMCTL|0x0002; //Set LDOK
}



// ------ Timer C0 interrupt handler ------
void timedcode(void)
{
#pragma interrupt saveall
static short pwmval = 20;
unsigned int i, j;
short temp;

Rec_Data[2] = *TMRC0_CNTR; //Start time of interrupt
GRNLED_OFF;
REDLED_ON;
Rec_Data[0] = *TMRB0_CNTR;
Rec_Data[1] = delta_TMRB0();
send_lint();
Rec_Data[3] = *TMRC0_CNTR; //Time for send_lint
rec_lint();
Rec_Data[4] = *TMRC0_CNTR; //Time for rec_lint
temp = *ADCA_ADRSLT4;
*ADCA_ADCR1 |= 0b0010000000000000; //Set ADCA start bit
pwmval++;
*PWMA_PWMVAL0 = pwmval;
PWMA_update();

if (pwmval==990)
{
pwmval=20;
}

*TMRC0_SCR &= 0x7fff; //Clear timer compare flag (?)

REDLED_OFF;
Rec_Data[4] = *TMRC0_CNTR; //Time for interrupt handler

}

// ------ Timer A0, set as quad decoder ------
void init_tmra0( void )
{
*TMRA0_CMP1 = 0xffff;
*TMRA0_CMP2 = 0x0000;
*TMRA0_CAP = 0x0000;
*TMRA0_LOAD = 0x0000;
*TMRA0_HOLD = 0x0000;
*TMRA0_CNTR = 0x0000;
*TMRA0_CTRL = 0b1000000010000010;
*TMRA0_SCR = 0x0000;
}

// ------ Timer A1, set as quad decoder ------
void init_tmra1( void )
```

```
{
*TMRA1_CMP1 = 0xffff;
*TMRA1_CMP2 = 0x0000;
*TMRA1_CAP = 0x0000;
*TMRA1_LOAD = 0x0000;
*TMRA1_HOLD = 0x0000;
*TMRA1_CNTR = 0x0000;
*TMRA1_CTRL = 0b1000010110000010;
*TMRA1_SCR = 0x0000;
}


// ------ Timer B0, set as quad decoder ------
void init_tmrb0( void )
{
*TMRB0_CMP1 = 0xffff;
*TMRB0_CMP2 = 0x0000;
*TMRB0_CAP = 0x0000;
*TMRB0_LOAD = 0x0000;
*TMRB0_HOLD = 0x0000;
*TMRB0_CNTR = 0x0000;
*TMRB0_CTRL = 0b1000000010000010;
*TMRB0_SCR = 0x0000;
}

// ------ Timer B1, set as quad decoder ------
void init_tmrb1( void )
{
*TMRB1_CMP1 = 0xffff;
*TMRB1_CMP2 = 0x0000;
*TMRB1_CAP = 0x0000;
*TMRB1_LOAD = 0x0000;
*TMRB1_HOLD = 0x0000;
*TMRB1_CNTR = 0x0000;
*TMRB1_CTRL = 0b1000010110000010;
*TMRB1_SCR = 0x0000;
}

// ------ Timer C0 with interrupt init routine and 1mS loop time------
void init_tmrc0( void )
{
*TMRC0_CMP1 = 0x9c40; //Count to 40000, interrupt and
reset counter
*TMRC0_CMP2 = 0x0000;
*TMRC0_CAP = 0x0000;
*TMRC0_LOAD = 0x0000;
*TMRC0_HOLD = 0x0000;
*TMRC0_CNTR = 0x0000;
*TMRC0_CTRL = 0b0011000000100000; //40 MHz clock, count to cmp1 and
reset
*TMRC0_SCR = 0x4000;
*ITCN_GRP8 = *ITCN_GRP8|0b0000000100000000;
*CORE_IPR = *CORE_IPR|0b1000000000000000;
}

// ------ PortD init routine --------
void init_gpiod( void )
{
*GPIOD_PUR = 0x0000;
*GPIOD_IAR = 0x0000;
*GPIOD_IENR = 0x0000;
*GPIOD_IPOLAR = 0x0000;
*GPIOD_IESR = 0x0000;
*GPIOD_DR = 0x0000;
*GPIOD_DDR = 0b0000000000000111;
*GPIOD_PER = 0b0000000011000000;
}

// ------ ADCA analog-to-digital converter init routine ------
void init_ADCA( void )
```

```
{
*ADCA_ADCR1 = 0b0000000000000001;
*ADCA_ADCR2 = 0b0000000000001001; //2 MHz ADC clock
*ADCA_ADZCC = 0b0000000000000000; //Zero crossing disabled
*ADCA_ADLST1 = 0b0011001000010000; //Default channel order
0,1,2,3
*ADCA_ADLST2 = 0b0111011001010100; //Default channel order
4,5,6,7
*ADCA_ADSDIS = 0b0000000000000000; //All channels enabled, test
mode off
*ADCA_ADLLMT0 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT1 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT2 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT3 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT4 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT5 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT6 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADLLMT7 = 0b0000000000000000; //Low limit checking
disabled
*ADCA_ADHLMT0 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT1 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT2 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT3 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT4 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT5 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT6 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADHLMT7 = 0b1111111111111111; //High limit checking
disabled
*ADCA_ADOFS0 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS1 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS2 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS3 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS4 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS5 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS6 = 0b0000000000000000; //Zero offset adjustment -
pos. output
*ADCA_ADOFS7 = 0b0000000000000000; //Zero offset adjustment -
pos. output
}


void main(void) //Run initialization routines, then enter untimed
endless loop.
{
short temp;
init_tmra0();
init_tmra1();
init_tmrb0();
init_tmrb1();
init_tmrc0();
```

```
init_gpiod();
init_SCI1();
init_ADCA();
init_PWMA();
*PWMA_PWMVAL0 = 0x1f4;
PWMA_update();

Rec_Data[0x0]=0x00000000;
Rec_Data[0x1]=0x00000000;
Rec_Data[0x2]=0x00000000;
Rec_Data[0x3]=0x00000000;
Rec_Data[0x4]=0x00000000;
Rec_Data[0x5]=0x00000000;
Rec_Data[0x6]=0x00000000;
Rec_Data[0x7]=0x00000000;
Rec_Data[0x8]=0x00000000;
Rec_Data[0x9]=0x00000000;
Rec_Data[0xa]=0x00000000;
Rec_Data[0xb]=0x00000000;
Rec_Data[0xc]=0x00000000;
Rec_Data[0xd]=0x00000000;
Rec_Data[0xe]=0x00000000;
Rec_Data[0xf]=0x00000000;

while (1) //Main endless loop
{
GRNLED_ON;

/* wait until next interrupt
asm(wait); */

}
}



/* ----- end of Isopod control framework ------ */
```

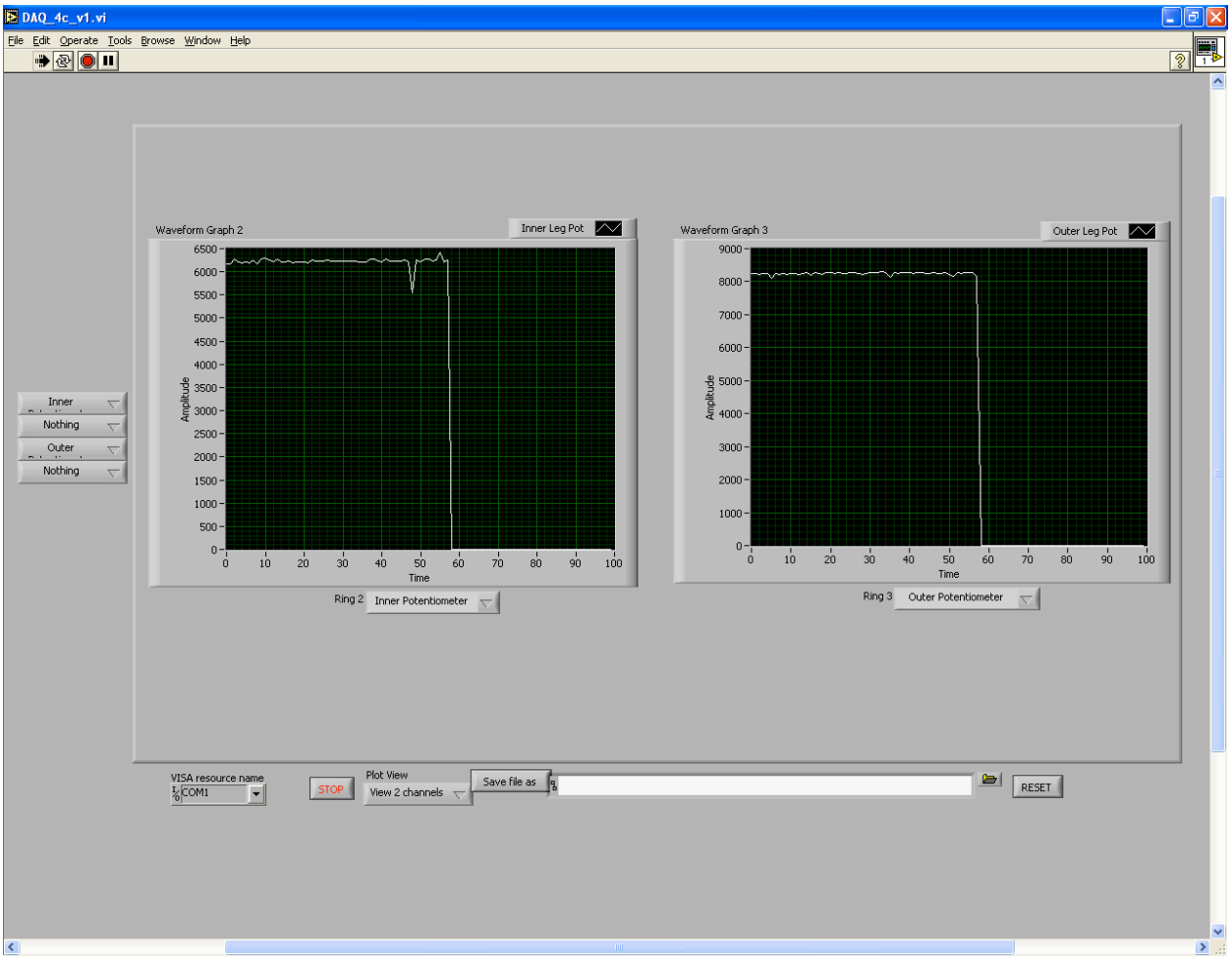


Figure 5. Front Panel showing 2 different plots with 2 parameters being saved.

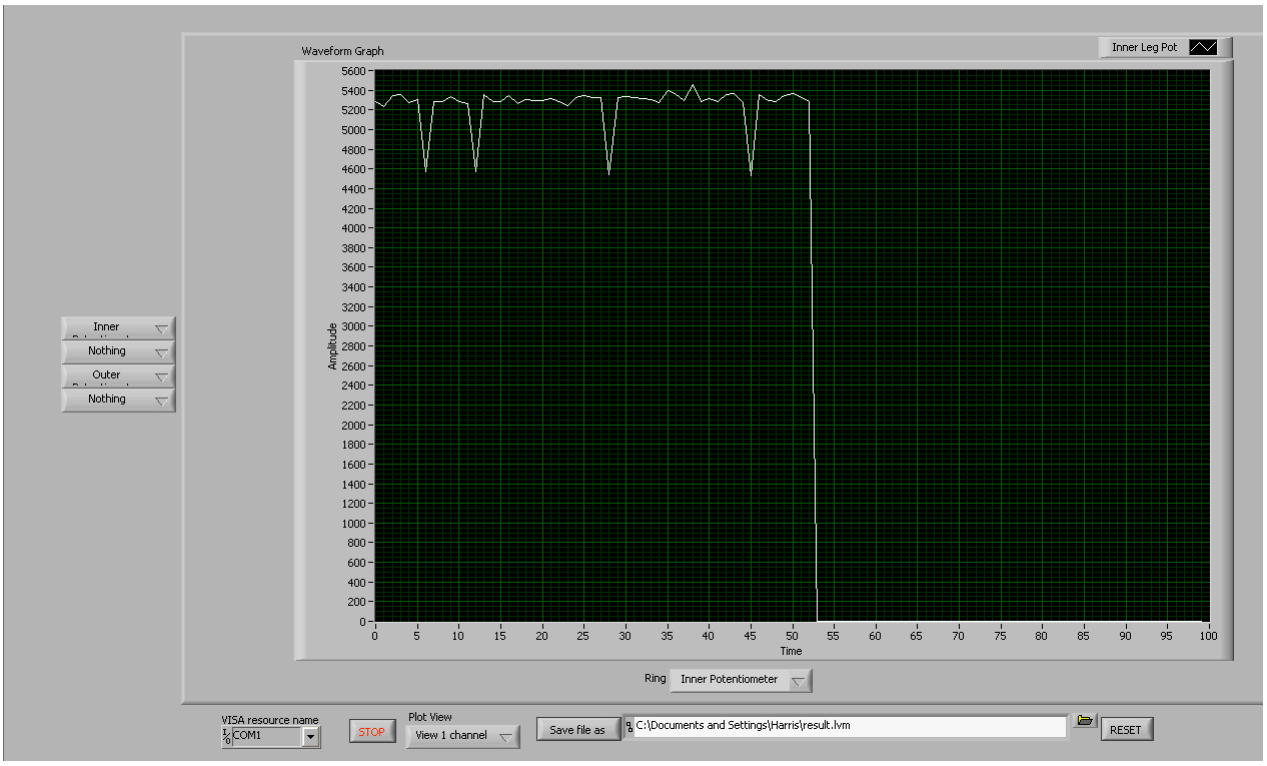


Figure 6. Front Panel showing 1 plots with 2 parameters being saved.

During one of the tests the data being plotted was also saved in result.lvm. The data is shown below.

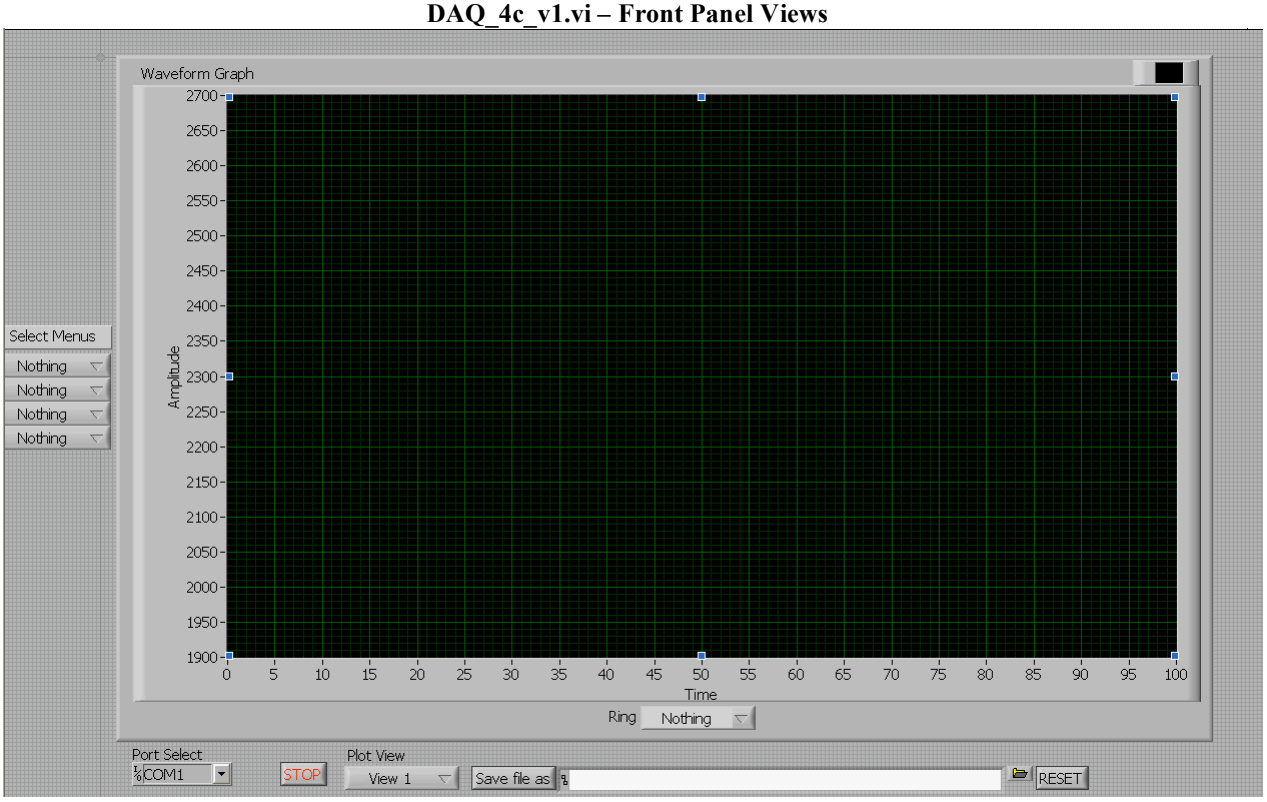
```
LabVIEW Measurement
Writer_Version    0.92
Reader_Version    1
Separator    Tab
Multi_Headings    Yes
X_Columns    No
Time_Pref    Relative
Operator    CAD
Date    2006/05/21
Time    16:03:55.604
***End_of_Header***

Channels    4
Samples    100    100    100    100
Date    2006/05/21    2006/05/21    2006/05/21    2006/05/21
Time    16:03:55.613999    16:03:55.613999    16:03:55.613999    16:03:55.613999
X_Dimension Time    Time    Time    Time
X0    0.0000000000000000E+0    0.0000000000000000E+0    0.0000000000000000E+0
      0.0000000000000000E+0
Delta_X    1.000000    1.000000    1.000000    1.000000
***End_of_Header***
X_Value                              Comment
      4808.000000 0.000000    7136.000000 0.000000
```

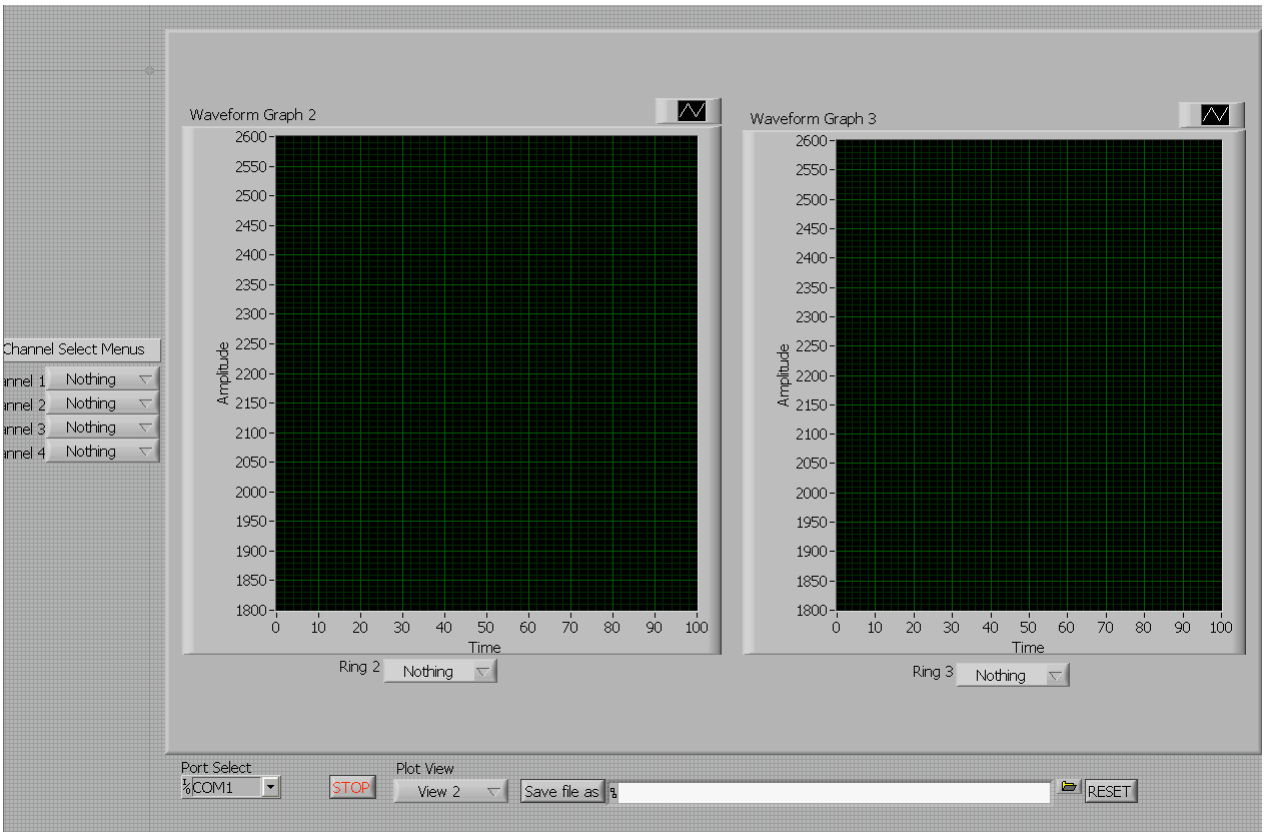
```
4832.000000 0.000000    7128.000000 0.000000
4816.000000 0.000000    7120.000000 0.000000
4800.000000 0.000000    7088.000000 0.000000
4816.000000 0.000000    7112.000000 0.000000
4792.000000 0.000000    7096.000000 0.000000
4816.000000 0.000000    7088.000000 0.000000
4776.000000 0.000000    7104.000000 0.000000
4784.000000 0.000000    7104.000000 0.000000
4768.000000 0.000000    7080.000000 0.000000
4816.000000 0.000000    7112.000000 0.000000
4752.000000 0.000000    7112.000000 0.000000
4792.000000 0.000000    7112.000000 0.000000
4784.000000 0.000000    7136.000000 0.000000
4760.000000 0.000000    7120.000000 0.000000
4776.000000 0.000000    7096.000000 0.000000
4752.000000 0.000000    7120.000000 0.000000
4776.000000 0.000000    7080.000000 0.000000
4776.000000 0.000000    7112.000000 0.000000
4720.000000 0.000000    6960.000000 0.000000
4792.000000 0.000000    7120.000000 0.000000
4784.000000 0.000000    7128.000000 0.000000
4752.000000 0.000000    6936.000000 0.000000
4848.000000 0.000000    7128.000000 0.000000
4808.000000 0.000000    7176.000000 0.000000
4720.000000 0.000000    7096.000000 0.000000
4800.000000 0.000000    7104.000000 0.000000
4776.000000 0.000000    7120.000000 0.000000
4728.000000 0.000000    7104.000000 0.000000
4760.000000 0.000000    7112.000000 0.000000
4720.000000 0.000000    7080.000000 0.000000
4752.000000 0.000000    6992.000000 0.000000
4792.000000 0.000000    7104.000000 0.000000
4752.000000 0.000000    7112.000000 0.000000
4760.000000 0.000000    6944.000000 0.000000
4800.000000 0.000000    7104.000000 0.000000
4760.000000 0.000000    7120.000000 0.000000
4760.000000 0.000000    7112.000000 0.000000
4752.000000 0.000000    7080.000000 0.000000
4728.000000 0.000000    7096.000000 0.000000
4736.000000 0.000000    6952.000000 0.000000
4848.000000 0.000000    7096.000000 0.000000
4752.000000 0.000000    7096.000000 0.000000
4800.000000 0.000000    7112.000000 0.000000
4752.000000 0.000000    7112.000000 0.000000
4744.000000 0.000000    6944.000000 0.000000
4920.000000 0.000000    7072.000000 0.000000
4728.000000 0.000000    7080.000000 0.000000
4800.000000 0.000000    7096.000000 0.000000
4760.000000 0.000000    7080.000000 0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
0.000000    0.000000    0.000000    0.000000
```

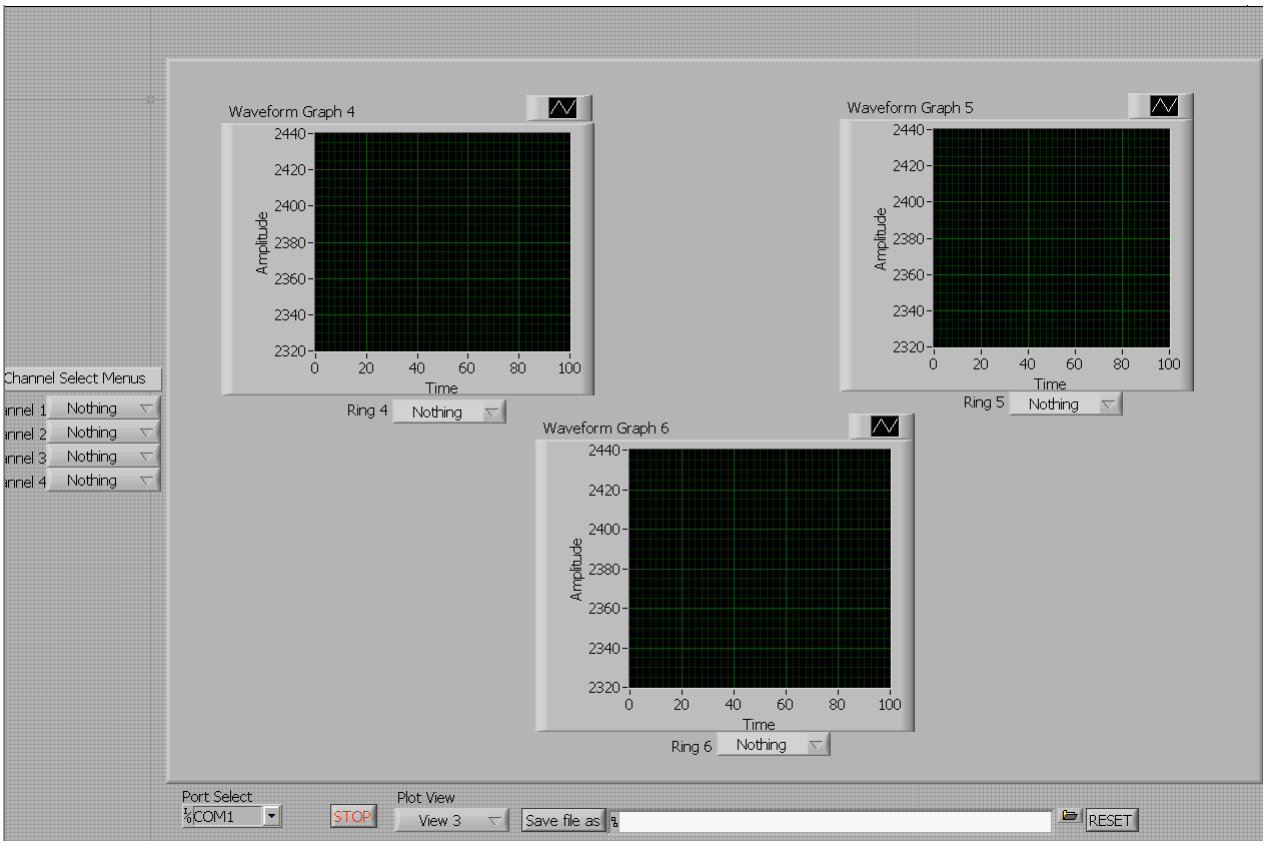
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000

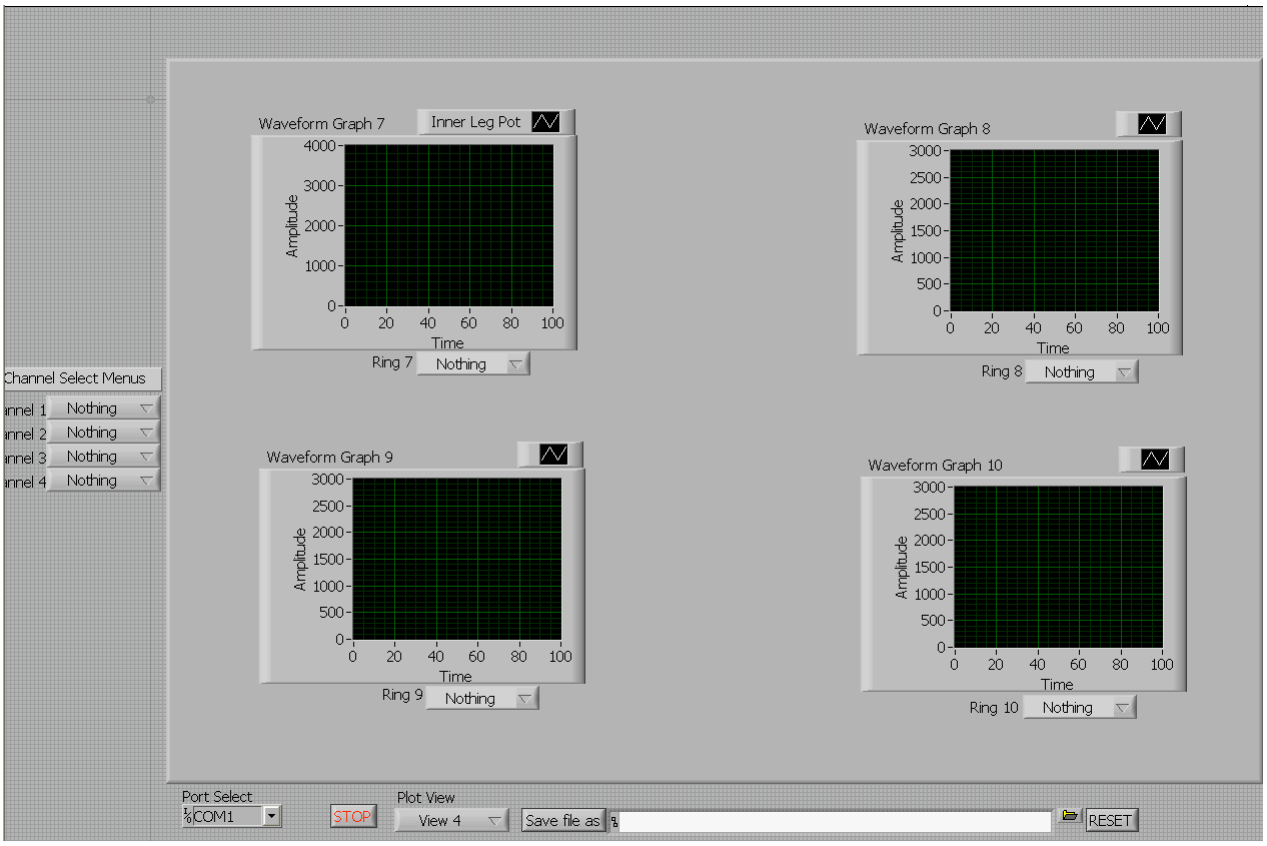
Appendix II – LabView Program

DAQ_4c_v1.vi – Front Panel Views

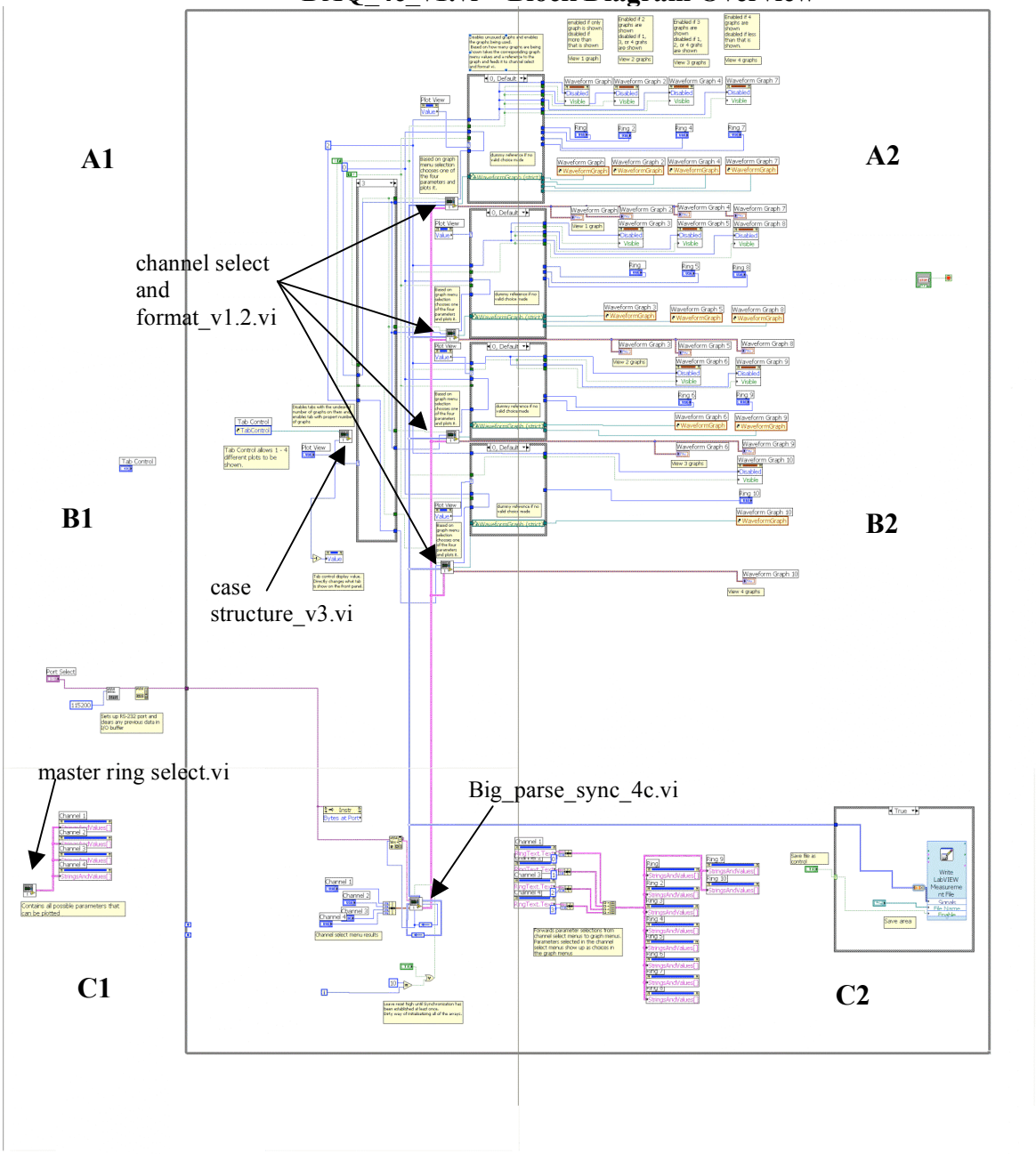


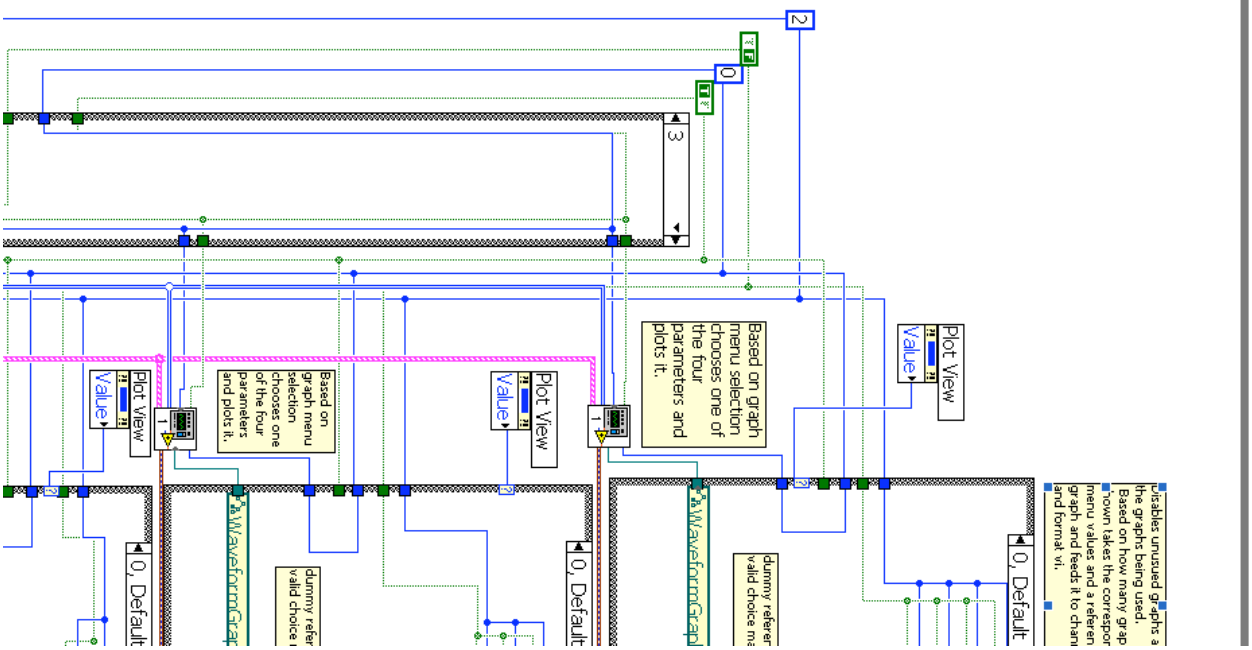



Waveform Graph 4
Amplitude
Time
Ring 4
Nothing
Waveform Graph 5
Amplitude
Time
Ring 5
Nothing
Waveform Graph 6
Amplitude
Time
Ring 6
Nothing
Channel Select Menus
Nothing
Nothing
Nothing
Nothing
Port Select
COM1
STOP
Plot View
View 3
Save file as
RESET

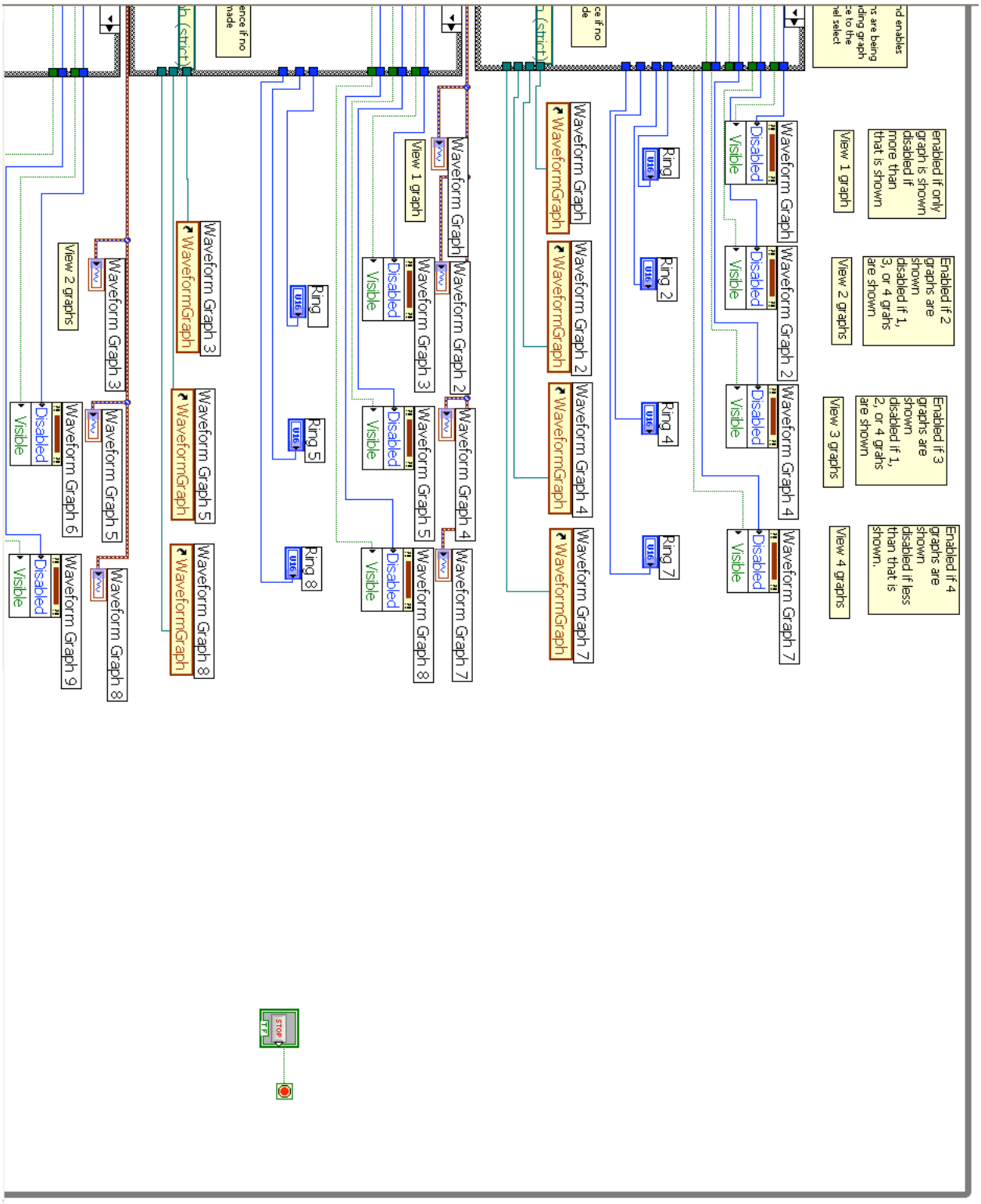

Waveform Graph 7
Inner Leg Pot
Amplitude
Time
Ring 7
Nothing
Waveform Graph 8
Amplitude
Time
Ring 8
Nothing
Waveform Graph 9
Amplitude
Time
Ring 9
Nothing
Waveform Graph 10
Amplitude
Time
Ring 10
Nothing
Channel Select Menus
Nothing
Nothing
Nothing
Nothing
Port Select
COM1
STOP
Plot View
View 4
Save file as
RESET

DAQ_4c_v1.vi – Block Diagram Overview

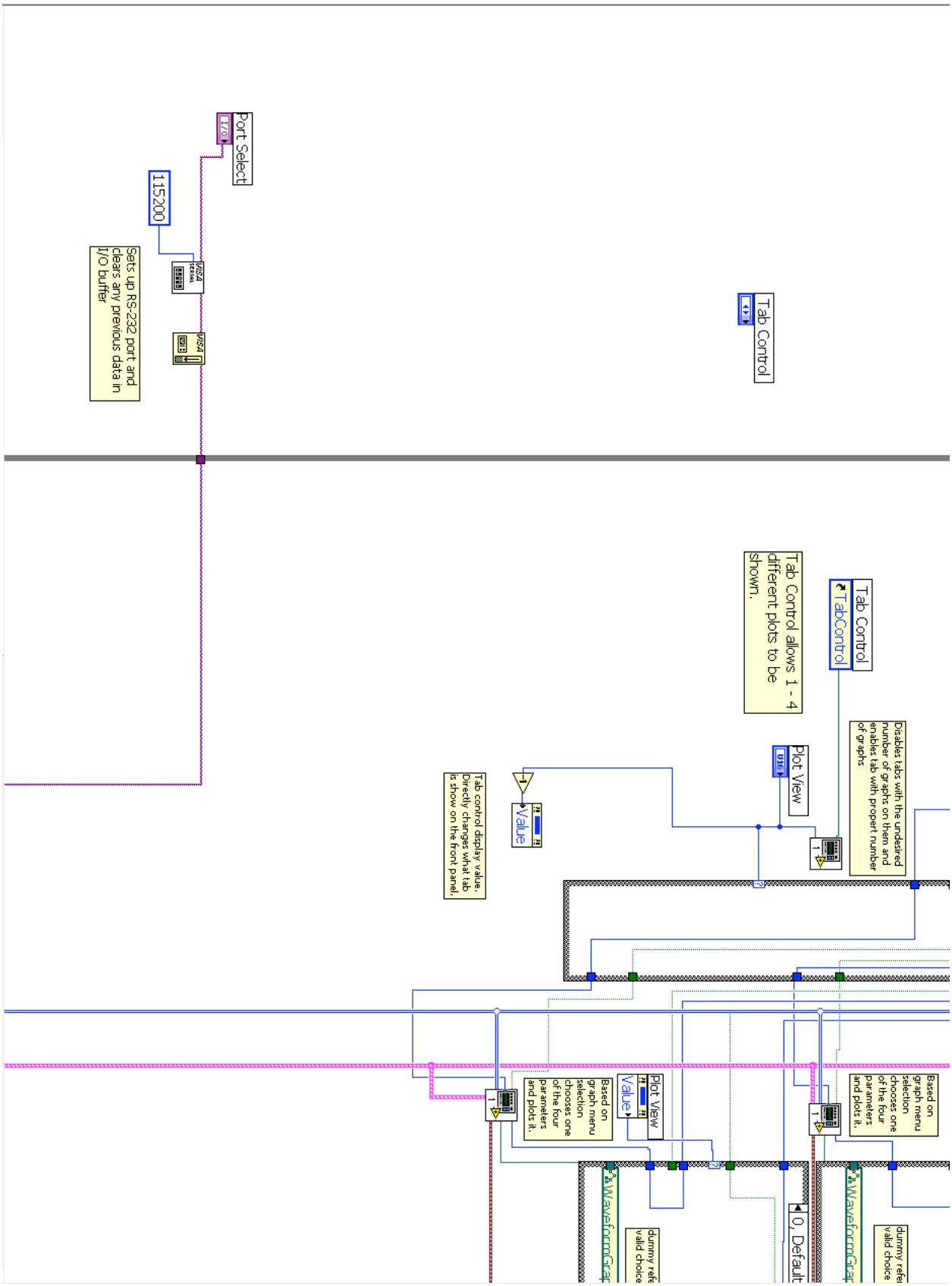




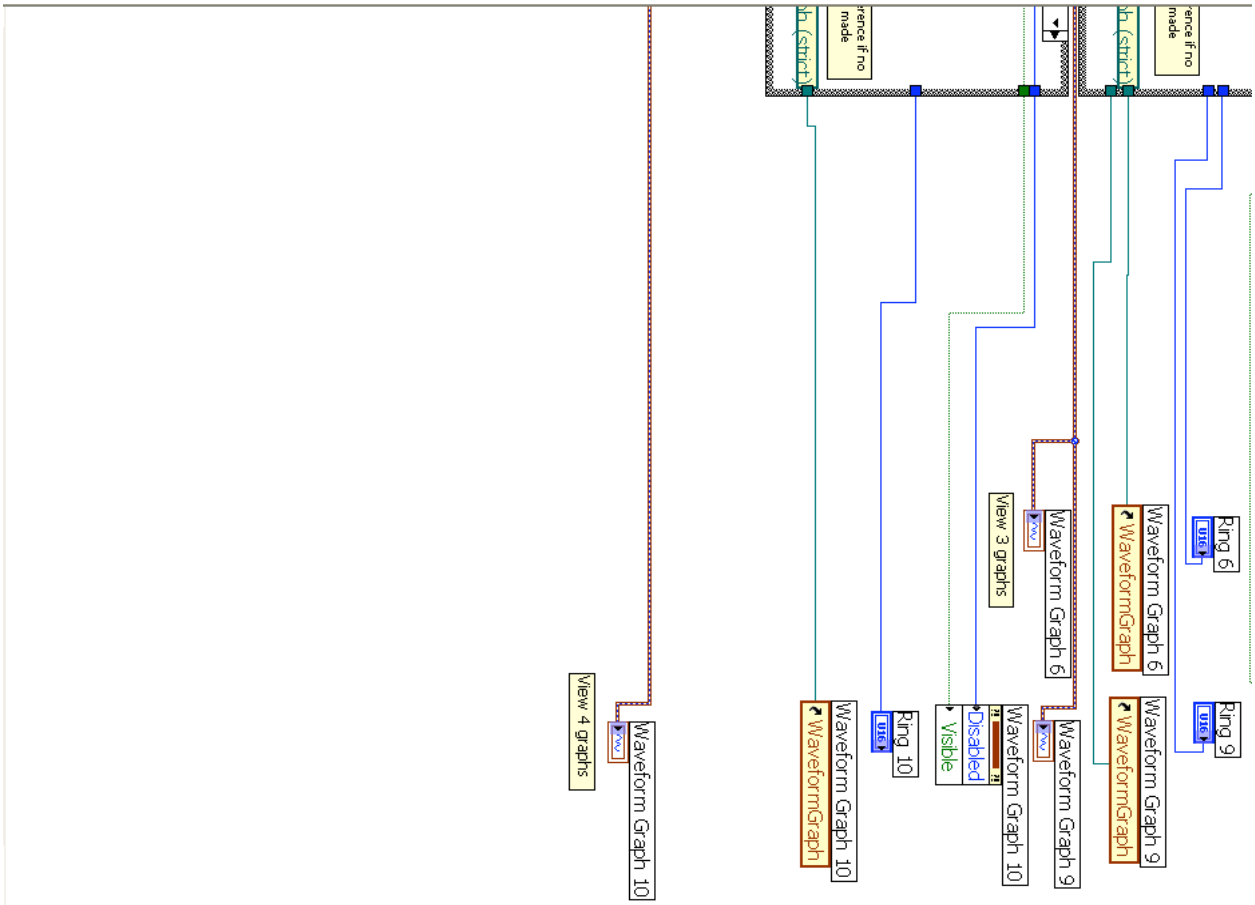
DAQ_4c_v1.vi Section A1



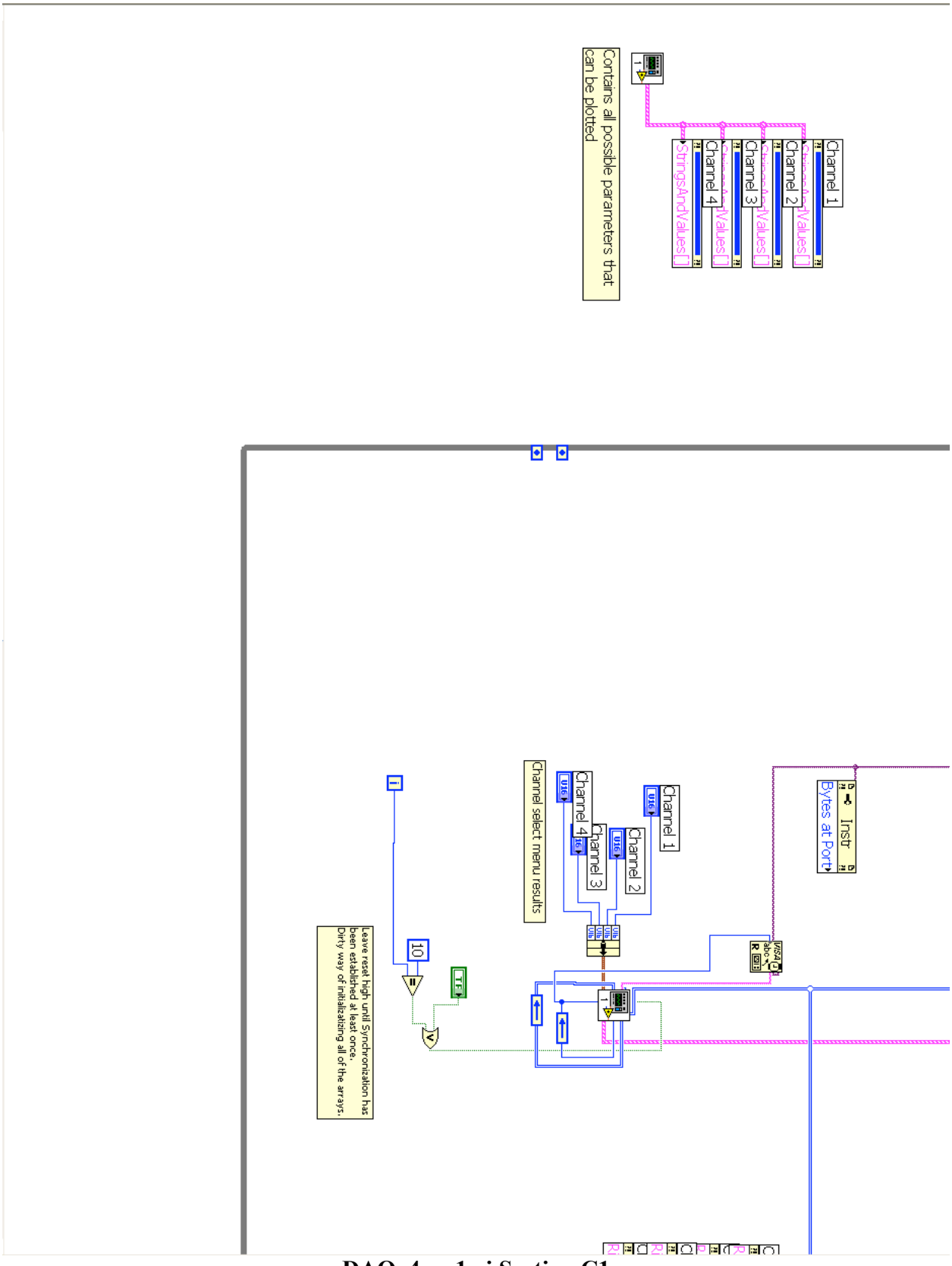
DAQ_4c_v1.vi Section A2



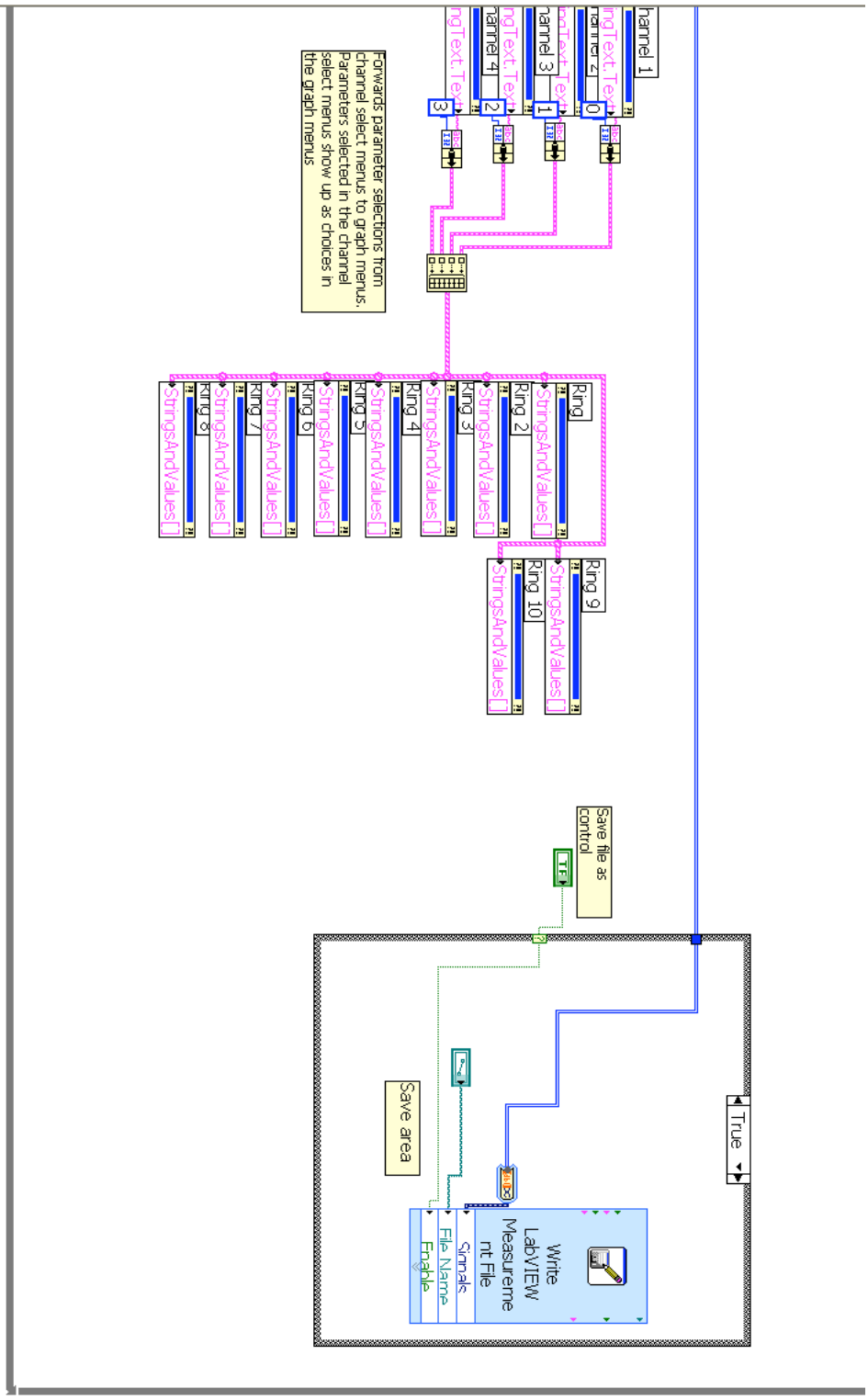
DAQ_4c_v1.vi Section B1



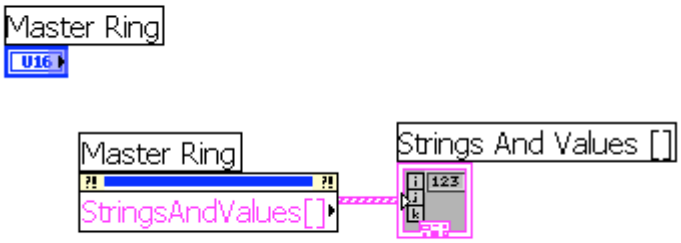
DAQ_4c_v1.vi Section B2



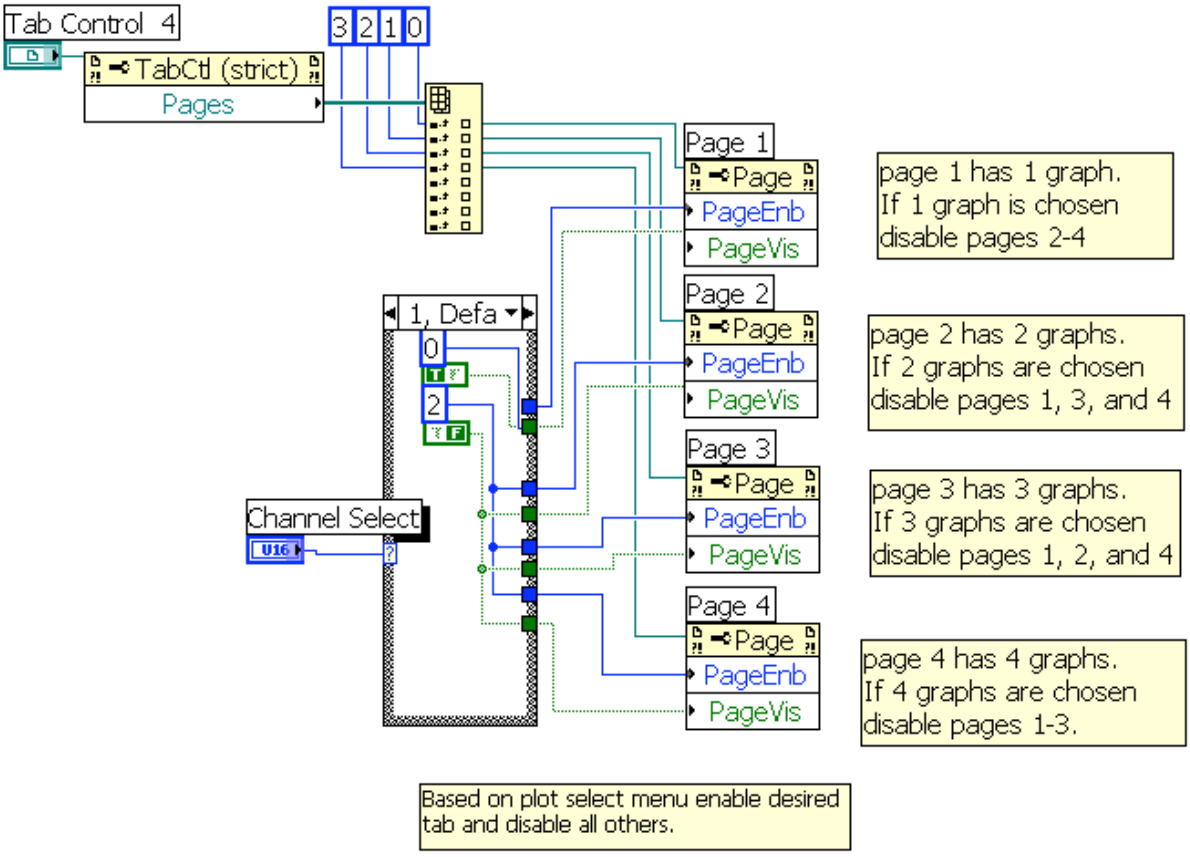
DAQ_4c_v1.vi Section C1



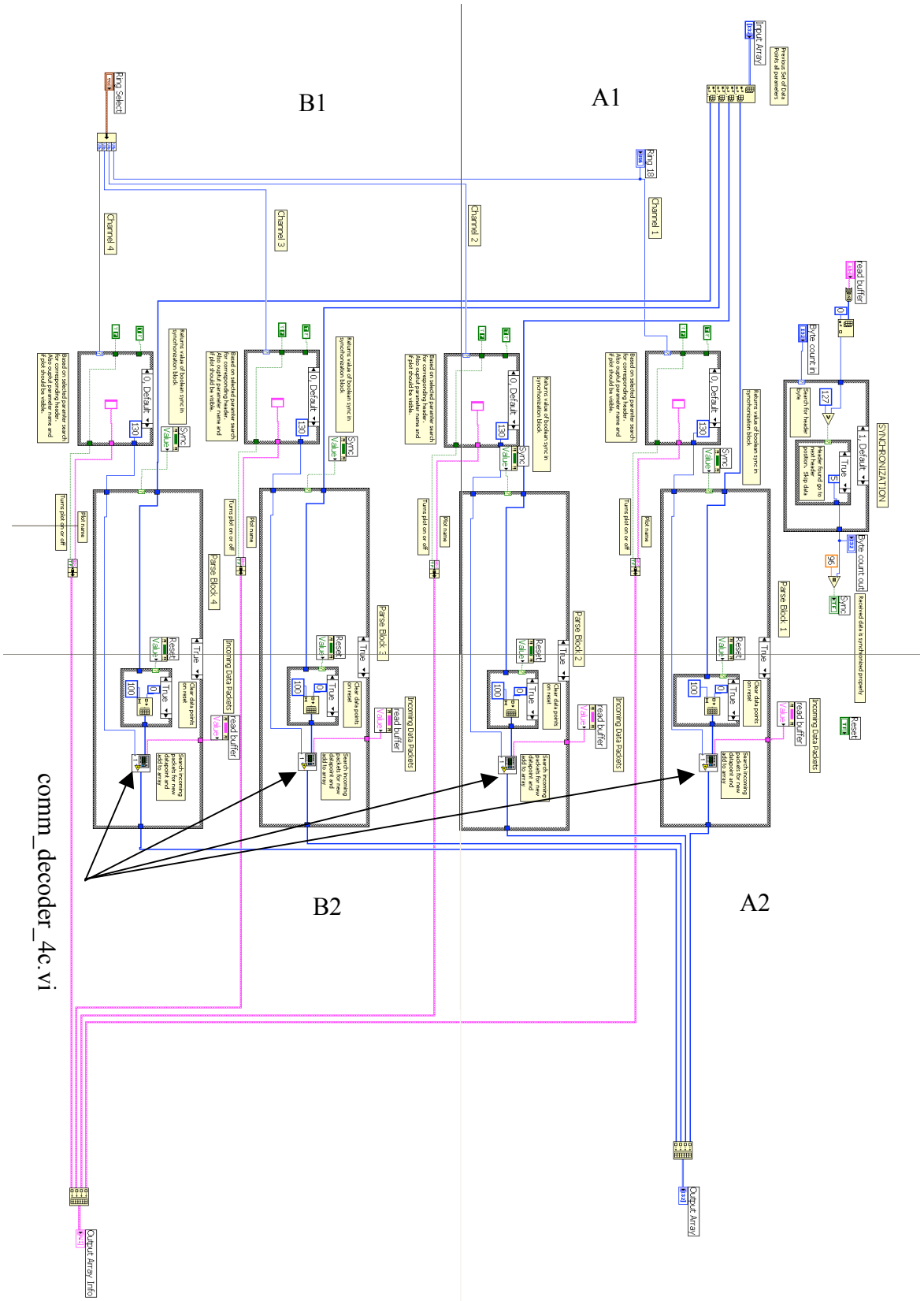
DAQ_4c_v1.vi Section C2



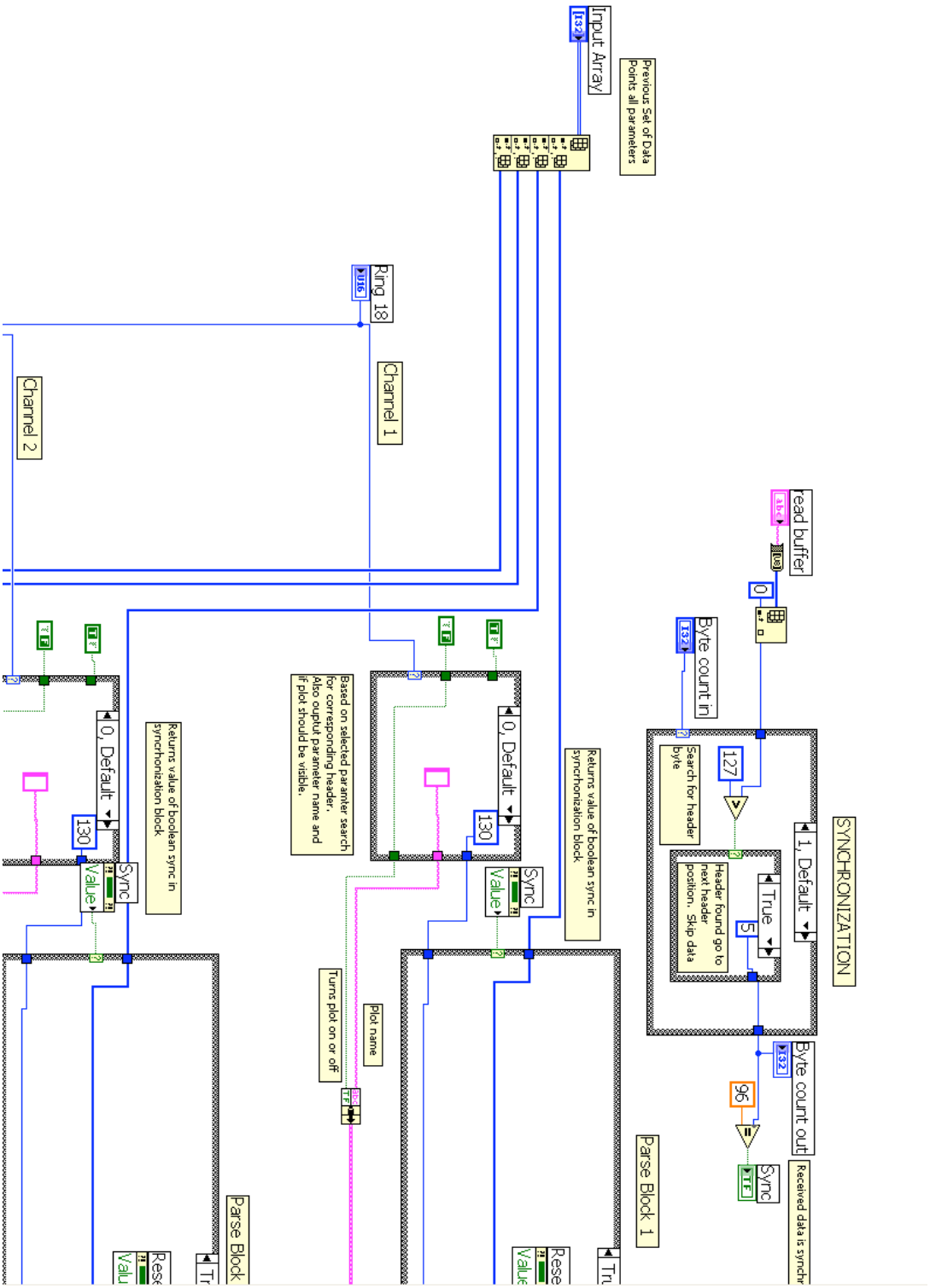
master ring select.vi



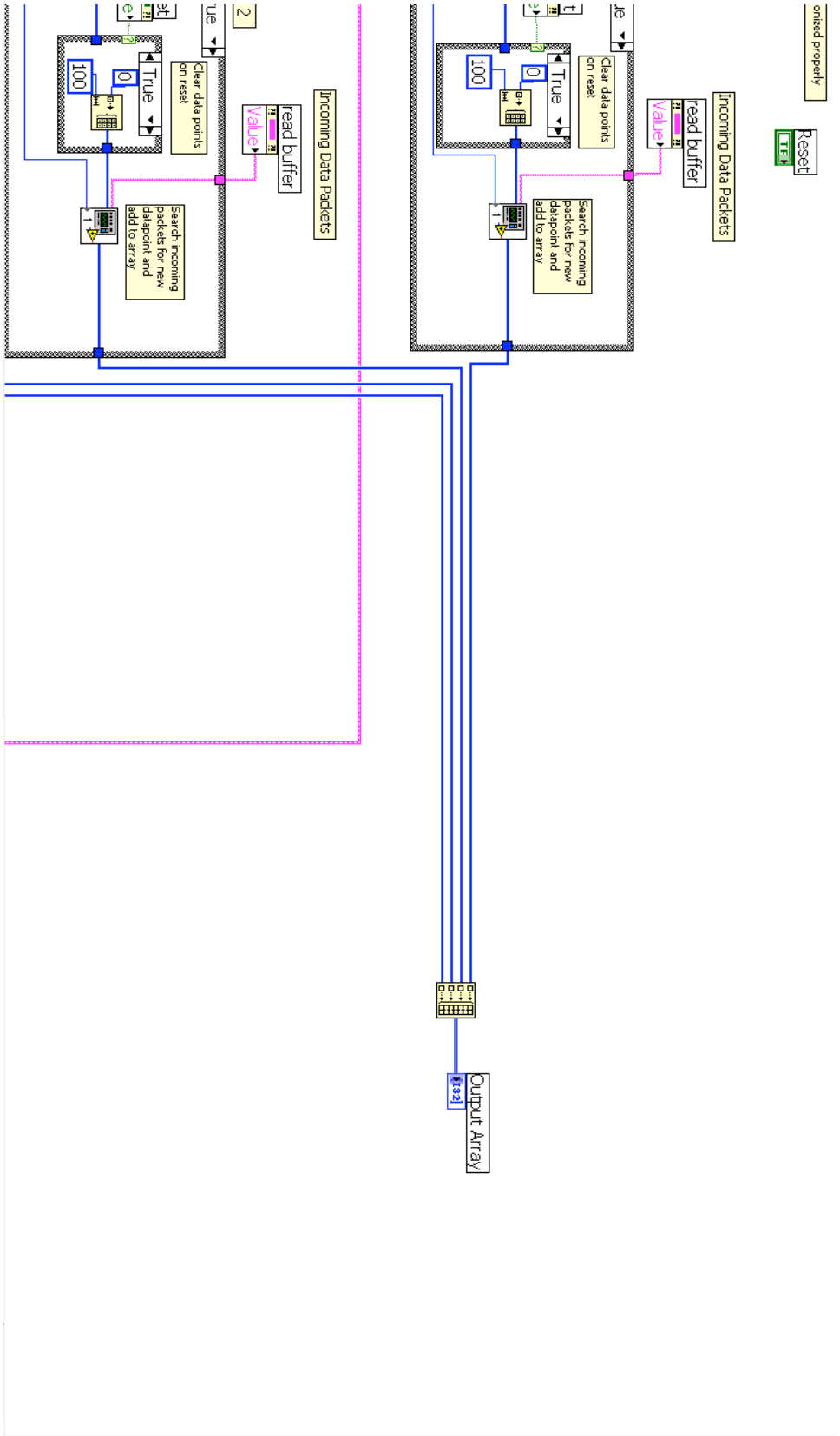
case structure_v3.vi



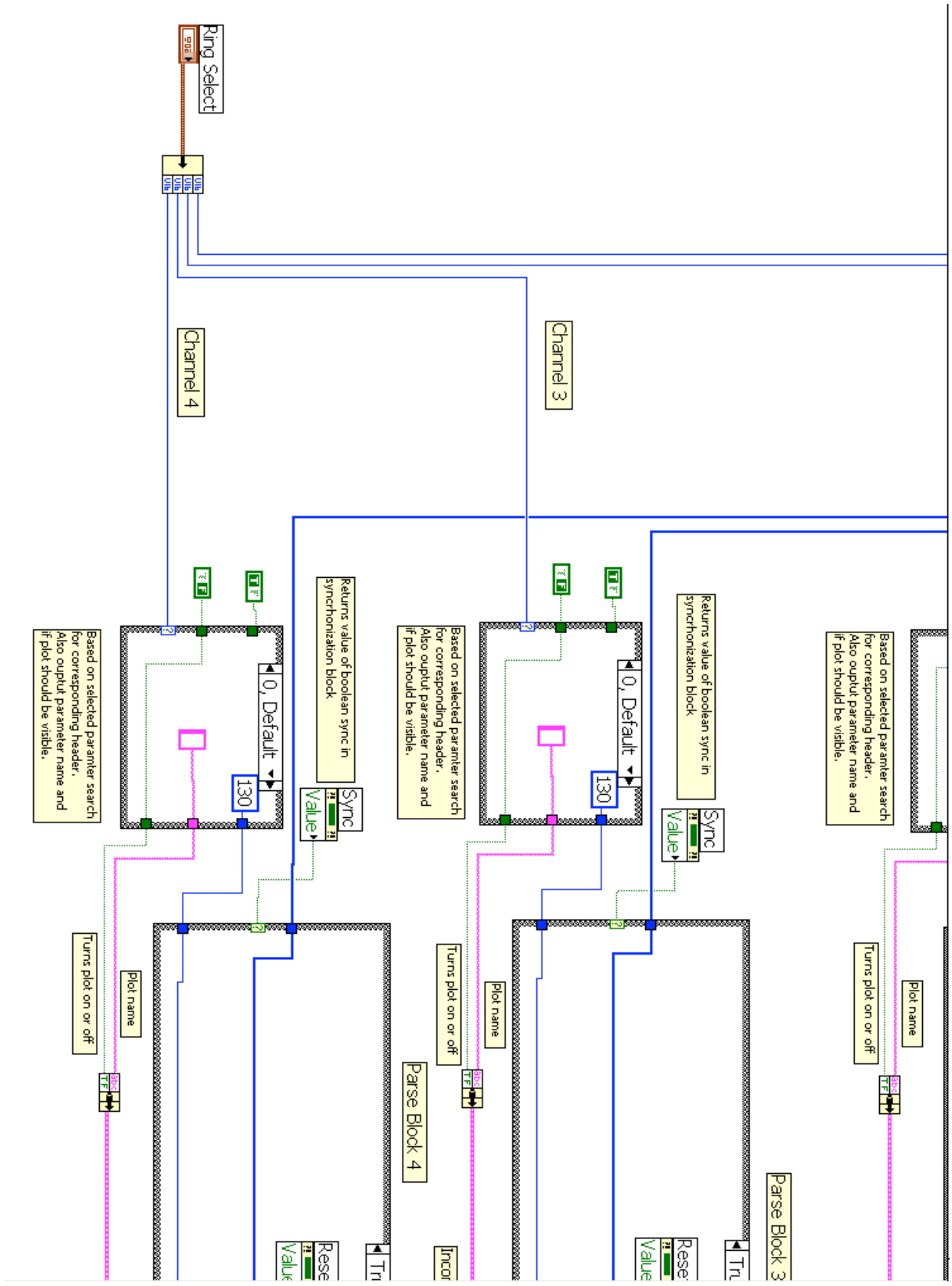
Big_parse_sync_4c.vi



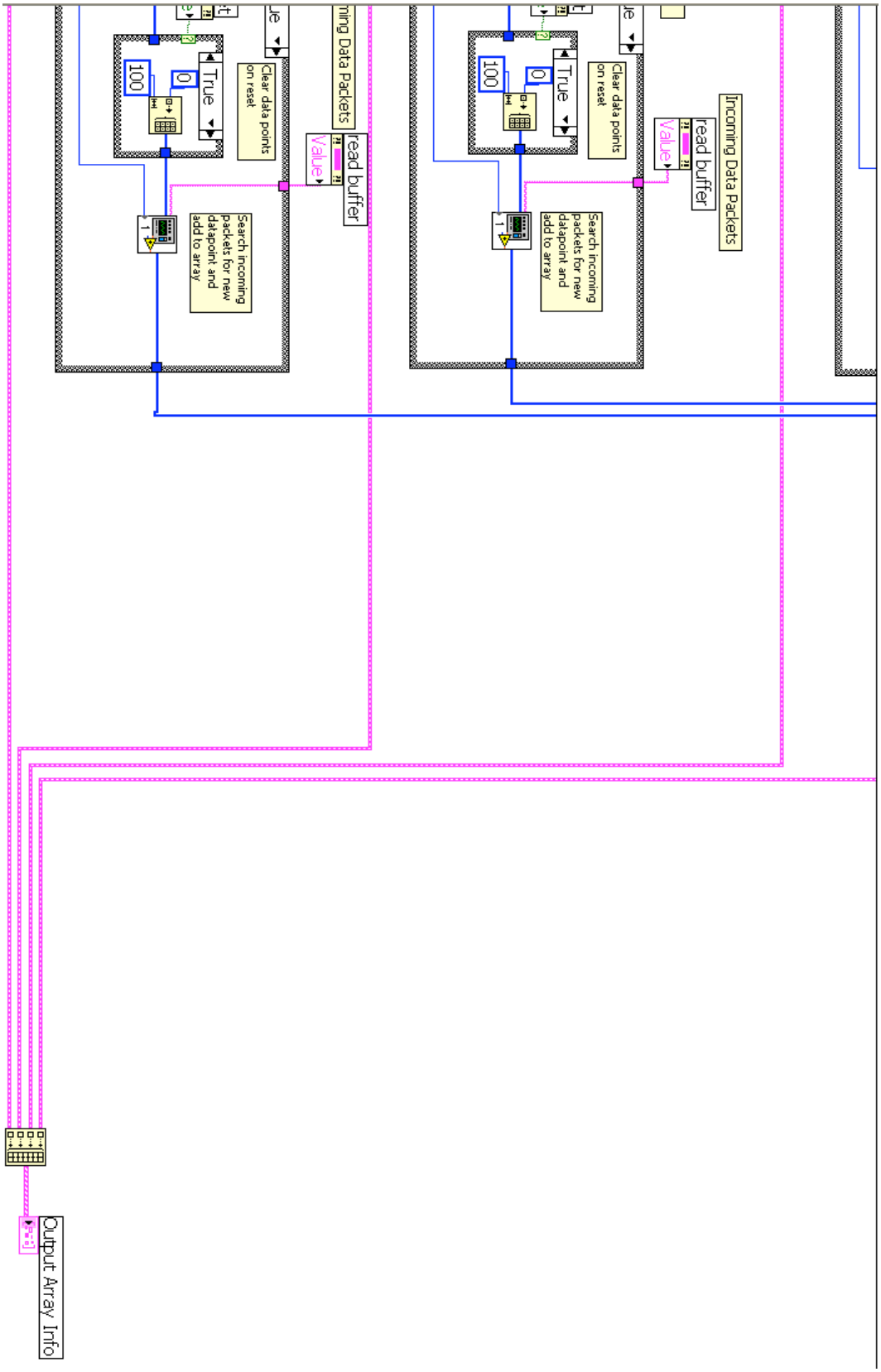
Big_parse_sync_4c.vi Section A1



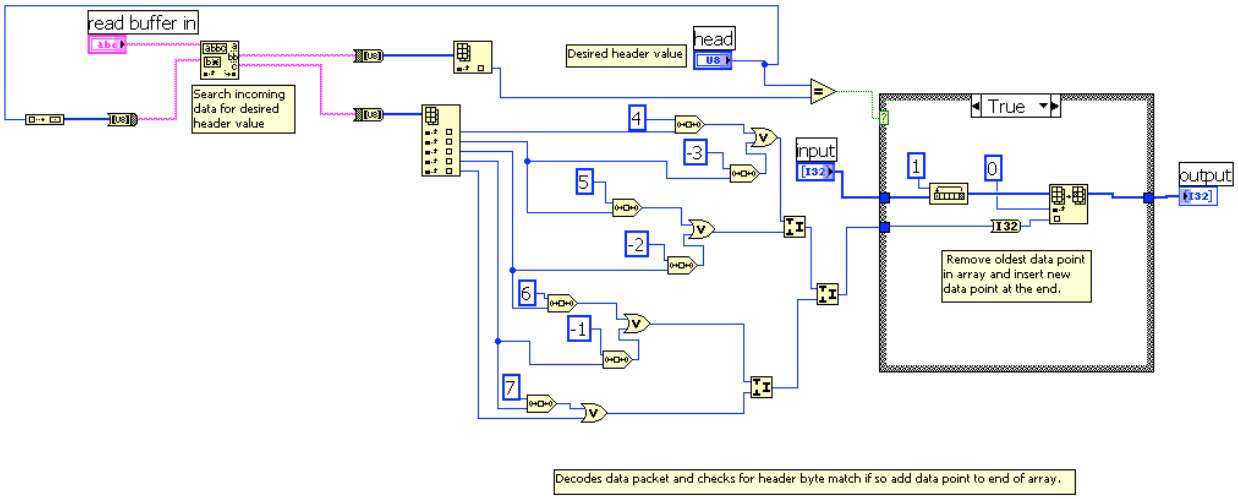
Big_parse_sync_4c.vi Section A2



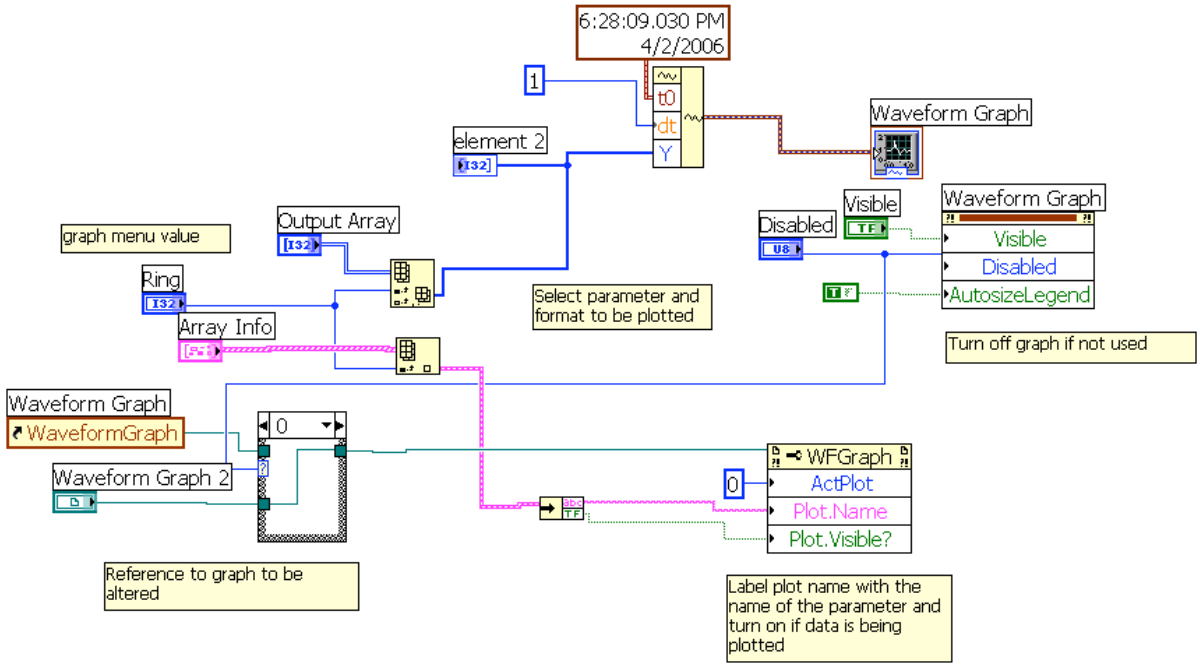
Big_parse_sync_4c.vi Section B1



Big_parse_sync_4c.vi Section B2



comm_decoder_4c.vi



channel select and format_v1.2.vi