Project Alarm Clock

ECE 299

2021-08-17



Section Number: B03 - Group ID: G09

Name	V Number
Hoang Tuan Kiet Vu	
Bolaji-Oyenekan Tioluwatimofe	



Acknowledgment

We acknowledge with respect the Lekwungen peoples on whose traditional territory the university stands and the Songhees, Esquimalt and WSÁNEĆ peoples whose historical relationships with the land continue to this day.

We acknowledge and would like to thank the following people who have helped us tremendously in completing this project:

- The Electrical Engineering Department, especially those who have formatted the course ECE 299. This course has taught us how to use SolidWorks and KiCad, as well as, providing us great opportunities to get hands-on experience with real-world problems, such as creating a fully functioning Alarm Clock.
- Mr. Levente Buzas for being our Teaching Assistant, he has been supportive and understanding during the labs.
- Mr. Brent Sirna for helping us whenever we are having technical difficulties in our project.
- The manufacturer of our PCB Seeed Technology Inc for helping us creating our PCB, as well as, noting our specifications
- Professor Ilamparithi Thirumarai Chelvan for all of his efforts in running ECE 299 during such a difficult time as COVID19. His kindness has helped our team tremendously, especially the time when we forgot to submit part of the report, or during the final software demonstration when David (one of the members) was dealing with sickness.

Contents

Acknowledgment i
Contents ii
List of Tables iii
List of Figures iv
I Project Goal 1
II Constraints 1
III Requirement Specifications 1
Hardware 1
Software 2
Design Calculations 2
IV Design & Validation 2
V Bill of Materials (*One Table*) 3
VI Circuit Schematic & PCB 3
VII Conclusion & Recommendations 6
References 7
Appendix A – Enclosure Details 8
Appendix B – Software code with comments 10

List of Tables

Table 1: Bill of Materials. 3

List of Figures

Figure 1: The schematic of the STM board and the Switches. 4
Figure 2: The schematic of the LED display, and the NMOS MOSFET. 4
Figure 3: The final layout of the PCB.. 5
Figure 4: The manufactured PCB's Top layer..... 5
Figure 5: The manufactured PCB's Bottom layer..... 6
Figure 6: The Design of the Digimon Tamagotchi..... 8
Figure 7: The front view of the Enclosure..... 8
Figure 8: The side view of the Enclosure 9
Figure 9: The back view of the Enclosure. 9

I Project Goal

The goal of this project is to create a fully functioning Alarm Clock that is capable of:

- Receiving user input using push buttons.
- Showing time on a 4-digit 7-segment LED display.
- Allowing the user to set time for the clock.
- Allowing the user to set alarm at any desired time
- Triggering light-based alarm as and when clock time reaches alarm time.
- Be able to snooze the alarm at any desired time.
- Be able to turn the alarm off at any desired time.

The Alarm Clock would consist of an STM board, a PCB that is designed by the team, and an LED that displays the time. The enclosure would be created and demonstrated on SolidWorks.

II Constraints

The constraints that define this project are:

- The use of an LTC 4727JR 4-digit 7-segment LED display to tell the time on the alarm clock.
- The use of an STM32F407 Discovery kit to provide intelligence for the alarm clock i.e., taking in user input and producing output in form of sound or light.
- The use of Push-buttons to provide user input to the alarm clock.
- The use of KiCAD to create schematics and PCB layout design for the PCB that will connect the Push-buttons, the STM32F407 Discovery kit, and the LTC 4727JR 4-digit 7-segment LED display.
- The use of SolidWorks to create an enclosure design for the alarm clock.
- The use of Eclipse IDE to program the STM board and test its functionality using the alarm clock computer program provided.

III Requirement Specifications

The requirements that need to be met for the successful completion of the project goals based on the constraint are:

Hardware

- The use of a current limiting resistor of R = 50 Ω ± 5% P ≥ 21mW in series with the segments of the digits on the Four digit 7-segment display (LTC-4742JR). The value of the current limiting resistor is derived in the design calculations section below.
- The use of an STM discovery kit as intelligence for the alarm clock with 3V and 20mA at its output pins.
- The use of Push-buttons with the internal pull up resistors of the STM board in a pull up microcontroller-switch configuration.
- The use of NMOS MOSFET transistors to switch between each digit of the Four digit 7-segment display (LTC-4742JR) with $I_d \geq 160mA$ and maximum switching voltage

$V_{GS(th)\ max} \geq 3V$. The calculation for I_d is shown in the design calculations section below while $V_{GS(th)\ max}$ is gotten from the voltage drop between the output pin and ground.

- The creation of a PCB that is approximately the size of the STM board with adequate consideration and provision of space for mounting the components stated above with proper connections and traces that pass the validation tests using KiCAD.
- The designing of an enclosure that will house the STM board and the populated PCB with dimensions that correspond with combination of the PCB and STM board with provision for opening and closing the enclosure and connecting the board to a power source.
- The designing of an enclosure that is mechanically stable, safe, and stylish.

Software

- The use of Eclipse IDE to program the STM board and test Alarm clock's Set Alarm, Set Time, Light-based alarm, and 10 mins snooze functionality using the alarm clock computer program provided.

Design Calculations

Using Ohms law V = IR; then R = V/I

R = 1/V; V = 1V (3V of output pin - 2V of display) I = 20mA [1]; 1

R = 50 Ω

Using $P = I^2R$

$P = (20*10^{-3})^2(50+5\%\ (50))$; $P \geq 21mW$ 2

Using Kirchhoff's law

I_d = 8(in the case all 7 segments and L are on) *I; 8*20mA 3

I_d = 160mA

Note:

I is the current from the board in A

V is the voltage at the output pin in V

R is the resistance of the resistor in Ω

I_d is the current through the transistor in A

IV Design & Validation

The display sub-system mainly the PCB was reviewed with our lab instructor and design rules were implemented based on the manufacturer's abilities and checked to ensure the board will function as proposed after it gets populated with components. Its proposed functionality was made by using TinkerCAD circuits simulation. Also, the display and Push-button subsystem were tested with the aid of the lab technician by first writing a program for the alarm clock computer program using Eclipse IDE after which the display and push-buttons were debugged and tested

for functionality based on the written code. The push-buttons tested include the set time, set alarm, snooze, and alarm off functional buttons.

V Bill of Materials (*One Table*)

Table 1: Bill of Materials.

Label in schematic	Component Description	Part number	Cost/unit quantity	Source of cost information
STM1	V_{pin}= 3V, I_{pin} = 20mA	STM32F407G-DISC1	$26.81/1pc	Digikey
U1	V_F = 2V, I = 20mA[1]	LTC-4727JR	$5.28/1pc	Digikey
R (1-8)	50Ω ± 5%, power rating = 5W	55J50RE	$9.23 /10pcs	Mouser
SW (1-5)	Thru hole 6mm x 6mm, Stem height 16.3 mm, SPST 0.05A 12VDC	TL1105YF100Q	$1.95/ 5pcs	Mouser
Q (1-4)	NMOS MOSFET. I_D = 200mA, V_{DSS} (Drain-source voltage) = 60V $V_{GS(th)\ max}$ = 3V[2] V_{GSS} (Gate-Source voltage) = 20V P_{MAX} = 400mW	2N700	$2.64/4pcs	Digikey

VI Circuit Schematic & PCB

Below are the schematics of the Alarm Clock:

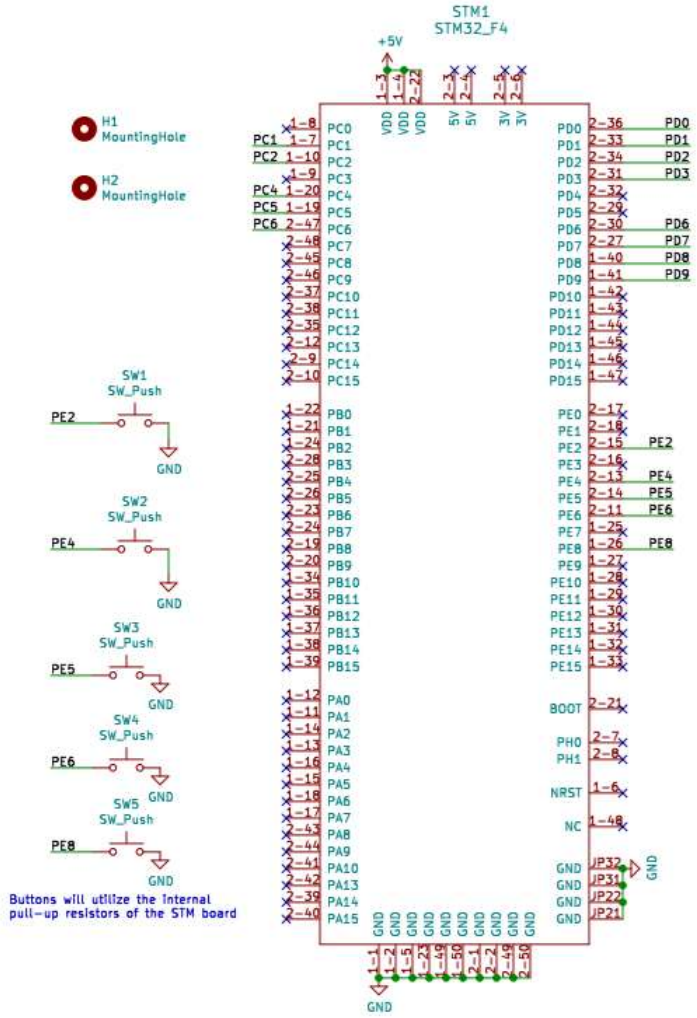


Figure 1: The schematic of the STM board and the Switches.

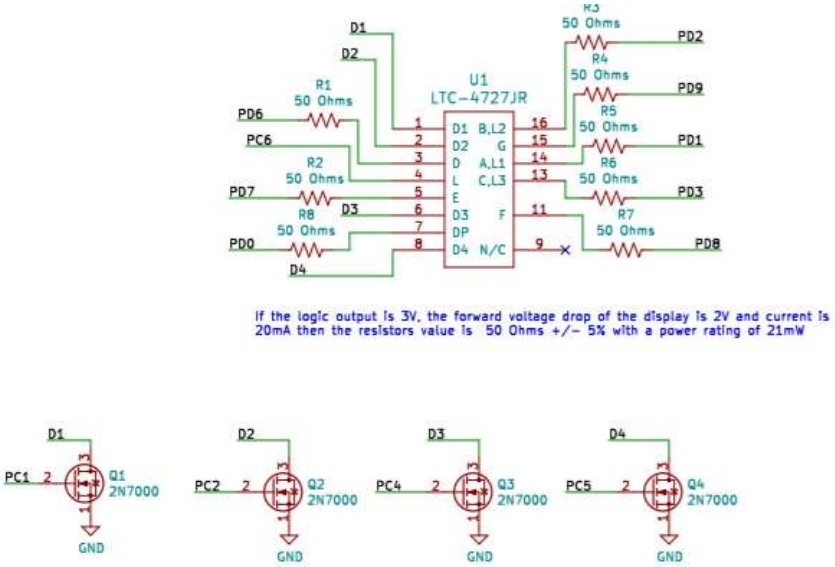


Figure 2: The schematic of the LED display, and the NMOS MOSFET.

The dimension of our PCB is 99.1mm by 69.9mm and below is the layout of the PCB:

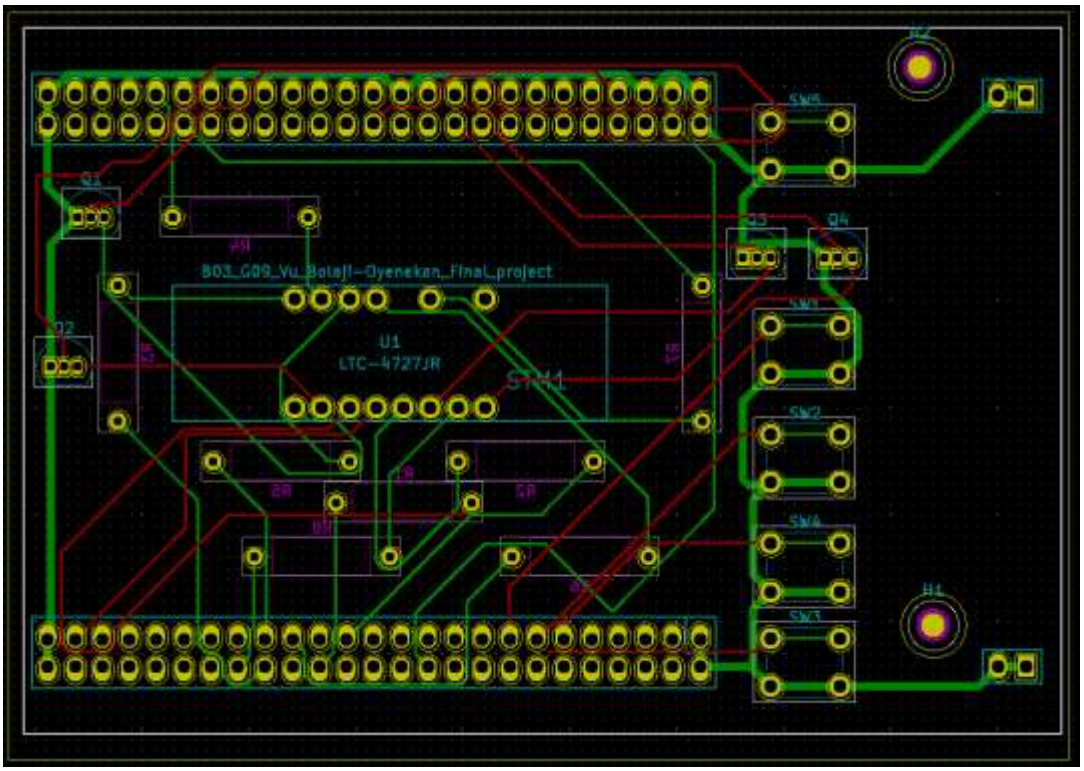


Figure 3: The final layout of the PCB.

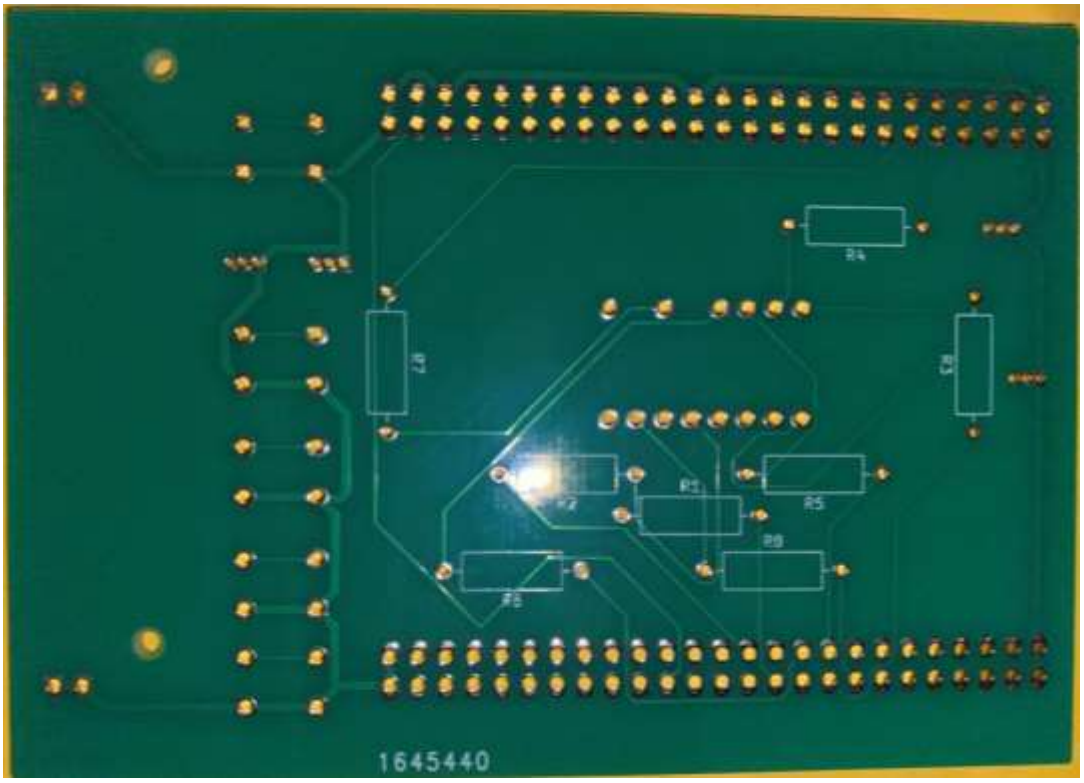


Figure 4: The manufactured PCB's Top layer.

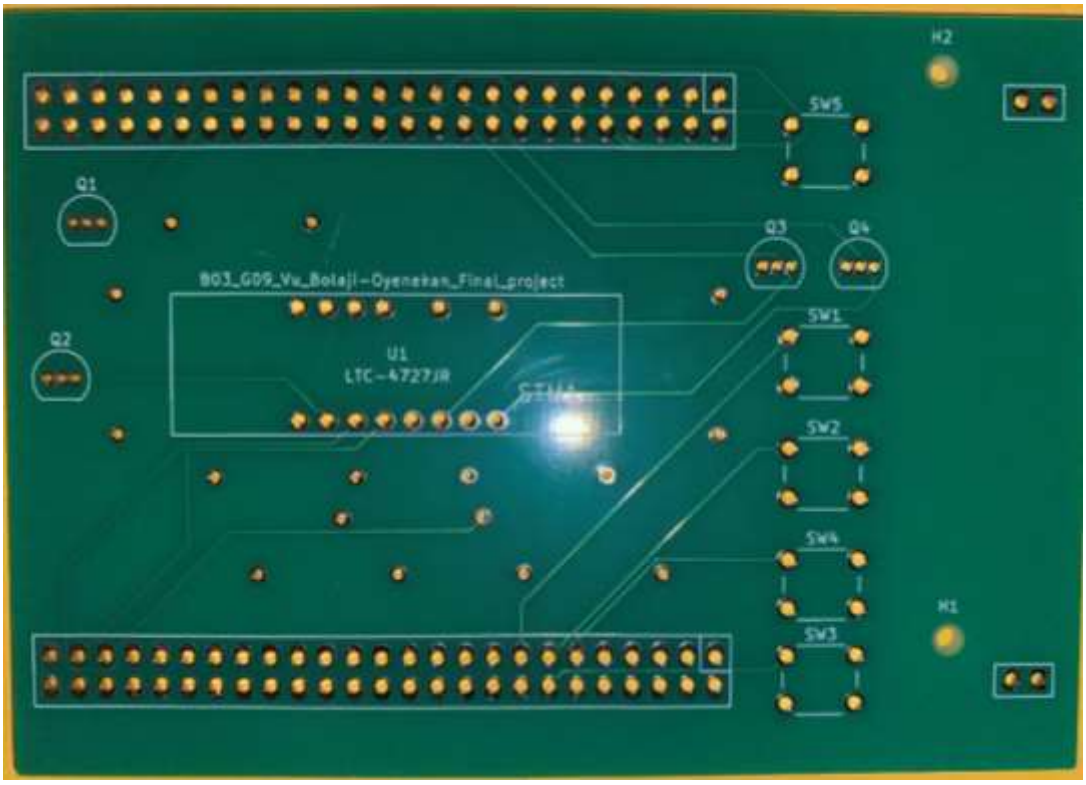


Figure 5: The manufactured PCB's Bottom layer.

VII Conclusion & Recommendations

In the end, our team has successfully created a functioning Alarm Clock that satisfied all of the specifications. Although the PCB design, the Enclosure, and the Code might not be perfect, we believe all of them are working as desired.

There have been many difficulties that occurred in this project. First of all, the project required each team to learn how to use SolidWorks and KiCAD, which our team has never used these programs before. Secondly, the limited time given during the lab has somewhat affected our demonstrations, as we are the last group in the lab and most of the time our demonstrations took place right before the lab ends, making it harder to ask the TAs for suggestions. Last but not least, David (One of the members) has fallen ill and was hospitalized during the time of the final milestone. As a result of this, we could not finish the code in time for our demonstrations. Fortunately, professor Ilamparithi Thirumarai Chelvan has extended our deadline for the demonstration of the code.

If we are to redo the project, we would make sure to be more comfortable with both SolidWorks and KiCAD, to spend less time making the Enclosure and the PCB, and spend more time perfecting other aspects of the project. Our suggestions for the project would be carefully creating the PCB, especially the dimensions of each component because the layout of the PCB would heavily affect the creation of the Enclosure.

References

[1] LITE-ON Technology Corporation's Editors, “LED Display Product Datasheet (LTC-4727 JR),” 16-Jul-2009. [Online]. Available: https://optoelectronics.liteon.com/upload/download/DS30-2000-193/C4727JR.pdf. [Accessed: 17-Aug-2021].

[2] Semiconductor Components Industries, LLC, “2N7000 / 2N7002 / NDS7002AN-Channel Enhancement Mode Field Effect Transistor,” October-2017, [Online]. Available: https://rocelec.widen.net/view/pdf/orqxwkxkq1/ONSM-S-A0003544006-1.pdf?t.download=true&u=5oefqw. [Accessed: 17-Aug-2021].

Appendix A – Enclosure Details

The Enclosure was designed from the inspirations we have from the Tamagotchi (A Japanese toy) of the show Digimon, below is the image of the actual toy that inspired us:



Figure 6: The Design of the Digimon Tamagotchi.

After spending time evaluating the similarities of the toy and the Alarm Clock we were building, we decided to create the Enclosure below:

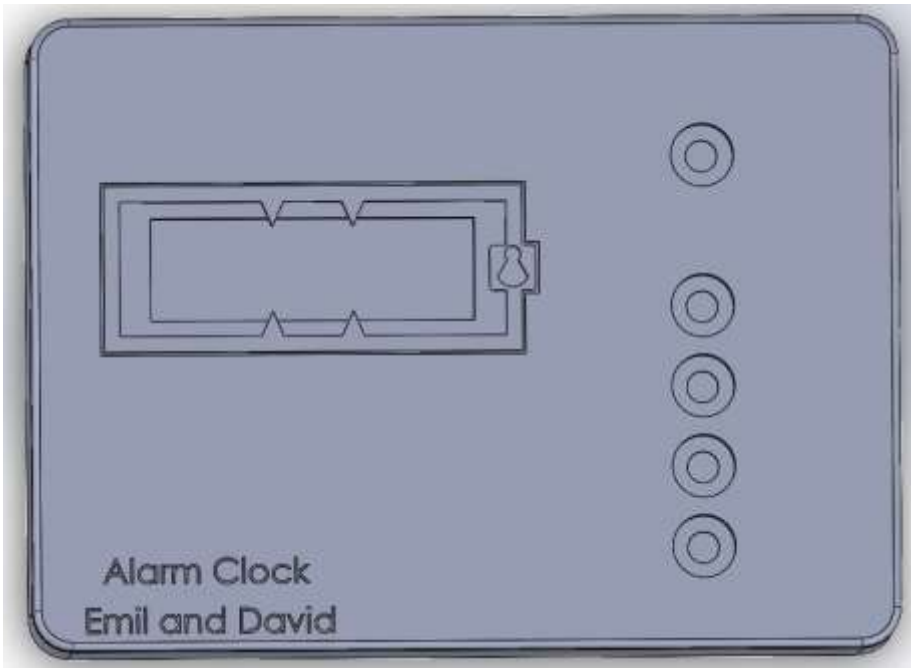


Figure 7: The front view of the Enclosure.

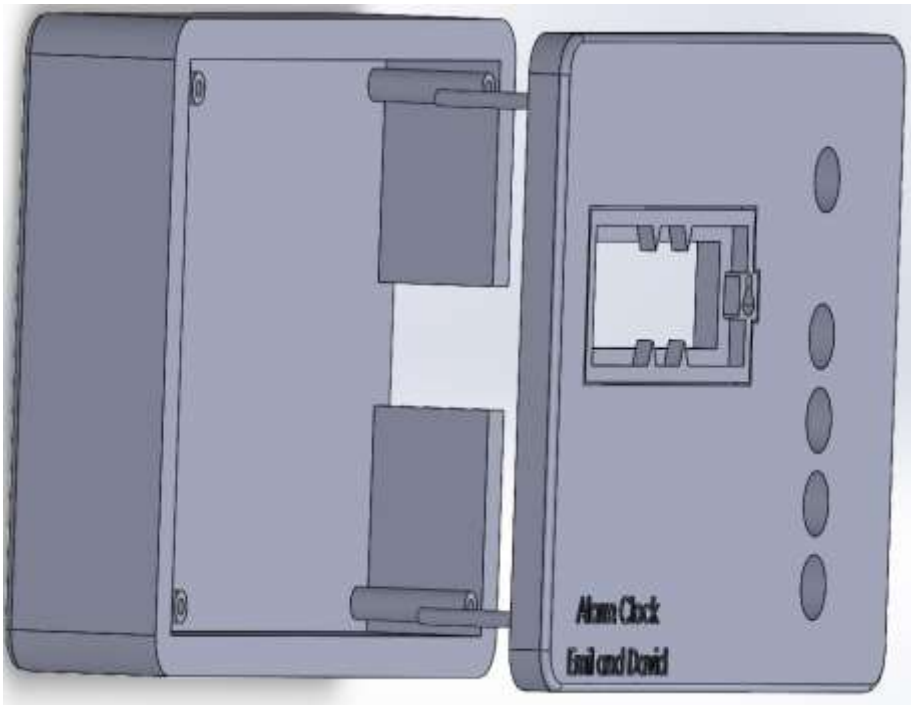


Figure 8: The side view of the Enclosure.

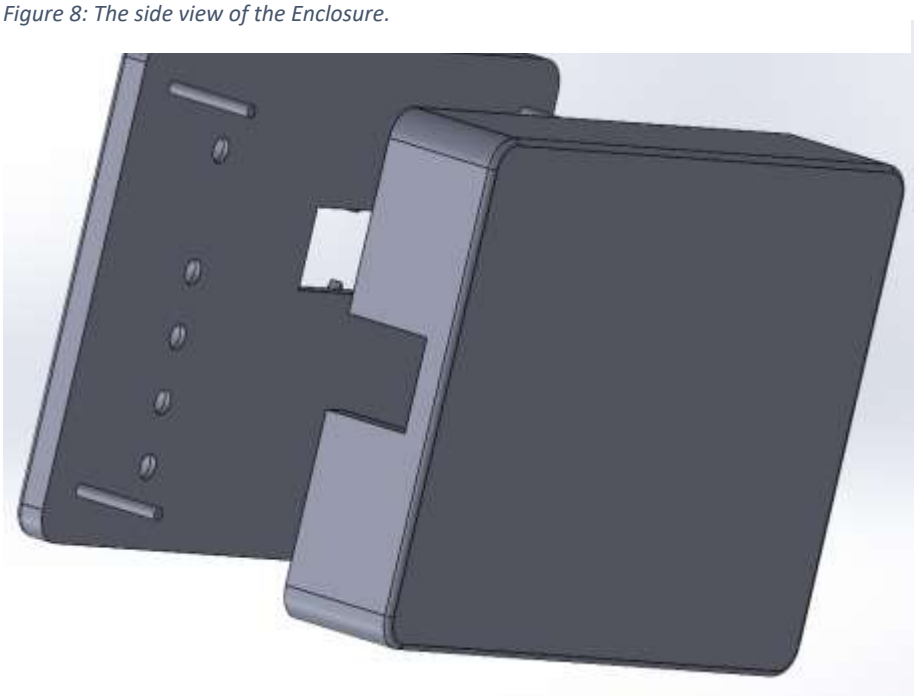


Figure 9: The back view of the Enclosure

Our closing mechanism requires the corners of the boards to have a 2 mm hole. Based on the design, the small rod would pierce through the boards and securely seal the enclosure as the back component has four 4 mm rods with 2 mm being filleted to make room for the front component.

Appendix B – Software code with comments

```
//
//   EEEEE   CCC   EEEEE   222    999    999
//   E      C   C  E      2   2  9   9  9   9
//   E      C      E          2  9   9  9   9
//   EEEEE  C      EEEEE     2    9999   9999
//   E      C      E        2        9      9
//   E      C   C  E       2        9      9
//   EEEEE   CCC   EEEEE  22222   99     99
//
//
//    AAA   L       AAA    RRRR   M   M       CCC   L       OOO    CCC   K   K
//   A   A  L      A   A   R   R  MM MM      C   C  L      O   O  C   C  K  K
//   A   A  L      A   A   R   R  M M M      C      L      O   O  C      K K
//   AAAAA  L      AAAAA   RRRR   M   M      C      L      O   O  C      KK
//   A   A  L      A   A   R R    M   M      C      L      O   O  C      K K
//   A   A  L      A   A   R  R   M   M      C   C  L      O   O  C   C  K  K
//   A   A  LLLLL  A   A   R   R  M   M       CCC   LLLLL   OOO    CCC   K   K
//
// Version 1.01
//
// ------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include "diag/Trace.h"
#include "stm32f4xx_hal.h"
//#include "Timer.h"


#define CLOCK_HOUR_FORMAT_12    (0)       // Display the time with AM/PM format
#define CLOCK_HOUR_FORMAT_24    (1)       // Display the time with 24 Hour format

#define TRUE  ( 1 == 1 )
#define FALSE ( 1 == 0 )

//
// Disable specific warnings
//

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"

//
// remove comment slashes on line below to enable real time clock support
//
#define USE_RTC

// ------------------------------------------------------------------------
//
// Standalone STM32F4 Simple Alarm Clock Stub Code
//
// This code just plays an MP3 file off of a connected USB flash drive.
//
// Trace support is enabled by adding the TRACE macro definition.
```

```
// By default the trace messages are forwarded to the DEBUG output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//

void
	Display7Segment( void ),
	SetTime( void ),
	SetAlarm( void ),
	Snooze( void ),
	ProcessButtons( void ),
	GetCurrentTime( void ),
	SystemClock_Config( void );


uint16_t
	CheckButtons( void );


//
// Global variables
//

#ifdef USE_RTC

RTC_InitTypeDef
	ClockInit;			// Structure used to initialize the real time
clock

RTC_HandleTypeDef
	RealTimeClock;			// Structure for the real time clock subsystem

RTC_TimeTypeDef
	ClockTime;			// Structure to hold/store the current time

RTC_DateTypeDef
	ClockDate;			// Structure to hold the current date

RTC_AlarmTypeDef
	ClockAlarm;			// Structure to hold/store the current alarm
time

#endif



TIM_HandleTypeDef
	DisplayTimer;		// Structure for the LED display timer subsystem



volatile int
	Alarm = FALSE,			// Flag indicating alarm
	DebounceCount1 = 0,	// Buttons debounce count
    DebounceCount2 = 0,
    DebounceCount3 = 0,
    DebounceCount4 = 0,
```

```
    DebounceCount5 = 0,
       a = 0,
       b = 0;

volatile uint16_t
       ButtonsPushed = 0x0000;                  // Bit field containing the bits of which
buttons have been pushed

volatile int
       BcdTime[4],                         // Array to hold the hours and minutes in BCD
format
       DisplayedDigit = 0,         // Current digit being displayed on the LED display
    Button1,
       Button2,
       Button3,
       Button4,
       Button5,
       SelectedDigitAlarm = 0,
       SelectedDigitClock = 0,
       DisplayMode = 1,
                                                // Current format for the displayed time
( IE 12 or 24 hour format )
       ClockHourFormat = CLOCK_HOUR_FORMAT_12,
       AlarmPmFlag = 0,
       TimePmFlag = 0;

void ConfigureDisplay( void )
{
//     GPIO_InitTypeDef
//            GPIO_InitStructure;

//
//  Enable clocks for PWR_CLK for RTC, GPIOE, GPIOD, GPIOC and TIM5.
//

//
// Enable the LED multiplexing display and push button timer (TIM5) at a frequency of
250Hz
//

//
// Configure the 7 segments, 5 digit controls and alarm signal as outputs
//

//
// Configure the push button switches ( maximum 6 ) as inputs with pull-ups
//

       __HAL_RCC_GPIOD_CLK_ENABLE(); // enabling clock for port E
       __HAL_RCC_GPIOE_CLK_ENABLE(); // enabling clock for port D
       __HAL_RCC_GPIOC_CLK_ENABLE(); // enabling clock for port C
       __HAL_RCC_GPIOA_CLK_ENABLE(); // enabling clock for port A(TIM5)
       __HAL_RCC_RTC_ENABLE(); // enabling clock for RTC

       GPIO_InitTypeDef GPIO_InitStructure; // handle for pointing GPIO
       GPIO_InitStructure.Pin = GPIO_PIN_1;// assign pin 1 as timer
       GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP; //set mode to output
       GPIO_InitStructure.Speed = GPIO_SPEED_LOW; // set speed to be low
```

```
	GPIO_InitStructure.Pull = GPIO_NOPULL; // assign no pull-up resistor
	GPIO_InitStructure.Alternate = GPIO_AF2_TIM5; // assign TIM5 alternate function

	HAL_GPIO_Init(GPIOA, &GPIO_InitStructure); // initialize port A with the handle

	__HAL_RCC_TIM5_CLK_ENABLE();
	DisplayTimer.Instance = TIM5;
	DisplayTimer.Init.Prescaler = 839;
	DisplayTimer.Init.CounterMode = TIM_COUNTERMODE_UP;
	DisplayTimer.Init.Period = 399 ;
	DisplayTimer.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
	DisplayTimer.Init.RepetitionCounter = 0;
	HAL_TIM_Base_Init( &DisplayTimer );
	// Set priority for the interrupt.
	// Value 0 corresponds to highest priority.

		HAL_NVIC_SetPriority( TIM5_IRQn, 0, 0);

	//Enable interrupt function request of Timer5
		HAL_NVIC_EnableIRQ( TIM5_IRQn );


	// Enable timer interrupt flag to be set when timer count is
	// reached
		__HAL_TIM_ENABLE_IT( &DisplayTimer, TIM_IT_UPDATE );

	// Enable timer to start
		__HAL_TIM_ENABLE( &DisplayTimer );


	GPIO_InitTypeDef GPIO_InitStructure1; // handle for pointing GPIO
	GPIO_InitStructure1.Pin =
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9;/
/ assign pins 0 for Alarm, and 1,2,3,6,7,8,9 for led segments a-g
	GPIO_InitStructure1.Mode = GPIO_MODE_OUTPUT_PP; //set mode to be push-pull output
	GPIO_InitStructure1.Speed = GPIO_SPEED_LOW; // set speed to be low
	GPIO_InitStructure1.Pull = GPIO_NOPULL; // assign no pull-up resistor
	GPIO_InitStructure1.Alternate = 0; // assign no alternate function
	HAL_GPIO_Init(GPIOD, &GPIO_InitStructure1); // initialize port D with the handle

	GPIO_InitTypeDef GPIO_InitStructure2; // handle for pointing GPIO
	GPIO_InitStructure2.Pin =
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6;// assign pins 1,2,4,5 for digit
control and 6 for dots
	GPIO_InitStructure2.Mode = GPIO_MODE_OUTPUT_PP; //set mode to output
	GPIO_InitStructure2.Speed = GPIO_SPEED_LOW; // set speed to be low
	GPIO_InitStructure2.Pull = GPIO_NOPULL; // assign pull-up resistor
	GPIO_InitStructure2.Alternate = 0; // assign no alternate function

	HAL_GPIO_Init(GPIOC, &GPIO_InitStructure2); // initialize port C with the handle

	GPIO_InitTypeDef GPIO_InitStructure3; // handle for pointing GPIO
	GPIO_InitStructure3.Pin =
GPIO_PIN_2|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_8;// assign pins 2 and 4 for button
1 and 2
	GPIO_InitStructure3.Mode = GPIO_MODE_INPUT; //set mode to input
	GPIO_InitStructure3.Speed = GPIO_SPEED_LOW; // set speed to be low
	GPIO_InitStructure3.Pull = GPIO_PULLUP; // assign pull-up resistor
```

```
	GPIO_InitStructure3.Alternate = 0; // assign no alternate function

	HAL_GPIO_Init(GPIOE, &GPIO_InitStructure3); // initialize port E with the handle




}


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

//
// Reset of all peripherals, Initializes the Flash interface and the System timer.
//
	HAL_Init();

//
// Configure the system clock
//
	SystemClock_Config();

//
// Display project name with version number
//
	trace_puts(
			"*\n"
			"*\n"
			"* Alarm clock project for stm32f4discovery board V2.00\n"
			"*\n"
			"*\n"
			);

//
// Initialize the seven segment display pins and push buttons
//
	ConfigureDisplay();

//
// Send a greeting to the trace device (skipped on Release).
//
	trace_puts("Initialization Complete");

//
// At this stage the system clock should have already been configured at high speed.
//

	trace_printf("System clock: %u Hz\n", HAL_RCC_GetSysClockFreq() /* SystemCoreClock
*/ );
    trace_printf( "HClk frequency %u\r\n", HAL_RCC_GetHCLKFreq());
    trace_printf( "PClk 1 frequency %u\r\n", HAL_RCC_GetPCLK1Freq());
    trace_printf( "PClk 2 frequency %u\r\n", HAL_RCC_GetPCLK2Freq());

	Alarm = FALSE;
```

```
	RCC_OscInitTypeDef
		RCC_OscInitStruct1;

	RCC_PeriphCLKInitTypeDef
		PeriphClkInitStruct1;

//
// Configure LSI as RTC clock source
//
	RCC_OscInitStruct1.OscillatorType = RCC_OSCILLATORTYPE_LSI;
	RCC_OscInitStruct1.PLL.PLLState = RCC_PLL_NONE;
	RCC_OscInitStruct1.LSIState = RCC_LSI_ON;

	if( HAL_RCC_OscConfig(&RCC_OscInitStruct1) != HAL_OK )
	{
		trace_printf( "HAL_RCC_OscConfig failed\r\n");
		while( TRUE );
	}

//
// Assign the LSI clock to the RTC
//
	PeriphClkInitStruct1.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
	PeriphClkInitStruct1.PeriphClockSelection = RCC_PERIPHCLK_RTC;
	if(HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct1) != HAL_OK)
	{
		trace_printf( "HAL_RCCEx_PeriphCLKConfig failed\r\n");
		while( TRUE );
	}

//
// Enable the RTC
//
	__HAL_RCC_RTC_ENABLE();

//
// Configure the RTC format and clock divisor
//

	RealTimeClock.Instance = RTC;
	RealTimeClock.Init.HourFormat = RTC_HOURFORMAT_12;

	RealTimeClock.Init.AsynchPrediv = 127;
	RealTimeClock.Init.SynchPrediv = 0xFF;
	RealTimeClock.Init.OutPut = RTC_OUTPUT_DISABLE;
	RealTimeClock.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
	RealTimeClock.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
	HAL_RTC_Init(&RealTimeClock );

//
// Disable the write protection for RTC registers
//
	__HAL_RTC_WRITEPROTECTION_DISABLE( &RealTimeClock );

//
// Disable the Alarm A interrupt
```

```
//
	__HAL_RTC_ALARMA_DISABLE( &RealTimeClock );

//
// Clear flag alarm A
//
	__HAL_RTC_ALARM_CLEAR_FLAG(&RealTimeClock, RTC_FLAG_ALRAF);

//
// Structure to set the time in the RTC
//
	ClockTime.Hours = 11;
	ClockTime.Minutes = 00;
	ClockTime.Seconds = 00;
	ClockTime.SubSeconds = 0;
	ClockTime.TimeFormat = RTC_HOURFORMAT_12;
	ClockTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
	ClockTime.StoreOperation = RTC_STOREOPERATION_RESET;

//
// Structure to set the date in the RTC
//

	ClockDate.Date =     21;
	ClockDate.Month =    RTC_MONTH_JUNE;
	ClockDate.WeekDay = RTC_WEEKDAY_WEDNESDAY;
	ClockDate.Year =     17;

//
// Set the date and time in the RTC
//

	HAL_RTC_SetDate(&RealTimeClock, &ClockDate, RTC_FORMAT_BIN);
	HAL_RTC_SetTime(&RealTimeClock, &ClockTime, RTC_FORMAT_BIN);



	HAL_NVIC_SetPriority( RTC_Alarm_IRQn, 0, 0 );
	HAL_NVIC_EnableIRQ( RTC_Alarm_IRQn );


//
// Set the initial alarm time
//
	ClockAlarm.Alarm = RTC_ALARM_A;
	ClockAlarm.AlarmTime.TimeFormat = RTC_HOURFORMAT12_PM;
	ClockAlarm.AlarmTime.Hours = 0x0C;
	ClockAlarm.AlarmTime.Minutes = 0x00;
	ClockAlarm.AlarmTime.Seconds = 0x05;
	ClockAlarm.AlarmMask = RTC_ALARMMASK_DATEWEEKDAY;
	ClockAlarm.AlarmDateWeekDay = 1;
	HAL_RTC_SetAlarm_IT( &RealTimeClock, &ClockAlarm, RTC_FORMAT_BIN );



	Button1 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_2);
	Button2 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_4);
	Button3 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_5);
```

```
        Button4 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_6);
        Button5 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_8);
        DebounceCount1 = 10;
        DebounceCount2 = 10;
        DebounceCount3 = 10;
        DebounceCount4 = 10;
        DebounceCount5 = 10;

//
// Start the display timer (TIM5)
//
        HAL_TIM_Base_Start_IT( &DisplayTimer );

        while ( TRUE )
        {

//
// Wait for an interrupt to occur
//
                __asm__ volatile ( "wfi" );

        }
}

/*
 *
 *  System Clock Configuration
 *
 */

void SystemClock_Config(void)
{

        RCC_OscInitTypeDef RCC_OscInitStruct;
        RCC_ClkInitTypeDef RCC_ClkInitStruct;
        RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

        __HAL_RCC_PWR_CLK_ENABLE();

        __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

        RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
        RCC_OscInitStruct.HSEState = RCC_HSE_ON;
        RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
        RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
        RCC_OscInitStruct.PLL.PLLM = 4;
        RCC_OscInitStruct.PLL.PLLN = 168;
        RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
        RCC_OscInitStruct.PLL.PLLQ = 7;
        HAL_RCC_OscConfig(&RCC_OscInitStruct);

        RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                        |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
        RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
        RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
        RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
        RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
        HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
```

```
	PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_I2S;
	PeriphClkInitStruct.PLLI2S.PLLI2SN = 192;
	PeriphClkInitStruct.PLLI2S.PLLI2SR = 2;
	HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct);

	HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

	HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

	/* SysTick_IRQn interrupt configuration */
	HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}


/*
 * Function: TIM5_IRQHandler
 *
 * Description:
 *
 *            Timer interrupt handler that is called at a rate of 250Hz.
 *
 * functionality:
 *            Poll the time and displays it on the 7 segment display
 *            Checks for button presses and handle any bounce conditions
 *            Do the appropriate operation based on the button pressed
 *
 */


void TIM5_IRQHandler(void)
{

//
// Validate the correct interrupt has occurred
//

	if( __HAL_TIM_GET_FLAG( &DisplayTimer, TIM_IT_UPDATE ) != RESET )
	{


		GetCurrentTime();

		Display7Segment();

//
// Process any button events
//
		if ( TRUE == CheckButtons())
		{
//
// Debug code
//
		trace_printf( "%04X\n", ButtonsPushed );

			ProcessButtons();
		}
//
```

```
// clear the timer interrupt flag
//
              __HAL_TIM_CLEAR_FLAG( &DisplayTimer, TIM_IT_UPDATE );
    }
}


/*
 * Function: RTC_Alarm_IRQHandler
 *
 * Description:
 *
 * When alarm occurs, clear all the interrupt bits and flags then start playing music.
 *
 */

#ifdef USE_RTC

void RTC_Alarm_IRQHandler(void)
{

//
// Verify that this is a real time clock interrupt
//
       if( __HAL_RTC_ALARM_GET_IT( &RealTimeClock, RTC_IT_ALRA ) != RESET )
       {

//
// Clear the alarm flag and the external interrupt flag
//
       __HAL_RTC_ALARM_CLEAR_FLAG( &RealTimeClock, RTC_FLAG_ALRAF );
       __HAL_RTC_EXTI_CLEAR_FLAG( RTC_EXTI_LINE_ALARM_EVENT );

//
// Restore the alarm to it's original time. This could have been a snooze alarm
//
       HAL_RTC_SetAlarm_IT( &RealTimeClock, &ClockAlarm, RTC_FORMAT_BCD );

       Alarm = TRUE;

       }
}

#endif

/*
 * Function: Display7Segment
 *
 * Description:
 *
 * Displays the current time, alarm time or time format
 *
 */

void Display7Segment(void)
{
//
// clear digit selection bits
```

```
//
	HAL_GPIO_WritePin( GPIOC, GPIO_PIN_1, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOC, GPIO_PIN_2, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOC, GPIO_PIN_4, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOC, GPIO_PIN_5, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOC, GPIO_PIN_6, GPIO_PIN_RESET );
//
// clear segment selection bits
//


	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_0, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_RESET );
	HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_RESET );
//
// Select current digit
//

	if(DisplayedDigit==0){
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_1, GPIO_PIN_SET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_2, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_4, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_5, GPIO_PIN_RESET );
	}
	if(DisplayedDigit==1){
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_1, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_2, GPIO_PIN_SET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_4, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_5, GPIO_PIN_RESET );
	}
	if(DisplayedDigit==2){
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_1, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_2, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_4, GPIO_PIN_SET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_5, GPIO_PIN_RESET );
	}
	if(DisplayedDigit==3){
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_1, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_2, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_4, GPIO_PIN_RESET );
		HAL_GPIO_WritePin( GPIOC, GPIO_PIN_5, GPIO_PIN_SET );
	}
	if(DisplayedDigit == 4){
	       HAL_GPIO_WritePin( GPIOC, GPIO_PIN_6, GPIO_PIN_SET );
	}

//
// Enable segments to be illuminated
//
		if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==0){

			HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
			HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
```

```
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_RESET );
}
if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==1){

      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_RESET );

   }
if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==2){

      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

   }
if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==3){

      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

   }
if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==4){

      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

   }
if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==5){

      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_SET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
      HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_SET );
```

```
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

                }
                if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==6){

                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_RESET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

                }
                if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==7){

                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_RESET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_RESET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_RESET );

                }
                if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==8){

                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

                }
                if(DisplayedDigit< 4 && BcdTime[DisplayedDigit]==9){

                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_1, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_2, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_3, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_6, GPIO_PIN_RESET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_7, GPIO_PIN_RESET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_8, GPIO_PIN_SET );
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_9, GPIO_PIN_SET );

                }
                if(Alarm == TRUE){
                    HAL_GPIO_WritePin( GPIOD, GPIO_PIN_0, GPIO_PIN_SET );
                }
//
// Advance to the next digit to be display on next interrupt
//
              if( DisplayedDigit == 4){
                DisplayedDigit = 0;
              }
              else{
                DisplayedDigit++;
```

```
              }
}

/*
 * Function: SetTime
 *
 * Description:
 *
 * Advance either the time hours or minutes field. Validate the new time and then update
the clock
 *
 */

void SetTime(void)
{
       if ( HAL_RTC_WaitForSynchro( &RealTimeClock ) == HAL_OK ){
              if(ButtonsPushed == 0x0011){
                if(SelectedDigitClock == 0 ){
                     if(ClockTime.Hours > 10 ){
                            ClockTime.Hours = ClockTime.Hours-10;
                     }
                }
                if(SelectedDigitClock == 1 ){
                     if(ClockTime.Hours > 01){
                            ClockTime.Hours = ClockTime.Hours-1;
                     }
                }
                if(SelectedDigitClock == 2 ){
                     if(ClockTime.Minutes > 1){
                            ClockTime.Minutes = ClockTime.Minutes-10;

                     }
                }
                if(SelectedDigitClock == 3 ){
                     if(ClockTime.Minutes > 0){
                            if(ClockTime.Minutes % 10 > 0){
                                   ClockTime.Minutes = ClockTime.Minutes-1;
                            }

                     }
                }
            }

                if(ButtonsPushed == 0x0012){
              if(SelectedDigitClock == 0 ){
                if(ClockTime.Hours < 3 ){
                       ClockTime.Hours = ClockTime.Hours+10;
                }
               }
            if(SelectedDigitClock == 1 ){
              if(ClockTime.Hours < 10){
                     ClockTime.Hours = ClockTime.Hours+1;
              }
            }
            if(SelectedDigitClock == 2 ){
              if(ClockTime.Minutes < 50){
                            ClockTime.Minutes = ClockTime.Minutes + 10;
              }
```

```
            }
            if(SelectedDigitClock == 3 ){
                    if(ClockTime.Minutes < 59){
                           if(ClockTime.Minutes % 10 < 9){
                                  ClockTime.Minutes = ClockTime.Minutes+1;
                           }

                }
            }
         }
                  ButtonsPushed = 0x0010;
                     HAL_RTC_SetDate(&RealTimeClock, &ClockDate, RTC_FORMAT_BIN);
                     HAL_RTC_SetTime(&RealTimeClock, &ClockTime, RTC_FORMAT_BIN);
       }
}

/*
 * Function: SetAlarm
 *
 * Description:
 *
 * Advance either the alarm hours or minutes field. Validate the new alarm time and then
set the alarm
 *
 */

void SetAlarm(void)
{
       if ( HAL_RTC_WaitForSynchro( &RealTimeClock ) == HAL_OK ){
              if(ButtonsPushed == 0x0009){

            if(SelectedDigitAlarm == 0 ){
              if(ClockAlarm.AlarmTime.Hours > 10 ){
                     ClockAlarm.AlarmTime.Hours = ClockAlarm.AlarmTime.Hours-10;
              }
            }
            if(SelectedDigitAlarm == 1 ){
              if(ClockAlarm.AlarmTime.Hours > 1){
                     ClockAlarm.AlarmTime.Hours = ClockAlarm.AlarmTime.Hours-1;
              }
            }
            if(SelectedDigitAlarm == 2 ){
              if(ClockAlarm.AlarmTime.Minutes > 9){
                     ClockAlarm.AlarmTime.Minutes = ClockAlarm.AlarmTime.Minutes-10;

              }
            }
            if(SelectedDigitAlarm == 3 ){
                    if(ClockAlarm.AlarmTime.Minutes > 0){
                           if(ClockAlarm.AlarmTime.Minutes % 10 > 0){
                                  ClockAlarm.AlarmTime.Minutes =
ClockAlarm.AlarmTime.Minutes-1;
                           }

                    }
            }
              }
              if(ButtonsPushed == 0x000A){
```

```
            if(SelectedDigitAlarm == 0 ){
              if(ClockAlarm.AlarmTime.Hours < 3 ){
                     ClockAlarm.AlarmTime.Hours = ClockAlarm.AlarmTime.Hours+10;
              }
            }
            if(SelectedDigitAlarm == 1 ){
              if(ClockAlarm.AlarmTime.Hours < 9){
                     ClockAlarm.AlarmTime.Hours = ClockAlarm.AlarmTime.Hours+1;
              }
            }
            if(SelectedDigitAlarm == 2 ){
              if(ClockAlarm.AlarmTime.Minutes < 49){
                     ClockAlarm.AlarmTime.Minutes = ClockAlarm.AlarmTime.Minutes + 10;
              }
            }
            if(SelectedDigitAlarm == 3 ){
                     if(ClockAlarm.AlarmTime.Minutes < 59){
                            if(ClockAlarm.AlarmTime.Minutes % 10 < 9){
                                   ClockAlarm.AlarmTime.Minutes =
ClockAlarm.AlarmTime.Minutes+1;
                            }

                     }
            }
                  }
              a = ClockAlarm.AlarmTime.Hours;
              b = ClockAlarm.AlarmTime.Minutes;
              ButtonsPushed = 0x0008;
              HAL_RTC_SetAlarm_IT( &RealTimeClock, &ClockAlarm, RTC_FORMAT_BIN );
       }
}


/*
 * Function: Snooze
 *
 * Description:
 *
 * Add 10 Minutes to the current time and validate. Update the alarm and enable.
 *
 */

void Snooze(void)
{
       if(Alarm == TRUE){
              if ( HAL_RTC_WaitForSynchro( &RealTimeClock ) == HAL_OK ){
              if(ClockAlarm.AlarmTime.Minutes > 49 ){
                     if(ClockTime.Hours == 12 ){
                            ClockAlarm.AlarmTime.Hours =  1;
                     }else{
                     ClockAlarm.AlarmTime.Hours = ClockTime.Hours + 1;
                     }
                     ClockAlarm.AlarmTime.Minutes = (ClockTime.Minutes+10) % 60 ;
                     HAL_RTC_SetAlarm_IT( &RealTimeClock, &ClockAlarm, RTC_FORMAT_BIN );
              }else{
              ClockAlarm.AlarmTime.Hours = ClockTime.Hours;
              ClockAlarm.AlarmTime.Minutes = ClockTime.Minutes+10;
```

```
                HAL_RTC_SetAlarm_IT( &RealTimeClock, &ClockAlarm, RTC_FORMAT_BIN );
                }
                }
        }
        Alarm = FALSE;
        ButtonsPushed = 0x0000;
}


/*
 * Function: GetCurrentTime
 *
 * Description:
 *
 * Return either the alarm time or current time in binary coded decimal format store in
the array BcdTime.
 *
 */

void GetCurrentTime(void)
{
        if(DisplayMode == 1){
                HAL_RTC_GetTime(&RealTimeClock, &ClockTime, RTC_FORMAT_BIN);
                HAL_RTC_GetDate(&RealTimeClock, &ClockDate, RTC_FORMAT_BIN);

                BcdTime[0] = ClockTime.Hours/10;
                BcdTime[1] = ClockTime.Hours%10;
                BcdTime[2] = ClockTime.Minutes/10;
            BcdTime[3] = ClockTime.Minutes%10;
        }
        if(DisplayMode == 2){
                HAL_RTC_GetAlarm( &RealTimeClock, &ClockAlarm,RTC_ALARM_A,
RTC_FORMAT_BIN );
                BcdTime[0] = ClockAlarm.AlarmTime.Hours/10;
                BcdTime[1] = ClockAlarm.AlarmTime.Hours%10;
                BcdTime[2] = ClockAlarm.AlarmTime.Minutes/10;
            BcdTime[3] = ClockAlarm.AlarmTime.Minutes%10;

        }

}


/*
 * Function: CheckButtons
 *
 * Description:
 *
 * Check the current state of all the buttons and apply debounce algorithm. Return TRUE
with the ButtonPushed
 * variable set indicating the button or buttons pushed if button press is detected.
 *
 */

uint16_t CheckButtons( void )
{

        if(Button1 != HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_2)){
```

```
		DebounceCount1--;
		if(DebounceCount1 > 0){
		}
		else{
			Button1 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_2);
			DebounceCount1 = 10;
			if(Button1 == GPIO_PIN_RESET){

					ButtonsPushed = 0x0010;

				return (TRUE);
			}
			else{
				ButtonsPushed = 0x0000;
			}
		}
	}
	else{
		DebounceCount1 = 10;
	}

	if(Button2 != HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_4)){
		DebounceCount2--;
		if(DebounceCount2 > 0){
		}
		else{
			Button2 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_4);
			DebounceCount2 = 10;
			if(Button2 == GPIO_PIN_RESET){

				ButtonsPushed = 0x0008;
				return (TRUE);
			}else{
				ButtonsPushed = 0x0000;
			}
		}
	}
	else{
		DebounceCount2 = 10;
	}

	if(Button3 != HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_5)){
		DebounceCount3--;
		if(DebounceCount3 > 0){
		}
		else{
			Button3 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_5);
			DebounceCount3 = 10;
			if(Button3 == GPIO_PIN_RESET){

					ButtonsPushed = (0x0004|ButtonsPushed);

				return (TRUE);
			}
		}
	}
	else{
		DebounceCount3 = 10;
```

```
	}

	if(Button4 != HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_6)){
		DebounceCount4--;
		if(DebounceCount4 > 0){
		}
		else{
			Button4 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_6);
			DebounceCount4 = 10;
			if(Button4 == GPIO_PIN_RESET){
				ButtonsPushed = (0x0002|ButtonsPushed);
				return (TRUE);
			}
		}
	}
	else{
		DebounceCount4 = 10;
	}

	if(Button5 != HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_8)){
		DebounceCount5--;
		if(DebounceCount5 > 0){
		}
		else{
			Button5 = HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_8);
			DebounceCount5 = 10;
			if(Button5 == GPIO_PIN_RESET){
				ButtonsPushed = (0x0001|ButtonsPushed);

				return (TRUE);
			}
		}
	}
	else{
		DebounceCount5 = 10;
	}

	return( FALSE );
}


/*
 * Function: ProcessButtons
 *
 * Description:
 *
 * Test for which button or buttons has been pressed and do the appropriate operation.
 *
 */

void ProcessButtons( void )
{
	if(ButtonsPushed == 0x0000){
		DisplayMode = 1;
	}
	if(ButtonsPushed == 0x0001){
      if(Alarm == TRUE){
         Alarm = FALSE;
```

```
          if ( HAL_RTC_WaitForSynchro( &RealTimeClock ) == HAL_OK ){
          ClockAlarm.AlarmTime.Hours = a;
                 ClockAlarm.AlarmTime.Minutes = b;
                 HAL_RTC_SetAlarm_IT( &RealTimeClock, &ClockAlarm, RTC_FORMAT_BIN );
          }
        }
        ButtonsPushed = 0x0000;
         }
         if(ButtonsPushed == 0x0002){
                ButtonsPushed = 0x0000;
         }
         if(ButtonsPushed == 0x0004){
        Snooze();
         }
         if(ButtonsPushed == 0x0008){
        DisplayMode = 2;
         }
         if(ButtonsPushed == 0x0009){
                SetAlarm();
         }
         if(ButtonsPushed == 0x000A){
                SetAlarm();
         }
         if(ButtonsPushed == 0x000C){
                if(SelectedDigitAlarm == 3){
                       SelectedDigitAlarm = 0;
                }
                else{
                       SelectedDigitAlarm++;
                }
                ButtonsPushed = 0x0008;
         }
         if(ButtonsPushed == 0x0010){
       DisplayMode = 1;
         }
         if(ButtonsPushed == 0x0011){
                SetTime();
         }
         if(ButtonsPushed == 0x0012){
                SetTime();
         }
         if(ButtonsPushed == 0x0014){
                if(SelectedDigitClock == 3){
                       SelectedDigitClock = 0;
                }
                else{
                       SelectedDigitClock++;
                }
                ButtonsPushed = 0x0010;
         }


}


#pragma GCC diagnostic pop

// ----------------------------------------------------------------------------
```