



Nanyang Technological University

School of Mechanical and Aerospace Engineering

# Major Programming Assignment

MA4830: Realtime Software for Mechatronic Systems

[\*Group picture removed for no misuses 😊]

Lecturer: Prof. Assoc. Prof. Gerald Seet Gim Lee

Written By: Ahladita Neti

Dihan Jannatan Mutaalim

Soh Zheng Da

Steven Edbert Winata

Tan Qian Yan

Wong Jia Ying

Yang Xunsheng

AY SEM 2 2022/2023

# Table of Contents

1. Introduction.....	4
1.1 Background.....	4
1.2 Objective.....	4
1.3 Technical Equipment.....	4
2. Implementation and Usage.....	5
2.1 User Manual.....	5
2.1.1 Program Execution.....	5
2.1.2 Parameters input from PCI Circuit Board.....	7
2.1.3 Keyboard Input.....	7
2.2 Program Highlight.....	10
2.3 Program Limitation.....	11
2.3.1 Hardware Limitation.....	11
2.3.2 Software Limitation.....	11
3. Solution.....	12
3.1 Program Listing.....	12
3.1.1 Program Initialization.....	12
3.1.2 Program Execution.....	13
3.1.3 Program Termination.....	13
3.2 Program Output.....	14
3.2.1 Sine Wave Generator.....	14
3.2.2. Converting functions to output other waveforms.....	15
4. Conclusion.....	15
5. Appendices.....	16
Appendix A – C Source Code.....	16

Appendix B – FlowCharts .....	32
B.1 Program Initialization.....	32
B.1.2 void WaveGeneration() .....	32
B.1.3 void checkArgs(int argc, char* argv[]) .....	33
B.1.4 void MemoryAllocation(void) .....	34
B.1.5 int getInt(int lowlimit, int highlimit) .....	34
B.1.6 float getFloat(float lowlimit, float highlimit) .....	34
B.2 Program Execution .....	35
B.2.1 void changeWaveform().....	35
B.2.2 void changeFrequency() .....	35
B.2.3 void saveyourfile(char *filename, FILE *fp, char *data) .....	36
B.2.4 void saveyourfilePrompt().....	36
B.2.5 void readfile(char *filename, FILE *fp) .....	37
B.2.6 void readFilePrompt() .....	37
B.2.7 void *ThreadWave(void* arg) .....	38
B.2.8 void *ThreadforHardwareInput(void *arg).....	38
B.2.9 void *MainPageOutput(void* arg).....	39
B.2.10 void *userinterface() .....	40
B.2.11 void signalHandler2().....	41
B.2.12 int main(int argc, char* argv[]) .....	41
B.3 Program Termination .....	42
B.3.1 void terminate() .....	42
B.3.2 void signalHandler().....	42

## **Lists of Figures**

Figure 1: (a) PCI Circuit Board, (b) Power Supply, (c) Oscilloscope .....	5
Figure 2: (a)Toggle Switches (b) Potentiometers .....	7
Figure 3: Real-Time Inputs Parameter Table.....	8
Figure 4: Main Menu after Ctrl+Z.....	8
Figure 5: Memory Fault After Hardware Termination .....	12
Figure 6: Different types of waveforms (a) Sine (b) Triangular (c) Square (d) Sawtooth .....	15

## **Lists of Tables**

Table 1: Initial Wave and Param Selection Guide.....	6
Table 2: Min, Max, and Default value Parameters .....	6
Table 3: Initialisation Command Examples.....	6
Table 4: Toggle Switch Settings .....	7
Table 5: Main Menu Input and Output .....	10
Table 6: Sine Wave with various Param Output.....	14

# 1. Introduction

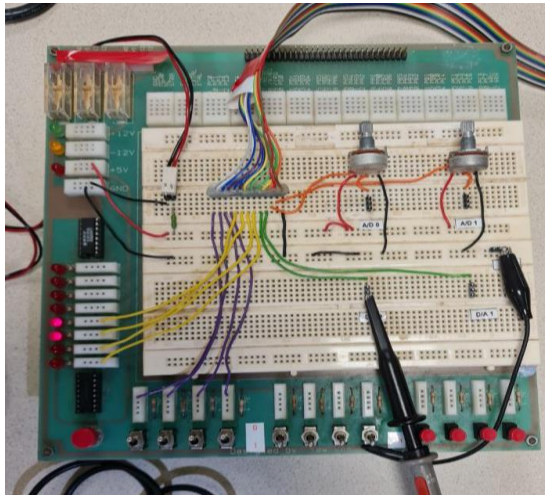
## 1.1 Background

This project is a component of the MA4830 Realtime Software for Mechatronics Systems course. This report acts as a guide for creating a waveform generator using real-time systems in C programming.

## 1.2 Objective

The objective of the assignment is to create a waveform generator and project it to the oscilloscope which allows user to configure waves parameters such as type of waveform, frequency, mean and amplitude. It needs to be capable of generating sine wave, sawtooth wave, triangle wave and square wave with the implementation of Realtime techniques, including multi-threads.

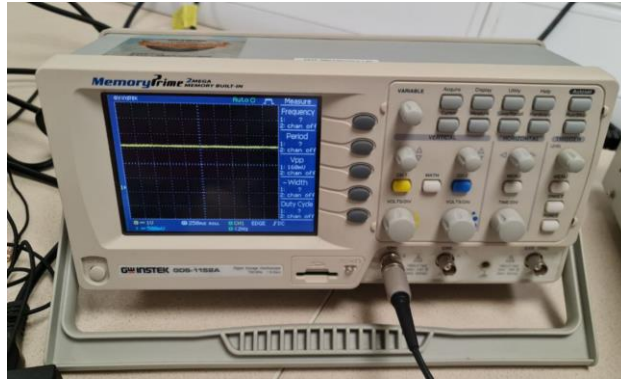
## 1.3 Technical Equipment



(1a)



(1b)



(1c)

Figure 1: (a) PCI Circuit Board, (b) Power Supply, (c) Oscilloscope

The PCI circuit board controls waveform parameters via the switches and potentiometer, while the DC power supply provides a direct current voltage of 5V to electronic devices. An oscilloscope is connected to the PCI circuit board to display waveforms in real-time and is also a vital tool needed for troubleshooting.

## 2. Implementation and Usage

### 2.1 User Manual

#### 2.1.1 Program Execution

The program is executed using these commands in the terminal.

Use this following command to compile the file:

```
cc -lm -o CA2 CA2Final.c
```

Use this following command to run the file:

```
./CA2 -[param][wave_type] -[param][value] -[param][value] -[param][value]
```

Description:

Compulsory Arguments: ./CA2

Optional Arguments: -[param][wave\_type] -[param][value]

Optional arguments can be set with any combination of parameters such as wave type, amplitude, means, and frequency, and the order of these parameters is not significant. Refer to table 3 for command line examples with arguments. Should there be any empty `-[param][value]` argument or invalid input, that respective parameter will be set to default value.

This following table is the key input to define a specific wave and parameters.

	Description	Argument
[wave_type]	Sine Wave	sine
	Square Wave	square
	Sawtooth Wave	saw
	Triangular Wave	tri
[param]	Wave Type	t
	Amplitude	a
	Means	m
	Frequency	f

*Table 1: Initial Wave and Param Selection Guide*

	Parameters	Minimum	Maximum	Default
[value]	Amplitude	0	5	1
	Frequency	0.5	10	1
	Mean	0	1	1
[wave_type]	Wave Type			sine

*Table 2: Min, Max, and Default value Parameters*

These are the examples of the command to run the CA2 file.

Command line examples	Wave Type	Amplitude	Means	Frequency
<code>./CA2 -tsine -m0.3 -f2.5 -a4</code>	Sine	4.0	0.3	2.5
<code>./CA2 -ttri -f12 -m0.4</code>	Triangular	Default = 1.0	0.4	(out of range) change to default = 1

*Table 3: Initialisation Command Examples*

### 2.1.2 Parameters input from PCI Circuit Board

Another capability of this program is allowing the user to configure wave parameters input through PCI circuit board. The usable input from the board consists of four toggle switches and one potentiometer (A/D 0). This potentiometer is used to change the values of specific parameters, and the real-time values will feedback to the console as the values are being changed.

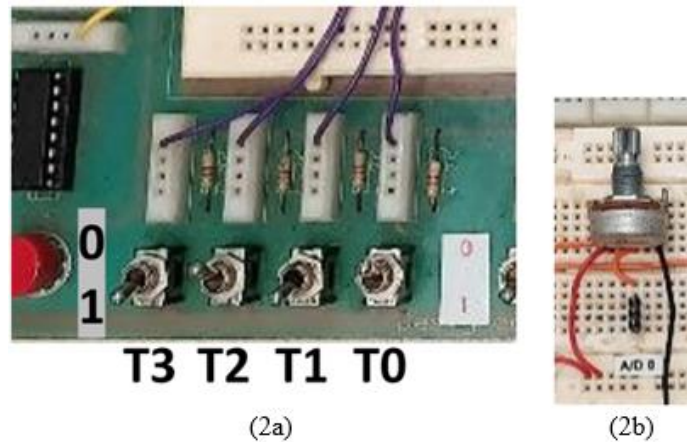


Figure 2: (a)Toggle Switches (b) Potentiometers

Settings	T3	T2	T1	T0
Change Frequency	1	0	0	1
Change Means	1	0	1	1
Change Amplitude	1	0	1	0
Kill all threads	1	1	X	
Disable all inputs	0	X		

Table 4: Toggle Switch Settings

### 2.1.3 Keyboard Input

The program not only enables users to change waveform parameters using the PCI board (Hardware Input), but it also allows them to interact with the system through the keyboard and access additional features. Once the user executes the program and initialise the parameters, it will display the real-time inputs parameter table. The user may choose to press Ctrl + C to exit program or Ctrl + Z to enable keyboard menu.



```

=====CAZ Assignment=====

To Exit Program, press Ctrl + C
To Enter Keyboard Menu, press Ctrl + Z

+-----+
|          WAVEFORM GENERATOR          |
+-----+
|          Real Time Inputs            |
+-----+
| Amp | Mean | Freq |
+-----+
| 2.00 | 0.50 | 9.00 |
+-----+

```

Figure 3: Real-Time Inputs Parameter Table

By hitting Ctrl + Z, the main menu is displayed to select different action.

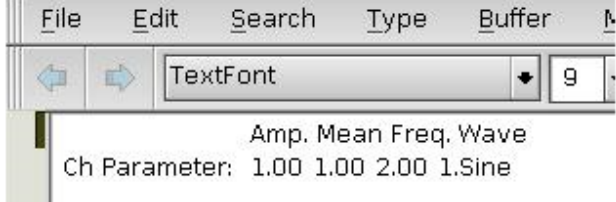
```

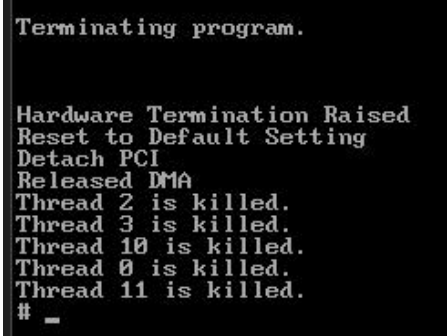
+-----+
|          SETTINGS                    |
+-----+
|          MAIN MENU                  |
| Please select an action (1-5):      |
| 1. Change Waveform Settings        |
| 2. Change Frequency of the wave    |
| 3. Save Current Settings to a File |
| 4. Read a File and Load Settings  |
| 5. Return to Main Page             |
| 6. End the Program                |
+-----+
|
| Enter an integer between 1 and 6:
| _

```

Figure 4: Main Menu after Ctrl+Z

Main Menu Input Choice	Output
1. Change Waveform Type	<p>The user has the options to choose the new type of waveform from the list shown.</p> <pre> Enter an integer between 1 and 6: 1 Select a number 1-4: 1. Sine Wave 2. Square Wave 3. Sawtooth Wave 4. Triangular Wave 0. Return to Main Menu </pre> <p>Any invalid input will result in re-prompting the user.</p> <pre> 6 Invalid input! Please try again. Select a number 1-4: </pre> <p>After selection, new type of wavefrom is updated on the oscilloscope with same parameters.</p>

Main Menu Input Choice	Output
2. Change Frequency of the Wave	<p>Other than PCI board, selecting this option will enable user to change the wave frequency via keyboard.</p> <pre>Enter a float between 0.50 and 10.00 5</pre> <p>Any invalid input will result in re-prompting the user.</p> <pre>Enter a float between 0.50 and 10.00 100 Your number should be within 0.500000 and 10.000000. Please enter a valid number.</pre> <p>After selection, the waveform will be updated with a new frequency.</p>
3. Save Current Settings to a File	<p>This option will prompt user to enter a file name. The program will then save the current waveform parameters into a text file located in the current directory.</p> <pre>You have indicated to save the output to a file. Please enter a filename (.txt extension will be added): 0. Return to Main Menu wavegenfile File saving in progress, please wait... File saved!</pre> 
4. Read a File and Load Settings	<p>Once the selected file is read, the current wave parameters are updated accordingly to the data saved inside the file.</p> <pre>You have indicated to read the file. Please name your file(.txt): 0. Return Main Menu wavegenfile File reading in progress, please wait... File Read Successfully</pre> <p>The program will re-prompt if the file cannot be opened or located.</p> <pre>File reading in progress, please wait... Cannot open file: No such file or directory</pre>
5. Return to Main Page	<p>After selecting this option, it will transport back to the Waveform Generator page where user can see Amp, Freq and Mean again.</p>

Main Menu Input Choice	Output
6. End the Program	<p>The whole program is terminated after each thread is killed accordingly.</p>  <pre> Terminating program.  Hardware Termination Raised Reset to Default Setting Detach PCI Released DMA Thread 2 is killed. Thread 3 is killed. Thread 10 is killed. Thread 0 is killed. Thread 11 is killed. # _ </pre>

*Table 5: Main Menu Input and Output*

## 2.2 Program Highlight

- Modification of Waveform Parameters:
  - a. Waveform type: Alterable using keyboard input through the User Interface.
  - b. Frequency: Changeable via keyboard input from the User Interface or through hardware.
  - c. Amplitude: Adjustable using hardware.
  - d. Mean: Modifiable using hardware.
- File Operations:
  - a. READ and WRITE functions to load and save current parameters in a .txt file.
- User Interface:
  - a. User-friendly interface with prompts for users to re-enter input if invalid data is provided.
  - b. Beeping alert sound triggered for invalid input or error messages.
- Thread and Process Management:
  - a. Effective monitoring of threads from creation to termination.
  - b. Termination of unnecessary threads for improved efficiency.

- Efficiency and Robustness:
  - a. Dynamic memory allocation for new variables when necessary.
  - b. Memory-efficient system with variable cleanup upon termination.
- Command-line Arguments:
  - a. Parsing of several command-line arguments to configure waveform type, amplitude, frequency, and average at program startup.
  - b. Flexibility in the order of argument inputs, allowing users to input arguments in any sequence.

## **2.3 Program Limitation**

### **2.3.1 Hardware Limitation**

- Potentiometer

It is used to modify the wave parameters. A single potentiometer can adjust all wave parameters by toggling the switches. Due to the electrical circuitry of the hardware, a parameter setting will instantly change based on the potentiometer's adjustment as soon as the dedicated toggle switch for that parameter is activated. For example, when a user toggles the switch to adjust the amplitude after altering the frequency, the amplitude will immediately change and adopt the setting previously applied to the frequency.

- Oscilloscope

Users must manually adjust the oscilloscope's screen to zoom in and out. The 'AUTOSET' feature may not function properly, and often, it does not display the expected output. Therefore, users need to adjust the output themselves.

### **2.3.2 Software Limitation**

- SIGINT or Ctrl + C

The use of `raise(SIGINT);` (Ctrl + C) in the `userinterface()` function may lead to a memory fault. When there is a call `raise(SIGINT);` it sends the SIGINT signal to the process. The signal is caught by the `signalHandler()` function.

The `signalHandler()` function is responsible for cancelling all threads except for the current one and then terminating the current thread. However, when `raise(SIGINT);` is called from the `userinterface()` function, the program might not have a chance to exit the `userinterface()` thread properly before it terminates.

To be more specific, the `userinterface()` function might still be executing some code when the `signalHandler()` function starts to cancel threads and terminate the current thread. As a result, the `userinterface()` function may not have a chance to exit gracefully and clean up any resources it was using. This leads to unexpected behaviour or memory faults.



*Figure 5: Memory Fault After Hardware Termination*

## 3. Solution

### 3.1 Program Listing

#### 3.1.1 Program Initialization

Please refer to Appendix A for the full code and Appendix B for the flowchart of each function.

- 3.1.1.1 `void PCISetup()` – Setup the PCI Circuit Board.
- 3.1.1.2 `void WaveGeneration()` – Formula for generating various wave.
- 3.1.1.3 `void checkArgs(int argc, char* argv[])` – Checks if the correct number of arguments is provided when executing the program.
- 3.1.1.4 `void MemoryAllocation(void)` – Allocate memory by using `malloc`.
- 3.1.1.5 `int getInt(int lowlimit, int highlimit)` – Gets integer input from user within a specified range.
- 3.1.1.6 `float getFloat(float lowlimit, float highlimit)` – Gets float input from user within a specified range.

### 3.1.2 Program Execution

Please refer to Appendix A for the full code and Appendix B for the flowchart of each function.

- 3.1.2.1        void changeWaveform() – Function to change waveform by using case.
- 3.1.2.2        void changeFrequency() – Function to change frequency.
- 3.1.2.3        void saveyourfile(char \*filename, FILE \*fp, char \*data) – Enable file saving.
- 3.1.2.4        void saveyourfilePrompt() – Prompt user to save filename.
- 3.1.2.5        void readfile(char \*filename, FILE \*fp) – Read saved file.
- 3.1.2.6        void readFilePrompt() – Enter the filename and read.
- 3.1.2.7        void \*ThreadWave(void\* arg) – Thread for generating wave to oscilloscope.
- 3.1.2.8        void \*ThreadforHardwareInput(void \*arg) – Thread created for hardware input like potentiometer and switches.
- 3.1.2.9        void \*MainPageOutput(void\* arg) – The main page that shows the amp, freq and mean in realtime.
- 3.1.2.10       void \*userinterface() – Setting menu that allows user to change stuff.
- 3.1.2.11       void signalHandler2() – Signal to create the user interface thread.
- 3.1.2.12       int main(int argc, char\* argv[]) – The main program of how different function and thread works together.

### 3.1.3 Program Termination

Please refer to Appendix A for the full code and Appendix B for the flowchart of each function.

- 3.1.2.1        void terminate() – Set the DAC oscilloscope output back to 5V and clear the unused variables.
- 3.1.3.2        void signalHandler() – Kill thread and call terminate()

## 3.2 Program Output

### 3.2.1 Sine Wave Generator

The 16-bit D/A port registers are configured to output voltage and unipolar range.

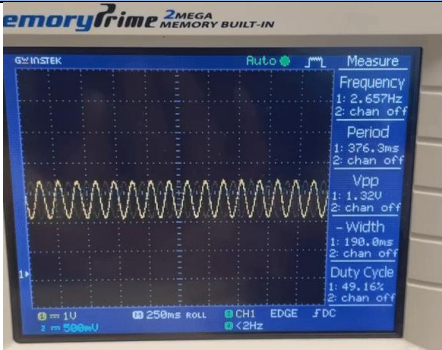
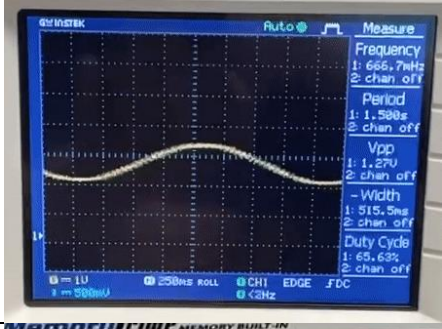
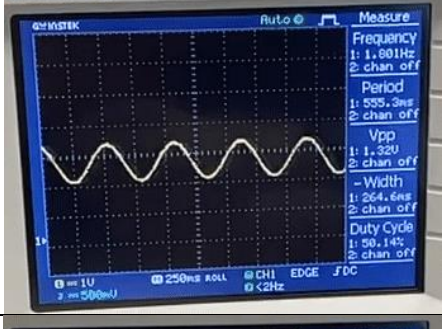
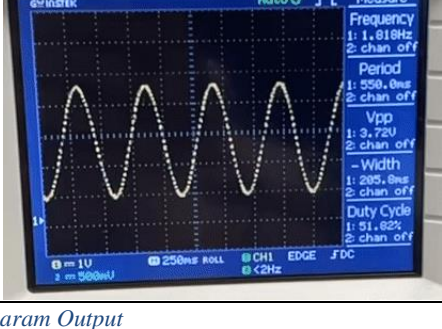
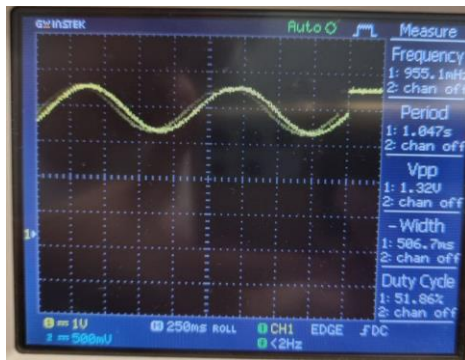
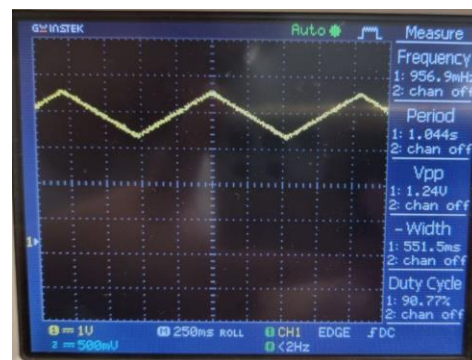
Amplitude	Mean	Frequency	Output
2	1	10	
2	1	0.5	
2	1	2	
5	1	2	

Table 6: Sine Wave with various Param Output

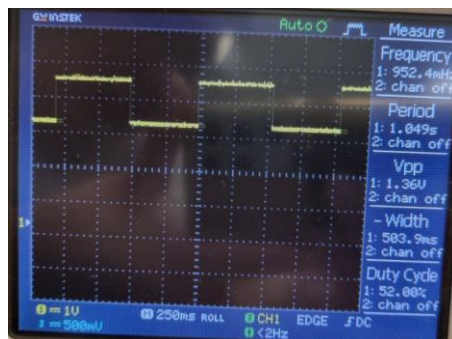
### 3.2.2. Converting functions to output other waveforms.



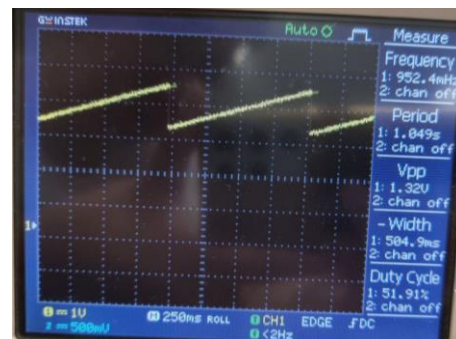
(5a)



(5b)



(5c)



(5d)

Figure 6: Different types of waveforms (a) Sine (b) Triangular (c) Square (d) Sawtooth

## 4. Conclusion

To summarise the features implemented in the project, the wave generator is able to produce sine waves, triangle waves, sawtooth waves and square waves. In addition, the program is presented with a basic UI that is user-friendly and easy to follow. It also displays error messages if the user input is invalid.



# 5. Appendices

## Appendix A – C Source Code

```
#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>

#include <sys/resource.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 numthreads 4
#define steps 50
#define billion 1000000000L

//=====
// Global Variable
//=====
int baddr[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[numthreads];
char Select_c[10];
int Select;
float amplitude=1;
float average=1;
float frequency=1;

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

//relabelling int datatype as wave_pt and wave_pt can store an array of ints of size steps
//so this is just an int array of 50 elements
typedef int wave_pt[steps];    //store points of waveforms with resolution steps

//initialise a pointer for datatype channel_para for variable ch
//to get address of the value stored in ch, printf(ch);
// *ch=1; --> *ch=1, then ch=address where 1 is stored,&ch=address of the pointer reference
channel_para *ch;

//create a 2D int array
wave_pt *wave_type;

//can be initialised as int *wave;
int wave[2];    //store wave type for each channel

//=====
// Functions
//=====

```

```

void PCIssetup(){
    struct pci_dev_info info;
    unsigned int i;
    // set all PCI_dev_info variables to 0 i.e. initialising
    memset(&info,0,sizeof(info));
    if(pci_attach(0)<0) {
        printf("\a");
        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) {
        printf("\a");
        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) {
        printf("\a");
        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 WaveGeneration(){
int i;

float sineTable[steps];
const float delta = (2.0*3.142)/steps;

const float deltaSaw = 2.0/steps;
const float deltaTri = 4.0/steps;

//Sine wave array
for(i=0; i<steps; i++){
    sineTable[i]= (sinf(i*delta));
}
for(i=0; i<steps; i++){
    sineTable[i]= (sinf(i*delta));
    wave_type[0][i] =( sineTable[i] * 0x7fff/5);
}

//Square wave array
// if i less then or equal to half the steps then dummy = 0x7fff else ~0x7fff
for(i=0;i<steps;i++){
    float dummy = (i<=steps/2)?0x7fff:-0x7fff;
    wave_type[1][i] = dummy /5;
}
//Saw-tooth wave array
// if i less then or equal to steps/2 ,(true) i * deltasaw * 0x7fff, (False) (-2 + i * deltaSaw) * 0x7fff
    dummy = (i <= steps / 2) ? i * deltaSaw * 0x7fff : (-2 + i * deltaSaw) * 0x7fff;
    wave_type[2][i] = dummy / 5;
//Triangular wave array
// if i <= steps/4 , (true)i * deltaTri * 0x7fff (else) check if (i <= steps * 3 / 4), (true) (2 - i *
deltaTri) * 0x7fff , (False) (-4 + i * deltaTri) * 0x7fff
    dummy = (i <= steps / 4) ? i * deltaTri * 0x7fff :
        (i <= steps * 3 / 4) ? (2 - i * deltaTri) * 0x7fff :
        (-4 + i * deltaTri) * 0x7fff;
    wave_type[3][i] = dummy / 5;
}

}

void checkArgs(int argc, char* argv[]){
    int i;

```

```

char argument;
char* argument_value;
printf("Reading command line arguments...\n");

if(argc>=2){
    for (i = 1; i < argc; i++) {
        argument = argv[i][1];
        argument_value = &(argv[i][2]);

        switch (argument) {
            case 't':
                if (strcmp(argument_value, "sine") == 0) {
                    printf("\n%d. Sine wave chosen.\n", i);
                    wave[0] = 1;
                }
                else if (strcmp(argument_value, "square") == 0) {
                    printf("\n%d. Square wave chosen.\n", i);
                    wave[0] = 2;
                }
                else if (strcmp(argument_value, "saw") == 0) {
                    printf("\n%d. Saw wave chosen.\n", i);
                    wave[0] = 3;
                }
                else if (strcmp(argument_value, "tri") == 0) {
                    printf("\n%d. Triangular wave chosen.\n", i);
                    wave[0] = 4;
                }
                delay(500);
                break;
            case 'a':
                amplitude = atof(argument_value);
                if(amplitude < 0.0 || amplitude > 5.0){
                    printf("\a");
                    printf("\nERROR: Amplitude must be between 0 and 5.\n");
                    printf("\nDefault amplitude is set to 1 instead\n");
                    amplitude = 1.00;
                }
                printf("\n%d. Amplitude: %f\n", i, amplitude);
                delay(500);
                break;
            case 'f':
                frequency = atof(argument_value);
                if(frequency < 0.5 || frequency > 10.0){
                    printf("\a");
                }
                printf("\nERROR: Frequency must be between 0.5 and 10.");
        }
    }
}

```

```

printf("\nDefault frequency is set to 1 instead\n");
frequency = 1.00;
}
printf("\n%d. Frequency: %f\n", i, frequency);
delay(500);
break;
case 'm':
    average = atof(argument_value);
    if(average < 0.0 || average >1.0){
        printf("\a");
        printf("\nERROR: Mean must be between 0 and 1.");
        printf("\nDefault mean is set to 1 instead\n");
        average = 1.00;
    }
    printf("\n%d. Mean: %f\n", i, average);
    delay(500);
    break;
default:
    printf("\a");
    printf("\nInvalid argument: %s\n", argv[i]);
    break;
}
}
}
if (argc==1) { //if no terminal argument is input, set sine wave as default waveform
    wave[0]=1;
    printf("No arguments are inserted. Values are set to default\n");
}
delay(2000);
}

void MemoryAllocation(void){
    // makes sure there is the space to store in pointer Ch
    if((ch = (channel_para*)malloc(1 * sizeof(channel_para))) == NULL) {
        printf("\a");
        printf("Not enough memory.\n");
    }
    (*ch).amp = amplitude;
    (*ch).mean = average;
    (*ch).freq = frequency;
    // makes sure there is 4x the space to store in pointer wave_type for the 4 different wave generated.
    if((wave_type = (wave_pt*)malloc(4 * sizeof(wave_pt))) == NULL) {
        printf("\a");
        printf("Not enough memory.\n");
        exit(1);
    }
}

```

```

}
}

//Reset DAC to 5V
void terminate(){
    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);

    // unassign the memory of pointers ch and wave_type.
    free((void *) ch);
    free((void *) wave_type);
    printf("Reset to Default Setting\nDetach PCI\nReleased DMA\n");
}

//number checker to ensure that it is within the limits.
int getInt(int lowlimit, int highlimit) {
    int outnum;
    char c;

    while (1) {
        printf("\nEnter an integer between %d and %d: \n", lowlimit, highlimit);
        if (scanf("%d", &outnum) != 1) {
            // The input is not a valid integer
            while ((c = getchar()) != '\n' && c != EOF) {}//loop forever until its not enter or EOF
            printf("\a");
            printf("Invalid input. Please enter an integer.\n");
            continue; // restart the loop in while
        }
        if (outnum < lowlimit || outnum > highlimit) {
            printf("\a");
            printf("Your number should be within %d and %d. Please enter a valid number.\n", lowlimit,
highlimit);
            continue; //restart the loop in while
        }
        while ((c = getchar()) != '\n' && c != EOF) {}//loop forever until its not enter or EOF
        return outnum;
    }
}

```

```

}

float getFloat(float lowlimit, float highlimit) {
    float outnum1 = 0.0;
    char c;
    printf("\nEnter a number between %.2f and %.2f\n", lowlimit, highlimit);
    while (1) {
        if (scanf("%f", &outnum1) != 1) {
            // The input is not a valid float
            while ((c = getchar()) != '\n' && c != EOF) {} //loop forever until its not enter or EOF
            printf("\a");
            printf("Invalid input. Please enter a number.\n");
            continue; // restart the loop in while
        }
        if (outnum1 < lowlimit || outnum1 > highlimit) {
            printf("\a");
            printf("Your number should be within %f and %f. Please enter a valid number.\n", lowlimit,
highlimit);
            continue; //restart the loop in while
        }
        while ((c = getchar()) != '\n' && c != EOF) {} //loop forever until its not enter or EOF
        return outnum1;
    }
}

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

    int Select;

    while (1) {
        printf("Select a number 1-4:\n"
            "1. Sine Wave\n"
            "2. Square Wave\n"
            "3. Sawtooth Wave\n"
            "4. Triangular Wave\n\n"
            "0. Return to Main Menu\n\n");

        if (scanf("%d", &Select) != 1) {
            // The input is not a valid integer
            char c;
            while ((c = getchar()) != '\n' && c != EOF) {}
            printf("\a");
            printf("Invalid input. Please enter an integer.\n");
            continue;
        }
    }
}

```



```

    }

    switch (Select) {
        case 0:
            return; // Return to main menu
        case 1:
        case 2:
        case 3:
        case 4:
            wave[0] = Select;
            printf("\n%s Wave Selected\n\n", wave_str[wave[0]-1]);
            return; // Return to main menu
        default:
            printf("\a");
            printf("Invalid input! Please try again.\n");
    }
}

}

void changeFrequency() {
    float temp_freq;
    temp_freq = getFloat(0.5, 10.0);
    (*ch).freq = temp_freq;
}

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

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

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

```



```

        wave[0] = wavetemp;
        (*ch).amp = chtemp.amp;
        (*ch).mean = chtemp.mean;
        (*ch).freq = chtemp.freq;
        printf("File Read Successfully\n\n");
    } else {
        printf("\a");
        printf("File Read Fail\n\n");
    }

    fclose(fp);
}

void readFilePrompt() {
    char filename[100];
    FILE *fp;

    printf("\n\nYou have indicated to read the file."
        "\nPlease name your file(.txt):\n\n"
        "0. Return Main Menu\n\n");

    scanf("%s", filename);

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

    readfile(filename, fp);
}

//=====
// P_Threads
//=====

void *ThreadWave(void* arg) {

    unsigned int current[steps];
    struct timespec start, stop;
    double accum = 0;
    unsigned int i;

    while (true) {
        // Generate wave data for the current channel
        for (i = 0; i < steps; i++) {
            current[i] = (wave_type[wave[0]-1][i] * (*ch).amp)*0.6 + ((*ch).mean*0.8 + 1.2)*0x7fff/5 * 2.5 ;

```

```

        // Critical Formula
        // Scale and offset the wave data
    }

    // Send the wave data to the DA converter
    if (clock_gettime(CLOCK_REALTIME, &start) == -1) {
        printf("\a");
        perror("clock_gettime");
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < steps; i++) {
        out16(DA_CTLREG, 0x0a23);    // DA Enable, #0, #1, SW 5V unipolar
        out16(DA_FIFOCLR, 0);        // Clear DA FIFO buffer
        out16(DA_Data, (short)current[i]);
        delay((1.0 / (*ch).freq*1000 - accum) / steps);
    }
    // Critical Formula
    }

    if (clock_gettime(CLOCK_REALTIME, &stop) == -1) {
        printf("\a");
        perror("clock_gettime");
        exit(EXIT_FAILURE);
    }

    accum = (double)(stop.tv_sec - start.tv_sec) + (double)(stop.tv_nsec - start.tv_nsec) / billion;
}

void *ThreadforHardwareInput(void *arg){

    int mode;
    unsigned int count;

    struct timespec delay_time = {0,100000000}; //100ms delay

    while(1) {
        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.
    nanosleep(&delay_time, NULL);    // allow mux to settle
    out16(AD_DATA,0);            // start ADC
    while(!(in16(MUXCHAN) & 0x4000));
    adc_in[(int)count]=in16(AD_DATA);
    count++;
    nanosleep(&delay_time, NULL);    // Write to MUX register - SW trigger, UP, DE, 5v, ch 0-7
}
}

switch ((int)mode) {
case 1:
    (*ch).freq = (float)adc_in[0] * 9.5 / 0xffff + 0.5; //scale from 16 bits to 0.5 ~ 10
    break;
case 2:
    (*ch).amp = (float)adc_in[0] * 5.00 / 0xffff; //scale from 16 bits to 0 ~ 5
    break;
case 3:
    (*ch).mean = (float)adc_in[0] * 1.00 / 0xffff; //scale from 16 bits to 0.00 ~ 1.00
    break;
}
} // end if keyboard
nanosleep(&delay_time, NULL);
} //end while
} //end thread

void *MainPageOutput(void* arg){
    delay(100);
    printf("\nTo Exit Program, press Ctrl + C\nTo Enter Keyboard Menu, press Ctrl + Z \n");
    printf("+-----+\n");
    printf("|          WAVEFORM GENERATOR          |\n");
    printf("+-----+\n");
    printf("|          Real Time Inputs          |\n");
    printf("+-----+\n");
    printf("|    Amp    |    Mean    |    Freq    |\n");
    printf("+-----+\n");
    while(1){
        printf("\r|    %6.2f    |    %6.2f    |    %6.2f    |\n", ch[0].amp , ch[0].mean, ch[0].freq);
        printf("+-----+\033[A");
        fflush(stdout);
        delay(100);
    }
}

```

```

    }

}

void *userinterface() {
    int input;
    bool running = true;

    // Stop a possible bug by cancelling the screen output thread
    if (pthread_cancel(thread[2]) == 0) {
        while (running) {
            printf("\n");
            printf("\n+-----+");
            printf("\n|                SETTINGS                |");
            printf("\n+-----+");
            printf("\n|                MAIN MENU                |");
            printf("\n| Please select an action (1-6):         |");
            printf("\n| 1. Change Waveform Settings           |");
            printf("\n| 2. Change Frequency of the wave       |");
            printf("\n| 3. Save Current Settings to a File    |");
            printf("\n| 4. Read a File and Load Settings     |");
            printf("\n| 5. Return to Main Page                |");
            printf("\n| 6. End the Program                    |");
            printf("\n+-----+\n\n");

            input = getInt(1, 6);

            switch (input) {
                case 1:
                    changeWaveform();
                    break;
                case 2:
                    changeFrequency();
                    break;
                case 3:
                    saveyourfilePrompt();
                    break;
                case 4:
                    if ((dio_in & 0x08) == 0x08) {
                        printf("\a");
                        printf("\n\nPlease switch off first toggle switch\n\n");
                        delay(1000);
                    }
                    else {
                        readFilePrompt();
                    }
                }
            }
        }
    }
}

```

```

        }
        break;
    case 5:
        // Clear console screen
        system("clear");
        if (pthread_create(&thread[2], NULL, &MainPageOutput, NULL)) {
            printf("\a");
            printf("ERROR: thread \"MainPageOutput\" not created.");
        }
        running = false;
        break;
    case 6:
        printf("\nTerminating program.\n");
        raise(SIGINT);
        break;
    }
}
}

//=====
//signalHandler
//=====

void signalHandler() {
    int i;
    printf("\n\nHardware Termination Raised\n");

    terminate();

    // Cancel all threads except for the current one
    for (i = 0; i < numthreads; i++) {
        if (pthread_self() != thread[i]) {
            pthread_cancel(thread[i]);
            printf("Thread %ld is killed.\n", thread[i]);
        }
    }
    // Exit the current thread
    printf("Thread %ld is killed.\n", pthread_self());
    pthread_exit(NULL);
}

void signalHandler2(){
    pthread_create(NULL, NULL, &userinterface, NULL);

```

```

}

//=====
//MAIN
//=====

int main(int argc, char* argv[]) {

    struct rlimit rlim;
    int j=0; //thread count
    rlim.rlim_cur=0;
    rlim.rlim_max=0;
    setrlimit(RLIMIT_CORE,&rlim);
    PCIsetup();
    checkArgs(argc, argv);
    MemoryAllocation();
    WaveGeneration();

    signal(SIGINT, signalHandler);

    signal(SIGTSTP, signalHandler2);

    system("clear");
    printf("\n=====CA2 Assignment=====\\n\\n");

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

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

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

    pthread_exit(NULL);
}

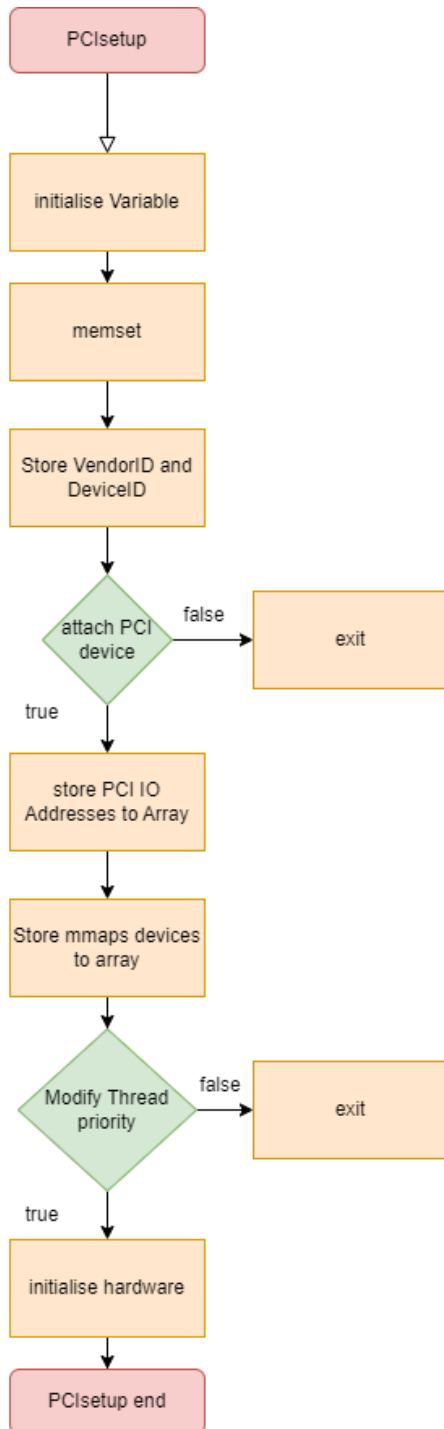
```



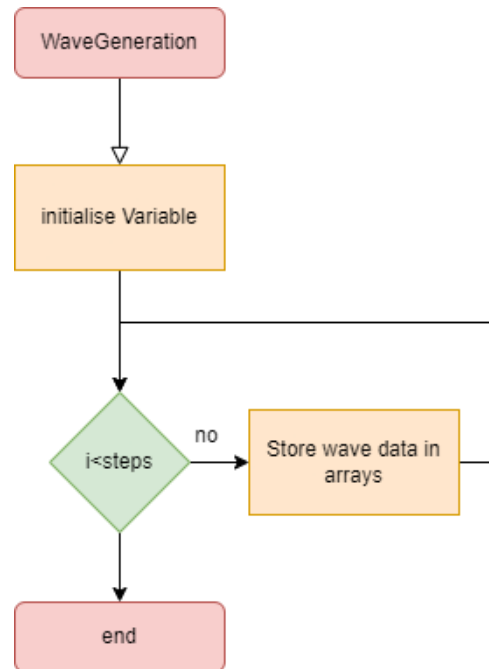
## Appendix B – FlowCharts

### B.1 Program Initialization

#### B.1.1 void PCISetup()



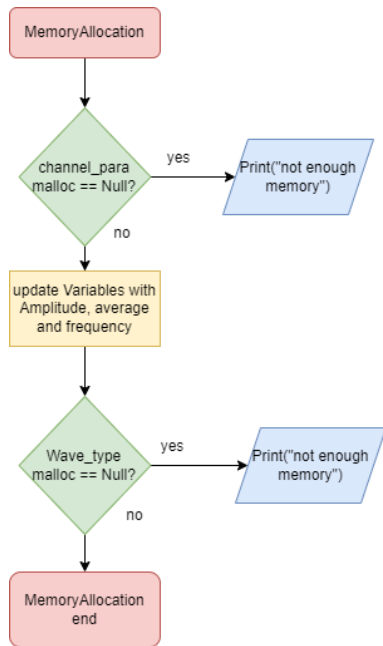
#### B.1.2 void WaveGeneration()



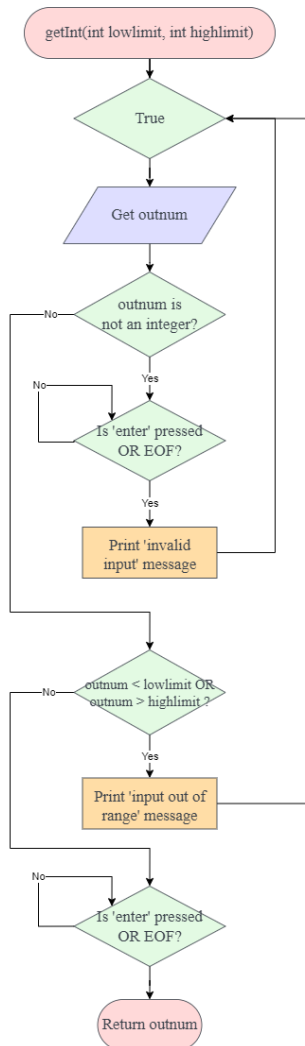
### B.1.3 void checkArgs(int argc, char\* argv[])



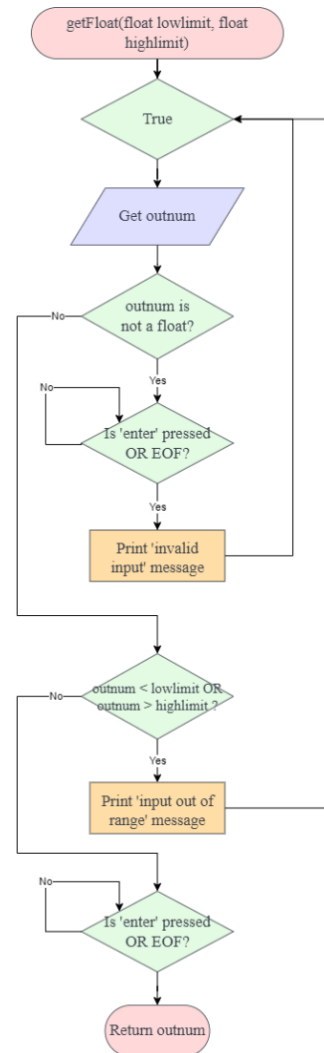
B.1.4 void  
MemoryAllocation(void)



B.1.5 int getInt(int  
lowlimit, int highlimit)

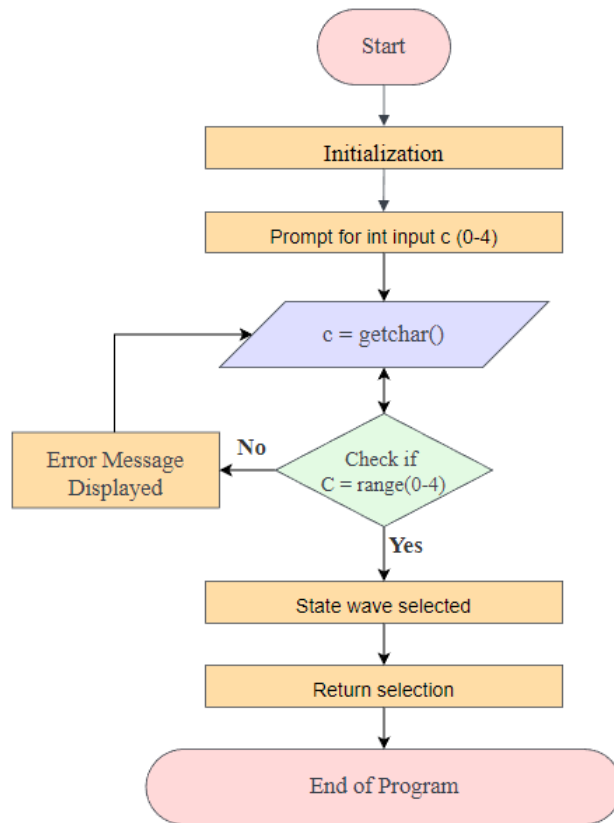


B.1.6 float getFloat(float  
lowlimit, float highlimit)

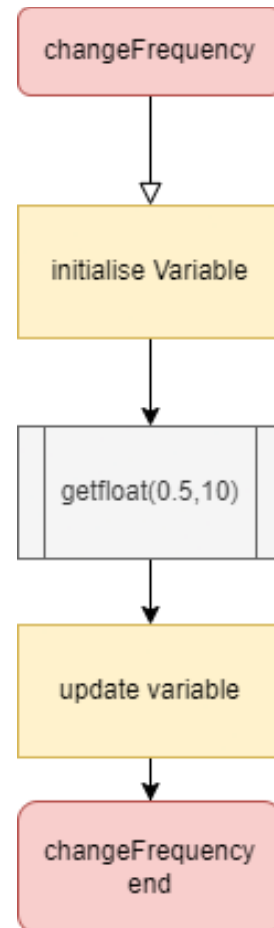


## B.2 Program Execution

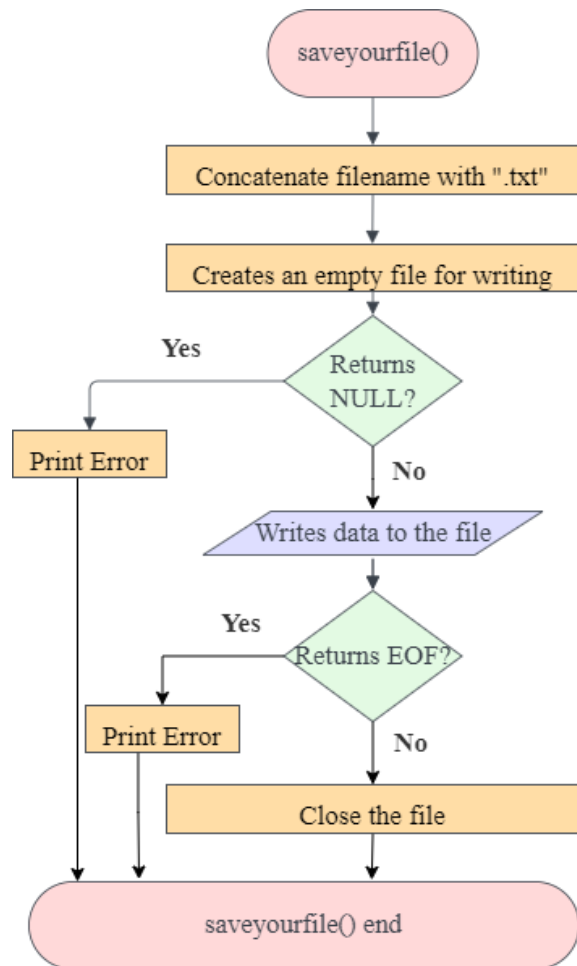
### B.2.1 void changeWaveform()



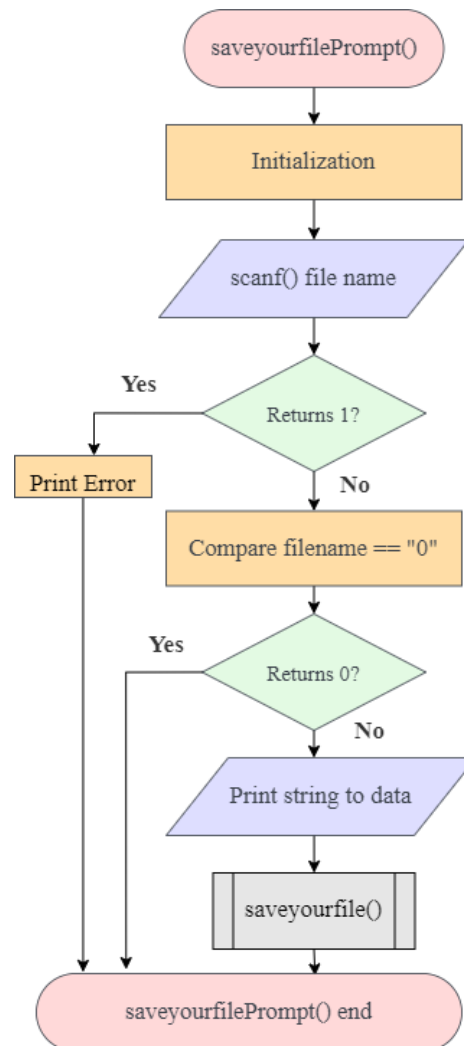
### B.2.2 void changeFrequency()



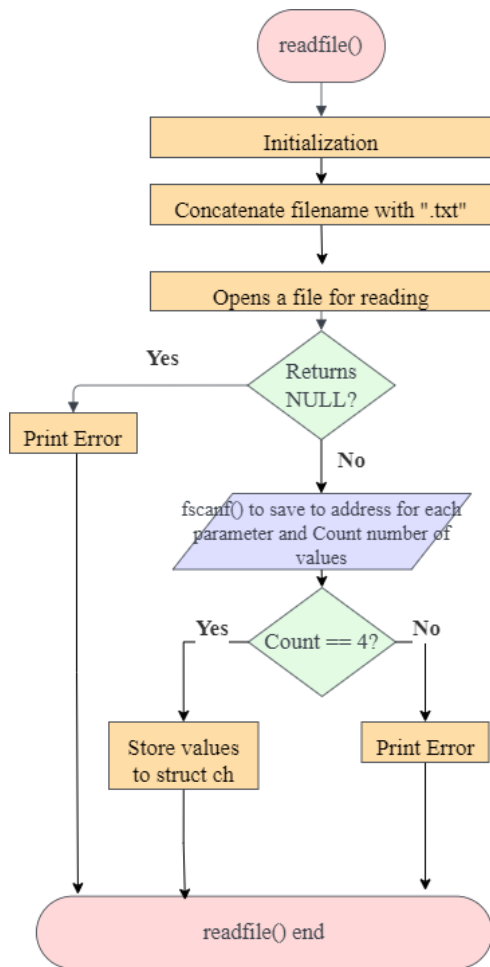
B.2.3 void saveyourfile(char \*filename,  
FILE \*fp, char \*data)



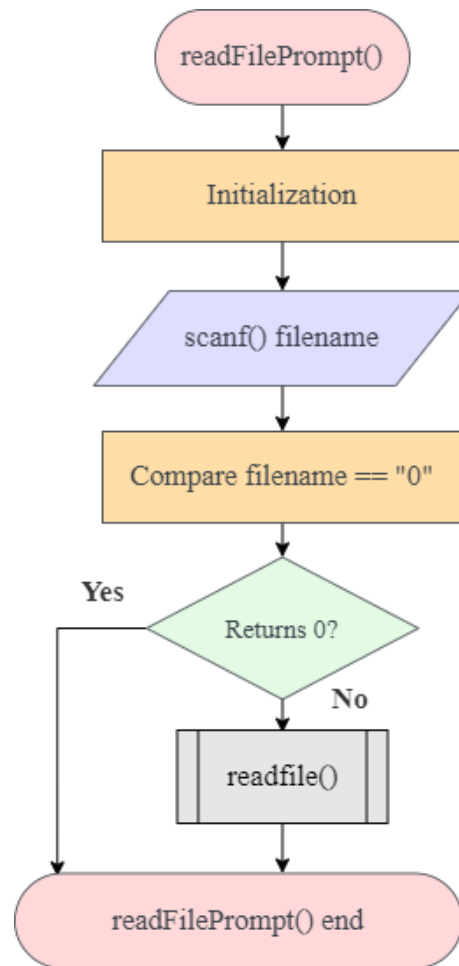
B.2.4 void saveyourfilePrompt()



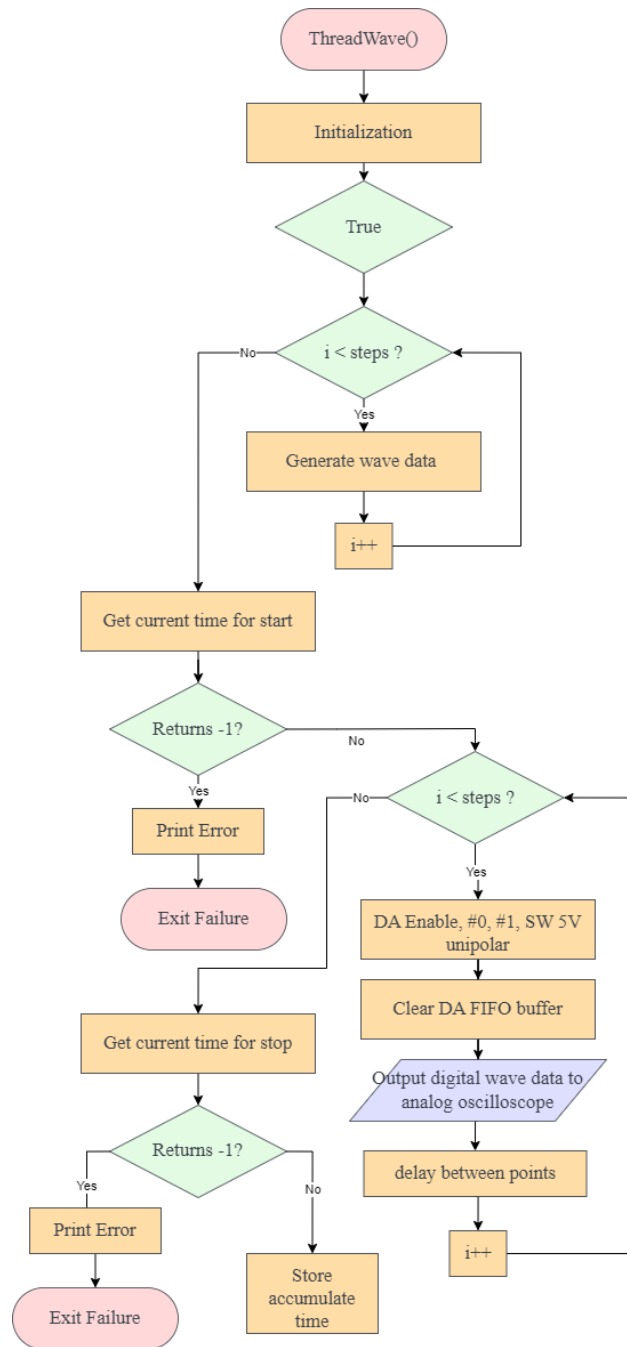
B.2.5 void readfile(char \*filename, FILE \*fp)



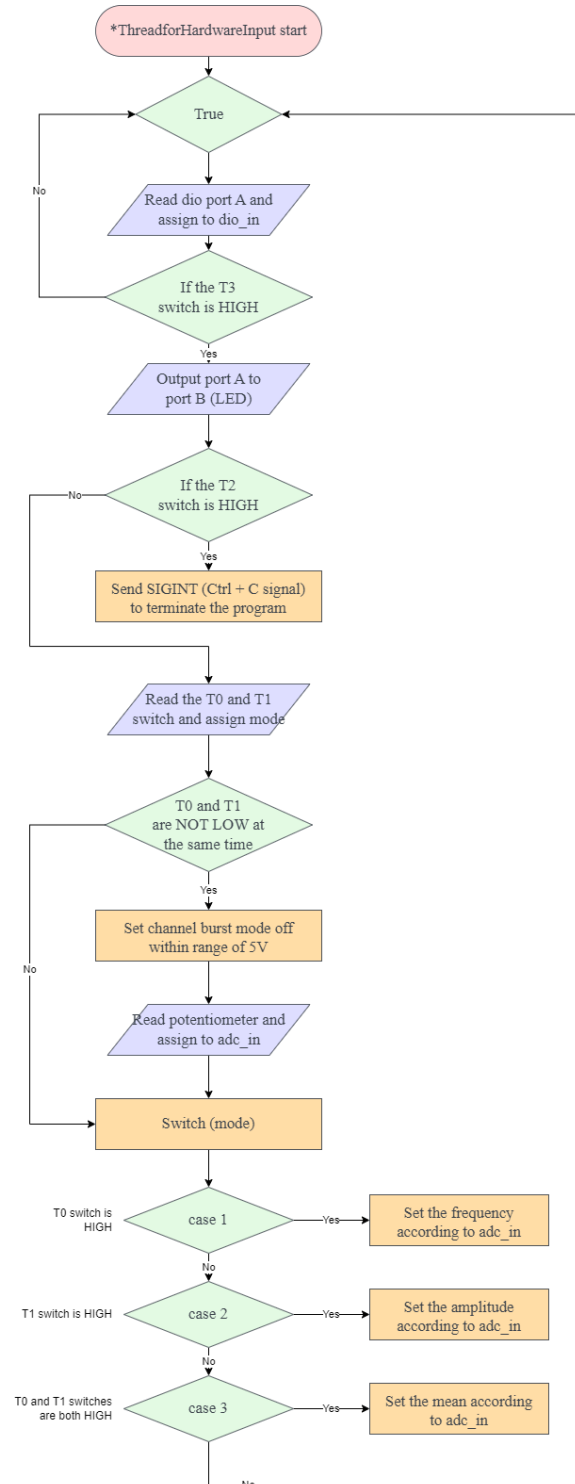
B.2.6 void readFilePrompt()



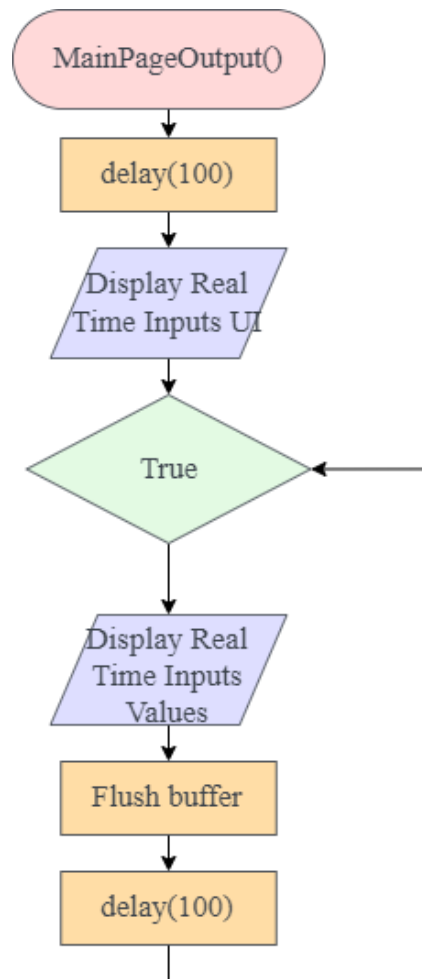
### B.2.7 void \*ThreadWave(void\* arg)



### B.2.8 void \*ThreadforHardwareInput(void \*arg)

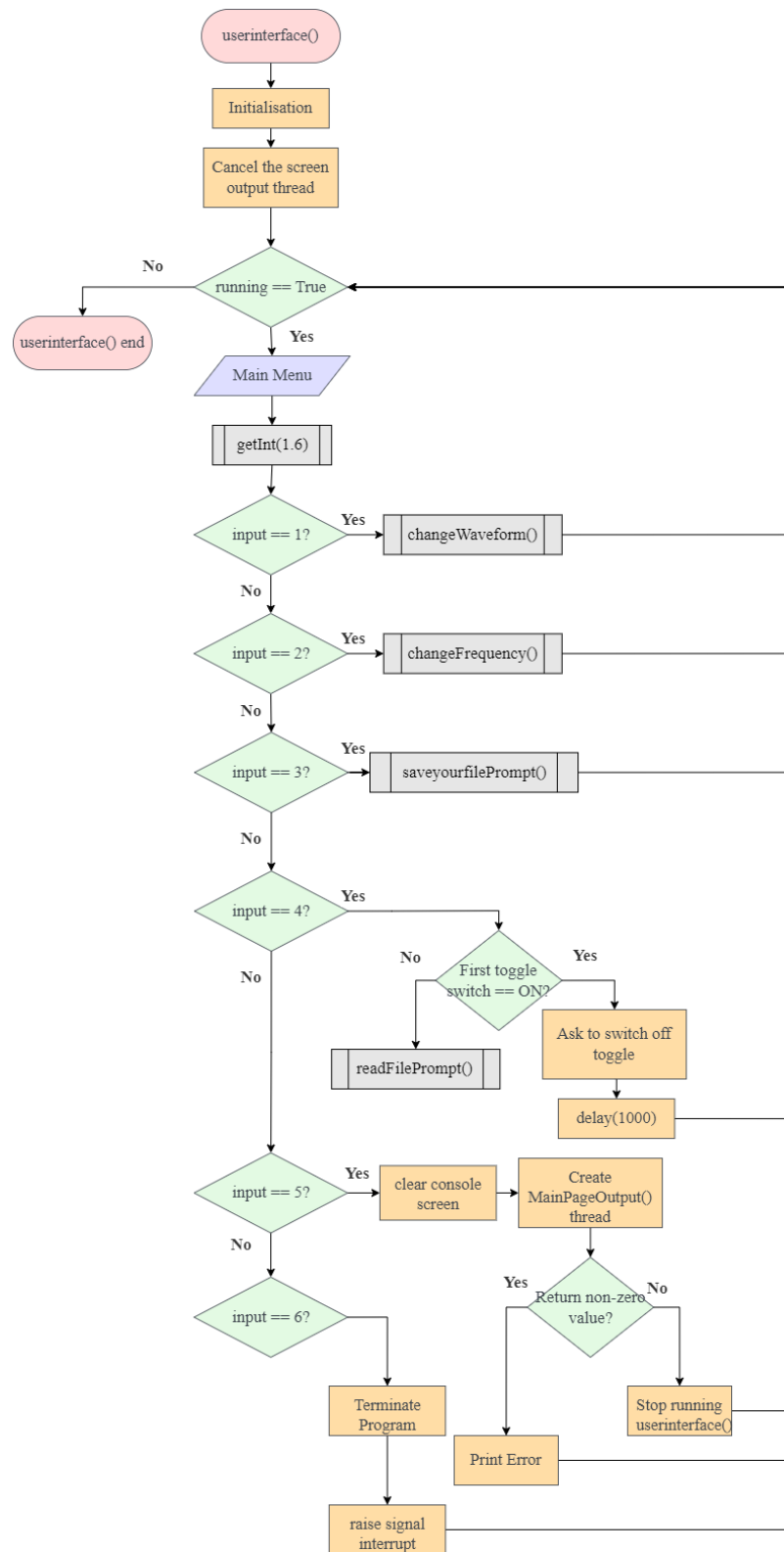


B.2.9 void \*MainPageOutput(void\* arg)

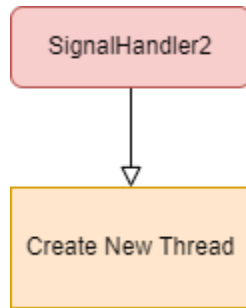




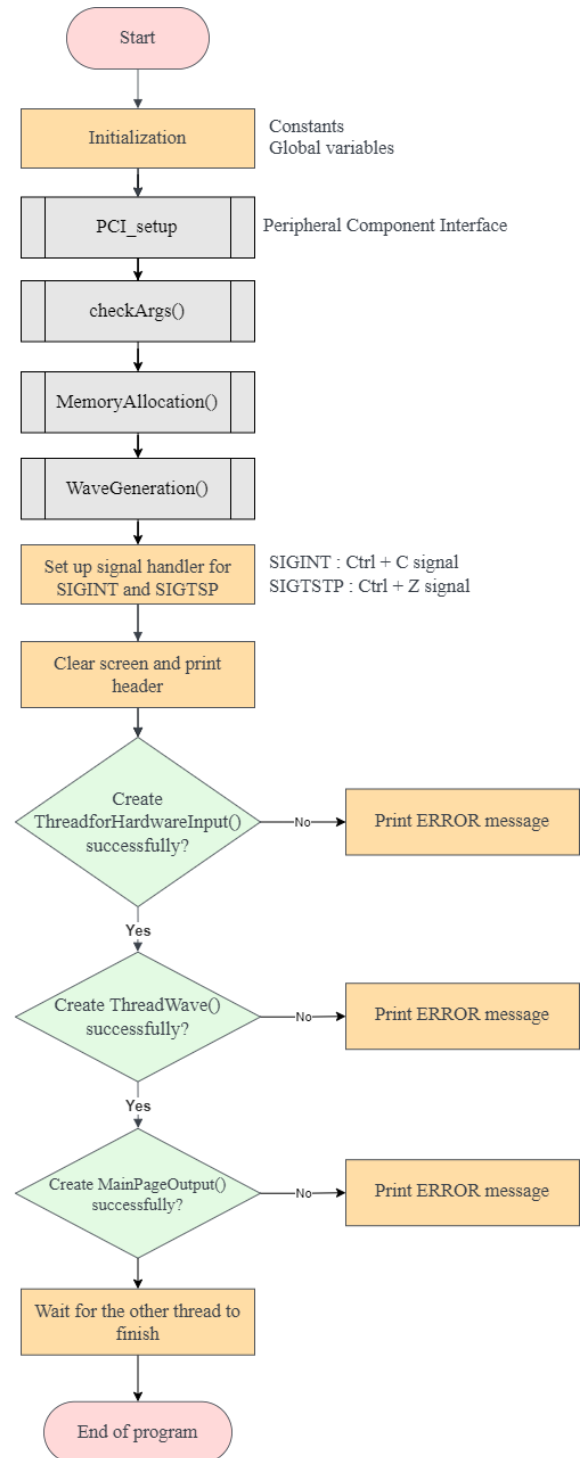
## B.2.10 void \*userinterface()



### B.2.11 void signalHandler2()

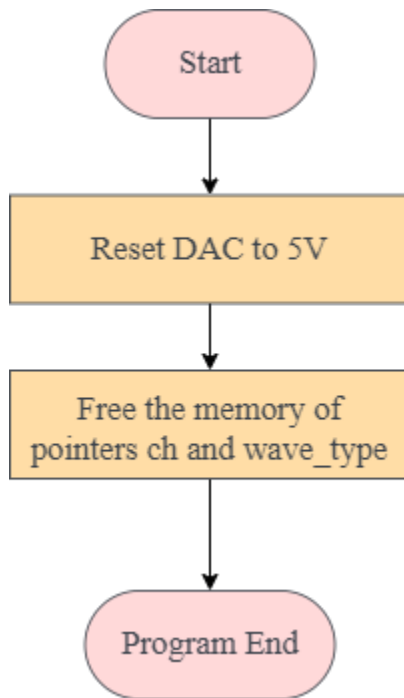


### B.2.12 int main(int argc, char\* argv[])



## B.3 Program Termination

### B.3.1 void terminate()



### B.3.2 void signalHandler()

