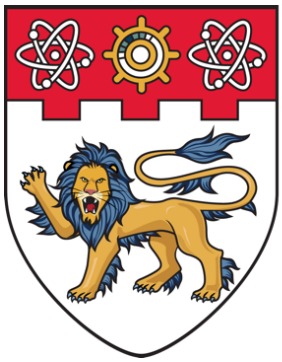


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

MA4830: Realtime Software for Mechatronic Systems

Major Assignment (CA2)

Group Members: Hoo Shi Bin (U1520038B)
Poon Jun Nam (U1520047C)
Heng Ming Ji (U1523049F)
Chow Jiun Fatt (U1520021C)
Tan Pak Zan (U1520002D)
Chong Bing Sheng (U1520033F)
Pius Lim Zhen Ye (U1520003A)

Table of Content

1. Introduction

This program serves as an application that generates waveform according to user's input. It is capable of generating two waves and projecting them on the oscilloscope at the same time. Besides, it allows user to configure wave parameters such as Type of waveform, Amplitude, Frequency and Mean.

1.1. Key Features

- Allows full configuration of waveform parameters (Waveform type, Amp., Freq., Mean)
    a. Amp. Freq. Mean

    These parameters can be configured by adjusting the hardware. Parameters type is determined the limit switches while the value of the determined parameters can be modified by turning the potential meters.
    b. Waveform type

    Waveform type can be configured by users through keyboard input. A user interface was developed and serves to prompt users for input.

- Allows READ & WRITE of parameters into a file.
    a. WRITE: The program allows user to save current waveform parameters into a file with desired file name. The file will then be saved in the current directory.
    b. READ: This program allows user to read waveform parameters from file that is saved earlier. The data read will be act as the parameters of waveforms that are being projected.

- Robustness of program
    a. Memory allocation for variables

    f_Malloc() function allows dynamics memory allocation when new variables are introduced (e.g. struct. variable *channel_para* to store the wave parameters). When the variables are not used, the memory is freed. This feature enhances the efficiency of the program.
    b. Thread management

    All the threads created in this program are well-tracked to ensure a good management of CPU resources. The thread (e.g. main()) is terminated when it is not used. All remaining threads are terminated accordingly when user prompts to end the program.
    c. User-friendliness

    All the keyboard inputs entered by the user are checked to ensure a valid input. Given an invalid input, user will be prompted to re-enter the input with clear instructions.

2. User Manual

2.1. Program Execution

The program can be executed by typing the command shown below at the terminal:

E.g. $ ./main_FINAL -square -sine

The two arguments followed by the program name is to configure the type of waveform generated in channel 1 and channel 2. In the example shown above, channel 1 will generate square wave while channel 2 will generate sine wave.

This program is able to allow user to input argument without any restrictions. In other word, user can choose not to input any arguments at the point of program execution. In this case, default waveform will be generated at both channels, where default waveform for channel 1 is sine wave and channel 2 is square wave.

Table 1: Command line arguments available

Arguments	Waveform types
-sine	Sine wave
-square	Square wave
-tri	Triangular wave
-saw	Sawtooth wave

Once the program is executed, it will clear out the terminal and display only the necessary instructions and information. As shown in Figure 1, the instructions for exiting program and prompting keyboard input are displayed. Besides that, the 3 parameters of waveform in both channels are displayed in real time manner at the terminal. The wave generated are shown in Figure 2.

Figure 1: Real-time wave parameters displayed on screen

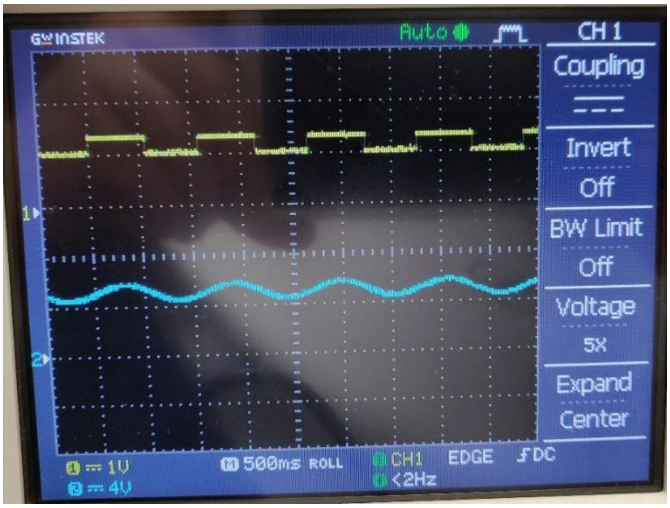


Figure 2: Waveform projected on oscilloscope

2.2. Change the wave parameters real-time via PCI board (Hardware Input)

This program also allow user to change the 3 parameters of waveform through the PCI board by switching on the toggle switch 1 (e.g. TS 1 as shown on the figure below). TS 1 works as a virtual "ON/OFF" switch for the external circuit board. Hence, the program will only takes input from PCI board when the TG 1 is switched on.

Next, TS 2 is the "ON/OFF" switch for the whole program. The whole program will be terminated once TS 2 is switched ON, provided TS 1 is switched ON as well.

TS 3 and TS 4 together functioned as wave parameters selection. The combination of the ON/OFF state of the 2 toggle switches is controlling the type of wave parameters to be changed. The combination are shown below in Table 3. The change of wave parameters is then done by adjusting the potentiometer, where 2 potentiometers are provided for channel 1 and channel 2 respectively.

The changes of the wave parameters are reflected in real-time as can see when projecting on the oscilloscope, as shown in Figure 4.

Table 2: The indication of ON/OFF state of each toggle switch

Toggle Switch / States	TS1	TS2	TS3	TS4
1 / ON	Enable hardware inputs	Terminate Program	Mode of Changing Parameters. (Refer to table below)	

0 / OFF	Disable hardware inputs (excluding keyboard)	-	

Table 3: The indication of ON/OFF state of TS 3 and TS 4 to choose which wave parameters to change

	State of switch TS 3	State of switch TS 4
NULL	0	0
Change Amplitude	0	1
Change Frequency	1	0
Change Mean	1	1

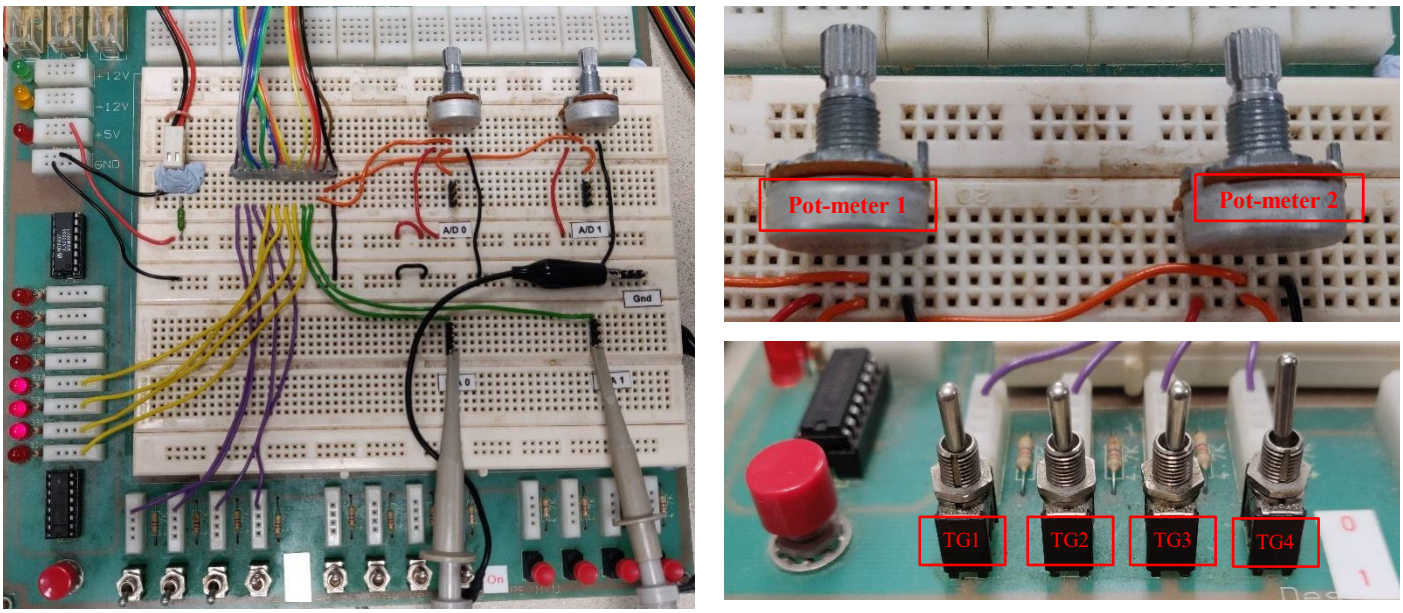
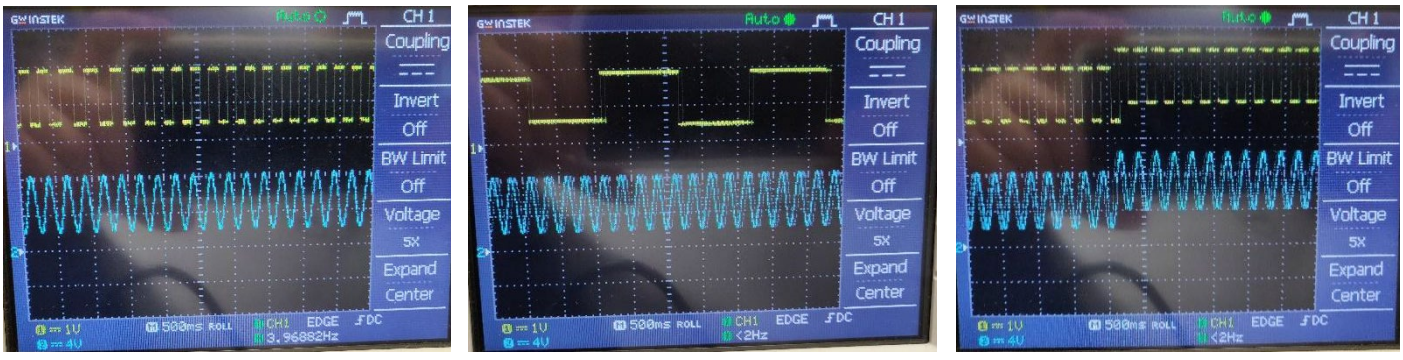


Figure 3: Toggle switches and potentiometers on the external circuit board

(Left: external circuit board; Top Right: potentiometers; Bottom Right: toggle switches)



Changing Amplitude | Changing Frequency | Changing Mean

Figure 4: The adjustment of the wave parameters displayed on the oscilloscope

2.3. Range of value of wave parameters

The Amplitude, Frequency and Mean's min and max value are as follow:

Table 4: Range of wave parameters

	Min	Max
Amplitude	0 Unit	5 Unit
Frequency	0.5 Hz	5 Hz
Mean	-5 Unit	5 Unit

2.4. Access additional functionalities via keyboard (Keyboard Input)

While the PCI board (Hardware Input) allow user to manipulate the waveform parameters, this program also provide the flexibility that user can interact with system through the keyboard and access additional functionalities. By hitting "Ctrl + \" keys on keyboard, the program will stop displaying the current parameters and change into MAIN MENU display. Additional functions mentioned below are then accessible for users.

From Main menu, user may opt to:

1. Change waveform type
2. Save and output file (wave parameters)
3. Read file
4. Return to display
5. Terminate the program

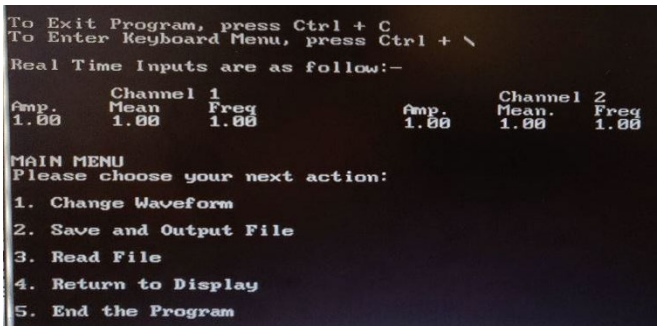


Figure 5: MAIN MENU prompt

2.4.1 Change Waveform Type

If option 1 is selected, user will be prompted to select the channel to be changed as shown in Figure 7. For instance, channel 1 is selected.

User has the option to choose the new type of waveform from the list below:

1. Sine wave
2. Square wave
3. Sawtooth wave
4. Triangular wave

After selection, new waveform is updated on the oscilloscope.

```
You have indicated to change waveform.
Please select the channel:

1. Channel 1
2. Channel 2

0. Return Main Menu
1
Channel 1 selected, please choose your desired waveform:
1. Sine Wave
2. Square Wave
3. Sawtooth Wave
4. Triangular Wave

0. Return Main Menu
```

Figure 7: Program prompt to user

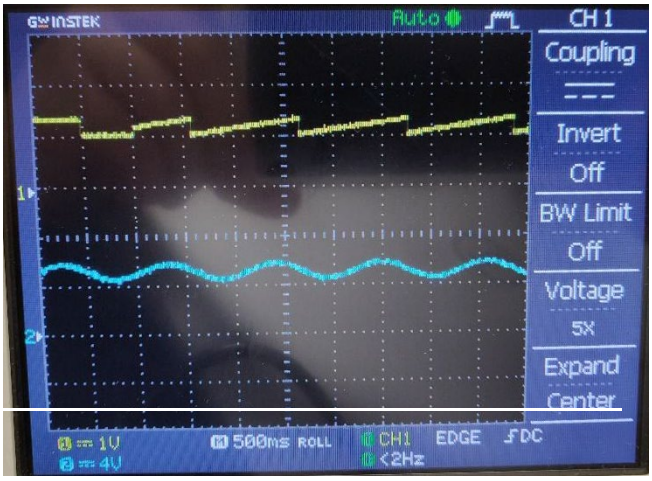


Figure 6: New waveform after selection done via keyboard

2.4.2 Create and Save file

Under option 2, user will be prompted to enter the name of file. Current wave parameters and type of waveform are then saved into a text file in the current directory.

```
You have indicated to save the output to a file.
Please name your file(.txt):

0. Return Main Menu
gerald
File saving in progress, please wait...
File saved!


MAIN MENU
Please choose your next action:
1. Change Waveform
2. Save and Output File
3. Read File
4. Return to Display
5. End the Program
```

Figure 8: Program prompt

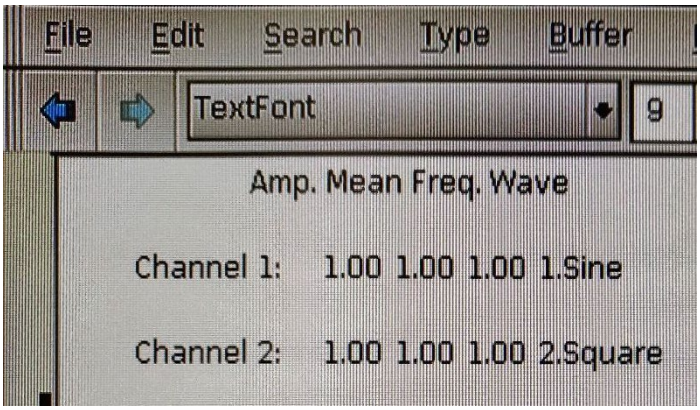


Figure 9: Content of the text file saved

2.4.3 Read file

For option 3, user will be prompted to enter the name of file to be read. Current wave parameters and wave type are updated according to the data saved in the file.

```
You have indicated to read the file.
Please name your file(.txt):

0. Return Main Menu
gerald
File reading in progress, please wait...
File Read Successfully


MAIN MENU
Please choose your next action:
1. Change Waveform
2. Save and Output File
3. Read File
4. Return to Display
```

2.4.4 Return to display

If option 4 is chosen, the program escapes from Main Menu and continues with displaying the current parameters.

2.4.5 Terminate the program

Program is terminated after each and every threads is killed accordingly.

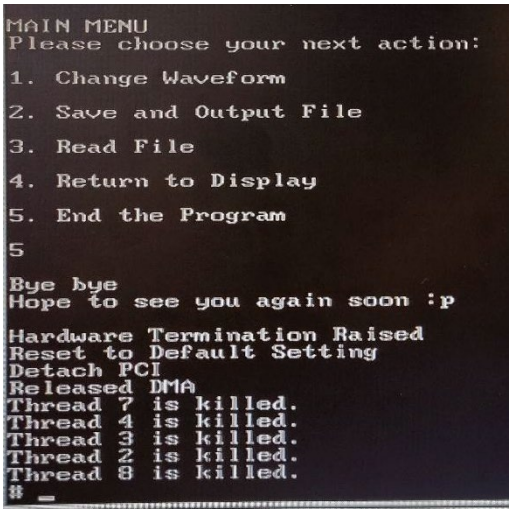


Figure 11: Program prompt

3. Backend Process

This section discusses the programming rationale and the approach to tackle multi-threading.

Our program includes Initialisation, Execution, and Termination.

3.1. Initialisation

To initialise the program, actions below are done in sequence.

1. Declarations of definitions and variables.
2. f_PCISetup() : initiates the mapping of PCI Board to process address space.
3. f_Malloc() : allocates dynamic memory spaces for the variables mentioned.
4. f_WaveGen : generates 4 wave arrays of different waveforms (Sine, Square, Triangular, Sawtooth)
5. Associating signal handlers with signals (SIGINT, SIGQUIT) respectively.
   - signal_handler() with SIGINT:-
     On SIGINT, signal_handler() calls f_termination to initiate clearing of DA registers, detachment of PCI board and releasing of dynamic memory allocated. Then signal_handler proceeds with exiting all running threads in order.
   - signal_handler2() with SIGQUIT:-
     On SIGQUIT, thread t_UserInterface is created.
6. Four threads (as below) are created.
   - t_Wave1() : scale and project wave to Channel 1 on oscilloscope.
   - t_Wave2() : scale and project wave to Channel 2 on oscilloscope.
   - t_HardwareInput() : read hardware input (toggle switches and potentiometer on the PCI Board).
   - t_ScreenOutput() : print real-time waveform parameters on screen.
7. Main thread is killed to save resources.

3.2. Execution

Five main threads are responsible for the execution of application. Four of them (t_UserInterface not included) are run in an infinite loop until SIGINT is prompted) Details of each threads are as below.

1. **t_Wave1()** : scales and projects wave to Channel 1 on oscilloscope.

Thread t_Wave1() takes in wave[] array* to determine the waveform type to be generated. The thread then acquires the point array of desired waveform from **wave_type[][] and scales it according to waveform parameters in struct ch[0] (Amp., Freq., Mean). The scaled point array is written to corresponding address to generate a waveform on oscilloscope.

Furthermore, to ensure the accurate frequency of waveforms, time function is used to calculate the time function is used to calculate the time taken for the waveform loop. Time variable is taken account into delay() and the duration of delay adjusts accordingly.

*wave[] : an array that contains configuration of waveform to be generated. It changes according to user input at the user interface. The property of wave[] is demonstrated in the table below.

Table 5: Properties of wave[] array

Value of wave[]	Waveform type to be generated
1	Sine Wave
2	Square Wave
3	Sawtooth Wave
4	Triangular Wave

**wave_type[][] : an array that servers as point arrays for different waveforms (Sine, Square, Triangular, Sawtooth). The properties are shown in the table below.

Table 6: Properties of wave_type[][] array, where NO_POINT is the number of points in one period

Wave_type[][]	Content
Wave_type[0][NO_POINT]	Sine wave array with n points
Wave_type[1][NO_POINT]	Square wave array with n points
Wave_type[2][NO_POINT]	Sawtooth wave array with n points
Wave_type[3][NO_POINT]	Triangular wave array with n points

2. **t_Wave2()** : scales and projects wave to Channel 2 on oscilloscope.
   Thread t_Wave2() works the same way as thread t_Wave1(). It takes in wave[] array and wave_type[][] array to generate waveform array. However, the scaling parameters is taken from struct ch[1] instead where ch[1] contains the Amp., Freq, and Mean for channel 2.

3. **t_HardwareInput()** : reads hardware input (toggle switches, potentiometer on PCI Board).

   Thread t_HardwareInput() first read the state of all 4 toggle switches (details of configuration of toggle switches is shown in the table below). While TS1 is ON and TS2 is OFF, the thread continuously reads the value of potentiometers, scales the ADC values to 16 bits, and saves it to ch[0] and ch[1]. For instance, given TS3 = 0 and TS4 = 1, turning the potentiometers will change the amplitude for Channel 1 and Channel 2. The values are then scaled and saved into ch[0].amp and ch[1].amp. In conclusion, this thread reads the hardware input, scales it to 16 bits digital values and saves it to ch[] for further usage.

Table 7: The indication of ON/OFF state of each toggle switch

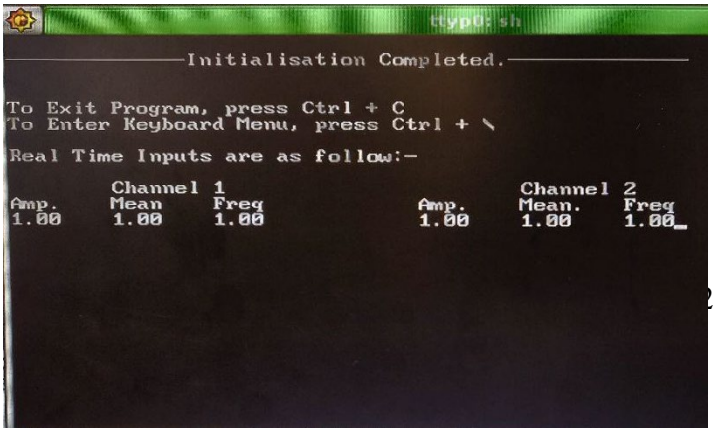
Toggle Switch / States	TS1	TS2	TS3	TS4
1 / ON	Enable hardware inputs	Terminate Program	Mode of Changing Parameters. (Refer to table below)	
0 / OFF	Disable hardware inputs (excluding keyboard)	-		

Table 8: The indication of ON/OFF state of TS 3 and TS 4 to choose which wave parameters to change

	State of switch TS 3	State of switch TS 4
NULL	0	0
Change Amplitude	0	1
Change Frequency	1	0
Change Mean	1	1

4. **t_ScreenOutput()** : prints real-time waveform parameters on screen.

   Thread t_ScreenOutput() serves as a platform to update user on the current (real-time) waveform parameters on the screen. It takes in struct ch[0], ch[1] and writes all the parameters into the display in an ordered manner. To optimize the application, the display is refreshed at 5 Hz to ensure the stable screen output and the accuracy of real-time parameters. A screenshot of the display is shown as below.



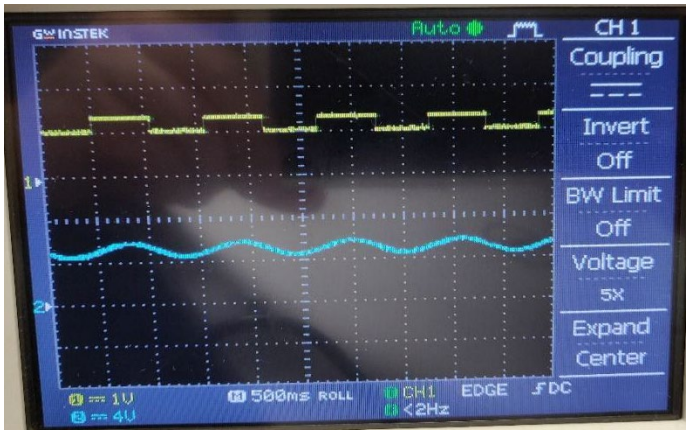


Figure 12: After initialisation, two waves (could be different waveforms) are projected onto oscilloscope. For instance, square wave (channel 1) and sine wave (channel 2) are shown respectively.

5. **t_UserInterface()** : prompts and takes in user input.

   Thread t_UserInterface() is created when SIGQUIT is called. It then kills thread t_ScreenOutput to stop the thread from printing the parameters. It is killed when user prompts to escape from User Interface (aka Main Menu).

   Thread t_UserInterface() serves as an user interface to prompt and take in user input. When created, a Main Menu is displayed on screen. From Main menu, user may opt to:

   a. Change waveform type
   b. Save and output file (wave parameters)
   c. Read file
   d. Return to display
   e. Terminate program

   Change waveform type: After user entering the desired new waveform type, this thread updates the value in wave[]. A new waveform is then displayed once new wave[] is taken as input by thread t_Wave1() or t_Wave2().

   Save and output file: User is prompted to enter the filename. The filename is then concatenated with file extension ".txt" and the file is created. Current parameters of both waveforms are saved into the file.

   Read file: User is prompted to enter the name of file to be read. If the file exists in the current directory, the data is extracted from the file and both waveform parameters are updated with the data. Furthermore, to ensure that the waveform parameters are updated accurately with the data, the thread makes sure that the Toggle Switch 1 (TS1) is switched before the parameters are updated.

   Return to display: The thread re-create thread t_ScreenOutput and then it exits (thread dies). This allow the user to escape from Main Menu and keep updated on the current waveform parameters.

Terminate Program: The thread raises SIGINT. With SIGINT and its associated signal handler, all running threads are killed in order and the program ends.

3.3. Termination

Termination of program is called when SIGINT is raised. With SIGINT associated with signal_handler(), f_termination is called to clear the Digital-to-Analog Registers, detach PCI board and free the dynamic memory allocated earlier. Signal_handler() then proceeds with killing all running threads in order. Afterall, the program ends.

4. Limitations

4.1. Hardware Input

The 2 potentiometers are assigned to adjust each channels' wave parameters respectively. One potentiometer is used to adjust all the wave parameters of one channel by switching the toggle switches. Due to the nature of the hardware's electrical circuit, a parameter setting will immediately change according to the potentiometer's adjustment once the parameter dedicated toggle switch is triggered. For instance, when someone trigger the toggle switch to adjust the amplitude after changing the frequency, the amplitude will immediately change and follow the setting applied to frequency before.

5. Discussion

We have attempted to integrate all the knowledge learnt during the lectures and hands-on practical sessions into our program design. For example, the use of multi-threading, formatted buffer I/O, PCI-DAS hardware input and output. However, there is some issue we faced and tried to solve them as stated below:

5.1. Range of frequency

We pre-defined the resolution of all waves into 200 points, to provide more accurate characteristic of different waves. As a trade-off, the maximum frequency can achieve is 5Hz, referring to the following explanation. We take sine wave as example, we initially compute the value of each point (with increment of 360°/ 200) and store into an array of integer. While displaying on output, we ensure the frequency by determining the gap period during each point. Due to the minimum timestamp of delay() function of 1ms, the maximum frequency is:

Period T (second) = 200 * 1ms = 0.2 s

Frequency (hz) = 1 / T = 1 / 0.2 = 5 hz

This problem can be solved by lowen the resolution, for example, if we use 100 points to describe a wave:

Period T (second) = 100 * 1ms = 0.1 s

Frequency (hz) = 1 / T = 1 / 0.1 = 10 hz

In future, we can expand the range of frequency by making resolution a variable factor and determine the correlation between gap period, frequency, and resolution.

5.2. Termination of multi-thread with signal handlers

As SIGINT can be captured by any running thread within same process, we cannot guarantee or predict which thread signal_handler() belongs to. Initially, we tried to terminate all threads orderly (using pthread_cancel with sequential thread id), but it had a chance to terminate the attached thread and caused signal_handler being terminated before it completed. After discussion, we solved the issue by checking the current thread id with pthread_self() and by pass itself from the pthread_cancel(), eventually we terminated the process with phtread_exit(NULL) within signal_handler.

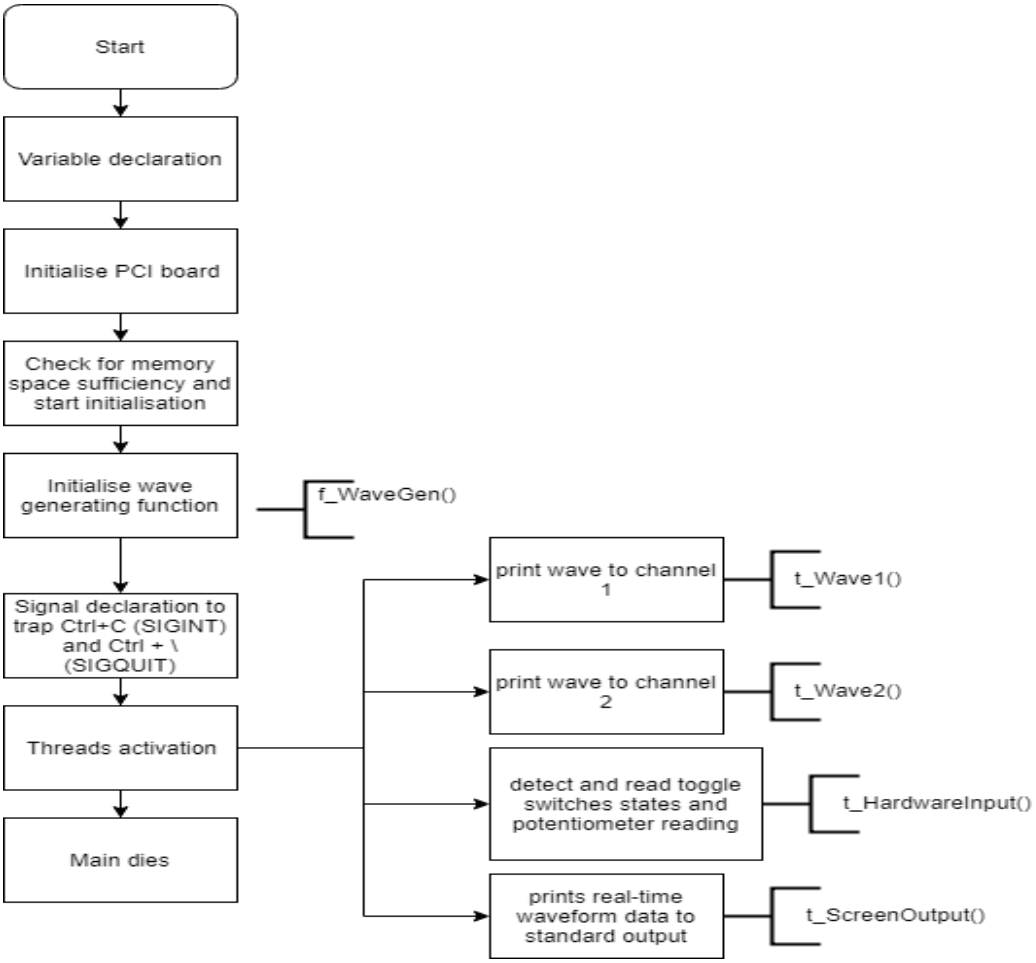
Same issue with SIGQUIT, initially we use signal_handler2() to capture this signal and trigger f_UserInterface(), but it will attach to any thread as IRQ (interrupt) and halt the resource from running thread until completely return from subroutine. Therefore, it will “pause” the running thread and cause

asynchronous multithreading. As a solution, we changed f_UserInterface() into t_UserInterface() to make it isolated thread, so that it would interrupt on-going other threads.

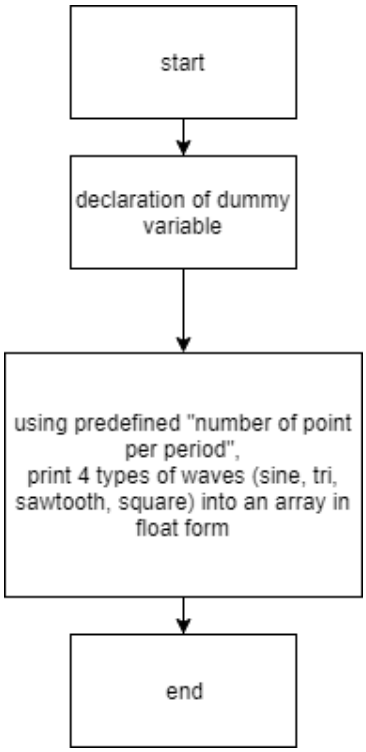
Appendix A – Flowcharts

1. Functions

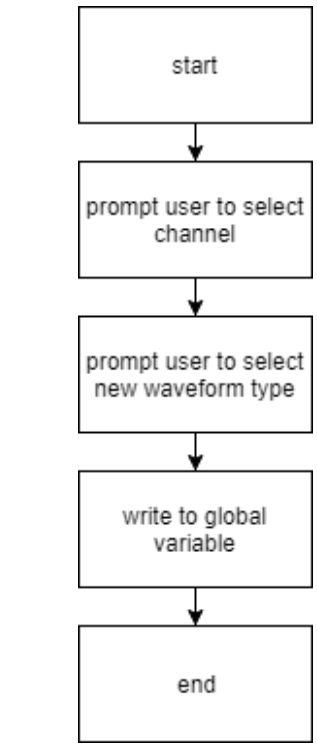
main( )



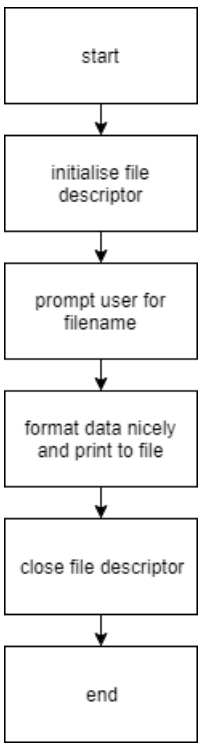
f_WaveGen( )



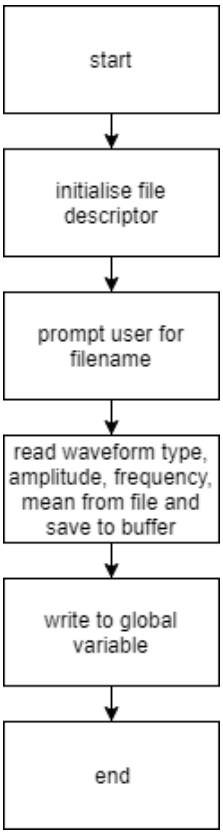
f_ChangeWavePrompt( )



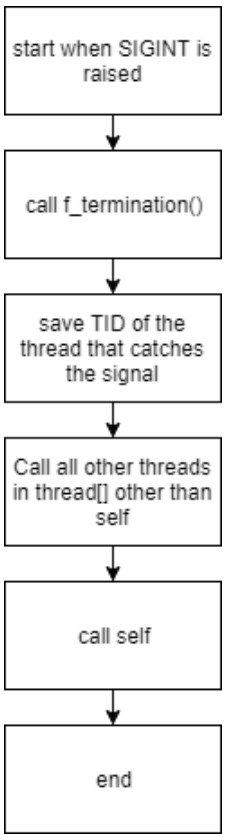
f_SaveFilePrompt( )



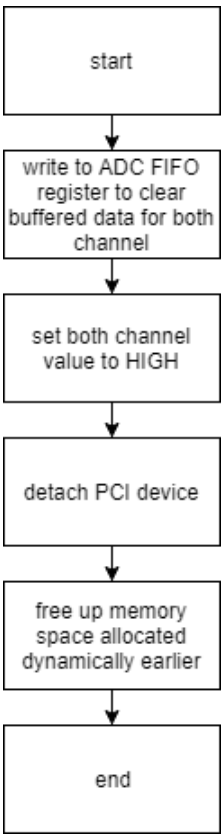
f_ReadFilePrompt( )



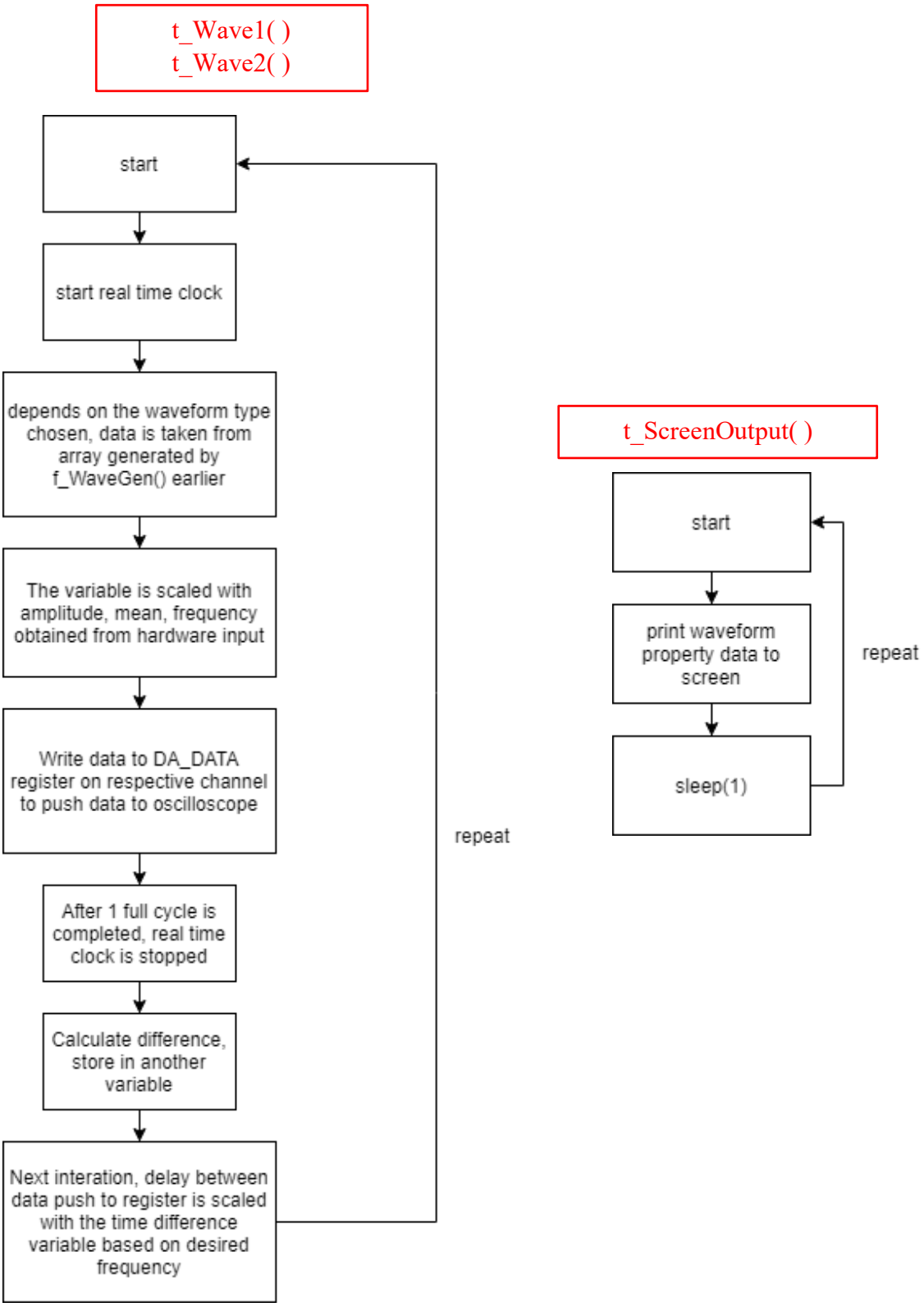
signalhandler( )



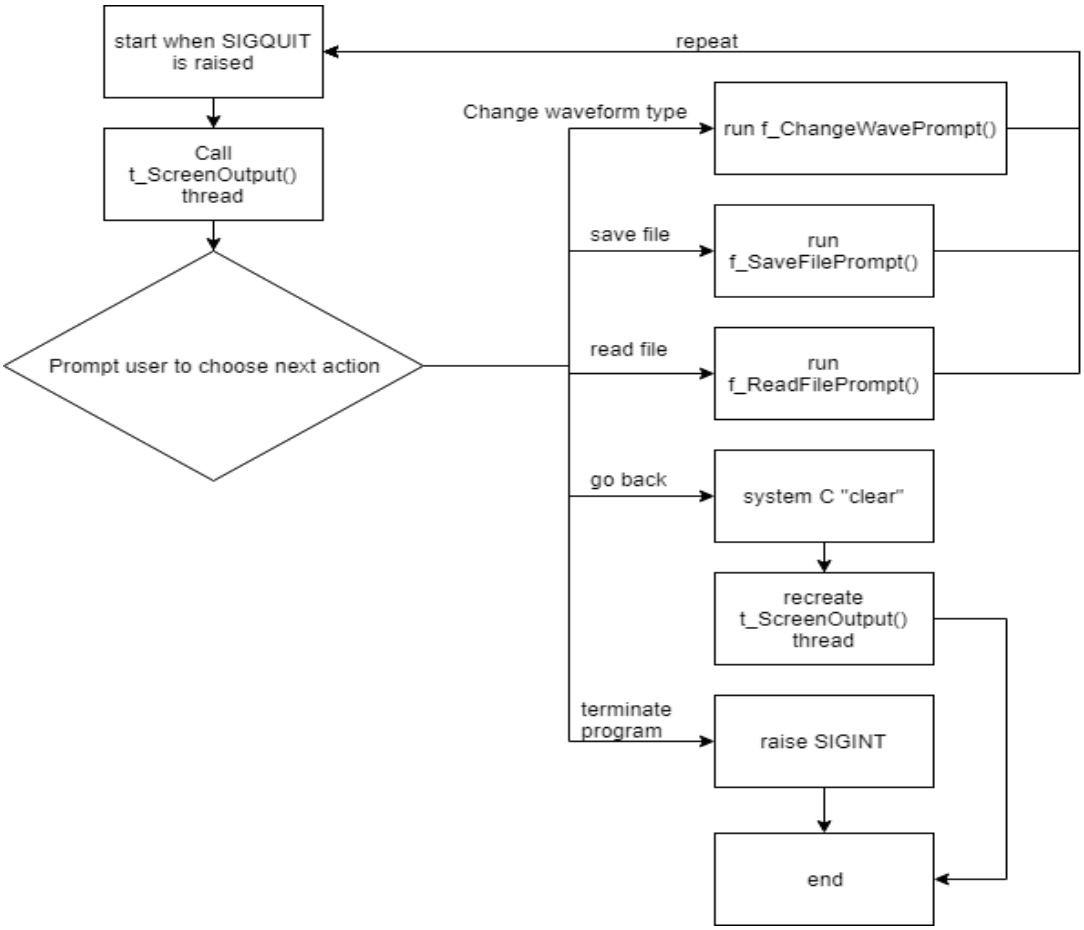
f_termination( )



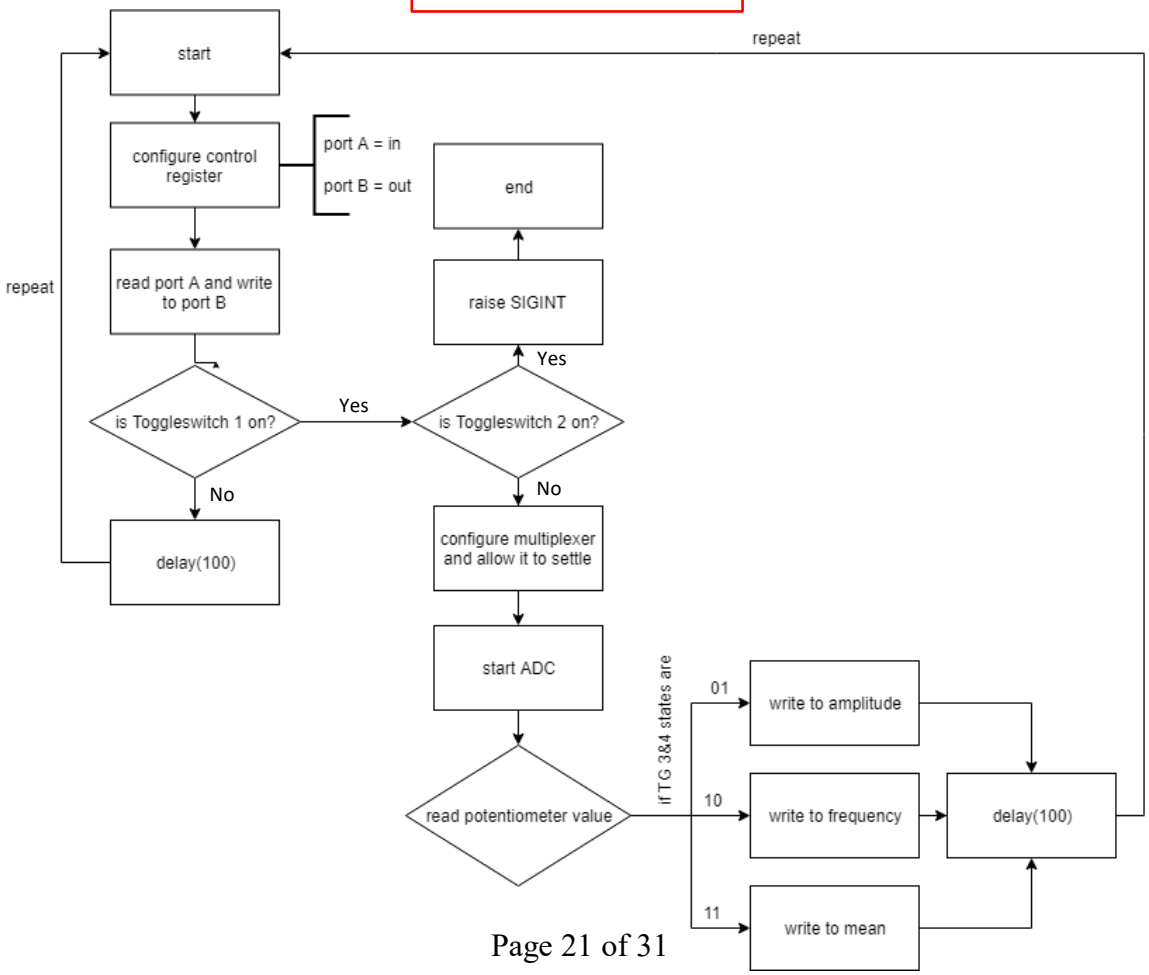
2. Threads


t_Wave1( )
t_Wave2( )
start
start real time clock
depends on the waveform type chosen, data is taken from array generated by f_WaveGen() earlier
The variable is scaled with amplitude, mean, frequency obtained from hardware input
Write data to DA_DATA register on respective channel to push data to oscilloscope
After 1 full cycle is completed, real time clock is stopped
Calculate difference, store in another variable
Next interation, delay between data push to register is scaled with the time difference variable based on desired frequency
repeat
t_ScreenOutput( )
start
print waveform property data to screen
sleep(1)
repeat

t_KeyboardInput( )



t_HardwareInput( )



Appendix B – Program Listing

```
//*****************************************************************************
//    Program : main_FINAL.c
//
//    Generates waveform according to user's input.
//    Capable of generating two waves and
//  projecting them on the oscilloscope at the same time.
//     Allows user to configure wave parameters:
//  (Type of waveform, Amplitude, Frequency, Mean)
//
//    Procedures:
//        "./main_FINAL"
//          - to open 2 channels with default(sine and square) waves
//        "./main_FINAL -saw"
//          - to open 2 channels with sawtooth and default(square) waves respectively
//        "./main_FINAL -saw -sine"
//          - to open 2 channels with sawtooth and sine waves respectively
//      Availabe arguments:
//            "-sine": Sine Wave; "-square": Square Wave;
//            "-tri": Triangular Wave; "-saw": Sawtooth Wave
//
//     23 November 2018 : Students of class 18/19
//     QNX 6.xx version - MA4830 Major Assignment
//*****************************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <hw/pci.h>
#include <hw/inout.h>
#include <sys/neutrino.h>
#include <sys/mman.h>
#include <math.h>
#include <time.h>
#include <pthread.h>
#include <signal.h>

#define INTERRUPT iobase[1] + 0 // Badr1 + 0 : also ADC register
#define MUXCHAN iobase[1] + 2   // Badr1 + 2
#define TRIGGER iobase[1] + 4   // Badr1 + 4
#define AUTOCAL iobase[1] + 6   // Badr1 + 6
#define DA_CTLREG iobase[1] + 8 // Badr1 + 8

#define AD_DATA iobase[2] + 0    // Badr2 + 0
#define AD_FIFOCLR iobase[2] + 2 // Badr2 + 2

#define TIMER0 iobase[3] + 0     // Badr3 + 0
#define TIMER1 iobase[3] + 1     // Badr3 + 1
#define TIMER2 iobase[3] + 2     // Badr3 + 2
#define COUNTCTL iobase[3] + 3   // Badr3 + 3
#define DIO_PORTA iobase[3] + 4  // Badr3 + 4
#define DIO_PORTB iobase[3] + 5  // Badr3 + 5
#define DIO_PORTC iobase[3] + 6  // Badr3 + 6
#define DIO_CTLREG iobase[3] + 7 // Badr3 + 7
#define PACER1 iobase[3] + 8     // Badr3 + 8
#define PACER2 iobase[3] + 9     // Badr3 + 9
#define PACER3 iobase[3] + a     // Badr3 + a
#define PACERCTL iobase[3] + b   // Badr3 + b

#define DA_Data iobase[4] + 0    // Badr4 + 0
#define DA_FIFOCLR iobase[4] + 2 // Badr4 + 2

#define BILLION 1000000000L
#define MILLION 1000000L
#define THOUSAND 1000L
#define NO_POINT 50
#define NUM_THREADS 5
#define AMP 1 //default wave parameters
#define MEAN 1
#define FREQ 1

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Global Variable
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int badr[5]; //PCI 2.2 assigns 6 IO base addresses
void *hdl;
uintptr_t dio_in;
uintptr_t iobase[6];
uint16_t adc_in[2];
int chan;
pthread_t thread[NUM_THREADS];

typedef struct
{
    float amp,
    mean,
    freq;
} channel_para; //store waveform parameters

typedef int wave_pt[NO_POINT]; //store points of wavefomrs with resolution NO_POINT
```

```
channel_para *ch;
wave_pt *wave_type;

int wave[2]; //store wave type for each channel

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Functions
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void f_PCIsetup()
{
    struct pci_dev_info info;
    unsigned int i;

    memset(&info, 0, sizeof(info));
    if (pci_attach(0) < 0)
    {
        perror("pci_attach");
        exit(EXIT_FAILURE);
    }

    // Vendor and Device ID
    info.VendorId = 0x1307;
    info.DeviceId = 0x01;

    if ((hdl = pci_attach_device(0, PCI_SHARE | PCI_INIT_ALL, 0, &info)) == 0)
    {
        perror("pci_attach_device");
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < 5; i++)
    {
        badr[i] = PCI_IO_ADDR(info.CpuBaseAddress[i]);
    }

    // map I/O base address to user space
    for (i = 0; i < 5; i++)
    { // expect CpuBaseAddress to be the same as iobase for PC
        iobase[i] = mmap_device_io(0x0f, badr[i]);
    }
    // Modify thread control privity
    if (ThreadCtl(_NTO_TCTL_IO, 0) == -1)
    {
        perror("Thread Control");
        exit(1);
    }

    // Initialise Board
    out16(INTERRUPT, 0x60c0); // sets interrupts     - Clears
    out16(TRIGGER, 0x2081);   // sets trigger control: 10MHz,clear,
    // Burst off,SW trig.default:20a0
    out16(AUTOCAL, 0x007f); // sets automatic calibration : default
    out16(AD_FIFOCLR, 0);   // clear ADC buffer
    out16(MUXCHAN, 0x0900); // Write to MUX register-SW trigger,UP,DE,5v,ch 0-0
    // x x 0 0 | 1  0  0 1  | 0x 7   0 | Diff - 8 channels
    // SW trig |Diff-Uni 5v| scan 0-7| Single - 16 channels

    //initialise port A, B
    out8(DIO_CTLREG, 0x90); // Port A : Input Port B: Output
}

void f_WaveGen()
{
    int i;
    float dummy;

    //Sine wave array
    float delta = (2.0 * 3.142) / NO_POINT;
    for (i = 0; i < NO_POINT; i++)
    {
        dummy = (sinf(i * delta)) * 0x7fff / 5;
        wave_type[0][i] = dummy;
    }
    //Square wave array
    //the value of 2 in dummy when i<NO_POINT/2 indicates the value of 'ON'
    //the value of 0 in dummy when i<NO_POINT indicates the value of 'OFF'
    for (i = 0; i < NO_POINT; i++)
    {
        if (i <= NO_POINT / 2)
            dummy = 1 * 0x7fff;
        if (i > NO_POINT / 2)
            dummy = -1 * 0x7fff;
        wave_type[1][i] = dummy / 5;
    }
    //Saw-tooth wave array
    //the delta used is similar to the one used for sine wave
    //the dummy is increased by multiple of delta when i <NO_POINT/2
    //then the dummy is decrease by multiple of delta when i<NO_POINT
    //the value of '1' in dummy when i<NO_POINT indicates the max value of wave form when i=NO_POINT/2
    delta = 2.0 / NO_POINT;
    for (i = 0; i < NO_POINT; i++)
```

```
    {
        if (i <= NO_POINT / 2)
            dummy = i * delta * 0x7fff;
        if (i > NO_POINT / 2 && i < NO_POINT)
            dummy = (-2 + i * delta) * 0x7fff;
        wave_type[2][i] = dummy / 5;
    }
    //Triangular wave array
    //the value of 2 in delta indicates the max vertical value of the wave form when i = NO_POINT-1, the value is closed to 2
    //the min vertical value of wave form is 0 when i = 0
    delta = 4.0 / NO_POINT;
    for (i = 0; i < NO_POINT; i++)
    {
        if (i <= NO_POINT / 4)
            dummy = i * delta * 0x7fff;
        if (i > NO_POINT / 4 && i <= NO_POINT * 3 / 4)
            dummy = (2 - i * delta) * 0x7fff;
        if (i > NO_POINT * 3 / 4 && i < NO_POINT)
            dummy = (-4 + i * delta) * 0x7fff;
        wave_type[3][i] = dummy / 5;
    }
}

void f_ArgCheck(int argc, char *argv[])
{
    int i;
    char **p_to_arg = &argv[1];

    //set default waveform for both channels
    wave[0] = 1;
    wave[1] = 2;
    //get waveform from user input
    for (i = 1; i < argc && i < 3; i++, p_to_arg++)
    {
        if (strcmp(*(p_to_arg), "-sine") == 0)
        {
            printf("\n%d. Sine wave chosen for Channel %d.\n", i, i);
            wave[i - 1] = 1;
        }
        else if (strcmp(*(p_to_arg), "-square") == 0)
        {
            printf("\n%d. Square wave chosen for Channel %d.\n", i, i);
            wave[i - 1] = 2;
        }
        else if (strcmp(*(p_to_arg), "-saw") == 0)
        {
            printf("\n%d. Saw wave chosen  for Channel %d.\n", i, i);
            wave[i - 1] = 3;
        }
        else if (strcmp(*(p_to_arg), "-tri") == 0)
        {
            printf("\n%d. Triangular wave chosen for Channel %d.\n", i, i);
            wave[i - 1] = 4;
        }
        else{
            printf("\nInput not valid!");
            printf("\n%d. Default wave chosen for Channel %d.\n", i, i);
        }
    } //end of for loop, checking argv[]
    delay(2000);
}

void f_Malloc()
{
    int i;
    if ((ch = (channel_para *)malloc(2 * sizeof(channel_para))) == NULL)
    {
        printf("Not enough memory.\n");
    }

    for (i = 0; i < 2; i++)
    {
        ch[i].amp = AMP;
        ch[i].mean = MEAN;
        ch[i].freq = FREQ;
    }

    if ((wave_type = (wave_pt *)malloc(4 * sizeof(wave_pt))) == NULL)
    {
        printf("Not enough memory.\n");
        exit(1);
    }
}

void f_termination()
{
    out16(DA_CTLREG, (short)0x0a23);
    out16(DA_FIFOCLR, (short)0);
    out16(DA_Data, 0x8fff); // Mid range - Unipolar

    out16(DA_CTLREG, (short)0x0a43);
    out16(DA_FIFOCLR, (short)0);
    out16(DA_Data, 0x8fff);
```

```
    pci_detach_device(hdl);

    free((void *)ch);
    free((void *)wave_type);
    printf("Reset to Default Setting\nDetach PCI\nReleased DMA\n");
}

int f_GetInt(int lowLim, int highLim)
{
    int outnum;
    while (true)
    {
        char c;

        //get int input
        scanf("%d", &outnum);
        //flush input buffer
        while ((c = getchar()) != '\n' && c != EOF);

        //check if outnum is within low and high limit
        //otherwise continue loop
        if (outnum >= lowLim && outnum <= highLim)
            return outnum;
        else
            printf("Please input a valid number!\nYour number should be within %d and %d\n\n", lowLim, highLim);
    }
}

void f_ChangeWavePrompt()
{
    int chn;
    const char *wave_str[] = {"Sine", "Square", "Sawtooth", "Triangular"};

    printf("\nYou have indicated to change waveform.\nPlease select the channel:
        \n1. Channel 1
        \n2. Channel 2
        \n\n0. Return Main Menu\n\n");

    //return main menu if input = 0
    if ((chn = f_GetInt(0, 2)) == 0)
        return;

    printf("\nChannel %d selected, please choose your desired waveform:
        \n1. Sine Wave
        \n2. Square Wave
        \n3. Sawtooth Wave
        \n4. Triangular Wave
        \n\n0. Return Main Menu\n\n", chn);
    //set wave for channel
    //return main menu if input = 0
    if ((wave[chn-1] = f_GetInt(0, 4)) == 0)
        return;

  // print selected wave
    printf("\n%s Wave Selected\n\n", wave_str[wave[chn-1]-1]);
}

void f_SaveFile(char *filename, FILE *fp, char *data)
{
    strcat(filename, ".txt");
    printf("\nFile saving in progress, please wait...\n");

    if ((fp = fopen(filename, "w")) == NULL)
    {
        printf("Cannot open\n\n");
        return;
    }
    if (fputs(data, fp) == EOF)
    {
        printf("Cannot write\n\n");
        return;
    }
    fclose(fp);
    printf("File saved!\n\n");
}

void f_SaveFilePrompt()
{
    const char *wave_str[] = {"Sine", "Square", "Sawtooth", "Triangular"};
    char filename[100];
    FILE *fp;
    char data[200];
    printf("\n\nYou have indicated to save the output to a file.
        \nPlease name your file(.txt):
        \n\n0. Return Main Menu\n\n");
    scanf("%s", &filename);

    //return main menu if input = 0
    if(strcmp(filename,"0")==0)
        return;

  //set data to store into file
    sprintf(data,
```

```
        "\t\t\t\t\tAmp. Mean Freq. Wave\n
        Channel 1: \t%2.2f\t%2.2f\t%2.2f\t%d.%s\n
        Channel 2: \t%2.2f\t%2.2f\t%2.2f\t%d.%s\n\n",
        ch[0].amp , ch[0].mean, ch[0].freq, wave[0], wave_str[wave[0]-1],
        ch[1].amp , ch[1].mean, ch[1].freq, wave[1], wave_str[wave[1]-1]);
    f_SaveFile(filename, fp, data);
}

float f_ValidValue(float value, float min, float max)
{
    //make sure value is between min and max
    if (value < min)
        return min;
    else if (value > max)
        return max;
    else
        return value;
}

void f_ReadFile(char *filename, FILE *fp)
{
    int count;
    channel_para temp_ch[2];
    int temp_wave[2];

    strcat(filename, ".txt");
    printf("\nFile reading in progress, please wait...\n");

    if ((fp = fopen(filename, "r")) == NULL)
    {
        printf("Cannot open\n\n");
        return;
    }
    //get variables from file
    count = fscanf(fp, "%*[^C]Channel 1: %f %f %f %d%*[^C] Channel 2: %f %f %f %d",
        &temp_ch[0].amp, &temp_ch[0].mean, &temp_ch[0].freq, &temp_wave[0],
        &temp_ch[1].amp, &temp_ch[1].mean, &temp_ch[1].freq, &temp_wave[1]);
    //if received all 8 values successfully, set values to variables
    if (count == 8)
    {
        int i;
        for (i = 0; i < 2; i++)
        {
            wave[i] = f_ValidValue(temp_wave[i], 1, 4);
            ch[i].amp = f_ValidValue(temp_ch[i].amp, 0, 5);
            ch[i].mean = f_ValidValue(temp_ch[i].mean, 0, 2);
            ch[i].freq = f_ValidValue(temp_ch[i].freq, 0.5, 5);
        }
        printf("File Read Successfully\n\n");
    }
    else
    {
        printf("File Read Fail\n\n");
    }
    fclose(fp);
}

void f_ReadFilePrompt()
{
    char filename[100];
    FILE *fp;
    char data[100];
    printf("\n\nYou have indicated to read the file.
        \nPlease name your file(.txt):
        \n\n0. Return Main Menu\n\n");
    scanf("%s", &filename);

    //return main menu if input = 0
    if(strcmp(filename,"0")==0)
        return;

    f_ReadFile(filename, fp);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Threads
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void *t_Wave1(void *arg)
{
    unsigned int i;
    unsigned int current1[NO_POINT];
    struct timespec start1, stop1;
    double accum1 = 0;

    while (true)
    {
        //Channel 1
        if (clock_gettime(CLOCK_REALTIME, &start1) == -1)
        {
            perror("clock gettime");
            exit(EXIT_FAILURE);
        }
```

```
        for (i = 0; i < NO_POINT; i++)
        {
            current1[i] = (wave_type[wave[0] - 1][i] * ch[0].amp) * 0.6 + (ch[0].mean * 0.8 + 1.2) * 0x7fff / 5 * 2.5;
            // scale + offset
        }
        for (i = 0; i < NO_POINT; i++)
        {
            out16(DA_CTLREG, 0x0a23); // DA Enable, #0, #1, SW 5V unipolar
            out16(DA_FIFOCLR, 0);     // Clear DA FIFO  buffer
            out16(DA_Data, (short)current1[i]);
            delay(((1 / ch[0].freq * 1000) - (accum1)) / NO_POINT);
        }

        if (clock_gettime(CLOCK_REALTIME, &stop1) == -1)
        {
            perror("clock gettime");
            exit(EXIT_FAILURE);
        }

        accum1 = (double)(stop1.tv_sec - start1.tv_sec) + (double)(stop1.tv_nsec - start1.tv_nsec) / BILLION;
    }
}

void *t_Wave2(void *arg)
{
    unsigned int i;
    unsigned int current2[NO_POINT];
    struct timespec start2, stop2;
    double accum2 = 0;

    while (true)
    {

        //Channel 2
        if (clock_gettime(CLOCK_REALTIME, &start2) == -1)
        {
            perror("clock gettime");
            exit(EXIT_FAILURE);
        }

        for (i = 0; i < NO_POINT; i++)
        {
            current2[i] = (wave_type[wave[1] - 1][i] * ch[1].amp) * 0.6 + (ch[1].mean * 0.8 + 1.2) * 0x7fff / 5 * 2.5;
            // scale + offset
        }
        for (i = 0; i < NO_POINT; i++)
        {
            out16(DA_CTLREG, 0x0a43); // DA Enable, #0, #1, SW 5V unipolar
            out16(DA_FIFOCLR, 0);     // Clear DA FIFO  buffer
            out16(DA_Data, (short)current2[i]);
            delay(((1 / ch[1].freq * 1000) - (accum2)) / NO_POINT);
        }

        if (clock_gettime(CLOCK_REALTIME, &stop2) == -1)
        {
            perror("clock gettime");
            exit(EXIT_FAILURE);
        }

        accum2 = (double)(stop2.tv_sec - start2.tv_sec) + (double)(stop2.tv_nsec - start2.tv_nsec) / BILLION;
    }
}

void *t_HardwareInput(void *arg)
{
    int mode;
    unsigned int count;

    while (true)
    {
        dio_in = in8(DIO_PORTA); // Read Port A

        if ((dio_in & 0x08) == 0x08)
        {
            out8(DIO_PORTB, dio_in); // output Port A value -> write to Port B
            if ((dio_in & 0x04) == 0x04)
            {
                raise(SIGINT);
            }
            else if ((mode = dio_in & 0x03) != 0)
            {
                count = 0x00;

                while (count < 0x02)
                {
                    chan = ((count & 0x0f) << 4) | (0x0f & count);
                    out16(MUXCHAN, 0x0D00 | chan); // Set channel     - burst mode off.
                    delay(1);                      // allow mux to settle
                    out16(AD_DATA, 0);             // start ADC

                    while (!(in16(MUXCHAN) & 0x4000));
                    adc_in[(int)count] = in16(AD_DATA);
```

```
                    count++;
                    delay(5); // Write to MUX register - SW trigger, UP, DE, 5v, ch 0-7
                }
            }

            switch ((int)mode)
            {
                case 1:
                ch[0].amp = (float)adc_in[0] * 5.00 / 0xffff; //scale from 16 bits to 0 ~ 5
                ch[1].amp = (float)adc_in[1] * 5.00 / 0xffff; //scale from 16 bits to 0 ~ 5
                break;
                case 2:
                ch[0].freq = (float)adc_in[0] * 4.50 / 0xffff + 0.5; //scale from 16 bits to 0.5 ~ 5
                ch[1].freq = (float)adc_in[1] * 4.50 / 0xffff + 0.5; //scale from 16 bits to 0.5 ~ 5
                break;
                case 3:
                ch[0].mean = (float)adc_in[0] * 2.00 / 0xffff; //scale from 16 bits to 0.00 ~ 2.00
                ch[1].mean = (float)adc_in[1] * 2.00 / 0xffff; //scale from 16 bits to 0.00 ~ 2.00
                break;
            }
        } //if take input from keyboard
        delay(100);
    } //end of while
} //end of thread

void *t_ScreenOutput(void *arg)
{
    delay(100);
    printf("\nTo Exit Program, press Ctrl + C\nTo Enter Keyboard Menu, press Ctrl + \\ \n");
    printf("\nReal Time Inputs are as follow:-\n\n");
    printf("\tChannel 1\t\t\tChannel 2\n");
    printf("Amp.\tMean\tFreq\t\tAmp.\tMean.\tFreq\n");
    while (true)
    {
        printf("\r%2.2f\t%2.2f\t%2.2f\t\t%2.2f\t%2.2f\t%2.2f", ch[0].amp, ch[0].mean, ch[0].freq, ch[1].amp, ch[1].mean, ch[1].freq);
        fflush(stdout);
        delay(100);
    }
}

void *t_UserInterface()
{
    int input;

    //to stop screen output
    if (pthread_cancel(thread[3]) == 0)
        while (true)
        {
            printf("\n\n\nMAIN MENU\nPlease choose your next action:\n\n");
            printf("1. Change Waveform
                \n2. Save and Output File
                \n3. Read File
                \n4. Return to Display
                \n5. End the Program\n\n");
            input = f_GetInt(1, 5);

            if (input == 1){
                f_ChangeWavePrompt();
            }else if (input == 2){
                f_SaveFilePrompt();
            }else if (input == 3){
                if ((dio_in & 0x08) == 0x08)
                {
                    printf("\n\nPlease switch off first toggle switch\n\n");
                    delay(1000);
                }
                else
                    f_ReadFilePrompt();
            }else if (input == 4){
                //clear console screen
                system("clear");
                if (pthread_create(&thread[3], NULL, &t_ScreenOutput, NULL))
                {
                    printf("ERROR; thread \"t_ScreenOutput\" not created.");
                }
                break;
            }else if (input == 5){
                printf("\nBye bye\nHope to see you again soon :p\n");
                raise(SIGINT);
            }
        }
    }

//++++++++++++++++++++++++++++++++
//SIGNAL_HANDLER
//++++++++++++++++++++++++++++++++

    void signal_handler()
    {
        int t;
        pthread_t temp;
```

```
        temp = pthread_self();
        printf("\nHardware Termination Raised\n");

        f_termination();

        for (t = 3; t >= 0; t--)
        {
            if (thread[t] != temp)
            {
                pthread_cancel(thread[t]);
                printf("Thread %d is killed.\n", thread[t]);
            }
    } // for loop
    printf("Thread %d is killed.\n", temp);
    pthread_exit(NULL);
    delay(500);
} // handler

void signal_handler2()
{
    pthread_create(NULL, NULL, &t_UserInterface, NULL);
}

//+++++++++++++++++++++++++++++++++
//MAIN
//+++++++++++++++++++++++++++++++++

int main(int argc, char *argv[])
{
    int j = 0; //thread count
    f_PCIsetup();
    f_Malloc();
    f_WaveGen();
    f_ArgCheck(argc, argv);

    signal(SIGINT, signal_handler);
    signal(SIGQUIT, signal_handler2);

    system("clear");
    printf("\n--------------Initialisation Completed.--------------\n\n");

    if (pthread_create(&thread[j], NULL, &t_HardwareInput, NULL))
    {
        printf("ERROR; thread \"t_HardwareInput\" not created.");
    }
    j++;

    if (pthread_create(&thread[j], NULL, &t_Wave1, NULL))
    {
        printf("ERROR; thread \"t_Wave1\" not created.");
    }
    j++;

    if (pthread_create(&thread[j], NULL, &t_Wave2, NULL))
    {
        printf("ERROR; thread \"t_Wave2\" not created.");
    }
    j++;

    if (pthread_create(&thread[j], NULL, &t_ScreenOutput, NULL))
    {
        printf("ERROR; thread \"t_ScreenOutput\" not created.");
    }
    j++;

    pthread_exit(NULL);
}
```

Appendix C – Group Photo