

**MA4830 Realtime Software for Mechatronic Systems**

**Major CA Project**

**Submitted To: Assoc. Prof Seet Gim Lee, Gerald**

**Group Members:**

Left to Right:

**Ng Ee Meng (U1420377G)**

**Khoo Kar Keng (U1420494C)**

**Joyce Lim Xin Yan (U1521972L)**

**Tan Jin Cai (U1421759E)**

**Joseph Ng Rong En (U1422346L)**

**Philip Puan Jun Kit (U1420409F)**

Table of Contents

[1. Introduction 1](#_Toc498872461)

[2. Description of Program 1](#_Toc498872462)

[3. Limitations 4](#_Toc498872463)

[4. User Manual 5](#_Toc498872464)

[4.1 Execute Program 5](#_Toc498872465)

[4.2 General Help Menu 5](#_Toc498872466)

[4.3 Keyboard Input 7](#_Toc498872467)

[4.4 Export and Import Data Configurations 11](#_Toc498872468)

[4.5 Termination using CTRL-C 13](#_Toc498872469)

[4.6 Output waveforms on Oscilloscope 14](#_Toc498872470)

[5. Flow Chart 16](#_Toc498872471)

[5.1 Main Function 16](#_Toc498872472)

[5.1.1 Description of main function 17](#_Toc498872473)

[5.2 Commandline Manager 18](#_Toc498872474)

[5.2.1 Description of command line Manager 18](#_Toc498872475)

[5.3 Thread “WaveGenManager” 20](#_Toc498872476)

[5.3.1 Description of Thread “WaveGenManager” 21](#_Toc498872477)

[5.4 Waveform Data Generation 22](#_Toc498872478)

[5.4.1 Description of Waveform Data Generation 23](#_Toc498872479)

[5.5 PushDAC FIFO 24](#_Toc498872480)

[5.5.1 Description of Push DAC FIFO 25](#_Toc498872481)

[5.6 Thread “peripheral input” 26](#_Toc498872482)

[5.6.1Description of thread peripheral input 27](#_Toc498872483)

[5.7 Thread “Main UI” 28](#_Toc498872484)

[5.7.1 Description of thread “Main UI” 29](#_Toc498872485)

[5.8 Change DAC configuration 30](#_Toc498872486)

[5.8.1 The description of how to change DAC configuration 31](#_Toc498872487)

[5.9 Show the status of ADC 32](#_Toc498872488)

[5.9.1 Description of show the status of ADC 32](#_Toc498872489)

[5.10 Export information of DAC to file 33](#_Toc498872490)

[5.10.1 Description of export information of DAC to file 34](#_Toc498872491)

[5.11 Import information to file 35](#_Toc498872492)

[5.11.1 Description of import information to file 35](#_Toc498872493)

[5.12 Signal Handling 36](#_Toc498872494)

[5.12.1 Description of signal handling 37](#_Toc498872495)

[6. Code Listing 39](#_Toc498872496)

List of Figures

Figure 1: Compile and Execute Code 5

Figure 2: Program Help Menu 5

Figure 3: Showing current DAC0 configurations 6

Figure 4: Showing current ADCs and their statuses 6

Figure 5: Menu for keyboard commands 7

Figure 6: Changing type of waveform using keyboard 7

Figure 7: Changing frequency of waveform using keyboard 8

Figure 8: Changing amplitude of waveform using keyboard 8

Figure 9: Changing the amplitude value of waveform using keyboard 9

Figure 10: Turning DAC[0] OFF/ON using keyboard 9

Figure 11: Resetting waveform to default values using keyboard 10

Figure 12: Export Configuration into file.txt 11

Figure 13: Export Options 11

Figure 14: Exported Waveform Data and Configuration 12

Figure 15: Importing saved waveform configuration 12

Figure 16: Termination of program with CTRL-C 13

Figure 17: Original Triangular Waveform (example) 14

Figure 18: Changed Frequency of Original 14

Figure 19: Changed Frequency & Amplitude of Original 14

Figure 20: Changed Frequency, Amplitude & Mean of Original 14

Figure 21: Sine Wave Output 15

Figure 22: Square Wave Output 15

Figure 23: Flow Chart of Main Function 16

Figure 24: Flow Chart of CL Manager 18

Figure 25: Flow Chart of Wave Gen Manager 20

Figure 26: Flow Chart of Waveform Data Generation 22

Figure 27: Flow Chart of PushDAC FIFO 24

Figure 28: Flow Chart of peripheral input 26

Figure 29: Flow Chart of MainUI 28

Figure 30: Flow Chart of ChangeDAC configuration 30

Figure 31: Flow Chart of show DAC 32

Figure 32: Flow Chart of export config 34

Figure 33: Flow Chart of import config 35

Figure 34: Flow Chart of signal handling 36

Figure 35: Flow Chart of signal handling 38

# Introduction

QNX is a very useful OS for doing real-time programming. It is used for various time-critical tasks, which are often present in a Mechatronic system. However, without the proper peripherals, its functionality is limited to the computing space.

Throughout the course of MA4830 Realtime Software for Mechatronic Systems, the computer is connected to a PCI-DAS1602/16 board. The PCI-DAS1602/16 is a multifunction measurement and control board designed to operate in computers with PCI bus accessory slots.

The use of the various peripherals, such as the General Purpose Input/Output (GPIO), Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC) on the PCI-DAS1602/16 board allows us to interface QNX with the external environment. The robustness of QNX and the rich functionalities of the PCI-DAS1602/16 board allows a myriad of tasks to be undertaken.

To demonstrate both the capabilities of the OS and the PCI board, we have programmed an application for wave generation. This application takes in input from both, a user interface and a set of physical peripherals, such as potentiometers and flip switches. Using these inputs, a waveform of choice is being produced by the system.

# Description of Program

The program interfaces with the PCI-DAS1602/16 board to produce waveform (either sinusoidal, triangular or square waveform) from the board’s DAC.

The main parameters of DAC are waveform type, frequency, amplitude, mean and isOn settings. The range for the inputs are:

1. Frequency – (ADC) from more than 0.00122Hz to less than 1750Hz, (0.0267, 1750) Hz.

(Keyboard) from more than 0 to less than 1750Hz, (0, 1750) Hz.

1. Mean – (ADC) from more than -9.8V to less than 9.8V, (-9.8, 9.8) V

(Keyboard) from more than -10V to less than 10V, (-10, 10) V

1. Amplitude – (ADC) from more than or equal to 0V to less than 9.8V, [0, 9.8) V

(Keyboard) from more than or equal to 0V to less than 10V, [0, 10) V

The user can use command line arguments to initialise starting parameters. However, the following order must be met:

<program\_name.exe> <waveform type> <frequency> <mean> <amplitude> <isOn>

Waveform type argument can be “-sin” (sine wave), “-tri” (triangular wave) or “-squ” (square wave). Frequency, mean and amplitude are floating point number and as such floating-point values can be inputted. The last argument is isOn argument where user insert 0/1 to turn off/on the DAC upon entry to the program.

Example execution is:

**./wavegen -sin 1000.3 2.4 5 1**

To change the parameters, the program takes in user input from the keyboard and/or from the potentiometers (using the board’s ADCs) and switches connected to the PCI board.

For the digital switches, bit description is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Bit number | Bit 3 | Bit 2 | Bit 1 and 0 |
| Description | Turn on/off parameters’ input acquisition from ADC and digital switches | Select the parameter to be changed by potentiometer 0 (ADC0) | Select the waveform type and change DAC operation |
| Bit description | 0 – Input acquisition from keyboard  1 – Input acquisition from ADC and digital switches (any keyboard input will be overwritten by continuous ADC acquisition) | 0 – Changes mean  1 – Changes amplitude | 0x00 – DAC is off  0x01 – Sine wave  0x02 – Triangular wave  0x03 – Square wave |

The formulae used for ADC (16 bit) conversion to frequency, amplitude or mean values are:

ADC0

ADC1 (Dedicated to frequency)

HIGHESTFREQ is 1750Hz.

The user can also export and import the DAC configuration to text file in the format of command line statement. Further descriptions are in the user manual.

The novelties of the program are:

1. Wide range of amplitude and mean inputs with data output range of (-10, 10) V
2. High achievable frequency with wide range of frequency input (0, 1750) Hz
3. Smart selection of DAC modes (bipolar/unipolar) to give the best DAC resolution
4. Robust system with input error checking and range checking
5. Returnable feature to the program when Ctrl+C is accidentally pressed
6. Simple, clean and user-friendly user interface
7. Export and import functions of DAC configuration and waveform data.
8. Smart resource management. Refresh UI and update waveform data only when necessary, as well as using less resource for low frequencies.

# Limitations

There are several limitations to this program, notably:

1. Only one wave generator, as multithreading makes resource management between wave generator unpredictable.  
   Suggestion: A scheduler to push FIFOs (two or more DACs) periodically.
2. The peripheral boards overwrite the user input from the keyboard if it is turned on.  
   Suggestion: Block the peripherals during specific functions. Unblock it after that.
3. Resource heavy when the frequency is high.
4. There is still some small mismatch in real frequency and ideal frequency. The mismatch is almost non-existant below 2Hz, but is around 10% below 2Hz. By doing some empirical adjustments, the discrepancies reduces as frequencies becomes higher, being only 1% at around 1kHz.

# User Manual

The user manual constitutes of the actual inputs using the keyboard, and the outputs on the screen and on the oscilloscope.

## 4.1 Execute Program

Figure 1: Compile and Execute Code

To compile the code and execute it, type in the commands as shown in Figure 1. In the execution, flags can be added to select type of wave form (-sin: sine, -tri: triangular, -squ: square), frequency, mean, amplitude and turning on/off DAC[0] (0: OFF, 1: ON) respectively. If no flags are added, the default settings will be sine wave, with frequency = 1 Hz, mean = 0 V, amplitude = 1 V and DAC0 is off.

## 4.2 General Help Menu

Figure 2: Program Help Menu

In Figure 2, it depicts the help menu when the program has been executed. The commands to be entered ranges from numerical values of 1-7. If input command is out of this range, “Invalid Choice” will be printed.

The following figures, Figure 3 and 4, illustrate how to get the various statuses and configurations of the DAC and ADCs.

Figure 3: Showing current DAC0 configurations

|  |  |
| --- | --- |
| **Keyboard Input:** 1  **Switches Input:** By turning the potentiometer, the frequency, amplitude or mean can be adjusted. | **Output:** Configurations of DAC0 which includes waveform types, number of samples per period, output resolution (V) and **current** values of frequency (Hz), amplitude (V) and mean (V). |



Figure 4: Showing current ADCs and their statuses

|  |  |
| --- | --- |
| **Keyboard Input:** 2 | **Output:** Shows the current ADC values and the DAC0 configurations. |

In addition, Figure 4 depicts how to control the switches to achieve desired configurations, which was explained in Section 2.

## 4.3 Keyboard Input

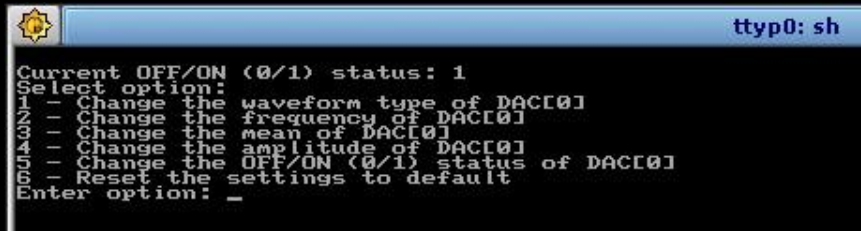
In Figures 5-11, they show how to change several values by using keyboard input. For these to occur, the PCI board must be switched off or the values of the board will overwrite those from the keyboard. From Figure 1, enter command **“3”** in the help menu to allow the program to change configurations by using the keyboard, and the screen is shown in Figure 5.

Figure 5: Menu for keyboard commands

The menu for keyboard commands are shown in Figure 5. Numerical commands ranging from 1-6 can be keyed in to produce various functions, which will be explained in Figures   
6-11.

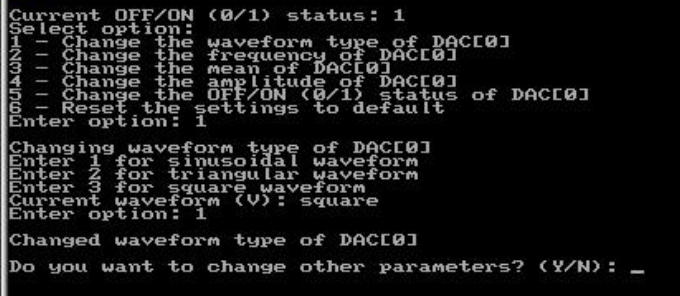


Figure 6: Changing type of waveform using keyboard

Upon entering command **“1”** from the menu for keyboard commands, the waveform type of DAC[0] can be changed. Figure 6 illustrates the current waveform type and commands to change the waveform to desired shapes. If the number entered is not an integer ranging from 1-3, an error message “Invalid input for waveform. (Must be integer)” will be printed.

|  |  |
| --- | --- |
| **Keyboard Input:** 1 | **Output:** Changes current waveform to sine wave |

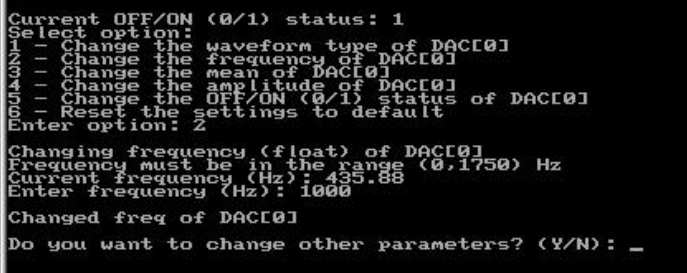
To change the frequency of the waveform of DAC[0] by using keyboard, input command **“2”** in the keyboard menu. Then, any value within the range can be used, as seen from Figure 7. If the value is out of range, an error message “Frequency is not changed” will be printed.

Figure 7: Changing frequency of waveform using keyboard

|  |  |
| --- | --- |
| **Keyboard Input:** 1000 | **Output:** Changes current frequency to 1000 Hz. |

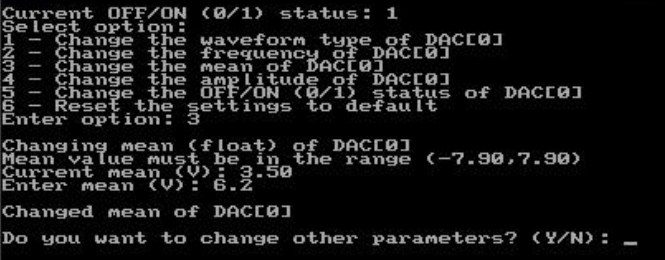


Figure 8: Changing amplitude of waveform using keyboard

Similarly, in order to change the mean value of the waveform, command **“3”** is to be keyed into the keyboard menu, as shown in Figure 8. The mean value can then be changed. If the mean value entered is not within range, an error message “Mean value is not changed” will be seen.

|  |  |
| --- | --- |
| **Keyboard Input:** 6.2 | **Output:** Changes current mean value to 6.2 V. |

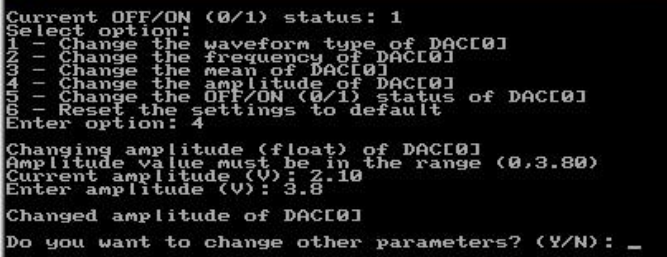
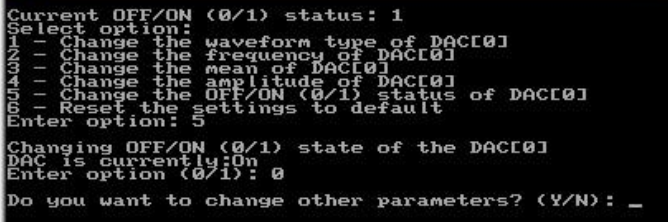
Also, to change the amplitude value, enter command **“4”** in the keyboard menu. Figure 9 demonstrates how to change the amplitude value of the waveform. If an error message of “Amplitude value is not changed” is seen, it means that the new amplitude value if out of range.

Figure 9: Changing the amplitude value of waveform using keyboard

|  |  |
| --- | --- |
| **Keyboard Input:** 3.8 | **Output:** Changes current amplitude value to 3.8 V. |

Figure 10: Turning DAC[0] OFF/ON using keyboard

Turning off DAC[0] can be achieved by entering command **“5”** in the keyboard menu. The current state of the DAC is shown in Figure 10. If the new option entered is not 0 or 1, an error message “Invalid options for OFF/ON” will be printed. Also, if the current state and new input state are the same, the program will feedback to the user that the DAC is already off or on.

|  |  |
| --- | --- |
| **Keyboard Input:** 0 | **Output:** If DAC[0] is ON, turn DAC[0] OFF. If DAC[0] is OFF, notify user that DAC[0] is already OFF. |

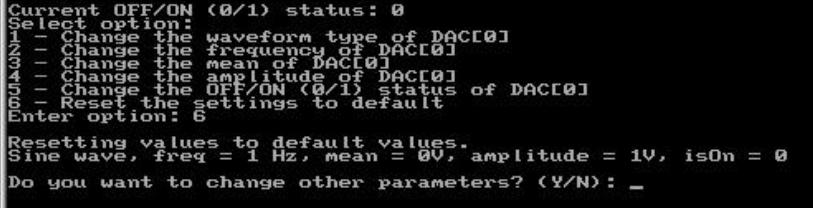
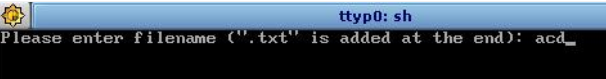
If the user desires to reset the waveform to default values, this can be done through the keyboard menu, as depicted in Figure 11.

Figure 11: Resetting waveform to default values using keyboard

|  |  |
| --- | --- |
| **Keyboard Input:** 6 | **Output:** Changes all previous values into default settings of sine wave, frequency = 1 Hz, mean = 0 V, amplitude = 1 V, and DAC to OFF. |

## 4.4 Export and Import Data Configurations

The program is capable of exporting and importing saved waveform configurations into a file, however, the PCI board must be turned off during importing of data importation to ensure that the current values on the board do not overwrite the saved file.

Enter command **“4”** in the help menu to export current configuration, followed by defining the file name of the configuration, as illustrated in Figure 12. The filename has a maximum of 30 characters, including the null terminator. If the filename is NULL, an error message will be printed.

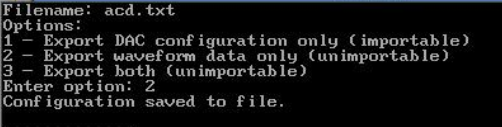


Figure 12: Export Configuration into file.txt

After entering the filename, there will be 3 export options available for the user, as shown in Figure 13. If the user would like to import the DAC configuration subsequently, it is recommended not to export the waveform data. However, if waveform data is desirable, the user can save the data into the defined filename. In addition, if the command sent is not within integers 1-3, the program will feedback to the user an error message.

Figure 13: Export Options

|  |  |
| --- | --- |
| **Keyboard Input:** 3 | **Output:** The DAC configuration and waveform data are saved into file “acd.txt”. |

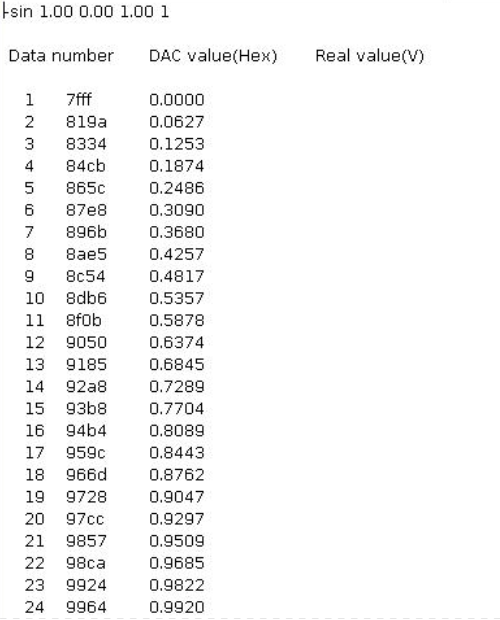
In Figure 14, it illustrates the saved file, ‘*acd.txt*’, which has the DAC configuration and values of the waveform at each point. The data consists of the data number, DAC Value (Hex) and Real Value (V), and the waveform is plotted using 100 points of data.

Figure 14: Exported Waveform Data and Configuration

To import the previously saved configuration, the user enters the command **“5”** in the help menu and enters the filename, without “.txt”, as demonstrated in Figure 15. The saved configuration is in file ‘*abc.txt’* in this example. The configuration will not be able to be imported if the saved file has waveform data.

Figure 15: Importing saved waveform configuration

|  |  |
| --- | --- |
| **Keyboard input:** abc | **Output:** Loads configurations of file *‘abc.txt’* and projects waveform onto oscilloscope. |

## 4.5 Termination using CTRL-C

Figure 16: Termination of program with CTRL-C

In Figure 16, it depicts the response of the program to the user when CTRL-C is entered. It allows the user to return to the program if CTRL-C was pressed unintentionally. If the user wishes to terminate the program through CTRL-C, he/she can enter **“Y”** or **“y”** to exit the program when prompted.

## 4.6 Output waveforms on Oscilloscope

In this section, picture illustrations will show the different actual outputs on the oscilloscope when parameters have been changed.

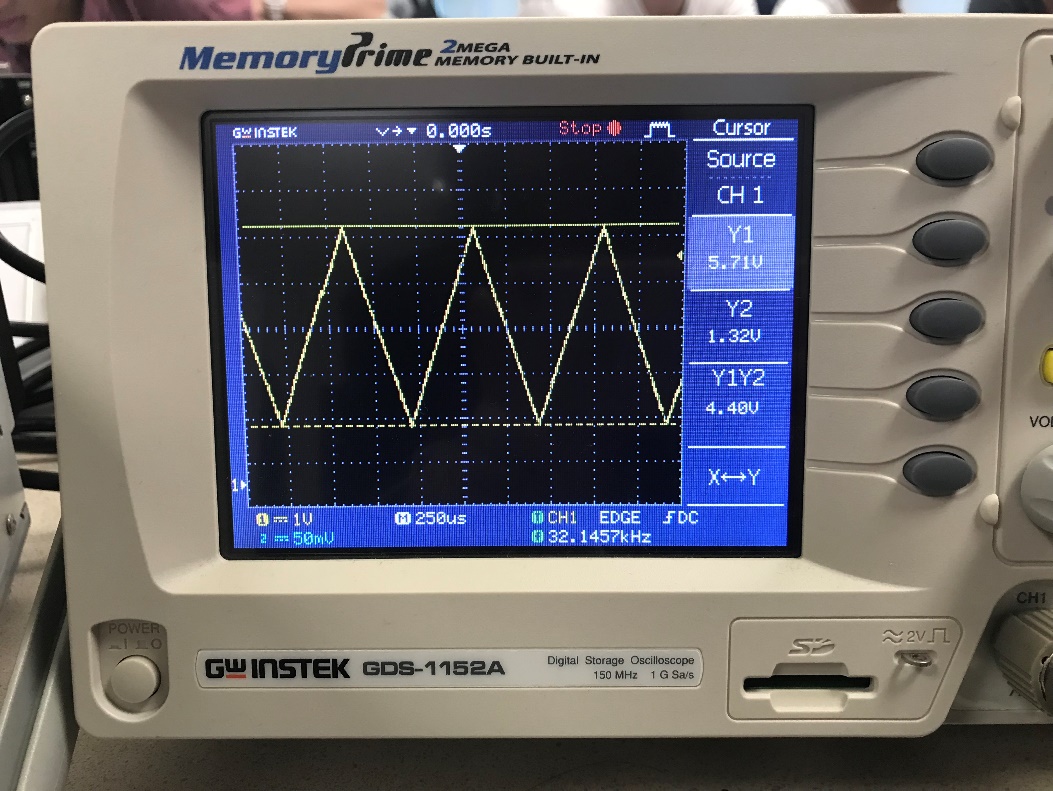


Figure 20: Original Triangular Waveform (example)

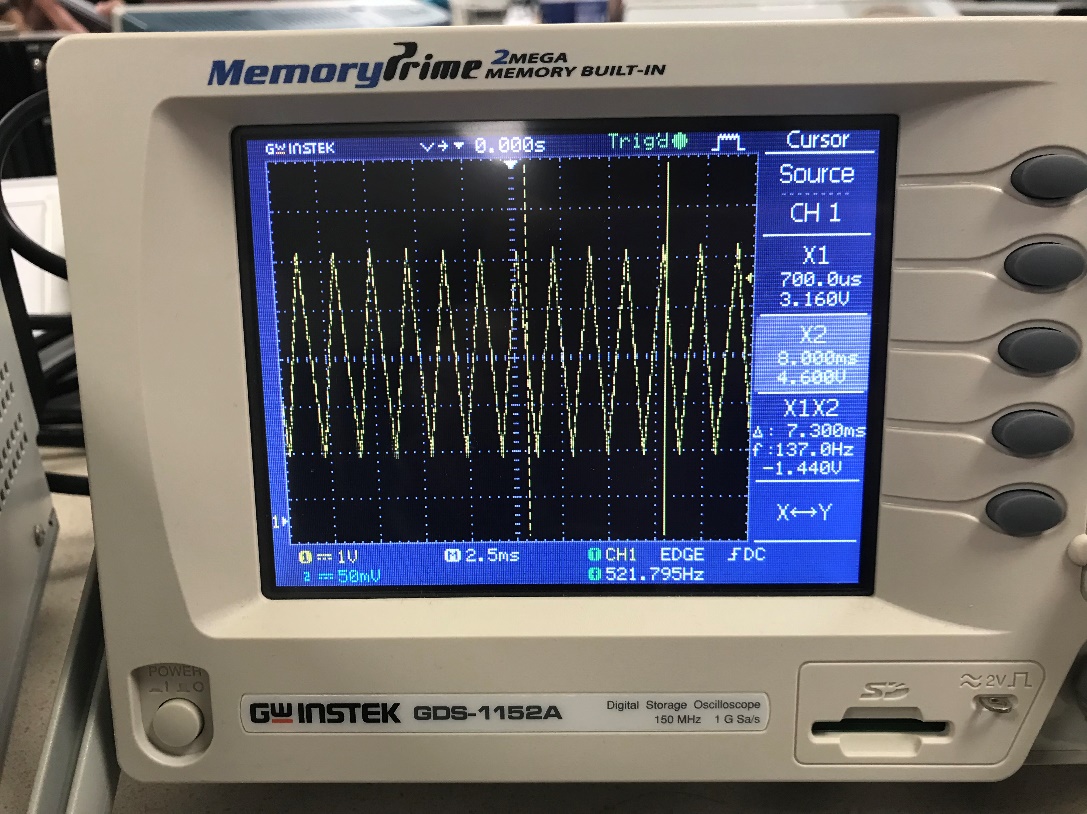


Figure 20: Changed Frequency of Original

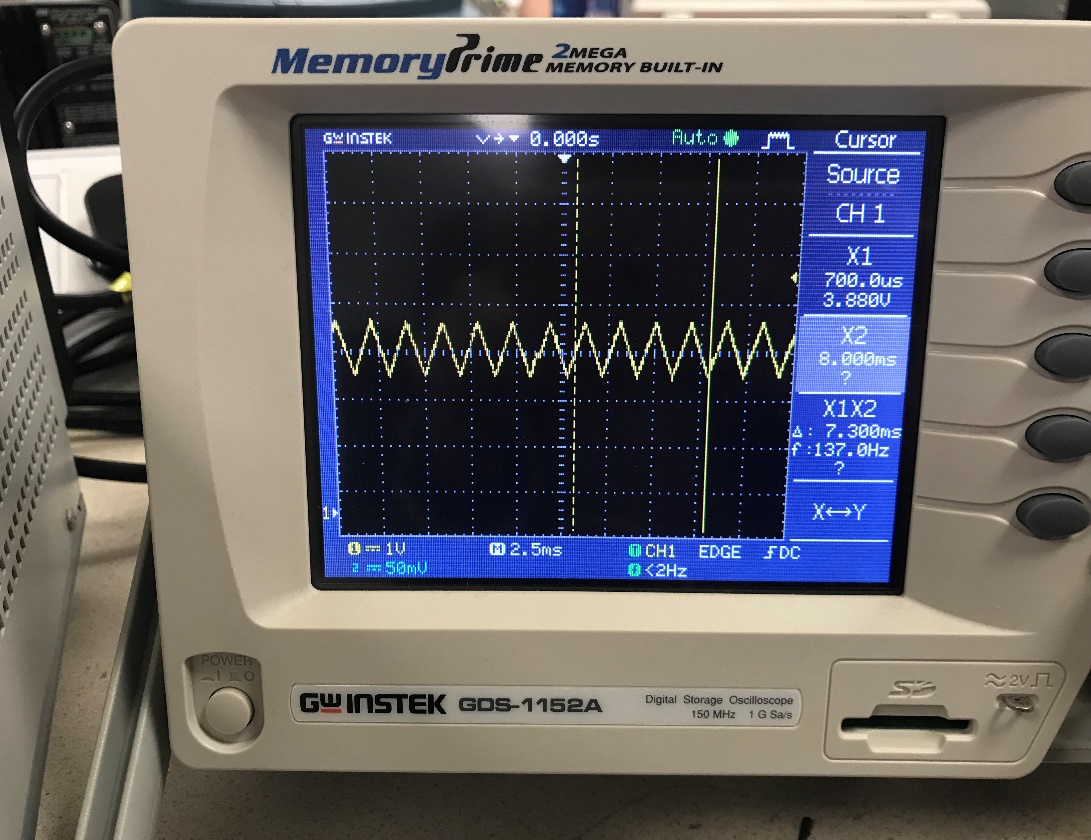


Figure 20: Changed Frequency & Amplitude of Original

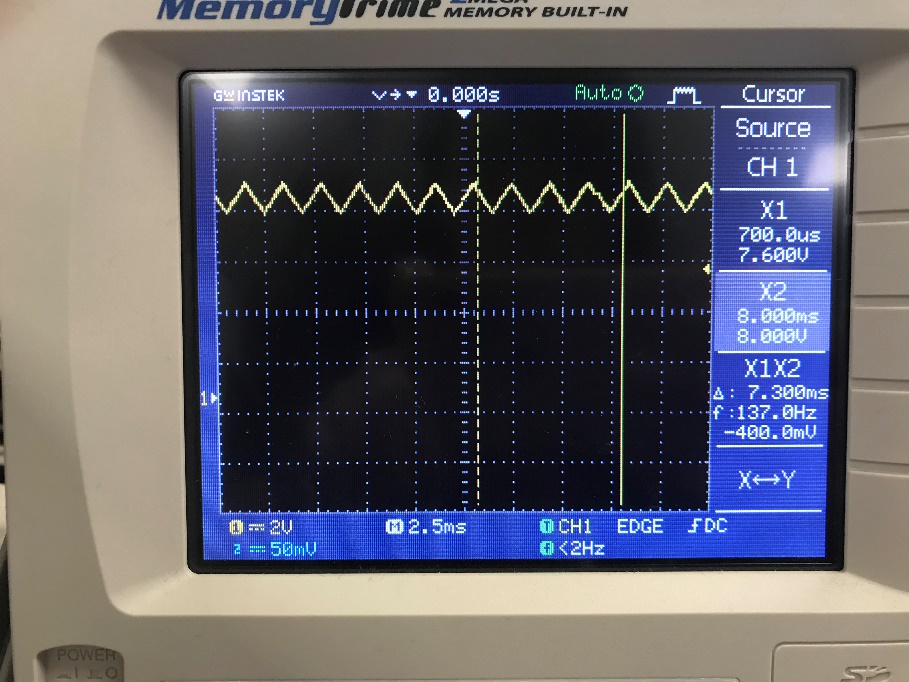


Figure 20: Changed Frequency, Amplitude & Mean of Original

Figures 17-20 depicts the variations of a triangular wave, which is used as an example.

Figures 21-22 illustrates two different waveforms shown on the oscilloscope; the sine wave and square wave, respectively.

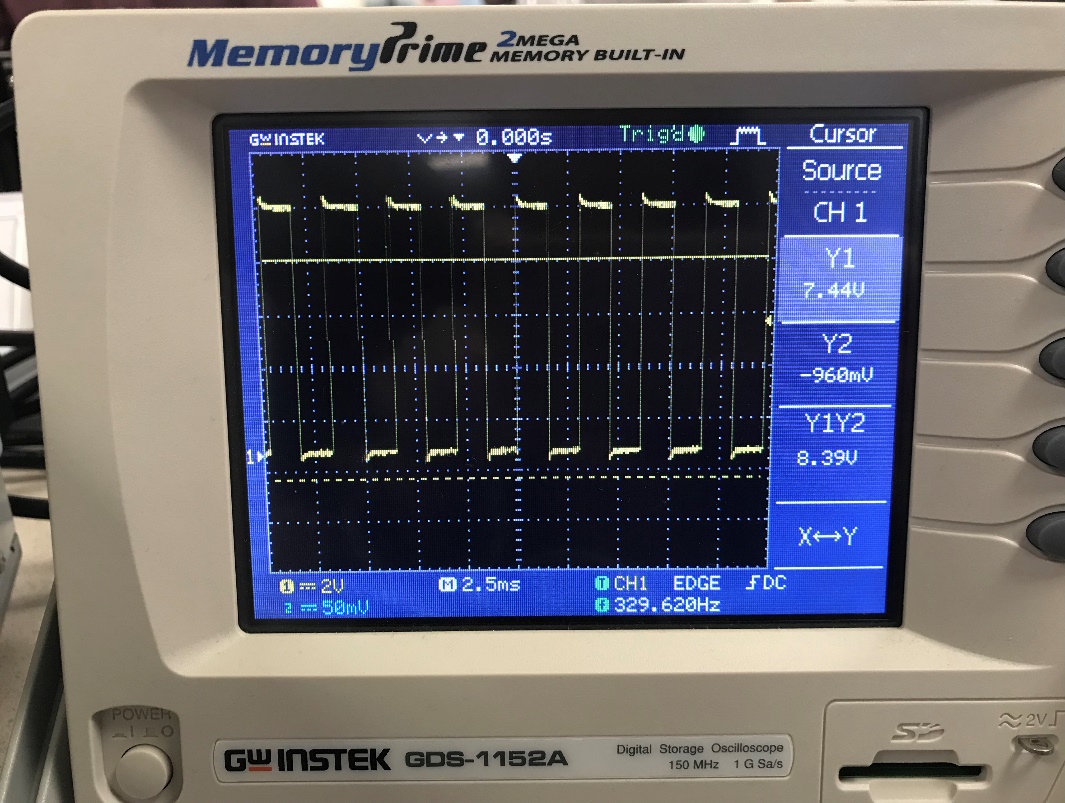
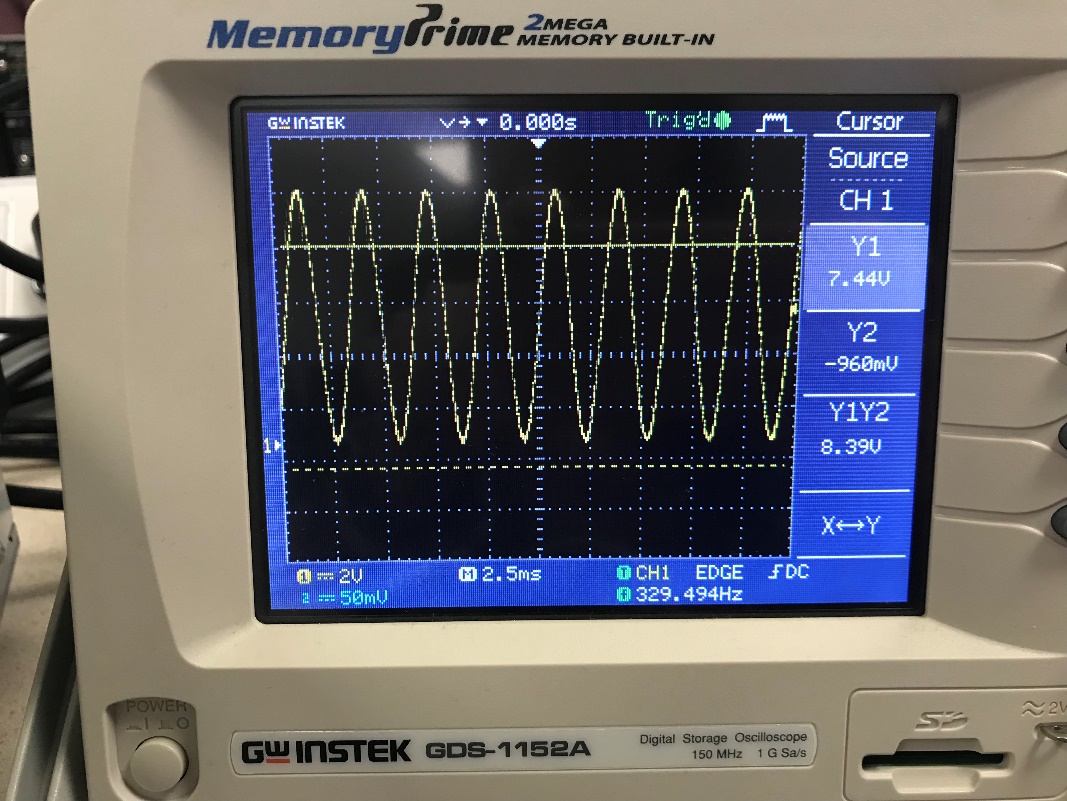


Figure 22: Sine Wave Output

Figure 22: Square Wave Output

# Flow Chart

## 5.1 Main Function

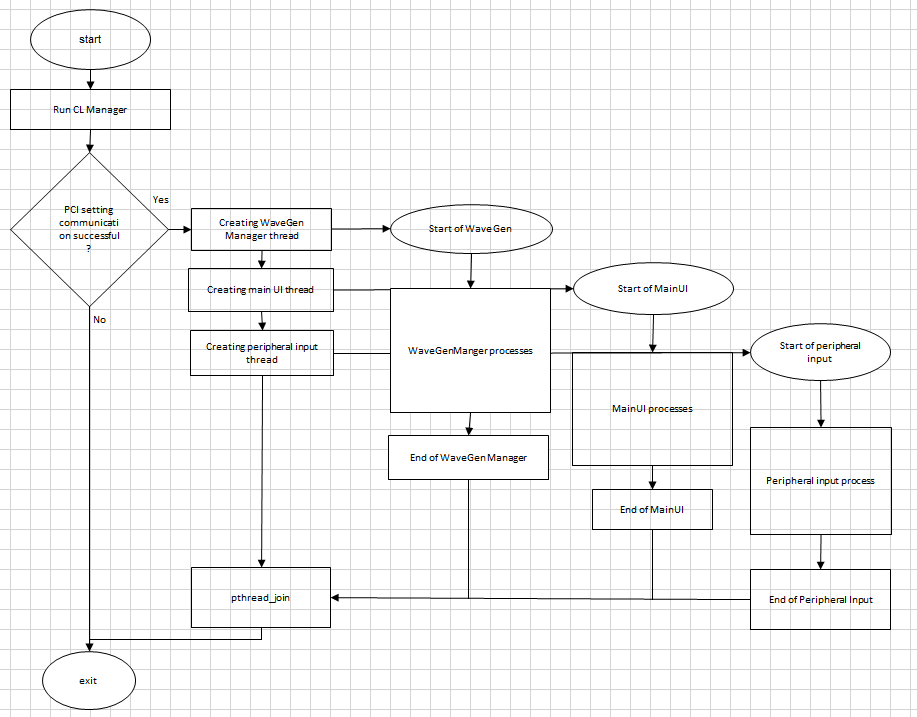


Figure 23: Flow Chart of Main Function

### 5.1.1 Description of main function

The diagram illustrate the flowchart of main function.

At first, function of “CL Manager” is invoked in order to determine the validity of string which inputted from user through terminal and value of data such as mean ,frequency and amplitude will being stored and eventually the configuration of DAC will being updated. The flowchart of “CL Manager” is being shown at Figure 24.

Main function will only continue if PCI setting communication is successful. Three new thread which are “WaveGen Manager”, “MainUI”, “Peripheral Input”will be created sequentially in main function and those 3 thread will run simultaneously with main thread.

P\_threadjoin function is used in main function in order to wait every threads until they complete their own execution.

## 5.2 Commandline Manager

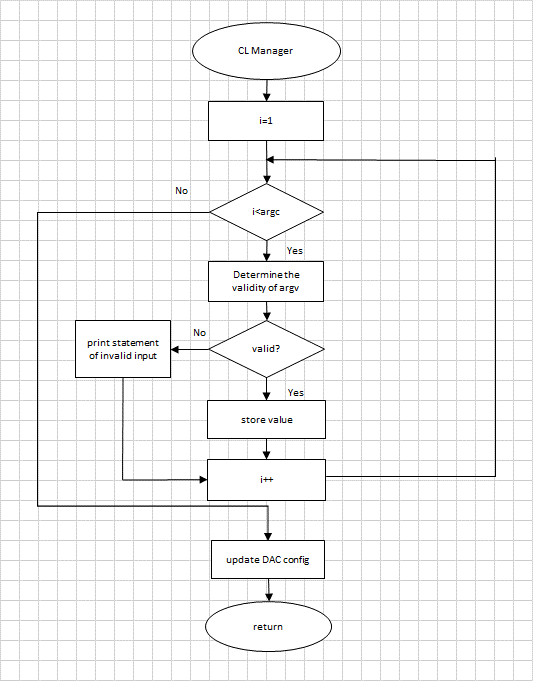


Figure 24: Flow Chart of CL Manager

### 5.2.1 Description of command line Manager

As mentioned in 5.1.1, Command Line Manager is used to determine the validity of string and check the value of frequency, mean, and amplitude whether those value exceed their range respectively.CL manager will only update the value if the value is in the range and valid string is inputted.

The flowchart of CL manager is shown in Figure 24 . By referring the user manual, user are required to input the format ./ <program\_name.exe> <waveform type> <frequency> <mean> <amplitude> <isOn> in order to execute the program. Once user complete the input process,

CL manager is invoked to check the string input whether is correct. The checking process will start from the string of <waveform.type>, if user input “tan” for waveform type. The invalid input statement will being shown in terminal. After that ,string of <frequency>,<mean>,<amplitude> will being converted to double type value and determine those value whether each value is inside their own range. If the value exceed the range, statement will being shown in order to notify user to provide correct input.

If the inputs are correct, the values of input will being stored inside memory and eventually updated to DAC. Once the counter exceed the number of arguments, the checking process will being terminated and other statements in main function will continue to run.

## 5.3 Thread “WaveGenManager”

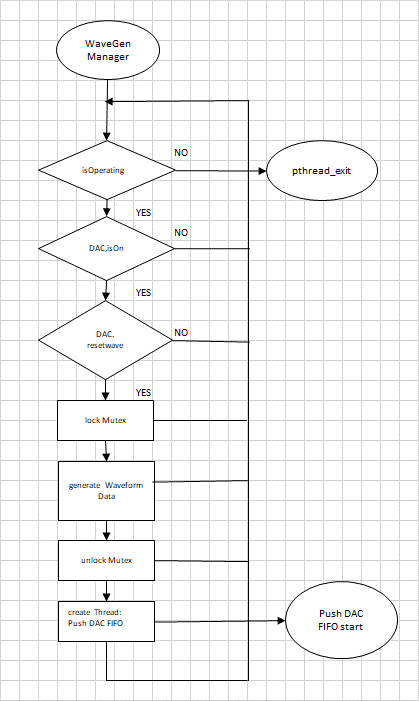


Figure 25: Flow Chart of Wave Gen Manager

### 5.3.1 Description of Thread “WaveGenManager”

The flowchart is shown in Figure 25. The thread “WaveGenManager” is used to supervise the status of DAC and resetwave which flag to reconfigure DAC data arrays and eventually create thread.

## 

## 5.4 Waveform Data Generation

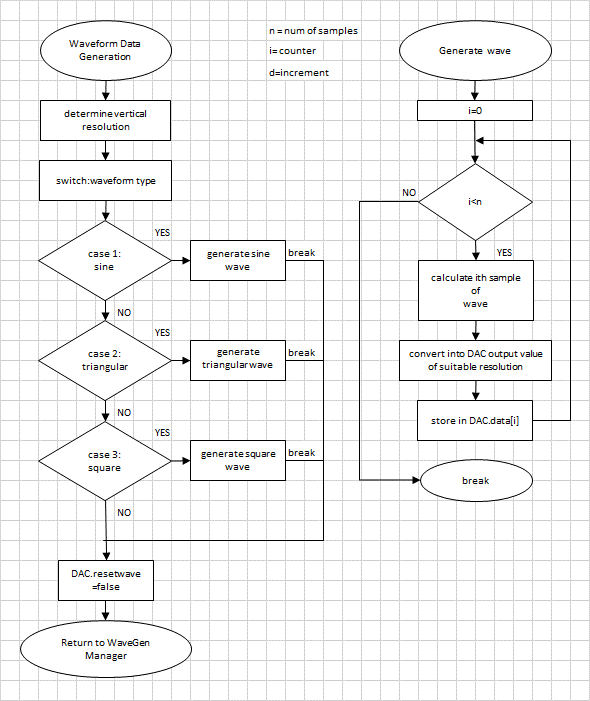


Figure 26: Flow Chart of Waveform Data Generation

### 

### 5.4.1 Description of Waveform Data Generation

This function will produce desired waveform from multiple options based on the decision of user . If user input 1, sine wave will being generated. If user input 2, triangular wave will being generated. If user input 3,square wave will being generated.

At the right of the diagram, it is the flowchart of how to generate the wave . The generation of sine, square and triangular wave are exactly similar. In our case, we have 100 sample in 1 period .In order to generate the wave, the value of ith data was being calculated sequentially

After that , the value will eventually converted into suitable DAC output value and stored in DAC data. The process will be completed once the number of iteration exceeds the total sample of data.

## 5.5 PushDAC FIFO

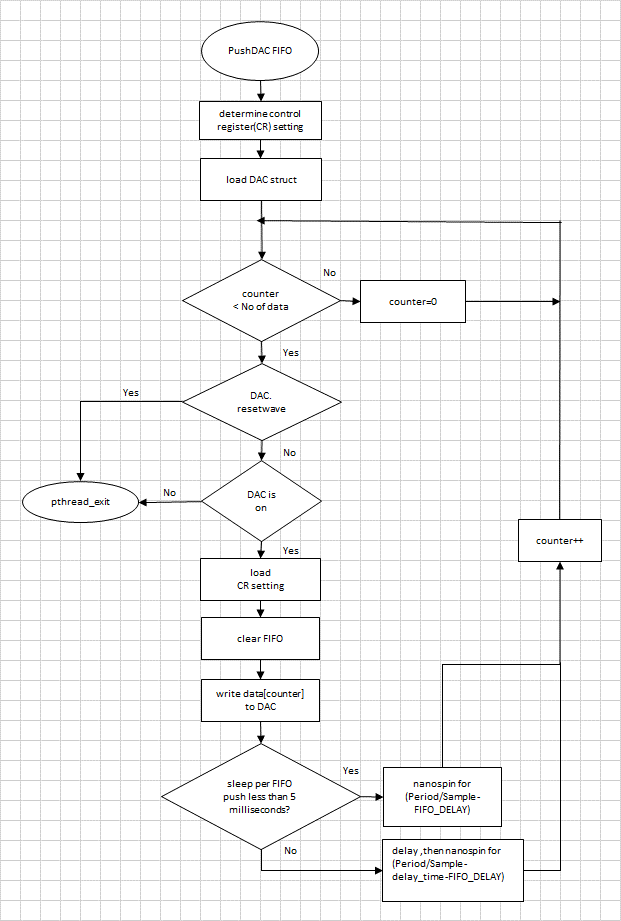


Figure 27: Flow Chart of PushDAC FIFO

### 5.5.1 Description of Push DAC FIFO

This thread will decide on the resolution of the DAC(whether 10v/5v unipolar/bipolar) and then periodically push out the corresponding analog value of wave into the DAC. This process will repeat until either DAC is switched off or a wave reset request is made, which will terminate this thread.

To conserve resources while still value performance, the duration between pushes is determine. If it is smaller than 5ms, we’ll use nanospin only to ensure accurate timing. Else, we’ll use a combination of nanospin and delay to conserve resource, while avoiding timing jitters. This is done by delaying for (duration – 2) milliseconds, determining how much time has passed using a clock, and then nanospin for the remaining duration.

FIFO\_DELAY is an empirical value for the time needed for the PCI board to perform analog output.

## 5.6 Thread “peripheral input”

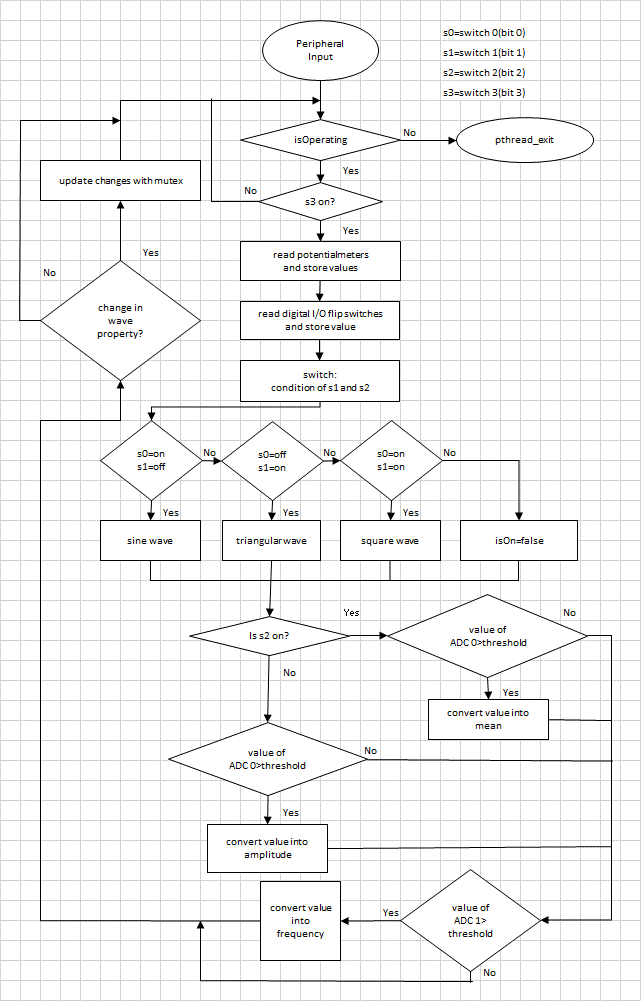


Figure 28: Flow Chart of peripheral input

### 5.6.1Description of thread peripheral input

The flowchart of thread peripheral input is shown in Figure 28. If the thread is running, the status of switch 3 will being checked . If switch 3 is on, the value of potential meter will being read and values is being stored. After that, the status of switch 0 and switch 1 are being recorded. If switch 0 is ON and switch 1 is OFF, sine wave will being generated. If switch 0 is OFF and switch 1 is ON , triangular wave will being generated. For the case switch 0 is ON and switch is OFF, square wave will being generated. If both switches are OFF, isOn will be set to false. After that, the status of switch 2 will being checked. If switch 2 is ON, the value of ADC 0 will being checked and eventually the value will being converted to mean provided the value larger than threshold value. However , the value of ADC 0 will being checked and the value will eventually converted into amplitude provided the value larger than threshold value. After that, the value of ADC 1 will being determined whether the value is larger than threshold value . The value will being converted into frequency provided the value is larger than threshold. Wave property wouldn’t be changed if the values do not being converted into mean, frequency and amplitude. Once the wave property is changed, the values of global variables is being updated with mutex. The process will run continuously to inspect the status of the switch.

## 5.7 Thread “Main UI”

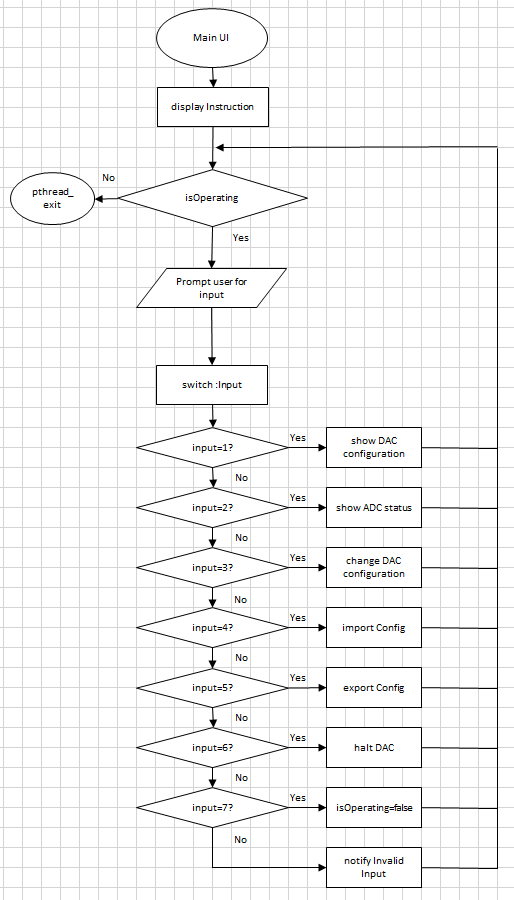


Figure 29: Flow Chart of MainUI

### 5.7.1 Description of thread “Main UI”

Instruction is displayed in order to guide user once the thread of main UI is executed. After that, User will being asked for providing valid input by referring the instruction menu. For example, user input "2", the status of ADC will being shown in terminal. However, invalid input statement will being shown in terminal if user provide incorrect input.

## 5.8 Change DAC configuration

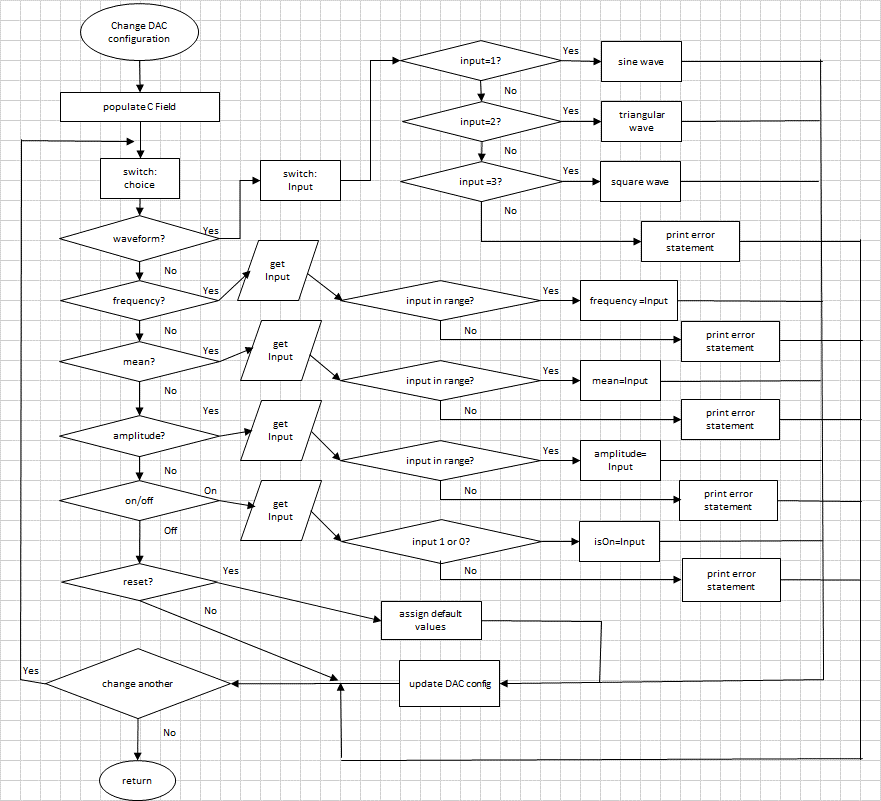


Figure 30: Flow Chart of ChangeDAC configuration

### 5.8.1 The description of how to change DAC configuration

The flowchart of change DAC configuration is shown in Figure 30. Basically, this function will determine the waveform type, frequency, mean, amplitude and on/off status and check the validity of those input. If the input is valid, the configuration of DAC will being updated with those valid input. However, the configuration of DAC will not updated and print error statement will being shown in terminal if user provide invalid input

## 5.9 Show the status of ADC

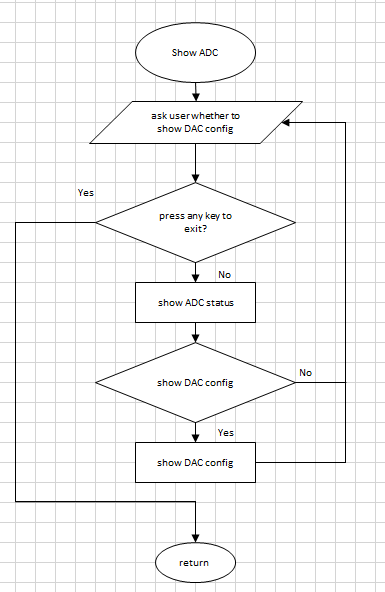


Figure 31: Flow Chart of show ADC

### 5.9.1 Description of show the status of ADC

This function will show the current ADC configuration. The function will also ask the user if they would like to show DAC configuration (Wave config) at the same time. The function will keep on running until a key is pressed, which will return the function to the main UI function.

To conserve resource, both show DAC config and show ADC config will only be run when there are changes in reading of the potentiometer or the orientation of the flip switches.

## 5.10 Export information of DAC to file

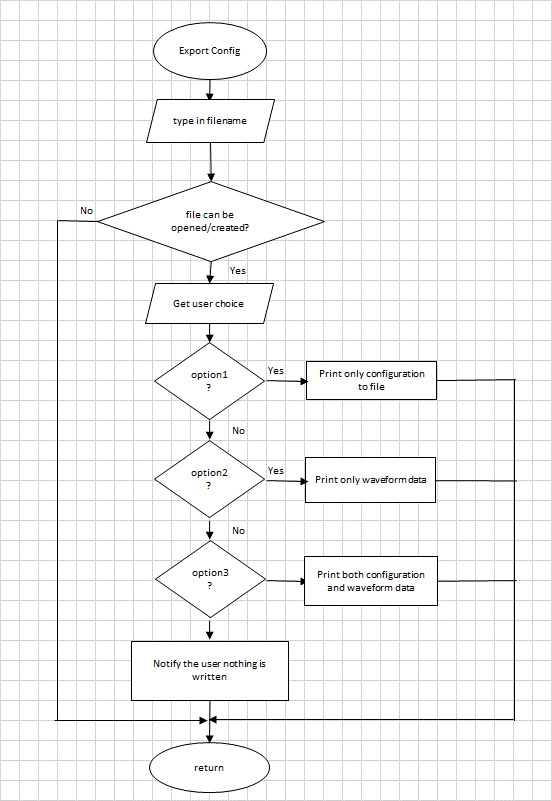


Figure 32: Flow Chart of export config

### 5.10.1 Description of export information of DAC to file

This function is used to export the configuration of DAC into a <filename.txt> file. At first, user are required to create or open a file. In our case, we have 3 option for user to choose what kind of parameters are required to be exported to the file. For case 1, the configuration of waveform will only be exported to the file. For case 2, the data of waveform will only be exported to the file. In case 3, both of them will be exported to the file. If user provide invalid choice, user will being notified nothing is written through the printed statement in terminal.

## 5.11 Import information to file



Figure 33: Flow Chart of import config

### 5.11.1 Description of import information to file

This function is used to import the information from <filename.txt>.At first, user will being asked to type the filename and the information inside the file will being extracted only if the file can be opened successfully.

## 5.12 Signal Handling

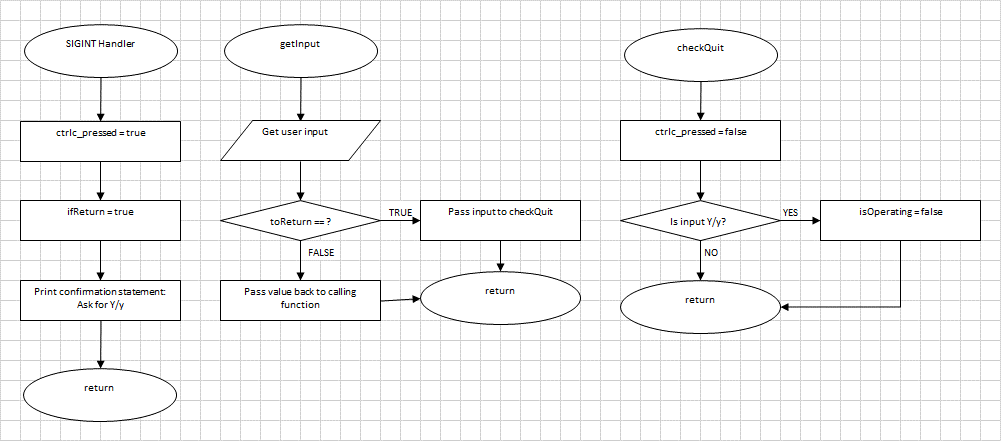


Figure 34: Flow Chart of signal handling

### 5.12.1 Description of signal handling

Scanf is a blocking function. So scanf is invoked, a user input is needed for the program to proceed. To overcome this issue, we replace scanf with a function getInput (which still uses scanf) and introduced a mechanism to “hijack” the getInput from its original caller, and use it to obtain quit confirmation from the user.

When SIGINT is caught, the Boolean ctrlc\_pressed and ifReturn becomes true. The signal handler will print out a confirmation message asking the user for permission to quit. Now, the original scanf function has been “hijacked” by signal handling process. Entering “Y” or “y” will lead to the function checkQuit. The checkQuit function will determine if a “Y” or “y” has been inputted.

If they are inputted, then the Boolean isOperating, which is the looping condition for all threads, will be set to false. Else, both checkQuit and getInput functions will terminate. The toReturn boolean will be used to abort the calling function of getInput. It will be true once SIGINT is raised.

The whole process, without the nitty gritty details, will look like this:

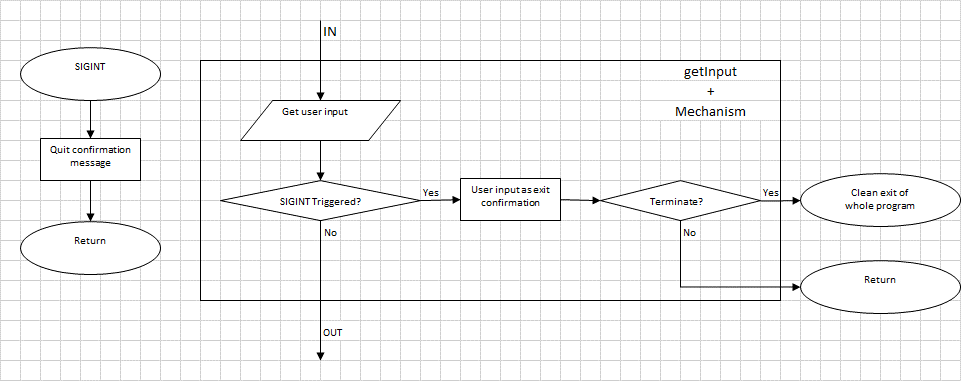


Figure 35: Flow Chart of signal handling

This mechanism will only work in our program, because of there will always be exactly one thread doing either of the following:

1. Getting input using getInput, which invokes scanf.
2. Monitoring the stdin, and return to the main menu once any button is pressed. The main menu will invoke getInput.

# Code Listing

1 /\* Program name: MA4830\_Waveform\_Generator.c

2

3 \* Author: Khoo Kar Keng (U1420494C)

4 Ng Ee Meng (U1420377G)

5 Joyce Lim Xin Yan (U1521972L)

6 Tan Jin Cai (U1421759E)

7 Joseph Ng Rong En (U1422346L)

8 Philip Puan Jun Kit (U1420409F)

9

10 \* Date of creation : 26/10/2017

11 \* Updated : 17/11/2017

12 \* Version : 1.3

13

14 \* This program generates waveform (sine, triangular, square) wave on PCI-DAS 1602

15 \* board. Maximum frequency achievable is 1750Hz while minimum and maximum output

16 \* voltages are -10 and 10V respectively.

17

18 \* Input variables: 1. Waveform type

19 \* 2. Frequency

20 \* 3. Mean

21 \* 4. Amplitude

22 \* 5. isOn flag

23

24 \* To change variables through arguments, the following order must be met:

25 \* Format: <program\_name.exe> <waveform type> <frequency> <mean> <amplitude> <isOn>

26 \* Waveform type: -sin =sine, -tri =triangular, -squ = square

27 \* e.g: ./wavegen -sin 1000.3 2.4 5 1

28

29 \* The user can change the DAC parameters (waveform properties) from keyboard

30 \* (MainUI) and switches & potentiometer (PeripheralInput). However, only one

31 \* device can change parameters at a time. PCI-DAS 1602's switches and ADC inputs

32 \* must be switched off before changing parameters through keyboard.

33

34 \* The ADC input ranges are:

35 \* 1. frequency - [2.67E-2, 1750]Hz

36 \* 2. amplitude - [0,9.8)V

37 \* 3. mean - (-9.8, 9,8)V

38

39 \* The keyboard input ranges are:

40 \* 1. frequency - (0, 1750)Hz

41 \* 2. amplitude - [0, 10)V

42 \* 3. mean - (-10, 10)V

43

44 \* However, in range checking, if the combined parameters gives

45 \* data not in range of (0, 1750)Hz and (-10,10)V (keyboard)/(-9.8, 9,8)V(ADC),

46 \* error message will show (keyboard) and parameter is not changed.

47

48 \* Description of each thread:

49 \* 1. Main thread - Initialise communications with PCI-DAS 1602 board,

50 \* create thread 2-4 and wait for them to finish

51 \* (pthread\_join) before quitting the program.

52 \* 2. MainUI thread - Interface with users with keyboard and display,

53 \* display real-time DAC and ADC status,

54 \* change DAC parameters from keyboard,

55 \* import and export DAC configuration, and

56 \* reset the isOperating flag if user quits

57 \* 3. PeripheralInput - Receive switches and potentiometers inputs,

58 \* convert data to correct forms,

59 \* changes the DAC parameters with range checking

60 \* 4. WaveGenManager - Supervises the isOn and resetWave flag to reconfigure

61 \* DAC data arrays and create thread

62 \* 5. PushDAC - Dedicated thread to output the data continuously,

63 \* thread exits when isOperating/isOn==false or resetWave==true

64

65 \* User can import and export the DAC configuration from and to .txt file

66 \* in command line argument format.

67 \*/

68 #include <stdio.h>

69 #include <stdlib.h>

70 #include <stdbool.h> //for boolean data type

71 #include <termios.h> //for tcischars();

72 #include <unistd.h>

73 #include <hw/pci.h>

74 #include <hw/inout.h>

75 #include <sys/neutrino.h>

76 #include <sys/mman.h>

77 #include <sys/types.h>

78 #include <process.h>

79 #include <pthread.h>

80 #include <math.h>

81

82 #define INTERRUPT iobase[1] + 0 // Badr1 + 0 : also ADC register

83 #define MUXCHAN iobase[1] + 2 // Badr1 + 2

84 #define TRIGGER iobase[1] + 4 // Badr1 + 4

85 #define AUTOCAL iobase[1] + 6 // Badr1 + 6

86 #define DA\_CTLREG iobase[1] + 8 // Badr1 + 8

87

88 #define AD\_DATA iobase[2] + 0 // Badr2 + 0

89 #define AD\_FIFOCLR iobase[2] + 2 // Badr2 + 2

90

91 #define TIMER0 iobase[3] + 0 // Badr3 + 0

92 #define TIMER1 iobase[3] + 1 // Badr3 + 1

93 #define TIMER2 iobase[3] + 2 // Badr3 + 2

94 #define COUNTCTL iobase[3] + 3 // Badr3 + 3

95 #define DIO\_PORTA iobase[3] + 4 // Badr3 + 4

96 #define DIO\_PORTB iobase[3] + 5 // Badr3 + 5

97 #define DIO\_PORTC iobase[3] + 6 // Badr3 + 6

98 #define DIO\_CTLREG iobase[3] + 7 // Badr3 + 7

99 #define PACER1 iobase[3] + 8 // Badr3 + 8

100 #define PACER2 iobase[3] + 9 // Badr3 + 9

101 #define PACER3 iobase[3] + 0xa // Badr3 + a

102 #define PACERCTL iobase[3] + 0xb // Badr3 + b

103

104 #define DA\_Data iobase[4] + 0 // Badr4 + 0

105 #define DA\_FIFOCLR iobase[4] + 2 // Badr4 + 2

106

107 #define PI acos(-1)

108

109 #define HIGHESTFREQ 1750

110 #define FIFO\_DELAY 6700 //Empirical value for delay of DAC push FIFO operation

111

112 #define THRESHOLD 30

113

114 // Struct for DAC waveform

115 typedef struct {

116 bool resetWave;

117 bool isOn;

118 const short identity;

119 unsigned short waveform\_type;

120 unsigned short data[1000];

121 unsigned short plus;

122 unsigned short DAC\_mode;

123 int samples\_per\_period;

124 float output\_res;

125 float mean;

126 float freq;

127 float amp;

128 }DACField ;

129

130 // Struct for intermediary field for changing global variables

131 typedef struct {

132 int waveform\_type;

133 float freq;

134 float mean;

135 float amp;

136 bool isOn;

137 }ChangeField ;

138

139 // PCI 2.2 assigns 6 IO base addresses

140 // PCI device global variables

141 int badr[5];

142 uintptr\_t iobase[5];

143

144 // Program global variables

145 bool isOperating=true; //boolean for program operation. False shutsdown the program.

146 bool ctrlc\_pressed=false; //boolean for SIGNINT. Used in checkQuit.

147 bool toReturn = false; //boolean for returning to MainUI(thread) after scanf. Used with Signal.

148 bool ADC\_Refresh = true; //boolean for refreshing ADC/GPIO display

149

150 // DACField struct global variables

151 DACField DAC={true, false, 0, 1, {}, 0, 1, 100, 0, 0, 1, 1};

152

153 // ADC global variables

154 uintptr\_t digital\_in;

155 uint16\_t adc\_in[2] = {};

156 uint16\_t old\_adc\_in[2] = {};

157

158

159 // Mutex (only one to change DAC variables)

160 pthread\_mutex\_t MainMutex = PTHREAD\_MUTEX\_INITIALIZER;

161

162 // Function Declaration for Housekeeping

163

164 // UI functionalities

165 void CLManager (int argc, char \*\*argv); //Manages Command line

166 void displayHelp(); //Display instructions for MainUI

167 void showDACConfig(); //Show current DAC configuration

168 void showADCStatus(); //Show ADC status

169 void importConfig(); //Import the configuration from .txt file

170 void exportConfig(); //Export the configuration to .txt file

171 void changeParam(); //Change the parameters of the DAC0

172 void stopOps(); //Stop operation of DAC (will turn on again if the switches are on)

173

174 // Quit signal

175 void checkQuit(char ch); //Reconfirm with user about quitting after SIGINT (Refer to function for more description)

176 void INThandler(int sig); //Signal handler for SIGINT

177

178 //Utilities

179 bool hasNegative();

180 short checkAbsMax(float mean, float amp); // Supporting function: check absolute maximum value

181 void change(bool onSignal, int wvty,

182 float f, float m, float a); // Function to directly change the parameters of the DAC

183 void setChangeField(ChangeField\* CF); //Set the change field to be equal to initial DAC parameters

184 void WaveformGen (); //Generate data for waveform

185 void chooseBestRes(); /\*Change the bipolar/unipolar mode based on mean and amplitude

186 to give best resolution\*/

187 long interval(struct timespec\* start, struct timespec\* end); // Used for calculating nanosec difference between timespec

188 void getInput(char\* in); //Get input from keyboard

189 int checkInput(char\* in); //Check the input validity in MainUI

190 float checkValidFloat(); //Check validity of floating point number

191 int checkValidInt(); //Check validity of integer

192

193 /\*\*\*\*\*\*\* Thread functions declaration \*\*\*\*\*\*\*/

194 // User Interface

195 void\* MainUI (void \*pointer); //General purpose thread for inputting from keyboard

196

197 // Peripherals (GPIO and ADC)

198 void\* PeripheralInputs(void \*pointer); //Thread for GPIO and ADC

199

200 // DAC

201 void\* WaveGenManager (void \* pointer); //Thread for managing waveform generating capabilities

202 void\* PushDAC (void\* Curr); //Thread to push-out data to DAC asynchronously

203

204

205 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

206 // Main function

207 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

208 int main(int argc, char\*\* argv) {

209 //PCI device variable

210 struct pci\_dev\_info info;

211 void \*hdl;

212 int rc;

213

214 uintptr\_t dio\_in;

215 uint16\_t adc\_in;

216

217 unsigned int i, count;

218 unsigned short chan;

219

220 pthread\_attr\_t attr;

221 pthread\_t thread[3];

222

223 // Invoke Signal

224 signal(SIGINT, INThandler);

225

226 // Call command line manager

227 CLManager (argc, argv);

228

229 // Set up the PCI

230 printf("\fSet-up Routine for PCI-DAS 1602\n\n");

231 memset(&info,0,sizeof(info));

232 if(pci\_attach(0)<0) {

233 perror("pci\_attach");

234 exit(EXIT\_FAILURE);

235 }

236

237 // Vendor and Device ID

238 info.VendorId=0x1307;

239 info.DeviceId=0x01;

240

241 // Attempt to attach to PCI device and populate info struct

242 if ((hdl=pci\_attach\_device(0, PCI\_SHARE|PCI\_INIT\_ALL, 0, &info))==0) {

243 perror("pci\_attach\_device");

244 exit(EXIT\_FAILURE);

245 }

246

247 // Determine assigned BADRn IO addresses for PCI-DAS1602

248 printf("\nDAS 1602 Base addresses:\n\n");

249 for(i=0;i<5;i++) {

250 badr[i]=PCI\_IO\_ADDR(info.CpuBaseAddress[i]);

251 printf("Badr[%d] : %x\n", i, badr[i]);

252 }

253

254 // Map I/O base address to user space

255 printf("\nReconfirm Iobase:\n");

256 for(i=0;i<5;i++) {

257 // Expect CPU Base Address to be the same as IO Base for PC

258 iobase[i]=mmap\_device\_io(0x0f,badr[i]);

259 printf("Index %d : Address : %x ", i,badr[i]);

260 printf("IOBASE : %x \n",iobase[i]);

261 }

262

263 // Modify thread control privity

264 if(ThreadCtl(\_NTO\_TCTL\_IO,0)==-1) {

265 perror("Thread Control");

266 exit(1);

267 }

268

269 // ADC write register

270 out16(INTERRUPT, 0x60c0);

271 out16(TRIGGER, 0x2081);

272 out16(AUTOCAL, 0x007f);

273 out16(AD\_FIFOCLR, 0);

274 out16(MUXCHAN, 0x0D00);

275

276 // Create joinable attribute

277 pthread\_attr\_init(&attr);

278 pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

279

280 /\*

281 Create three threads:

282 1. Waveform Manager

283 2. Keyboard I/O

284 3. ADC and Switches

285 \*/

286 delay(1500);

287 system("clear");

288

289 for(i=0;i<3;i++){

290 switch (i){

291 case 0:{rc = pthread\_create(&thread[i], &attr, &WaveGenManager, NULL);

292 break;}

293 case 1:{rc = pthread\_create(&thread[i], &attr, &MainUI, NULL);

294 break;}

295 case 2:{rc = pthread\_create(&thread[i], &attr, &PeripheralInputs, NULL);

296 break;}

297 }

298 if (rc){

299 printf("pthread\_create() #%d return error! Code %d\n", rc);

300 exit(-1);

301 }

302 }

303 pthread\_attr\_destroy(&attr);

304

305 // Joining MainUI input

306 for(i=0;i<3;i++){

307 rc = pthread\_join(thread[i], NULL);

308 if (rc){

309 printf("pthread\_join() #%d return error! Code %d\n", i, rc);

310 exit(-1);

311 }

312 }

313

314 // Exit message and clean screen

315 printf("Process quitting in 1 second...");

316 fflush(stdout);

317 sleep(1);

318 system("clear");

319 pci\_detach\_device(hdl);

320 return 0;

321 }

322

323 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

324 // Main Program Functionalities

325 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

326 //Display help menu

327 void displayHelp() {

328 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

329 printf("Help menu\n");

330 printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

331 printf("%\*s\t\t%s", 6, "1", "Show current DAC0 configurations.\n");

332 printf("%\*s\t\t%s", 6, "2" ,"Show current ADCs' statuses.\n");

333 printf("%\*s\t\t%s", 6, "3", "Change the configurations.\n");

334 printf("%\*s\t\t%s", 6, "4", "Export configurations or waveforms.\n");

335 printf("%\*s\t\t%s", 6, "5", "Import configurations.\n");

336 printf("%\*s\t\t%s", 6, "6", "Halt DAC operation "

337 "(must turn off peripheral input).\n");

338 printf("%\*s\t\t%s", 6, "7", "Exit program\n");

339 printf("\nFriendly reminder: please turn off peripheral input\n"

340 "before changing any variable through keyboard.\n");

341 printf("\nPlease enter your command: ");

342 }

343 //Get input from keyboard

344 void getInput(char\* in) {

345 // Flush out any output buffer (printf)

346 fflush(stdout);

347 // Flush to ensure input buffer is cleared

348 fflush(stdin);

349 scanf("%s", in);

350 // Clear input buffer as a safe practice

351 fflush(stdin);

352 // Check whether CTRL+C is pressed

353 if(toReturn) {

354 checkQuit(in[0]);

355 }

356 return;

357 }

358 //Check the input validity in MainUI

359 int checkInput(char\* in) {

360 int temp=0;

361 char\* endptr;

362 temp = strtol(in, &endptr, 10);

363 // check if valid integer is inputted

364 if(\*endptr == '\0'){

365 if(temp>0 && temp <8)

366 return temp;

367 }

368 else

369 printf("Invalid input! Inputted:%s\n", in);

370 return 0;

371 }

372 //Show current DAC configuration

373 void showDACConfig(){

374 printf("%\*s\n", 38, "DAC0");

375 printf("%\*s%\*d\n", 25,

376 "Running? (0-OFF, 1-ON)", 15, DAC.isOn);

377 printf("%\*s", 25, "Waveform type");

378 switch(DAC.waveform\_type){

379 case 1: { printf("%\*s", 15, "Sinusoidal"); break;}

380 case 2: { printf("%\*s", 15, "Triangular"); break;}

381 case 3: { printf("%\*s", 15, "Square"); break;}

382 }

383 printf("\n");

384 printf("%\*s%\*d\n", 25,

385 "Samples per period", 15, DAC.samples\_per\_period);

386 printf("%\*s%\*.2E\n", 25,

387 "DAC Output resolution (V)", 15, DAC.output\_res/1000000);

388 printf("%\*s%\*.2f\n", 25, "Frequency (Hz)", 15, DAC.freq);

389 printf("%\*s%\*.2f\n", 25, "Amplitude (V)", 15, DAC.amp);

390 printf("%\*s%\*.2f\n", 25, "Mean (V)", 15, DAC.mean);

391 return;

392 }

393 //Show ADC status

394 void showADCStatus(){

395 bool showDAC = false;

396 char input[5];

397 char in, key;

398 /\* Ask user whether to show DAC configuration to

399 supervise real-time changes on DAC \*/

400 printf("\nDo you want to show DAC config? (Y/N): ");

401 getInput(&input[0]);

402 /\* Return to MainUI if CTRL+C is pressed (same applies for

403 subsequent similar statements) \*/

404 if(toReturn)

405 return;

406 if (input[0] == 'y' || input[0] == 'Y') {

407 showDAC = true;

408 }

409 else if (!(input[0] == 'n' || input[0] == 'N')) {

410 printf("Invalid choice. DAC config will not show.\n");

411 }

412 // While to show ADC settings (optional: DAC settings)

413 while (1){

414 /\* Break loop if any key is pressed (tcischars detects number

415 of characters waiting to be read from the therminal \*/

416 if(tcischars(1)>0){

417 fflush(stdin);

418 printf("\f");

419 break;

420 }

421 delay(1000);

422 /\* Get user's confirmation character and return to

423 MainUI if CTRL+C is pressed \*/

424 if(toReturn){

425 getInput(&input[0]);

426 return;

427 }

428 if(ADC\_Refresh){

429 // Switches input and descriptions

430 printf("\f%\*s\nDescriptions\n", 50, "Digital input");

431 printf("Bit 3 --> 0 = Keyboard control only ");

432 printf("1 = Keyboard and peripheral inputs\n");

433 printf("Bit 2 --> P0/ADC0 changes: 0 = mean 1 = amplitude\n");

434 printf("%\*s", 14, "Bit 1 & 0 --> ");

435 printf("%-\*s\n", 26, "0x00 - DAC0 is off");

436 printf("%\*s%-\*s\n", 14, "", 40, "0x01 - Sine wave");

437 printf("%\*s%-\*s\n", 14, "", 40, "0x02 - Triangular wave");

438 printf("%\*s%-\*s\n", 14, "", 40, "0x03 - Square wave");

439 printf("\nPort A Bit\t\t3\t\t2\t\t1\t\t0\n");

440 printf("\t\t\t%1d", (digital\_in&0x08) >> 3);

441 printf("\t\t%1d", (digital\_in&0x04)>> 2);

442 printf("\t\t%1d", (digital\_in&0x02) >> 1);

443 printf("\t\t%1d\n", (digital\_in&0x01));

444 // ADC input

445 printf("\n\n%\*s\n", 38, "ADC Value (16 bit - Hex)");

446 printf("%\*s\t\t%04X\n", 10, "ADC0 ", adc\_in[0]);

447 printf("%\*s\t\t%04X\n", 10, "ADC1 ", adc\_in[1]);

448 // DAC settings

449 if (showDAC){

450 printf("\n\n\n");

451 showDACConfig();

452 }

453 printf("Press any key to quit the function\n");

454 ADC\_Refresh = false;

455 }

456 }

457 }

458 //Import the configuration from .txt file

459 void importConfig(){

460 int i=0;

461 int j=1;

462 bool isNull=false;

463 FILE\* fd;

464 char filename[30];

465 const char source[10]="imported";

466 char input100[100]={'\0'};

467 char \*input[12];

468 printf("Warning: Please turn off peripheral control before importing!\n");

469 printf("Please enter filename (\".txt\" is added at the end): ");

470 getInput(&filename[0]);

471 if(toReturn)

472 return;

473 strcat(filename, ".txt");

474 // Open file to read, return if fails

475 fd = fopen(filename,"r+");

476 if(fd == NULL){

477 printf("Failed to open %s.\n", filename);

478 return;

479 }

480 /\* Read the file character by character

481 If " " or "\n" is found, replace it by EOF.

482 Input the address of first character of each word

483 to char pointer array \*/

484 input[0]=(char \*)&source;

485 input[j]=&input100[0];

486 while(1){

487 if(isNull){

488 input[++j]=&input100[i];

489 isNull=false;

490 }

491 if(fscanf(fd, "%c", &input100[i])==EOF)

492 break;

493 if(input100[i]=='\n' || input100[i]==' '){

494 input100[i]='\0';

495 isNull=true;

496 }

497 i++;

498 }

499 /\* Use CLManager to read imported settings

500

501 input[0] = "imported"

502 input[i] = respective string

503

504 Use of mutex when changing shared global variables

505 \*/

506 pthread\_mutex\_lock(&MainMutex);

507 CLManager(j, (char\*\*) &input);

508 pthread\_mutex\_unlock(&MainMutex);

509 // Close file and print confirmation message

510 fflush(fd);

511 fclose(fd);

512 printf("\nConfiguration from %s is loaded.\n", filename);

513 sleep(1);

514 return;

515 }

516 //Export the configuration to .txt file

517 void exportConfig(){

518 FILE\* fd;

519 bool printConfig =false, printData = false;

520 char filename[30];

521 int i;

522 printf("Please enter filename (\".txt\" is added at the end): ");

523 getInput(&filename[0]);

524 if(toReturn) return;

525 strcat(filename, ".txt");

526 // Open file (buffered write)

527 fd = fopen(filename,"w+");

528 if(fd == NULL){

529 printf("Failed to open/create %s.\n", filename);

530 return;

531 }

532 // Display filename and export options

533 printf("\fFilename: %s\n", filename);

534 printf("Options:\n");

535 printf("1 - Export DAC configuration only (importable)\n");

536 printf("2 - Export waveform data only (unimportable)\n");

537 printf("3 - Export both (unimportable)\n");

538 printf("Enter option: ");

539 // Check options

540 switch(checkValidInt()){

541 case 1: {printConfig = true; break;}

542 case 2: {printData = true; break;}

543 case 3: {printConfig = true; printData = true; break;}

544 default:{if(toReturn) return;

545 printf("Invalid choice. File is not written.\n");}

546 }

547 // Printing configuration only

548 if(printConfig){

549 // Print setting in command line format

550 switch(DAC.waveform\_type){

551 case 1:{fprintf(fd, "%s ", "-sin"); break;}

552 case 2:{fprintf(fd, "%s ", "-tri"); break;}

553 case 3:{fprintf(fd, "%s ", "-squ"); break;}

554 }

555 fprintf(fd, "%.2f %.2f %.2f %d\n",

556 DAC.freq, DAC.mean, DAC.amp, DAC.isOn);

557 }

558 // Printing waveform data only

559 if(printData){

560 fprintf(fd, "\n Data number\t\tDAC value(Hex)\t\tReal value(V)\n\n");

561 for(i=0;i<DAC.samples\_per\_period;i++)

562 fprintf(fd, "\t%d\t\t%04x\t\t\t%.4f\n", i+1, DAC.data[i],

563 (DAC.data[i]-(int)(DAC.DAC\_mode < 2 ? 0x7FFF : 0))\*(DAC.output\_res/1000000));

564 }

565 // Close file and print confirmation message

566 fflush(fd);

567 fclose(fd);

568 printf("Configuration saved to file.\n");

569 return;

570 }

571 //Check validity of floating point numbers

572 float checkValidFloat(){

573 char input[12];

574 char\* pointer;

575 float temp=0;

576 getInput(&input[0]);

577 if(toReturn)

578 return -100;

579 // Convert string to double

580 temp = strtod(input, &pointer);

581 // Return the number if valid

582 if(\*pointer=='\0')

583 return temp;

584 // Return -100 if invalid

585 else

586 printf("Invalid floating point number!\n");

587 return -100;

588 }

589 // Check validity of integer

590 int checkValidInt(){

591 char input[5];

592 char\* pointer;

593 float temp=0;

594 getInput(&input[0]);

595 if(toReturn)

596 return -1;

597 // Convert string to long

598 temp = strtol(input, &pointer, 10);

599 // Return the number if valid, else return -1

600 if(\*pointer=='\0')

601 return temp;

602 return -1;

603 }

604 //Change the parameters of the DAC0

605 void changeParam(){

606 ChangeField CField;

607 bool isRepeat=false;

608 bool hasChanged = false;

609 char repeatchar;

610 char input[5];

611 int select1=0;

612 int select2=0;

613 float temp;

614 do{

615 // Load CField with current DAC values

616 setChangeField(&CField);

617 // Display options

618 printf("\nCurrent OFF/ON (0/1) status: %d\n", CField.isOn);

619 printf("Select option:\n");

620 printf("1 - Change the waveform type of DAC[0]\n");

621 printf("2 - Change the frequency of DAC[0]\n");

622 printf("3 - Change the mean of DAC[0]\n");

623 printf("4 - Change the amplitude of DAC[0]\n");

624 printf("5 - Change the OFF/ON (0/1) status of DAC[0]\n");

625 printf("6 - Reset the settings to default\n");

626 printf("Enter option: ");

627 // Integer validity check

628 if((select1=checkValidInt())>=0){

629 switch (select1){

630 // Option 1: change waveform

631 case 1: {

632 printf("\nChanging waveform type of DAC[0]\n");

633 printf("Enter 1 for sinusoidal waveform\n");

634 printf("Enter 2 for triangular waveform\n");

635 printf("Enter 3 for square waveform\n");

636 switch(DAC.waveform\_type){

637 case 1: printf("Current waveform (V): sinusoidal\n"); break;

638 case 2: printf("Current waveform (V): triangular\n"); break;

639 case 3: printf("Current waveform (V): square\n"); break;

640 }

641 printf("Enter option: ");

642 // Integer validity check

643 if((select2=checkValidInt())>=0)

644 // Selection validity check

645 if(select2>0 && select2<4){

646 hasChanged = true;

647 CField.waveform\_type=select2;

648 printf("\nChanged waveform type of DAC[0]\n");

649 }

650 else

651 printf("Invalid waveform selection.\n");

652 else{

653 if(toReturn) return;

654 printf("Invalid input for waveform. (Must be integer)");

655 }

656 break;

657 }

658 // Option 2: change frequency

659 case 2: {

660 printf("\nChanging frequency (float) of DAC[0]\n");

661 printf("Frequency must be in the range (0,%d) Hz\n", HIGHESTFREQ);

662 printf("Current frequency (Hz): %.2f\n", DAC.freq);

663 printf("Enter frequency (Hz): ");

664 // Range checking

665 if((temp=checkValidFloat())>0 && temp < HIGHESTFREQ){

666 hasChanged = true;

667 CField.freq=temp;

668 printf("\nChanged freq of DAC[0]\n");

669 }

670 else{

671 if(toReturn) return;

672 printf("Error: Frequency is not changed.\n");

673 }

674 break;

675 }

676 // Option 3: change mean

677 case 3: {

678 printf("\nChanging mean (float) of DAC[0]\n");

679 printf("Mean value must be in the range (%.2f,%.2f)\n",

680 -(10-CField.amp), 10-CField.amp);

681 printf("Current mean (V): %.2f\n", DAC.mean);

682 printf("Enter mean (V): ");

683 if((temp=checkValidFloat())>= -10)

684 // Range checking

685 if(checkAbsMax(temp, CField.amp)!=0){

686 hasChanged = true;

687 CField.mean=temp;

688 printf("\nChanged mean of DAC[0]\n");

689 }

690 else{

691 if(toReturn) return;

692 printf("Mean value is not changed.\n");

693 }

694 break;

695 }

696 // Option 4: change amplitude

697 case 4: {

698 printf("\nChanging amplitude (float) of DAC[0]\n");

699 printf("Amplitude value must be in the range (0,%.2f)\n"

700 , 10-CField.mean);

701 printf("Current amplitude (V): %.2f\n", DAC.amp);

702 printf("Enter amplitude (V): ");

703 if((temp=checkValidFloat())>= 0)

704 // Range checking

705 if(checkAbsMax(temp, CField.amp)!=0){

706 hasChanged = true;

707 CField.amp=temp;

708 printf("\nChanged amplitude of DAC[0]\n");

709 }

710 else{

711 if(toReturn) return;

712 printf("Amplitude value is not changed.\n");

713 }

714 break;

715 }

716 // Option 5: change isOn flag

717 case 5: {

718 printf("\nChanging OFF/ON (0/1) state of the DAC[0]\n");

719 printf("DAC is currently:");

720 DAC.isOn ? printf("On\n") : printf("Off\n");

721 printf("Enter option (0/1): ");

722 if((select2=checkValidInt())>=0)

723 if(select2==0)

724 if (CField.isOn==false)

725 printf("DAC[0] is already off.\n");

726 else

727 {

728 hasChanged = true;

729 CField.isOn=false;

730 }

731 else if(select2==1)

732 if (CField.isOn==false)

733 {

734 hasChanged = true;

735 CField.isOn=true;

736 }

737 else

738 printf("DAC[0] is already on.\n");

739 else{

740 if(toReturn) return;

741 printf("Invalid options for OFF/ON.\n");

742 }

743 break;

744 }

745 // Option 6: restore default values

746 case 6: {

747 printf("\nResetting values to default values.\n");

748 printf("Sine wave, freq = 1 Hz, mean = 0V, amplitude = 1V, isOn = 0\n");

749 hasChanged = true;

750 CField.waveform\_type=1;

751 CField.freq=1;

752 CField.mean=0;

753 CField.amp=1;

754 CField.isOn=false;

755 break;

756 }

757 default: {

758 if(toReturn) return;

759 printf("\nInvalid choice.\n");

760 }

761 }

762 // Use of mutex when changing shared global variables

763 if(hasChanged){

764 if(select1<7 && select1>0){

765 pthread\_mutex\_lock(&MainMutex);

766 change(CField.isOn, CField.waveform\_type,

767 CField.freq, CField.mean, CField.amp);

768 pthread\_mutex\_unlock(&MainMutex);

769 // Reset hasChanged flag

770 hasChanged = false;

771 }

772 // Wait for WaveGenManger to finish

773 delay(50);

774 }

775 }

776 if(toReturn) return;

777 // Ask user whether to repeat changeParam

778 printf("\nDo you want to change other parameters? (Y/N): ");

779 getInput(&input[0]);

780 if(toReturn) return;

781 if(input[0]=='y' ||input[0]=='Y')

782 isRepeat=true;

783 else if (input[0]=='n' ||input[0]=='N')

784 isRepeat=false;

785 else{

786 printf("Invalid choice. Returning to main UI.\n");

787 isRepeat=false;

788 }

789 printf("\f");

790 } while(isRepeat);

791 return;

792 }

793 //Stop operation of DAC(will turn on again if the switches are on)

794 void stopOps(){

795 // Use of mutex when changing shared global variables

796 pthread\_mutex\_lock(&MainMutex);

797 DAC.isOn=false;

798 pthread\_mutex\_unlock(&MainMutex);

799 return;

800 }

801 //The general purpose thread for inputting from keyboard

802 void\* MainUI (void \*pointer){

803 char input[10];

804 sleep(1); // Wait for WaveGenManager to finish its configuration

805 while (1) {

806 // Exit thread if isOperating is false

807 if(!isOperating)

808 pthread\_exit(NULL);

809 // Reset toReturn flag only when program reaches MainUI

810 if(ctrlc\_pressed==false)

811 toReturn = false;

812 delay(100);

813 displayHelp();

814 getInput(&input[0]);

815 // Continue loop if CTRL+C is pressed

816 if(toReturn)

817 continue;

818 printf("\f");

819 switch (checkInput(&input[0])) {

820 // case 1 - print current devices' configurations.

821 case 1: { showDACConfig(); break; }

822 // case 2 - print current devices' statuses.

823 case 2: { showADCStatus(); break; }

824 // case 3 - change the configurations.

825 case 3: { changeParam(); break; }

826 // case 4 - export configurations or waveforms.

827 case 4: { exportConfig(); break; }

828 // case 5 - import configurations.

829 case 5: { importConfig(); break; }

830 // case 6 - halt all operations.

831 case 6: { stopOps(); break; }

832 // case 7 - quit the program

833 case 7: { isOperating = false; break; }

834 //show error in input

835 default:{ printf("Invalid character. Please reenter. \n");}

836 }

837 }

838 return 0;

839 }

840

841 // Check if scanf has pickup a quit confirmation after signal

842 void checkQuit(char ch){

843 ctrlc\_pressed=false;

844 if (ch == 'y' || ch== 'Y'){

845 isOperating=false;

846 }

847 else printf("\f");

848 return;

849 }

850 /\* SIGINT handler function

851

852 When CTRL+C is pressed, INThandler is started.

853 INThandler sets toReturn and ctrlc\_pressed flags.

854 Confirmation to quit is asked and INTHandler quits.

855

856 Meanwhile, MainUI stops at getInput (waits user

857 input).

858

859 getInput will receive input regardless of said

860 flags settings. After receiving input, toReturn

861 flag is checked.

862

863 If true, getInput will run checkQuit for the

864 inputted character. If not, getInput progresses

865 as per normal.

866 \*/

867 void INThandler(int sig){

868 ctrlc\_pressed=true;

869 toReturn = true;

870 printf("\nYou pressed CTRL+C. "

871 "Do you really want to quit? (Y/y to quit): \n");

872 }

873

874 // Manages arguments

875 void CLManager (int argc, char \*\*argv){

876 int counter, temp2;

877 float temp;

878 ChangeField CField;

879 char\* endptr;

880 // Load CField with default DAC values

881 setChangeField(&CField);

882 // Argument checking (from argv[1])

883 for(counter=1;counter<argc;counter++){

884 // Check argv[1] - waveform type

885 if((counter%5) ==1){

886 if(strcmp(argv[counter],"-sin") == 0)

887 CField.waveform\_type=1;

888 else if(strcmp(argv[counter],"-tri") == 0)

889 CField.waveform\_type=2;

890 else if(strcmp(argv[counter],"-squ") == 0)

891 CField.waveform\_type=3;

892 else

893 printf("Invalid waveform.\n");

894 }

895 // Check argv[2] to argc[4]

896 else if ((counter%5) !=0){

897 // Convert string to double

898 temp = strtod(argv[counter], &endptr);

899 // Continue if valid

900 if(\*endptr == '\0'){

901 /\* Check argv[2] - freq, argv[3] - mean

902 and argv[4] - amplitude \*/

903 switch(counter%5){

904 case 2: {

905 // Range checking for all parameters

906 if(temp>0 && temp <HIGHESTFREQ)

907 CField.freq = temp;

908 else

909 printf("Frequency must be in the range (0, %d) Hz\n", HIGHESTFREQ);

910 break;

911 }

912 case 3: {

913 if(checkAbsMax(temp, CField.amp)!=0)

914 CField.mean = temp;

915 else

916 printf("Error: Out of range mean value: %.2f V\n", temp);

917 break;

918 }

919 case 4: {

920 if(checkAbsMax(CField.amp, temp)!=0)

921 CField.amp = temp;

922 else

923 printf("Error: Out of range amplitude value: %.2f V\n", temp);

924 break;

925 }

926 }

927 }

928 else

929 // Show invalid input (not double), if any

930 printf("Invalid input #%d: %s\n", counter, argv[counter]);

931 }

932 // Check argv[5] - isOn status

933 else {

934 temp2 = strtol(argv[counter], &endptr, 2);

935 if(temp2==1)

936 change(true, CField.waveform\_type, CField.freq, CField.mean, CField.amp);

937 else

938 change(false, CField.waveform\_type, CField.freq, CField.mean, CField.amp);

939 }

940 }

941 // CLManager ending message

942 printf("Ending command line manager function...\n");

943 sleep(1);

944 return;

945 }

946

947

948 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

949 // DAC parts

950 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

951 // Change the bipolar/unipolar mode based on mean and amplitude

952 // to give best resolution

953 void chooseBestRes(){

954 // Data has negative value(s)

955 if(hasNegative()){

956 // Absolute maximum <5V

957 if(checkAbsMax(DAC.mean, DAC.amp)==1){

958 DAC.plus=(0x0000<<DAC.identity);

959 DAC.DAC\_mode = 0;

960 DAC.output\_res=152.59;

961 }

962 // Absolute maximum <10V

963 else if (checkAbsMax(DAC.mean, DAC.amp)==2){

964 DAC.plus=(0x0100<<DAC.identity);

965 DAC.DAC\_mode = 1;

966 DAC.output\_res=305.14;

967 }

968 }

969 // Data has no negative value

970 else{

971 if(checkAbsMax(DAC.mean, DAC.amp)==1){

972 DAC.plus=(0x0200<<DAC.identity);

973 DAC.DAC\_mode = 2;

974 DAC.output\_res=76.29;

975 }

976 else if (checkAbsMax(DAC.mean, DAC.amp)==2){

977 DAC.plus=(0x0300<<DAC.identity);

978 DAC.DAC\_mode = 3;

979 DAC.output\_res=152.59;

980 }

981 }

982 return;

983 }

984 // Generate data for waveform

985 void WaveformGen (){

986 int i=0;

987 int z=0;

988 double delta\_incr, dummy, res;

989 unsigned short offset=0;

990 /\*

991 value = mean + x

992 sine wave : x= amp\*sin(2\*PI\*freq\*t)

993 triangular wave : x= (4\*amp/T)\*t (0<t<T/4)

994 x= 2\*amp-(4\*amp/T)\*t (T/4<t<3\*T/4)

995 x= -4\*amp+(4\*amp/T)\*t (3\*T/4<t<T)

996 square wave : x= amp (0<t<T/2)

997 x= -amp (T/2<t<T)

998 \*/

999 // Choose unipolar/bipolar DAC mode

1000 chooseBestRes();

1001 // Convert resolution to unit of V

1002 res = DAC.output\_res/1000000;

1003 // Add offset for bipolar mode

1004 if(DAC.DAC\_mode<2)

1005 offset+=0x7FFF;

1006 switch (DAC.waveform\_type){

1007 case 1: {// Sine wave waveform creation

1008 delta\_incr=2.0\*PI/DAC.samples\_per\_period; // increment

1009 for(i=0;i<DAC.samples\_per\_period;i++) {

1010 dummy= (sinf((float)(i\*delta\_incr)))\* DAC.amp + DAC.mean;

1011 dummy= offset + dummy/res;

1012 DAC.data[i]= (unsigned short) dummy;

1013 }

1014 break;

1015 }

1016 case 2: {// Triangular wave waveform creation

1017 delta\_incr=4\*DAC.amp/DAC.samples\_per\_period; // increment

1018 for(i=0;i<DAC.samples\_per\_period/4;i++) {

1019 dummy= delta\_incr\*i +DAC.mean;

1020 dummy= offset + dummy/res;

1021 DAC.data[i]= (unsigned short) dummy;

1022 }

1023 for(;i<(3\*DAC.samples\_per\_period/4);i++) {

1024 dummy= 2\*DAC.amp-delta\_incr\*i +DAC.mean;

1025 dummy= offset + dummy/res;

1026 DAC.data[i]= (unsigned short) dummy;

1027 }

1028 for(;i<DAC.samples\_per\_period;i++) {

1029 dummy= -4\*DAC.amp+delta\_incr\*i +DAC.mean;

1030 dummy= offset + dummy/res;

1031 DAC.data[i]= (unsigned short) dummy;

1032 }

1033 break;

1034 }

1035 case 3: {// Square wave waveform creation

1036 for(i=0;i<DAC.samples\_per\_period/2;i++) {

1037 dummy= DAC.amp + DAC.mean;

1038 dummy= offset + dummy/res;

1039 DAC.data[i]= (unsigned short) dummy;

1040 }

1041 for(;i<DAC.samples\_per\_period;i++) {

1042 dummy= -DAC.amp + DAC.mean;

1043 dummy= offset + dummy/res;

1044 DAC.data[i]= (unsigned short) dummy;

1045 }

1046 break;

1047 }

1048 }

1049 // Reset resetWave flag after finishing configuration

1050 DAC.resetWave=false;

1051 return;

1052 }

1053 // Used for calculating nanosec difference between timespec

1054 long interval(struct timespec\* start, struct timespec\* end){

1055 long temp = end->tv\_nsec - start->tv\_nsec;

1056 if(temp > 0) return temp;

1057 else return (temp + 1000000000);

1058 }

1059

1060 // Function to push-out data to DAC(thread function)

1061 void\* PushDAC (void\* Curr){

1062 // Obtained struct pointer from pthread\_create

1063 DACField\* Current = (DACField\*) Curr;

1064 struct timespec time\_start, time\_end;

1065 int i, delay\_time;

1066 unsigned short CTLREG\_content;

1067 long nanospin\_time;

1068 // Configure the DAC CTRL register data values

1069 CTLREG\_content=(unsigned short)((\*Current).plus+((\*Current).identity+0x1)\*0x20+0x3);

1070 nanospin\_time = (long)(1000000000.0/(Current->freq\*Current->samples\_per\_period));

1071 // While loop to push out data

1072 while (1){

1073 for(i=0;i<(\*Current).samples\_per\_period;i++) {

1074 // Exit thread if isOperating or isOn ==false, or when resetWave==true \*/

1075 if(!((\*Current).resetWave==false && (\*Current).isOn==true) || !isOperating)

1076 pthread\_exit(NULL);

1077 out16(DA\_CTLREG, CTLREG\_content); // Write setting to DAC CTLREG

1078 out16(DA\_FIFOCLR, 0); // Clear DA FIFO buffer

1079 out16(DA\_Data, (\*Current).data[i]); // Output data

1080 // Use nanospin if sleep time is below 5ms, else use clock and delay.

1081 if(nanospin\_time < 5 \* 1000000) nanospin\_ns(nanospin\_time - FIFO\_DELAY);// Busy wait

1082 else{

1083 clock\_gettime(CLOCK\_REALTIME, &time\_start);

1084 delay\_time = (nanospin\_time/1000000) - 2;

1085 delay(delay\_time);

1086 clock\_gettime(CLOCK\_REALTIME, &time\_end);

1087 nanospin\_ns(nanospin\_time - FIFO\_DELAY - interval(&time\_start, &time\_end));

1088 }

1089 }

1090 }

1091 return (0);

1092 }

1093 //Thread for managing waveform generating capabilities

1094 void\* WaveGenManager (void \* pointer){

1095 pthread\_t tid;

1096 while(1){

1097 delay(100);

1098 if(!isOperating){

1099 pthread\_exit(NULL);

1100 DAC.isOn = false;

1101 }

1102 // Continue checking if PushDAC needs no change

1103 if(DAC.isOn==false)

1104 continue;

1105 else {

1106 if(DAC.resetWave==false)

1107 continue;

1108 /\* Create a thread if DAC setting is changed, PushDAC

1109 automatically dies (see above descriptions)\*/

1110 else{

1111 // Use to Mutex when changing shared global variables

1112 pthread\_mutex\_lock(&MainMutex);

1113 // Set up data field for DAC

1114 WaveformGen();

1115 pthread\_mutex\_unlock(&MainMutex);

1116 // Create thread

1117 pthread\_create(&tid, NULL, &PushDAC, (void \*)&DAC);

1118 }

1119 }

1120 }

1121 }

1122

1123 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

1124 // Input manager for switches and analogue inputs

1125 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

1126 void \* PeripheralInputs(void \*pointer){

1127 bool isOn = false;

1128 bool hasChanged = false;

1129 unsigned short mean\_amp=0;

1130 unsigned short count = 0x00;

1131 unsigned short chan = 0x00;

1132 unsigned short wavef = 1;

1133 bool digital\_in\_old;

1134 float temp;

1135 ChangeField CField;

1136 ADC\_Refresh = true;

1137 while(1){

1138 // Exit thread if isOperating is false

1139 if(!isOperating)

1140 pthread\_exit(NULL);

1141 // Load the CField with default DAC values

1142 setChangeField(&CField);

1143 temp=0;

1144 count = 0x00;

1145 // Delay to allow DAC to work better

1146 delay(100);

1147 // Port A : Input, Port B : Output, Port C (upper | lower) : Output | Output

1148 out8(DIO\_CTLREG,0x90);

1149

1150 // Read Port A

1151 digital\_in =in8(DIO\_PORTA);

1152 // Output Port A value -> write to Port B (LEDs)

1153 out8(DIO\_PORTB, digital\_in);

1154

1155 // Read potentiometers

1156 while (count < 0x02) {

1157 chan = ((count & 0x0f) << 4) | (0x0f & count);

1158 out16(MUXCHAN, 0x0D00 | chan); // Set channel - burst mode off

1159 delay(1); // Allow mux to settle

1160 out16(AD\_DATA, 0); // Start ADC

1161 while (!(in16(MUXCHAN) & 0x4000)); // Wait until the data is filled

1162 adc\_in[count] = in16(AD\_DATA);

1163 count++;

1164 }

1165

1166 // Remove unneeded bits

1167 digital\_in = digital\_in & 0x0f;

1168

1169 // Check if switch configuration has changed

1170 if(digital\_in != digital\_in\_old) ADC\_Refresh = true;

1171

1172 digital\_in\_old = digital\_in;

1173

1174 // Continue if the peripheral input is turned off

1175 if (!(digital\_in & 0x08)) continue;

1176

1177 else {

1178 // Assume the DAC to be on

1179 isOn = true;

1180 switch ((digital\_in & 0x03)) {

1181 case 1: { wavef = 1; break; }

1182 case 2: { wavef = 2; break; }

1183 case 3: { wavef = 3; break; }

1184 // Set isOn to be off is bit 0 and 1 are off

1185 default: { isOn = false; break; }

1186 }

1187 /\* Only set the hasChanged flag if the current

1188 configuration is different from the previous one \*/

1189 if (CField.waveform\_type != wavef) {

1190 CField.waveform\_type = wavef;

1191 hasChanged = true;

1192 }

1193 if (CField.isOn != isOn) {

1194 CField.isOn = isOn;

1195 hasChanged = true;

1196 }

1197 // Get the bit value of bit 2

1198 mean\_amp = ((digital\_in & 0x04) >> 2);

1199 /\*

1200 ADC[0] - for mean and amplitude

1201 - check whether the difference in new & old values is

1202 more than threshold before trying to change

1203 \*/

1204 if (abs((int)adc\_in[0] - (int)old\_adc\_in[0]) > THRESHOLD) {

1205 // Change amplitude if bit 2 is set

1206 if (mean\_amp == 1) {

1207 temp = (float)(adc\_in[0]) \* 10 / 65535;

1208 // Range checking

1209 if (fabs(DAC.mean + temp) < 9.8 && fabs(DAC.mean - temp) < 9.8) {

1210 CField.amp = temp;

1211 hasChanged = true;

1212 }

1213 }

1214 // Change mean if bit 2 is not set

1215 else {

1216 temp = (float)(adc\_in[0]) \* 10 / 32767 - 10;

1217 // Range checking

1218 if (fabs(DAC.amp + temp) < 9.8 && fabs(DAC.amp - temp) < 9.8) {

1219 CField.mean = temp;

1220 hasChanged = true;

1221 }

1222 }

1223 }

1224 /\*

1225 ADC[1] - dedicated for frequency

1226 - check whether the difference in new & old values is

1227 more than threshold before trying to change

1228 \*/

1229 if (abs((int)adc\_in[1] - (int)old\_adc\_in[1])>THRESHOLD) {

1230 // Minimum input required (> 0x01)

1231 if (adc\_in[1] > 0x0001) {

1232 CField.freq = (float)(adc\_in[1]) \* HIGHESTFREQ / 65535;

1233 hasChanged = true;

1234 }

1235 else{

1236 CField.freq = (float)(0x0001) \* HIGHESTFREQ / 65535;

1237 hasChanged = true;

1238 }

1239 }

1240

1241 // Change the value(s) if hasChanged flag is set

1242 if (hasChanged) {

1243 ADC\_Refresh = true;

1244 // Use of mutex when changing shared global variables

1245 pthread\_mutex\_lock(&MainMutex);

1246 change(CField.isOn, wavef, CField.freq, CField.mean, CField.amp);

1247 pthread\_mutex\_unlock(&MainMutex);

1248 hasChanged = false;

1249 // Load new values to old values array

1250 old\_adc\_in[0] = adc\_in[0];

1251 old\_adc\_in[1] = adc\_in[1];

1252 }

1253 }

1254 }

1255 return(0);

1256 }

1257

1258

1259

1260 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

1261 // Supporting Programs

1262 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

1263 // Supporting function: check whether DAC will have negative data

1264 bool hasNegative(){

1265 if( (DAC.mean-DAC.amp) < 0)

1266 return true;

1267 return false;

1268 }

1269 // Supporting function: check absolute maximum value

1270 short checkAbsMax(float mean, float amp){

1271 float absMax=0;

1272 if(fabs(mean+amp)>fabs(mean-amp))

1273 absMax = fabs(mean+amp);

1274 else

1275 absMax = fabs(mean-amp);

1276 if(absMax<5)

1277 return 1;

1278 else if(absMax<10)

1279 return 2;

1280 else {

1281 printf("\nError: The absolute maximum is more than 10V.\n");

1282 printf("Either reset the data field or select small absolute value.\n");

1283 return 0;

1284 }

1285 }

1286 // Function to directly change the parameters of the DAC

1287 void change(bool onSignal, int wvty, float f, float m, float a){

1288 DAC.waveform\_type=wvty;

1289 DAC.freq=f;

1290 DAC.mean=m;

1291 DAC.amp=a;

1292 DAC.resetWave=true;

1293 DAC.isOn=onSignal;

1294 }

1295 //Set the change field to be equal to initial DAC parameters

1296 void setChangeField(ChangeField\* CF){

1297 (\*CF).waveform\_type=DAC.waveform\_type;

1298 (\*CF).freq=DAC.freq;

1299 (\*CF).mean=DAC.mean;

1300 (\*CF).amp=DAC.amp;

1301 (\*CF).isOn=DAC.isOn;

1302 }