CSE211 Introduction to Embedded Systems

Project Documentation

Submitted to:

Prof. Dr. Sherif Hammad

Eng. Mohamed Tarek

Submitted by:

Team 1

Youssef George Fouad		19P9824
Kerollos Wageeh Youssef		19P3468
Anthony Amgad Fayek		19P9880
Mostafa Nasrat Metwally		19P4619
Andrew Hany Emil		19P5685
CESS	G2 S3	Senior-1

Table of Contents

1. APPLICATION ARCHITECTURE

The program is built in an embedded layered architecture that consists of three layers, where each of them only communicates with the layer right under it and is divided as follows.

1.1. Microcontroller Abstraction Layer (MCAL)

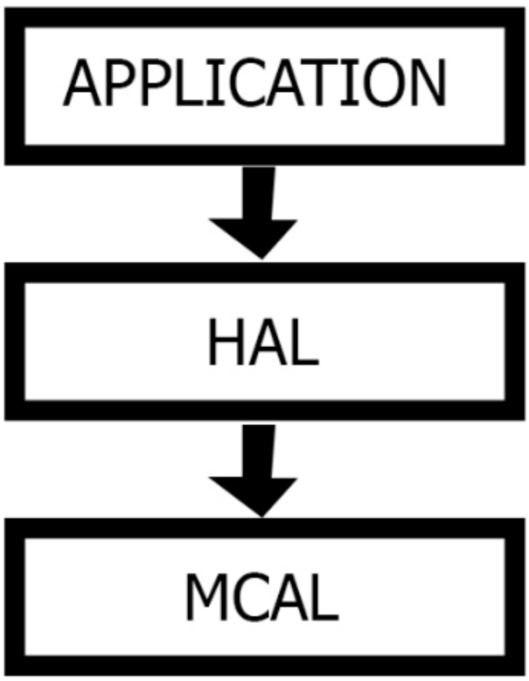
This layer resembles the lowest layer of our architecture where we have the firmware for all the microcontroller peripherals like DIO, timers, NVIC, and interrupt handling. This layer is totally handled by the Tivaware library that includes all the needed functions implemented in addition to files “SysTimer.h”, “SysTimer.c”, “Timers.h”, “Timers.c” written by us to implement delay functions for both systick timer and general-purpose timers respectively.

1.2. Hardware Abstraction Layer (HAL)

This layer has the drivers for all the external hardware parts in our system, which are the LCD screen, Keypad, 4 push buttons and a buzzer. The drivers’ implementation is found in files “LCD.h”, “LCD.c”, “Keypad.h”, “Keypad.c”, “Buttons.h”, and “Buttons.c”.

1.3. Application Layer

The highest layer of our system’s architecture including the software implementation of our 3-parts application logic, which can be found in the “main.c” file.



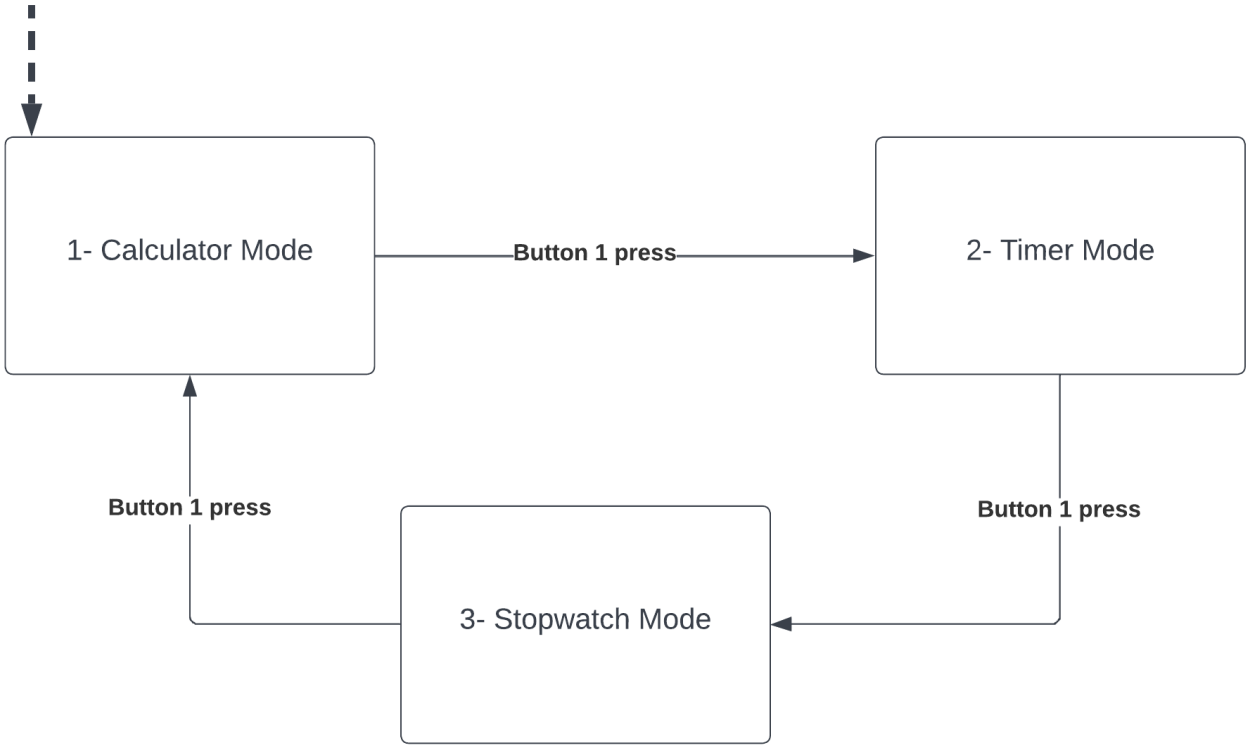
2. Application Overview

2.1. Flow of Operation

The embedded system consists of three applications, calculator, timer, and stopwatch between which the user can switch by pressing button 1 provided on the breadboard.

2.2. General Flowchart

This overview flowchart showcases the three operation modes of our embedded system and how pressing button 1 switches between the three sub-applications. Each sub-application flow of operation is provided in detail in chapters 4,5,6.



2.3. Common Implementation Details

2.3.1. Switching between applications

Switching between applications is an interrupt-based functionality, implemented in ***BtnHandlers()*** function in "main.c" file, checks which button was pressed and act upon. Button 1 connected to pin 2 on Port A is used to switch the ***state*** variable from one application mode to the other.

On switching to **state** = 1 which is the calculator app, the calculator apps variables are reset.

On switching to **state** = 2 or **state** = 3, no action is taken as both apps should continue running in background at all times.

```
void btnHandlers(void){
 switch(GPIOIntStatus(GPIO_PORTA_BASE,GPIO_INT_PIN_2)){

 case(GPIO_INT_PIN_2): //switch
  switch(state){
  case 1:
   opermode = 0;
   calciter = 0;
   oper1len = 0;
   oper1 = 0;
   memset(calcin, 0, sizeof(calcin));
   state = 2;
   break;
  case 2:
   state = 3;
   break;
  case 3:
   state = 1;
   break;
  }
  while((GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_2) == GPIO_PIN_2)); // for bouncing
  SysTimerDelay(50*16000); // for bouncing
  break;
```

2.3.2. Delay functions

Displaying the welcoming message of the whole embedded systems, "Team 1 CSE211 Project", in addition to each of the welcoming messages of each of the three applications require a delay that will allow them to appear on screen for a specified time.

Timer0 of the GPTM module was utilized in an interrupt-based manner and three functions were implemented in files "Timers.h" and "Timers.c".

- **TimerDelayInit():** initializes the timer0 configurations to be used when needed and sets the Tim0Handler function as the interrupt handler function.

```
void TimerDelayinit(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_WTIMER0);
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_WTIMER0));
  IntMasterEnable();
  IntEnable(INT_WTIMER0A);
  TimerIntEnable(WTIMER0_BASE,TIMER_TIMA_TIMEOUT);
  TimerIntRegister(WTIMER0_BASE,TIMER_A,Tim0Handler);
  TimerConfigure(WTIMER0_BASE,TIMER_CFG_ONE_SHOT);
}
```

- **TimerDelay(uint32_t delay):** loads timer0 with the needed duration and starts the countdown, blocking the system for some time.

```
void TimerDelay(uint32_t delay){
  TimerDisable(WTIMER0_BASE,TIMER_A);
  TimerLoadSet(WTIMER0_BASE,TIMER_A,delay*16000);
  TimerIntClear(WTIMER0_BASE,TIMER_TIMA_TIMEOUT);
  TimerEnable(WTIMER0_BASE,TIMER_A);
  while(Tim0delayflag);
  Tim0delayflag = true;
}
```

- **Tim0Handler():** executes when the timer0 counting down of the required duration elapses, unblocking the system by setting the flag variable false.

```
void Tim0Handler(void){
  Tim0delayflag = false;
  TimerIntClear(WTIMER0_BASE,TIMER_TIMA_TIMEOUT);
  return;
}
```

2.3.3. System initialization

On power up, all mentioned external hardware, GPTM timers 0,1,2, and GPIO ports A, B, C, E are initialized in the start of the main function before the while(1) loop. Also, a welcoming message is displayed.

```
int main()
{
  //Inititions
  uint16_t prevstate = 0;
  LCD_init();
  KeyPad_Init();
  TimerDelayinit();
  btns_init();
```

```
 GPIOIntRegister(GPIO_PORTA_BASE,btnHandlers);

GPIOIntEnable(GPIO_PORTA_BASE,GPIO_INT_PIN_2|GPIO_INT_PIN_3|GPIO_INT_PIN_4|GPIO_INT_PIN_5);

 SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
 while(!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER1));

 IntEnable(INT_TIMER1A);
 TimerIntEnable(TIMER1_BASE,TIMER_TIMA_TIMEOUT);
 TimerIntRegister(TIMER1_BASE,TIMER_A,Timer1IntHandler);
 TimerConfigure(TIMER1_BASE,TIMER_CFG_PERIODIC);
 TimerLoadSet(TIMER1_BASE,TIMER_A,16000000);

 SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
 while(!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER2));

 IntEnable(INT_TIMER2A);
 TimerIntEnable(TIMER2_BASE,TIMER_TIMA_TIMEOUT);
 TimerIntRegister(TIMER2_BASE,TIMER_A,Timer2IntHandler);
 TimerConfigure(TIMER2_BASE,TIMER_CFG_PERIODIC);
 TimerLoadSet(TIMER2_BASE,TIMER_A,16000000);

 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
 while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));

 IntPrioritySet(INT_TIMER1A,0xE0);
 IntPrioritySet(INT_TIMER2A,0xE0);
 IntPrioritySet(INT_GPIOA,0xE0);

 GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2|GPIO_PIN_3);

 // Intro Screen
 uint8_t ch = '\0';
 uint8_t sw = 0;
 GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,~GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_2));
 LCD_Print("     Team 1     "," CSE211 Project ");
 TimerDelay(3000);
 GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,~GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_2));
```

3. External Hardware

3.1. LCD Screen

Our 16x2 LCD is connected to Port B pins of the TivaC microcontroller. As previously mentioned, the file “LCD.h” declares the needed functions for dealing with the LCD screen and “LCD.c” defines them to support the correct logic needed. Seven functions are included as follows.

- **LCD_init ():** initiates our LCD screen to work in 4 bytes mode using pins 4,5,6,7 to send one half of a byte at a time. The function clears the character memory, initiates the cursor, and convert the operation mode from 8 to 4-bit mode.
- **LCD_command (unsigned char c):** sends byte commands to the screen.
- **LCD_Show ():** shows character where cursor stands.
- **LCD_Clear ():** totally clears the LCD screen.
- **LCD_Print (char *s, char *d):** takes two strings, one for each row. sets cursor position then shows character, before hiding the cursor.
- **LCD_PrintLn (char i, char *s):** choose 1 specific line to print on.
- **LCD_Cursor (char x, char y):** set cursor position on any row.

3.2. KEYPAD

Our 4x4 Keypad is connected to ports C, E pins of the TivaC microcontroller. Files “Keypad.h” and “Keypad.c” implements 2 functions needed to deal with the external hardware.

- **KeyPad_Init():** initiates our keypad by setting up ports C, E.
- **KeyPad_Read():** maps each user press to one of the 16 pre-defined characters (0-9), (A-D), (*), (#), and returns null character (\0) on long user presses.

3.3. Buttons

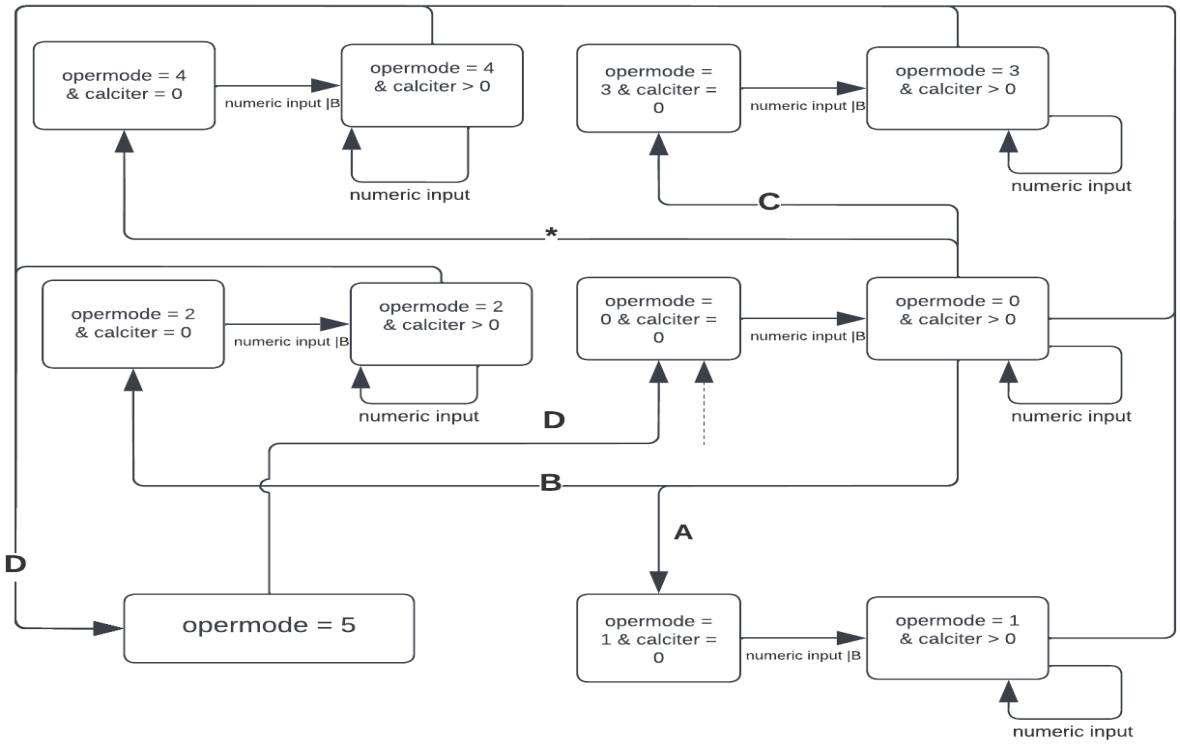
Four push buttons were needed to complete the project, where they were all connected to port A pins of the TivaC microcontroller and pulled down to input 1 on user presses. Function “btns_init()” declared in “Buttons.h” and defined in “Buttons.c” implements the required initializations of the buttons.

4. CALCULATOR MODE

4.1. Calculator Flow of Operation

The first mode on which the system initially starts is the calculator where the user can perform any two-operands operations from the four basic arithmetic ones: addition, subtraction, multiplication, division.

1- Entering this mode, the LCD screen shows welcoming text “Calculator Mode”
2- User inputs first operand, a positive or a negative number, where the negative sign “-“ is the “B” on the keypad.
3- If the user enters an operator (÷, x, +, =) before the first number, it is ignored.
4- The user then enters the operator on the keypad, where “A” on the keypad is “+”, “C” is “÷”, and “*” is the multiplication operator.
5- If the user presses “=” before entering the second operand, no action is taken.
6- The user then enters the second operand, before pressing “D” key on the keypad which denotes the “=” sign to calculate the result.
7- The result is displayed on the LCD screen.
8- The user can press “=” to reset the calculator to its initial state.



Where operModes 0 - 5 denotes the following:

0- Wait for first operand.
1- Wait for second operand of addition operation.

2- Wait for second operand of subtraction operation.
3- Wait for second operand of division operation.
4- Wait for second operand of multiplication operation.
5- Results Displayed.

4.2. Implementation Details

```
while(state == 1){  // checking KeyPad
      ch = KeyPad_Read();

      if ((ch != '\0') && sw == 0){
        if(ch == 'A'){   // +
          if(calciter != 0 && opermode == 0){
            oper1len = calciter - 1;
            calciter = 0;
            oper1 = atoll(calcin);
            memset(calcin, 0, sizeof(calcin));
            opermode = 1;
            LCD_Show('+');
          }
        }else if(ch == 'B'){   // -
          if(calciter != 0  && opermode == 0){
            oper1len = calciter - 1;
            calciter = 0;
            oper1 = atoll(calcin);
            memset(calcin, 0, sizeof(calcin));
            opermode = 2;
            LCD_Show('-');
          }else if(calciter == 0){ // for operand sign
            if(calciter < 14 - oper1len){
            calcin[calciter] = '-';
            calciter++;
            LCD_Show('-');
            }else{
            LCD_Print("#   OVERFLOW   #","#      !!      #");
            TimerDelay(2000);
            LCD_Clear();
            opermode = 0;
            calciter = 0;
            oper1len = 0;
            oper1 = 0;
            memset(calcin, 0, sizeof(calcin));
            }
          }
        }else if(ch == 'C'){   // /
          if(calciter != 0 && opermode == 0){
            oper1len = calciter - 1;
            calciter = 0;
            oper1 = atoll(calcin);
            memset(calcin, 0, sizeof(calcin));
            opermode = 3;
            LCD_Show('/');
```

```
    }
  }else if(ch == '*'){   // *
    if(calciter != 0 && opermode == 0){
      oper1len = calciter - 1;
      calciter = 0;
      oper1 = atoll(calcin);
      memset(calcin, 0, sizeof(calcin));
      opermode = 4;
      LCD_Show('X');
    }
  }else if(ch == 'D'){   // =
    if(calciter != 0){
      if(opermode == 0){  // only one operand
        LCD_PrintLn(1, calcin);
      }else if(opermode == 1){  // +
        int64_t result = oper1 + atoll(calcin);
        int32_t length = snprintf( NULL, 0, "%lld", result);
        char* str = malloc( length + 1 );
        snprintf(str, length + 1, "%lld", result );
        LCD_PrintLn(1,str);
        free(str);
      }
      else if(opermode == 2){ // -
        int64_t result = oper1 - atoll(calcin);
        int32_t length = snprintf( NULL, 0, "%lld", result);
        char* str = malloc( length + 1 );
        snprintf(str, length + 1, "%lld", result );
        LCD_PrintLn(1,str);
        free(str);
      }
      else if(opermode == 3){ // /
        double result = (double)oper1 / (double)atoll(calcin);
        int32_t length = snprintf( NULL, 0, "%lf", result);
        char* str = malloc( length + 1 );
        snprintf(str, length + 1, "%lf", result );
        LCD_PrintLn(1,str);
        free(str);
      }
      else if(opermode == 4){ // *
        int64_t result = oper1 * atoll(calcin);
        int32_t length = snprintf( NULL, 0, "%lld", result);
        char* str = malloc( length + 1 );
        snprintf(str, length + 1, "%lld", result );
        LCD_PrintLn(1,str);
        free(str);
      }else if(opermode == 5){ // if result is displayed already
        LCD_Clear();
        opermode = 0;
        calciter = 0;
        oper1len = 0;
        oper1 = 0;
```

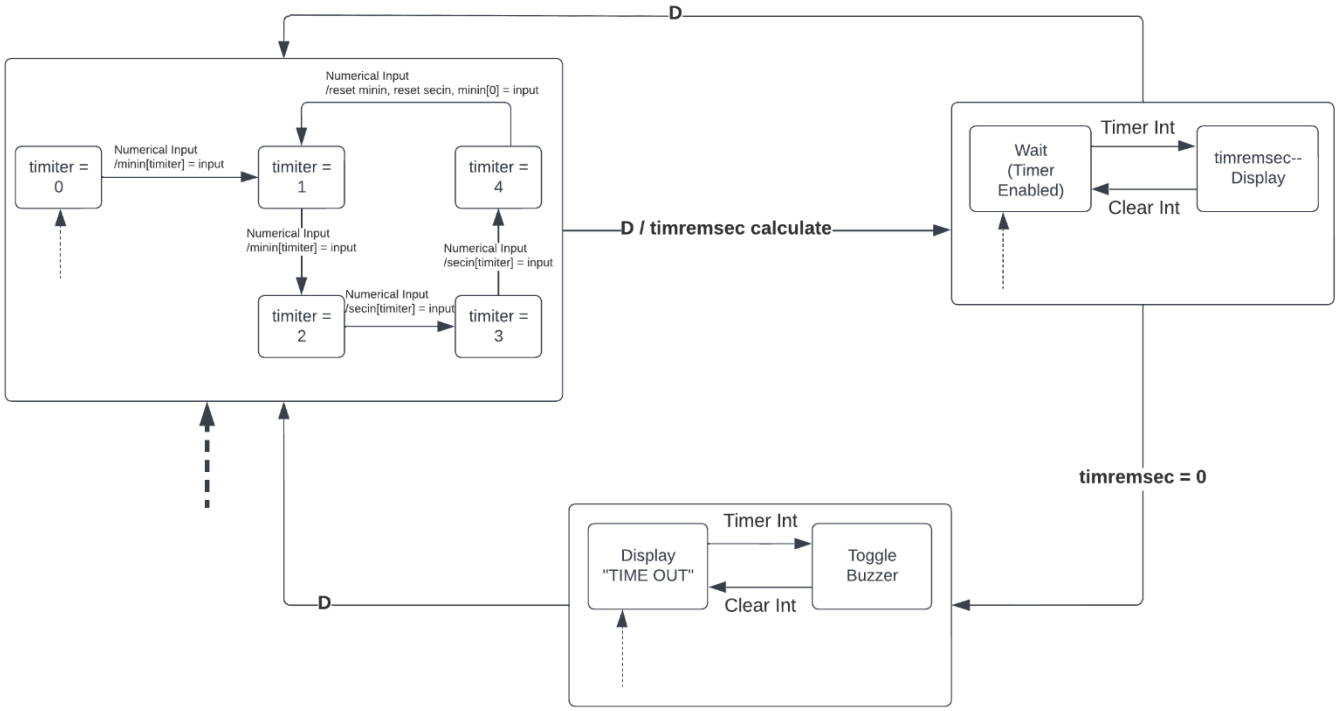
```
            memset(calcin, 0, sizeof(calcin));
            break;
          }
          oper1len = 0;
          calciter = 1;
          oper1 = 0;
          memset(calcin, 0, sizeof(calcin));
          opermode = 5;
        }
      }else{ //entering an operand normally
        if((calciter < 14 - oper1len)  && (ch != '#')){
          calcin[calciter] = ch;
          calciter++;
          LCD_Show(ch);
        }else if(ch != '#'){
          LCD_Print("#   OVERFLOW   #","#      !!      #");
          TimerDelay(2000);
          LCD_Clear();
          opermode = 0;
          calciter = 0;
          oper1len = 0;
          oper1 = 0;
          memset(calcin, 0, sizeof(calcin));
        }
      }
      sw = 1;
      break;
    }
  }
```

5. TIMER MODE

5.1. Timer Flow of Operation

The timer application allows the user to input a timer value in minutes and seconds and countdown the seconds from that value until reaching zero.

1- Entering this mode, the LCD screen shows welcoming text “Timer Mode”
2- User inputs value in minutes and seconds using the keypad (4 digits).
3- User presses the “D” key on the keypad to start the countdown.
4- The LCD screen updates every second to show the latest countdown value.
5- When it reaches “0:0”, the buzzer starts beeping and the green LED of the TivaC microcontroller starts toggling to alert the user that the countdown is over.
6- User represses “D” key to stop LED light and the buzzer beeping. The timer resets and returns to the initial state.



5.2. Implementation Details

Initially, the system allows the user to input the required value through the keypad up to 4 digits.

```
while(state == 2){ //checking keypad
      ch = KeyPad_Read();

      if ((ch != '\0') && sw == 0){
        if(timmode == 0){
          if((timiter < 2)  && (ch != '#') && (ch != '*') && (ch != 'A') && (ch != 'B') && (ch != 'C') && (ch != 'D')){ // first 2
digits (minutes)
            minin[timiter] = ch;
            timiter++;
            LCD_Show(ch);
            if(timiter == 2){
              LCD_Show(':');
            }
          }else if((timiter < 4) && (ch != '#') && (ch != '*') && (ch != 'A') && (ch != 'B') && (ch != 'C') && (ch != 'D')){ //
second 2 digits (seconds)
            secin[timiter-2] = ch;
            timiter++;
            LCD_Show(ch);
          }
          else if((timiter >= 4)  && (ch != '#') && (ch != '*') && (ch != 'A') && (ch != 'B') && (ch != 'C') && (ch != 'D')){ //
cycling
            timiter = 0;
            minin[0] = '0';
            minin[1] = '0';
            secin[0] = '0';
            secin[1] = '0';
            minin[timiter] = ch;
            LCD_PrintLn(0,"00:00");
            LCD_Cursor(0,0);
            LCD_Show(ch);
            timiter++;
          }
```

On user input “D” after entering the duration value, Timer2 of the GPTM module is enabled and the timer starts counting down.

```
          else if(ch == 'D'){ // start
            timremsec = atoll(secin);
            timremsec += (60*(atoll(minin)/100));
            timmode = 1;
            TimerEnable(TIMER2_BASE,TIMER_A);
          }
```

On user input “D” during timer counting process or on timeout, timer is reset, and buzzer and green LED are stopped.

```
}else if((timmode == 1 || timmode == 2) && ch == 'D'){ // reseting
          TimerDisable(TIMER2_BASE,TIMER_A);
          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,0x0);
          timmode = 0;
          minin[0] = '0';
          minin[1] = '0';
          secin[0] = '0';
          secin[1] = '0';
          timiter = 0;
          timremsec = 0;
          LCD_Print("00:00          "," ");
          LCD_Cursor(0,0);
        }
```

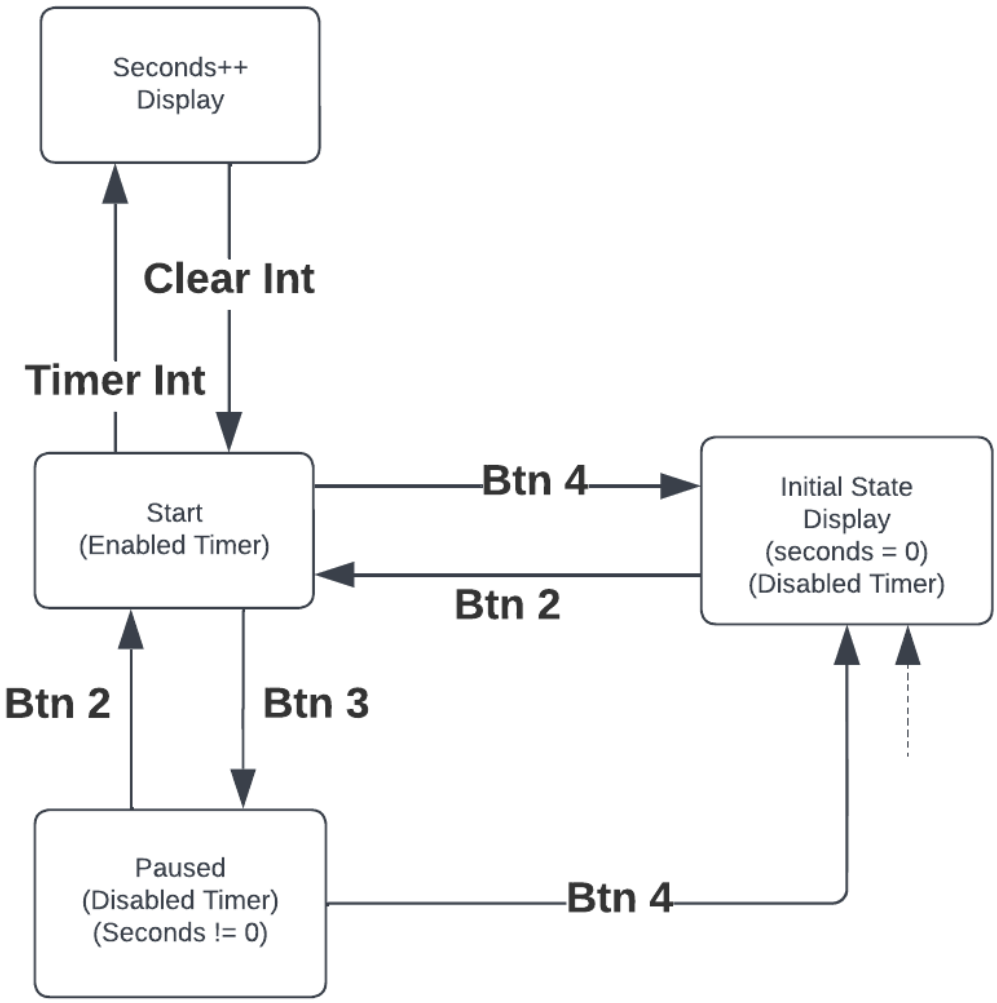
6. STOPWATCH MODE

6.1. Stopwatch Flow of Operation

The stopwatch allows the user to count time in seconds, with the aid of the three push buttons 2,3,4 provided on the breadboard to control the counting operation.

1- Initially, the stopwatch is set to 00:00 and is disabled until user presses button 2 to start.
2- As the counting starts, the LCD displays time increment periodically every second.
3- If the user presses button 3, stopwatch is paused until he presses button 2 to continue.
4- If at any given time the user presses button 4, time is reset back to 0.

In addition, the stopwatch continues counting even if the user switches the calculator mode or the timer mode, and he can check the stopwatch count by switching back to the timer mode



6.2. Implementation Details

Stopwatch application is based upon the GPTM module in TivaC where it utilizes the whole general purpose timer1 32-bits to update the screen every second.

Interrupt-based implementation including 2 main functions:

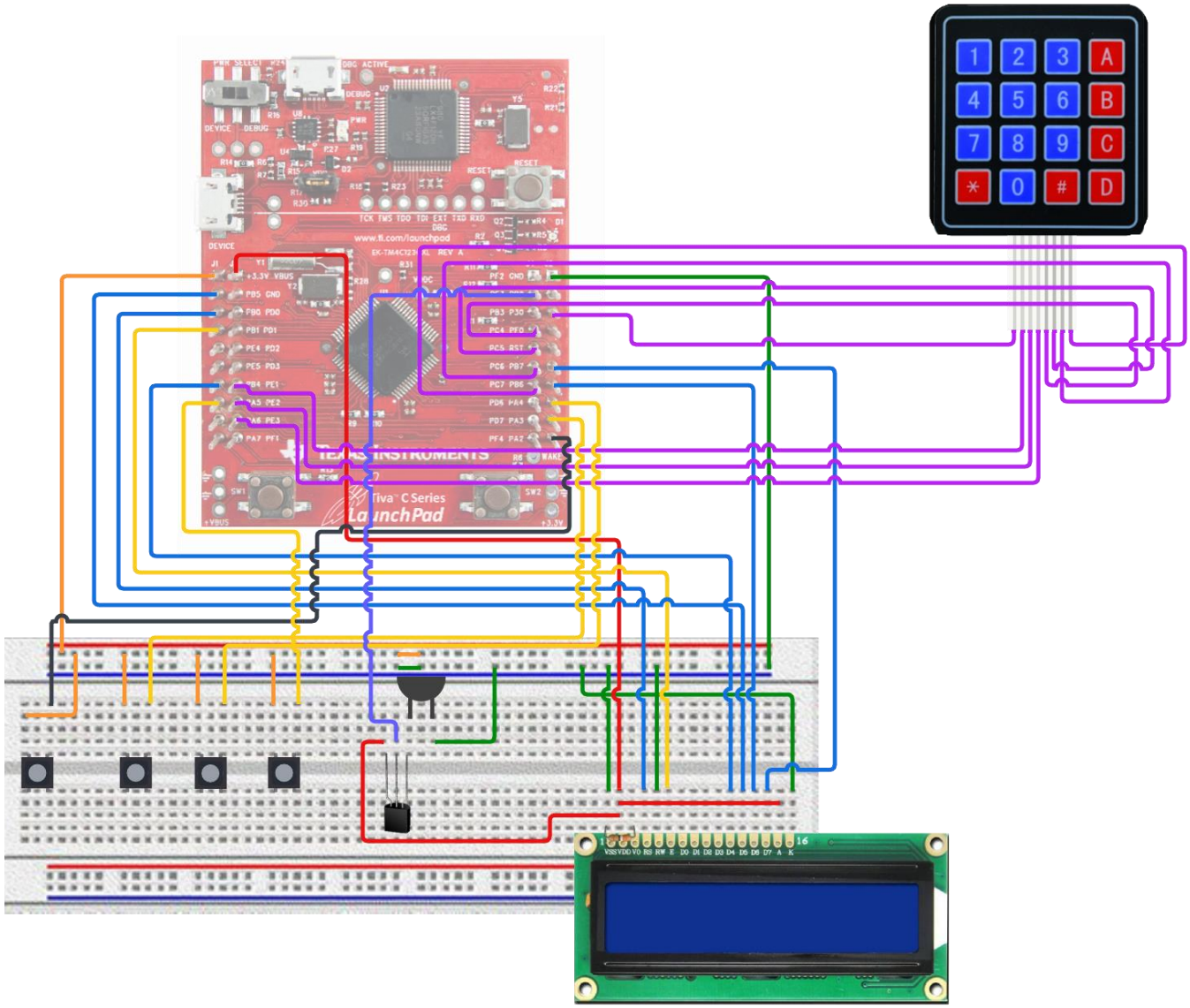
- **btnHandlers():** implemented to run on buttons presses events (interrupts). It enables timer 1 to start counting, pauses (disables timer 1), or resets the timer value (sets **seconds** variable to zero and disables timer 1) on pressing buttons 2, 3, 4 respectively.

```
case(GPIO_INT_PIN_3): //start stopwatch
   if(state == 3){
        TimerEnable(TIMER1_BASE,TIMER_A);}
   break;
 case(GPIO_INT_PIN_4): //pause stopwatch
   if(state == 3){
        TimerDisable(TIMER1_BASE,TIMER_A);}
   break;
 case(GPIO_INT_PIN_5): //reset stopwatch
   if(state == 3){
        TimerDisable(TIMER1_BASE,TIMER_A);
        LCD_PrintLn(0,"0:0  ");
        seconds = 0;}
   break;}
```

- **Timer1IntHandler():** implemented to run when timer1 elapses. It increments the **seconds** variable value by 1, calculates minutes and seconds value, and displays the new value on the LCD screen if the current state on screen is the stopwatch application. If not, the value of the seconds variable is still incremented as the app is running in the background.

```
// stopwatch handler
void Timer1IntHandler(void){
   seconds++;
   if(state == 3 && waits){
   uint16_t stod = seconds%60;
   uint16_t mtod = seconds/60;
   int32_t length = snprintf( NULL, 0, "%hu", stod);
   char* str1 = malloc( length + 1 );
   snprintf(str1, length + 1, "%hu", stod );
   length = snprintf( NULL, 0, "%hu", mtod);
   char* str2 = malloc( length + 1 );
   snprintf(str2, length + 1, "%hu", mtod );
   str2 = strcat(str2,":");
   str1 = strcat(str1,"  ");
   LCD_PrintLn(0,strcat(str2,str1));
   free(str1);
   free(str2);}
   TimerIntClear(TIMER1_BASE,TIMER_TIMA_TIMEOUT);}
```

7. HARDWARE IMPLEMENTATION



8. ATTACHED LINKS

- Demo video link on OneDrive.
- Project code GitHub repository link.