ELEC5620M
EMBEDDED MICROPROCESSOR SYSTEM DESIGN
ASSIGNMENT 1 REPORT

Phaneeshwar Mallakkagari
Student ID: 201387234

Contents

1. Abstract

This report covers the design, implementation and verification of a Digital Clock programmed using embedded C. The model is well structured using various concepts of embedded systems. This report briefs about the implemented design and the code was verified using the software tool and Hardware Chip.

2. Introduction

The design aims to create a digital clock using embedded C programming language. In this assignment, different drivers were built with source and header files. This task was demonstrated by using various timer operations along with accessing various peripherals like LEDs, 7-Segment display to view the current time, and push buttons inbuilt on the hardware board are used to set the time to start at any minute or hour we want. The constructed design code is feasible for the users to control various parameters of the digital clock like by adapting to the required time, by setting up the alarm and snoozing the time.

3. Code Design

3.1 Main Module

This Module is designed mainly by using various concepts of embedded systems like event timers, task schedulers, functions, pointers, memory- mapping etc. At the start of the main code, multiple parameters are being initialized using initialize Variable, initial Final and initialize functions. Initialize function is used for displaying zeros in all 7-Segment displays. While initialize Variable function is used for setting up the parameters to zero, these variables are used for displaying values on the 7-segment display, and “initialfinal” function initializes those variables that are specifically used when the system runs in alarm mode. Furthermore, exitOnFail function is used to very whether the timer functions are correctly initialized. To run the digital clock, concepts of task scheduler have been majorly used, as this would be an ideal way of implementation. Hence, different time intervals were assigned to various tasks. In this design, the timer operation generates 1 Hz clock frequency, hence the timer interrupt status register will be high for every second, and the same clock can be used for obtaining minutes and hours by setting up different intervals.

Load = (System frequency * Required frequency) / (4*(Prescaler + 1))

Above equation is used to determine the task time interval value to generate 1hz frequency for seconds counter. Here, **System frequency** is 900 MHz, and **Required frequency** is set to 1Hz. Furthermore, the **Prescaler** is set to a value of 0xC7. Hence to generate one second, timeinterval0 value is set to 1125000. Similarly, minutes counter will increment for every 60 seconds, hence timeinterval1 value will be 60 times of timeinterval0 and hours counter will increment for every 3600 seconds, hence timeinterval2 .

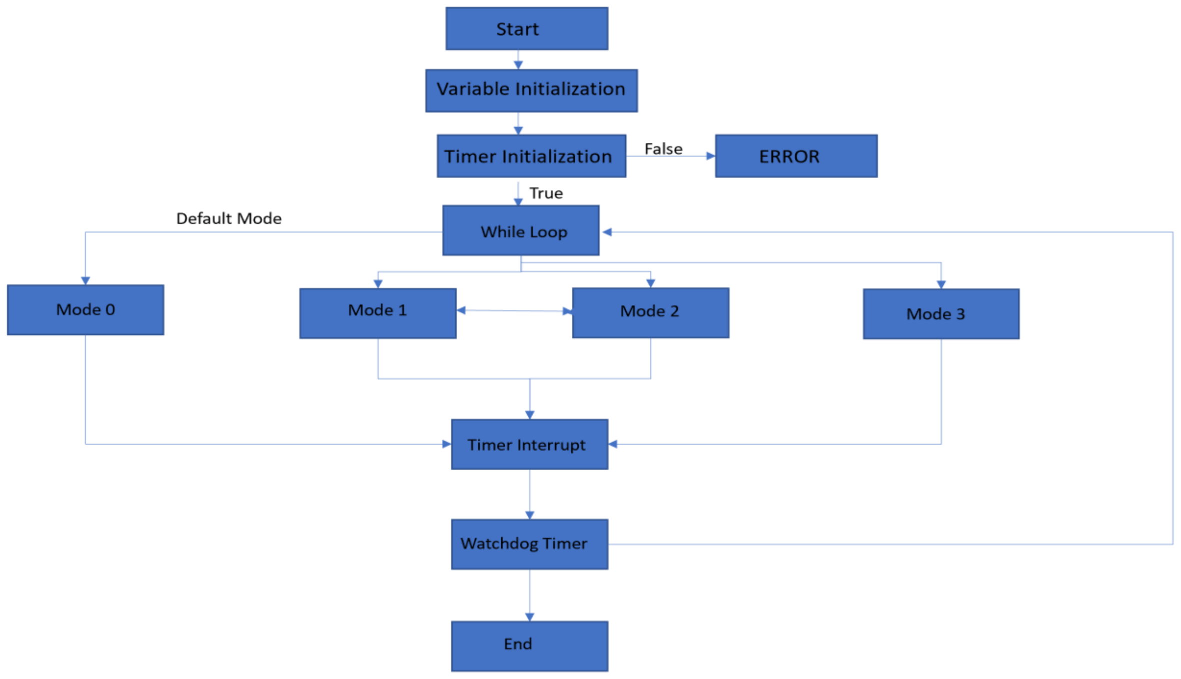

Figure 1: Main flow of the program

3.1.1 Mode 0 – Standard Mode

In this mode, the digital clock will run normally and displays the current time on the 7-segment LED. This mode operates three different tasks. The first task is to generate seconds of the clock. The units position of the seconds counter will change its value for every one second. And it ranges from 0 to 9, while the tens position of the seconds counter varies for every 10 seconds and ranges from 0 to 5. Hence these two-subblocks are interlinked with the help of an 'if' condition. Similarly, in second task, the minute's counter is designed and the only difference is that the unit's position of the minutes counter will increment for every 60 seconds and it ranges from 0 to 9, while the tens position will increment for every 600 seconds and it ranges from 0 to 5. In third task, the tens digit ranges from 0 to 2, while the units digit of the hours counter will go from 0 to 9, for first two rounds and for the third round it is limited to a maximum value of 3. Post that all the 7- segment displays are initialized to zero. This entire process is successfully achieved through task scheduling and function calls, and appropriate conditional loops were used to accomplish this Mode.

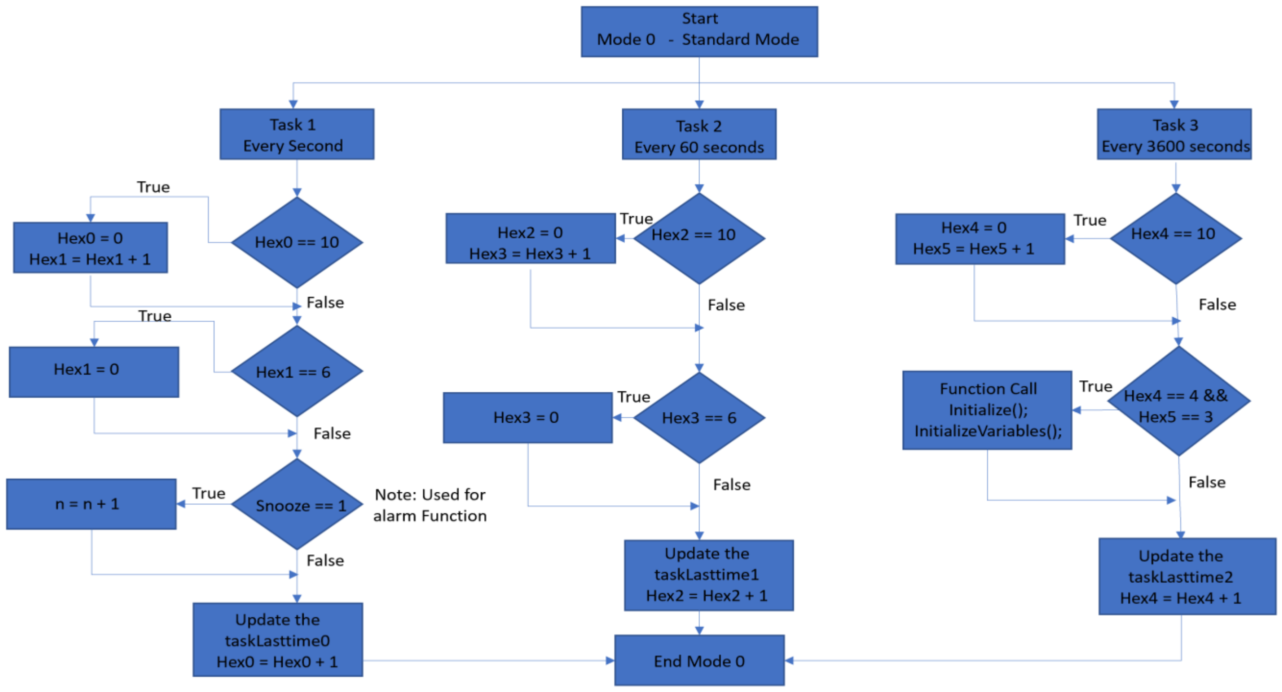

Figure 2: Mode 0 – Standard Mode flowchart

3.1.2 Mode 1- Time Edit Mode

In this mode, the user can change the time of the digital clock with the help of Pushbuttons. This peripheral is mapped into the memory space of the processor; hence a pointer is initialized to base address of the pushbutton as 0xFF200050 and the base address of this peripheral is typecasted for the compiler. An array variable called as ‘flag’ is initialized, where this variable is used for avoiding the effect of debouncing. When the button is pressed, the flag variable goes high, while another if-conditional statement is used to check if the button is not pressed and the flag is high. When this is true, then the time of the clock is changed, and the flag is initialized to zero. Pushbutton0 is used for changing the minutes counter of the clock and time delay for the task 3 is incremented by one minute every time the Pushbutton0 is pressed, in order to synchronize the minute and hours of the clock. In this similar manner Pushbutton1 is used for changing the hours counter of the clock. In addition to this, a KeyFunction is used in this operation and this function is helpful when the alarm is turned on. This subroutine function has dual functionality and it is controlled using the variable called ‘key’. When the alarm is off, key value will be equal to 1, this function would store the current timer value into a variable called ‘InitialValue’. When the alarm is turned on, key value would be equal to 0, it would behave like a counter that counts the number of times the user has pressed the button.

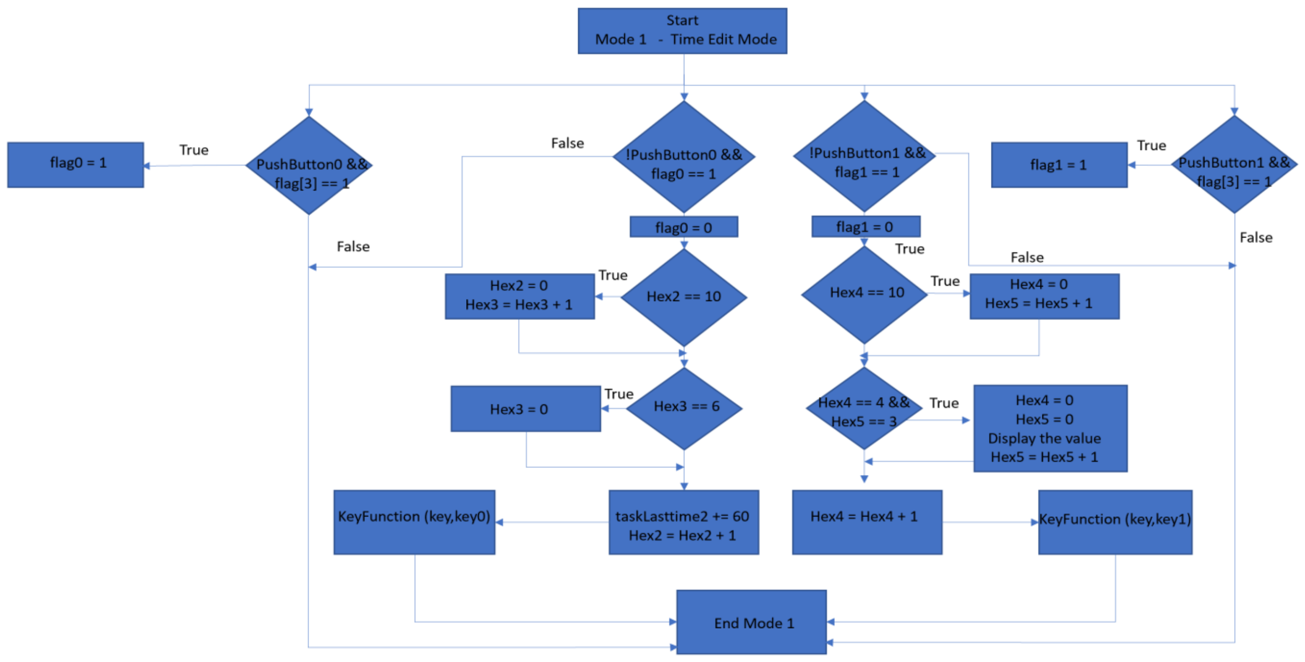

Figure 3: Mode 1 – Time Edit flowchart

3.1.3 Mode 2- Alarm Mode

This mode is used for setting the alarm. Firstly, for this mode to function, sliding switch has to turned on. When the switch goes high, the variable ‘key’ is set to zero. Then the user can set the time with help of pushbuttons and the Key Function will act like a counter which increments every time the user presses the button. The Alarm time being set when Pushbutton2 and Pushbutton3 are pressed simultaneously. While setting the alarm, it would decrement the hour counter by number of times the minute button has been pressed by the user. Furthermore, the 7-segment LEDs would display the current time once the alarm is set. If the digital clock reaches to the set time, then the LEDs will turn-on for about 10 seconds and then turns off. To deactivate the alarm mode, the user will have to turn off the switch. If New alarm is needed to be set then the user have to deactivate the present alarm by turning off switch and later turn on the switch. Furthermore, Once the alarm is set we cannot access Mode 1

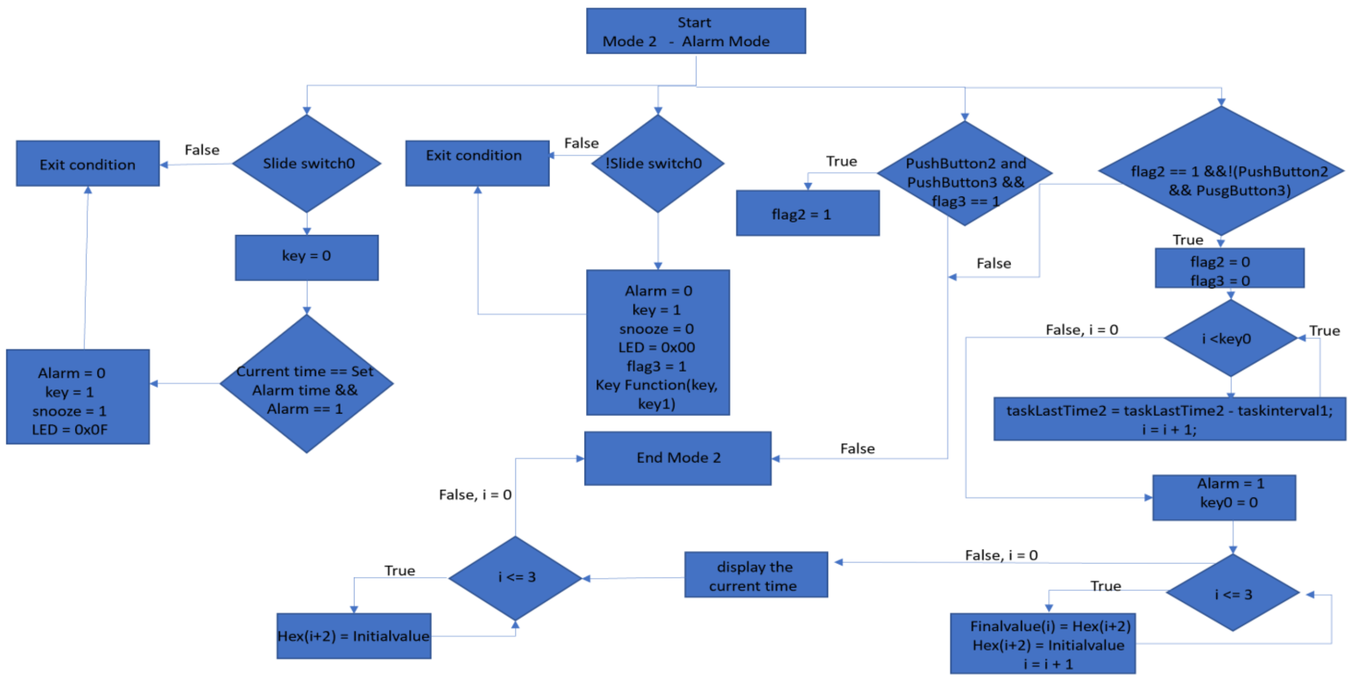

Figure 4: Mode 2 – Alarm Mode flowchart

3.1.4 Mode 3- Alarm Snooze

Once the Alarm is activated, this mode would snooze the set alarm for every 5 minutes till the alarm function is turned off. Here the variable 'n' is used to avoid conjunction between the Task4 and when the current time is equal to alarm time. To avoid this conjunction 'n' variable behaves like a delay only for the first call.

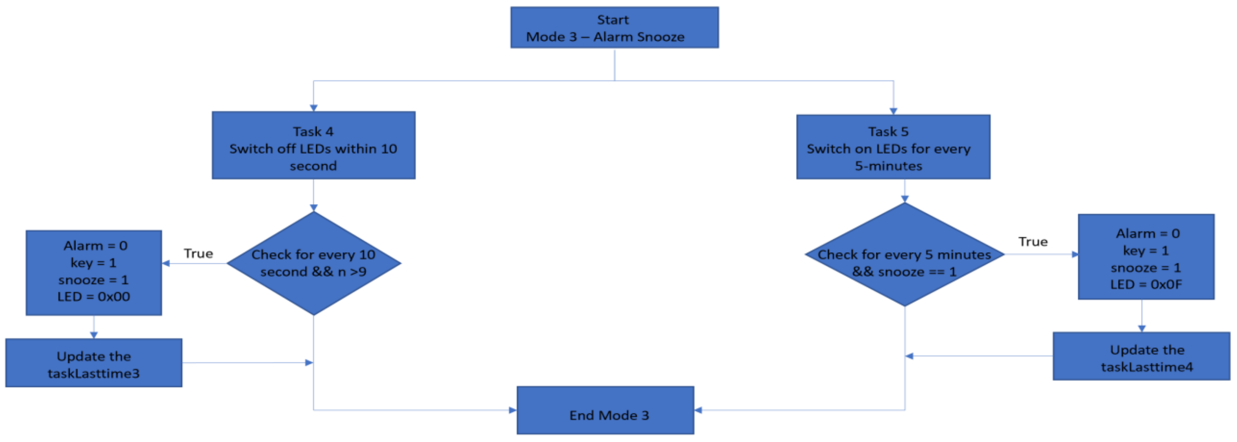

Figure 5: Mode 3 – Alarm Snooze flowchart

4. Testing and Debugging

Various testing techniques were used to test both software and hardware using the timer driver.

4.1.1 Software

In software testing and debugging the Eclipse IDE has been used to step through code and test variables to see if the results obtained are as expected. Several bugs were solved while creating header and source files for the timer operations and for the main function. Number of breakpoints were used to find how long it took for the DE1-SoC to run the test code. Moreover, linking different functions and flags for different modes of operations was a challenging task.

4.1.2 Hardware

When testing the hardware, the part of a code is checked each time to see if the number displayed on the 7-Segment is as expected. Several functions like timer functions were tested in the similar manner. After testing the normal operations of the digital clock, pushbuttons were tested by changing the minutes and hours of the clock. Furthermore, the alarm functions were tested for different time intervals.

5. Conclusion

In Conclusion, creating the timer function and the main function was an intellectually stimulating task in which numerous algorithms and testing techniques had been used. The Eclipse IDE was found to be a very useful and powerful tool for debugging and testing the code step by step with help of breakpoints and checking memory variables. This design was thoroughly tested and verified.

6. Appendix

6.1 Design Code

6.1.1 Timer11.h

```
#ifndef TIMER11_TIMER11_H_
#define TIMER11_TIMER11_H_

#include <stdbool.h>

#define TIMER_SUCCESS 0
#define TIMER_ERRORNOINT -1
#define INTERUPPT_HIGH 1
#define INTERUPPT_LOW 2

signed int Timer_intialise (unsigned int base_address);
bool Timer_isIntialised(void);

signed int Timer_load (unsigned int load);
signed int Timer_Prescaler (unsigned int prescaler);
signed int Timer_Control (unsigned int control);
signed int Timer_counterread (void);
signed int Timer_interuppt(void );
//signed int Timer_read_interuppt(void );
//signed int Timer_clear_interuppt(void);


#endif /* TIMER11_TIMER11_H_ */
```

6.1.2 Timer11.c

```
#include "Timer11.h"

volatile unsigned int *base_address_ptr = 0x00;

bool timer_intialized = false;
bool load_intialized = false;

#define Timer_Load      (0x00/sizeof(unsigned int))
#define Timer_counter   (0x04/sizeof(unsigned int))
#define Timer_control   (0x08/sizeof(unsigned int))
#define Timer_interrupt (0x0C/sizeof(unsigned int))

// to check if the timer function as initialized
signed int Timer_intialise (unsigned int base_address)
```

```
{
 base_address_ptr = (unsigned int *) base_address;
 base_address_ptr[Timer_control] = 0x00;

 timer_intialized = true;

 return TIMER_SUCCESS;
}

bool Timer_isIntialised(void)
{
 return timer_intialized;
}
// this function is used for loading the value
signed int Timer_load (unsigned int load)
{
 if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
 load_intialized = true;
     base_address_ptr[Timer_Load] = load;

     return TIMER_SUCCESS;

}
bool Load_isIntialised(void)
{
 return load_intialized;
}

// this function is used for setting up the Prescaler value
signed int Timer_Prescaler (unsigned int prescaler)
{
 unsigned int variable;
     unsigned int *timer_ptr;  // declare a seprate pointer
 if (!Timer_isIntialised()) return TIMER_ERRORNOINT;

 variable =  *base_address_ptr;
 timer_ptr = &variable;
 timer_ptr[Timer_control] &= 0x00FF;
 timer_ptr[Timer_control] |= (prescaler << 8 );
 base_address_ptr[Timer_control] |= timer_ptr[Timer_control];
 return TIMER_SUCCESS;
}
// this is the control functions to set controls I,A,E
signed int Timer_Control (unsigned int control)
{
 unsigned int *timer_ptr1;
 unsigned int variable1;
 if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
 if (!Load_isIntialised())  return TIMER_ERRORNOINT;

 variable1 = *base_address_ptr;
     timer_ptr1 = &variable1;
     timer_ptr1[Timer_control] &= ~0x7;
     timer_ptr1[Timer_control] = 0x7 & control;
     base_address_ptr[Timer_control] |= timer_ptr1[Timer_control];
```

```
    return TIMER_SUCCESS;
}
// to read the current value of the timer
signed int Timer_counterread (void)
{

 if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
 if (!Load_isIntialised())  return TIMER_ERRORNOINT;
    return base_address_ptr[Timer_counter];
}
// to check the interuppt, when set it will cleared
signed int Timer_interuppt(void)
{
 unsigned int value = 0;
 unsigned int *timer_ptr;
 if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
 if (!Load_isIntialised())  return TIMER_ERRORNOINT;
 value = base_address_ptr[Timer_interrupt];

 if (value == 1)
 {
        base_address_ptr[Timer_interrupt] = 0x01;
 }

```

6.1.3 Declare.h

```
#ifndef DECLARE_H_
#define DECLARE_H_
// Variables declared for timer Operations
volatile unsigned int base_add = 0xFFFEC600;
volatile unsigned int load = 0x00;
volatile unsigned int control = 0x0003;
volatile unsigned int prescaler = 0xC7;
unsigned int timer_interrupt = 0;
// Declarations of other peripherals
volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;
volatile unsigned int *pushbutton = (unsigned int *) 0xFF200050;
volatile unsigned int *SlideSwitch = (unsigned int *) 0xFF200040;
// Declarations of intermediate
unsigned int currentTimerValue;
unsigned int i = 0, k = 0, n = 0, key = 0, key0 = 0, key1 = 0, Alarm = 0, snooze = 0;
unsigned int taskLastTime[5];
unsigned int taskinterval[5];
unsigned int flag[4] = {0,0,0,0};
unsigned int nosec[5] = {1,60,3600,10,300}; // 1 sec,60sec,3600sec, 10sec to switch off the leds, 300 sec to snooze the time
unsigned int HexDisp[6] = {1,1,1,1,1,1};
unsigned int InitialValue[5];
unsigned int FinalValue[5];
unsigned int display[10] = { 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0X6D, 0x7D, 0x07, 0x7F, 0x6F}; // Array of Hex values ranging from 0 to 9
#endif /* DECLARE_H_ */
```

6.1.4 Hps_watchdog.h

```
#ifndef HPS_WATCHDOG_H_
#define HPS_WATCHDOG_H_

//#define for backwards compatibility
#define ResetWDT() HPS_ResetWatchdog()

// Function to reset the watchdog timer.
__forceinline void HPS_ResetWatchdog() {
*((volatile unsigned int *) 0xFFD0200C) = 0x76;
}

// Function to get value of the watchdog timer
__forceinline unsigned int HPS_WatchdogValue() {
return *((volatile unsigned int *) 0xFFD02008);
}

#endif /* HPS_WATCHDOG_H_ */
```

6.1.5 Main.c

```
#include "HPS_Watchdog/HPS_Watchdog.h"
#include "Timer11/Timer11.h"
#include "Declare/Declare.h"    // All the declared variables are present in this
header
#include <stdlib.h>

void exitOnFail(signed int status, signed int successStatus)
{
    if (status != successStatus)
    {
        exit((int)status);
    }
}

void initializeVariables(void)   // This function used for initialize the values of
variables
{
       for (k = 0; k < 6; k++)
       {
              HexDisp[k] = 0;           // making the display variables to zero, This
variable is used to display the values on the 7-Segment display
       }
       for (k = 0; k < 5; k++)
       {
              taskLastTime[k] = 0;
       }
}

void initialFinal (void)
{
	for (k = 0; k <= 3; k++)
		{
			InitialValue[k] = 0;	// this variable is used to store the
value of
			FinalValue[k] = 0;	// this variable will store the alarm
value set by the user
		}
}

// initialize the display to value Zero, So when this function is called all 7-
Segments display number zero.
void initialize(void)
{
	volatile unsigned int *display03 = (unsigned int*) 0xFF200020;
	volatile unsigned int *display45 = (unsigned int*) 0xFF200030;
	*display03 = (0x3F << 0) | (0x3F << 8) | (0x3F << 16) | (0x3F << 24) ;
	*display45 = (0x3F << 0) | (0x3F << 8);

}

// this function is used for displaying the different values on the 7-Segment display

void displayHex (int a, int b, int n)

{
	volatile unsigned int *display03 = (unsigned int*) 0xFF200020; // Can access
only first four 7-Segment Displays on board
	volatile unsigned int *display45 = (unsigned int*) 0xFF200030; // Can access
the Last two 7-Segment Displays
	unsigned int value;

	if (n == 0)

	{
		value = *display03;
		value &= ~(0x7F << b);
		value |= (a << b);
		*display03 = value;
	}

	if (n == 1)

	{
		value = *display45;
	    value &= ~(0x7F << b);
		value |= (a << b);
		*display45 = value;

	}
}
```

```
void KeyFunction (int K, int K12)    // here K is a flag
{
	unsigned int j;

	if (K == 0)

	{
		K12 += 1;    // this function is used to count number of times the user 
had pressed the button, Once the Switch is turned on then this statement is used
	}

	if (K == 1)
	{
		for (j = 0; j <= 3; j++)
		{
			InitialValue[j] = HexDisp[j+2] - 1; // this would store the 
initial values till the alarm switch is turned on. Once the Switch is turned on then 
this statement is not in use
		}
	}
}

// This function is used to initialize different variables, this function is useful 
for setting up and switching of Alarm
void AlarmVariables (int x1, int x2, int x3, int x4)
{
	Alarm = x1;
	key   = x2;
	snooze = x3;
	*LEDR_ptr = x4;

}

int main(void)
{
	*LEDR_ptr = 0x0; // Making all Leds on Board in Off state
	initialFinal();

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

	{
	    taskLastTime[i] = Timer_counterread();
	    taskinterval[i] = 1125000 * nosec[i];    // Initialize different Interval 
values
	 }

	load = 0xFFFFFFFF;
	exitOnFail (Timer_intialise(base_add),TIMER_SUCCESS);
	        exitOnFail (Timer_load(load),TIMER_SUCCESS);
	        exitOnFail (Timer_Prescaler(prescaler),TIMER_SUCCESS);
	        exitOnFail (Timer_Control(control),TIMER_SUCCESS);
	        ResetWDT();
```

```
    initialize();

    while(1)

    {
      signed int currentTimerValue = Timer_counterread();

   //***************** MODE 0 Standard Mode ************************************

      if ((taskLastTime[0] - currentTimerValue) >= taskinterval[0]) // This Task
Scheduler is True for Every Second, And useful for displaying the Seconds of the
digital clock, (task 1 for Mode 0)

      {

      if (HexDisp[0] == 10)   // To Check if the units of Seconds counter is above
9, It initialize the variable HexDisp[0](first 7-Segment Display) to zero and also
display 0 after it reaches 9.

      {
          HexDisp[0] = 0; // Initialize the variable to zero when it is above 9


          displayHex(display[HexDisp[1]],8,0); // it will display the tens digit of
the Seconds hand of the digital clock, here 8 Represents the amount shift within
          HexDisp[1] += 1; // Increment the variable(tens digit of the Seconds Hand
of the clock) for every 10 seconds.

       }

      if (HexDisp[1] == 6) // To check if Tens digit of the seconds counter is above
5. It will initialize the variable to zero

         {
           HexDisp[1] = 0; //Initialize the variable to zero when it is above 5

         }

       displayHex(display[HexDisp[0]],0,0);   // displayHex: it is a function
Displays the value in the 7-Segment, display(): array has the list of hex values,
                                              // HexDisp(): it defines which element
in the display array to be displayed.

       taskLastTime[0] = taskLastTime[0] - taskinterval[0]; // Update the
taskLastTime0
       HexDisp[0] += 1; // Increment the variable for every second

       if (snooze == 1)  // This Conditional Statement is specifically used when
Alarm Switch is turned On

          {
            n += 1;
           }
         }
```

```
        if (((taskLastTime[1] - currentTimerValue) >= taskinterval[1]))  // This task
is activated For Every 60 Seconds, as 1 min = 60 seconds.  (task 2 for Mode 0)

                {

                        if (HexDisp[2] == 10)     // to check if the units of the
Minutes counter is above 9, HexDisp[2] is used for displaying digit in the third 7-
segment display

                        {
                                HexDisp[2] = 0;  //initialize to zero,

                                displayHex(display[HexDisp[3]],24,0);   // display
function, it displays tens place of Minutes, fourth 7-segment display
                                HexDisp[3] += 1; // increment tens position Minutes
counter for every 600 seconds
                        }

                        if (HexDisp[3] == 6) // to check if the tens of the
Minutes counter is above 5, HexDisp[3] is used for displaying digit in the fourth 7-
segment display

                        {
                                HexDisp[3] = 0; // Initialize to zero once it is
greater than 5
                        }

                        displayHex(display[HexDisp[2]],16,0); // display
function, it displays units place of Minutes, third 7-segment display

                        taskLastTime[1] = taskLastTime[1] - taskinterval[1]; //
update the taskLastTime1
                        HexDisp[2] += 1;  // units place of Minutes
positIncrement the variable for every 60 seconds


                }

                if ((taskLastTime[2] - currentTimerValue) >= taskinterval[2]) //
This task is activated For Every 3600 Seconds, as 60 min = 3600 seconds.  (task 3 for
Mode 0)
                 {
                        if (HexDisp[4] == 10)  // to check if the units of the
hours counter is above 9, HexDisp[4] is used for displaying digit in the fifth 7-
segment display

                        {

                                HexDisp[4] = 0;   // initialize to zero after
reaches to 9

                                displayHex(display[HexDisp[5]],8,1); // display
function, it displays tens place of hours, sixth 7-segment display
                                HexDisp[5] += 1;  // increment the variable
                        }
```

```
                           if (HexDisp[5] == 3 && HexDisp[4] == 4)  // to check when
the timer goes beyond 23:59:59

                           {
                                  initializeVariables(); // initialize all the
variables
                                  initialize(); // initialize the display
                           }

                           displayHex(display[HexDisp[4]],0,1); // display function,
it displays units place of hours, fifth 7-segment display

                           taskLastTime[2] = taskLastTime[2] - taskinterval[2]; //
update the count
                           HexDisp[4] += 1;

                    }

// ********************** MODE 1 Time Edit MODE
**********************************************//

                    // To Avoid Debouncing of the switch
                    if ((*pushbutton & 0x01) && flag[3] == 1)

                       {
                              flag[0] = 1;

                       }

                    if ((*pushbutton & 0x02) && flag[3] == 1)

                        {
                           flag[1] = 1;

                         }
                // This function is for PushButton0 , To change the Minutes
                       if (flag[0] && !(*pushbutton & 0x01))

                              {
                                      flag[0] = 0;

                                             if (HexDisp[2] == 10) // to check if
the units of the Minutes counter is above 9, HexDisp[2] is used for displaying digit
in the third 7-segment display

                                             {

                                             HexDisp[2] = 0; // Initialize to zero
once it is greater than 9

                                             displayHex(display[HexDisp[3]],24,0);
                                             HexDisp[3] += 1;

                                             }
```

```
                                              if (HexDisp[3] == 6) // to check if
the tens of the Minutes counter is above 5, HexDisp[3] is used for displaying digit
in the fourth 7-segment display

                                              {
                                                  HexDisp[3] = 0; // Initialize to
zero once it is greater than 5

                                              }

                                              displayHex(display[HexDisp[2]],16,0);

                                              taskLastTime[2] += taskinterval[1]; //
hour time is incremented whenever pushbutton0 is pressed

                                              HexDisp[2] += 1;

                                              KeyFunction (key,key0); // this
function has dual functionality and used for storing the current time as initial
variable.
                                                                       // other
functionality is that when alarm is ON, it would count number of times the user
pressed the button
                                 }

               // This function is for PushButton1, To change the Hours
                      if (flag[1] && !(*pushbutton & 0x02))

                         {
                             flag[1] = 0;

                             if (HexDisp[4] == 10) // to check if the units of the
hours is above 9, HexDisp[4] is used for displaying digit in the fifth 7-segment
display

                                {
                               HexDisp[4] = 0;
                                 displayHex(display[HexDisp[5]],8,1); // display
function, it displays tens place of hours, sixth 7-segment display
                                   HexDisp[5] += 1;

                                   }

                            if (HexDisp[5] == 3 && HexDisp[4] == 4)

                                              {
                                                   HexDisp[4] = 0;
                                                   HexDisp[5] = 0;


      displayHex(display[HexDisp[5]],8,1); // display function, it displays tens
place of hours, sixth 7-segment display
                                                   HexDisp[5] += 1;
                                }
```

```
                                              displayHex(display[HexDisp[4]],0,1);
// display function, it displays units place of hours, fifth 7-segment display

                                              HexDisp[4] += 1; // increment the
value

                                              KeyFunction (key,key1); // this
function has dual functionality and used for storing the current time as initial
variable.
                                            // other functionality is that when
alarm is ON, it would count number of times the user pressed the button
                                 }

// ************************************* MODE 2 Alarm Mode
*************************************//
 if (*SlideSwitch && 0x01)

    {

       key = 0;

if (Alarm == 1 && HexDisp[2] == FinalValue[0] && HexDisp[3] == FinalValue[1] &&
HexDisp[4] == FinalValue[2] && HexDisp[5] == FinalValue[3]) // to check if the
current time is equal to
                                // the set time

                             {
                                AlarmVariables (0,1,1,0x0F); // function call to
initialize the values of the variables, alarm = 0, key =1 ,snooze = 1, Led On
                             }
                          }

                      if (!(*SlideSwitch && 0x01))

                         {

                          AlarmVariables (0,1,0,0x00); // function call to
initialize the values of the variables, alarm = 0, key =1 ,snooze = 0, Led Off
                          KeyFunction (key,key1); // In this case this function will
help to store the initial value immediately
                          n = 0;
                          flag[3] = 1;  // This flag is used when alarm function is
set, it make sure that Standard mode and snooze mode function properly. Once the
alarm is set then Mode 1 cannot be accessed


                          }

     if ((*pushbutton & 0x0C) && (*SlideSwitch && 0x01) && flag[3] == 1)
                          {
                          flag[2] = 1;

                          }
                      if (flag[2] && !(*pushbutton & 0x0C))

                              {
                                      flag[2] = 0;
                                      flag[3] = 0;

    for (i = key0; i > 0; i--)
        {
            taskLastTime[2] = taskLastTime[2] - taskinterval[1]; // to reduce the
time interval of hour , depending on number of times the pushbutton0 is changed
         }

    key0 = 0;
    key1 = 0;
    Alarm = 1;

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

    {
       FinalValue[i] = HexDisp[i + 2];    // Alarm value is stored in a variable call
FinalValue
      HexDisp[i + 2] = InitialValue [i]; // initialize the value before setting up
the alarm to HexDisp
    }
                                // display the current time
 displayHex(display[HexDisp[2]],16,0);
 displayHex(display[HexDisp[3]],24,0);
 displayHex(display[HexDisp[4]],0,1);
 displayHex(display[HexDisp[5]],8,1);

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

    {
       HexDisp[i + 2] = InitialValue [i] + 1;
    }
   }

  //******************************* Mode 3 Alarm Snooze
***********************************//

 if (((taskLastTime[3] - currentTimerValue) >= taskinterval[3]) && n>9) // n variable
is used to avoid congestion between this task and when the alarm is set(it acts like
a delay only for first call)

// it checks for every 10 secs
       {
          AlarmVariables (0,1,1,0x00);                          // turn off the led
          taskLastTime[3] = taskLastTime[3] - taskinterval[3];

           }

 if (((taskLastTime[4] - currentTimerValue) >= taskinterval[4]) && snooze == 1) //
Snnoze for every 5 min
```

```
     {
        AlarmVariables (0,1,1,0x0F);    // turn on the led
        taskLastTime[4] = taskLastTime[4] - taskinterval[4];
      }
//*************************************************************************************
******//
       Timer_interuppt(); // we clear the private timer interrupt flag if it is set,
hence this function is used
       ResetWDT();       // Reset Watchdog Timer
     }
       }
```