STM32 Alarm Clock

ECE 299

August 17th, 2021

B02 G08

Name	**V Number**
Nicholas Sandor	V00918770
Immanuel Hartono	V00962183

Acknowledgment

This project would not have been possible without all of the help we have received. We would like to acknowledge Brent Sirna, for his technical expertise with the STM boards and the software environment. We also thank our teaching assistant Narges Attarmoghaddam, and our professor Dr. Ilaparithi Thirumarai Chelvan for providing a wealth of information on project requirements, and answering our many questions. Finally, we thank our family and friends for their emotional support.

We would also like to thank PCBway, for providing us with an affordable, rapid and high quality PCB manufacturing service, and SOLIDWORKS for readily licensing its software to students like us.

Contents

List of Tables

List of Figures

1 Project Goal

In this project, the objective is to design an alarm clock that is capable of receiving inputs from push buttons, clocking a real time clock on 4-digit 7-segment LED display, set time and alarm on the clock, trigger light-based alarm when the clock reaches the alarm, snooze and turn off the alarm. The project requires the circuit PCB, software coding, and enclosure design. The project uses an STM 32 board which will be coded in eclipse, KiCAD to produce PCB schematics, and SolidWorks to create enclosure design. Overall, our project is successful in this regard.

2 Constraints

There are several constraints that need to be noticed. The LED display that is used must be LTC 4727JR which is considerably small for an LED display of alarm clocks. STM32F407 discovery kit must be used as a microcontroller. Push buttons must be used as user inputs.

KiCAD must be used for designing PCB schematics and PCB layout design. SolidWorks must be used for designing alarm clock enclosure designs. The Eclipse IDE is to be used for programming. The PCB must be manufactured and submitted to the ECE department by 30 July 2021. The enclosure design and software must be presented in the lab schedule (July 27 2021). Finally, the project reports must be submitted by 17 August 2021.

3 Requirement Specifications

The specification of the project is listed below:

1. The forward current through each LED segment in the LTC-4727JR is 25mA [1]. Therefore, the resistor must be installed in series with the stm board and LTC-4727JR so that the current went through the segment did not go beyond 25 mA
2. Use 4 MOSFET to multiplex each digit.
3. Displaying Real Time Clock in the LTC-4727JR.
4. Set time and alarm hours and minutes
5. Turn on ALARM display if alarm is triggered
6. Snooze the alarm for 10 minutes
7. Verify that the PCB and STM board fit on the enclosure dimensions.
8. The enclosure design can be open and closed
9. Design protections on the enclosure to ensure product safety
10. Make sure the enclosure is stable and creative
11. The PCB can be mounted to the STM board

The formula for the Resistor each segment is

$$V=IR \quad (1)$$

where
V is Voltage provided by STM (5V)
I is the current flow to the series circuit (A)
R is combination of R resistor and R each segment (Ω)

Another formula to help the calculation is

$$R_{series} = R1 + R2 + \dots \quad (2)$$

where
R series is the total of resistance in a series (Ω)
R1 is resistance of the first component (Ω)
R2 is resistance of another component (Ω)

Notice that the current on each component of the circuit will be the same since the current is in series.

4 Design and Validation

The following section outlines the design process for the board, justifying design decisions and showing relevant calculations for component values. Validation procedures for the display subsystem and pushbutton subsystem are also detailed.

4.1 Design procedure

The following steps were taken for our design.

4.1.1 7 Segment display

As per our requirements, the LTC-4727JR was used as our 4x7 segment display. Driving this display would require current limiting resistors on the LED elements, and MOSFETS to multiplex the 4 digits.

The following procedure was used to obtain the value for the current limiting resistors:

As per the datasheet for the LTC-4727JR, the forward current of the LEDs is 25mA with a typical forward voltage drop of 2V [1].

Since the voltage drop is 2V and the forwards current of each segment is 25 mA [1]
V=IR
2 = 25 * 10^-3 *R
R each segment = 80 Ω

The voltage given by each pin in STM board is 5V [3]
V = I (R resistor + R each segment)
5 = 25 *10^-3 *(R resistor + 80)
R resistor = 120 Ω

With 5 % tolerance, the resistor is still high enough to make the current below 25 mA.

Four MOSFETs were used to multiplex the four individual digits. The 2n7000 MOSFET was selected as it works with our respective logic levels at its gate and is capable of handling the requisite current and voltage from the 7 segment display.

It should be noted that the 2N7000 MOSFET gates have a capacitance [2]. This means that there will be an inrush current if the microcontroller outputs are hooked up directly to the gates. This inrush current may exceed the maximum allowable output current on the STM32, damaging the IO ports. As such, 1 Kohm gate resistors were added to create an RC system which will limit the inrush current. As the gate capacitance is not specified, a precise calculation of the ideal resistor value is not possible. As such, a standard ‘safe’ value of 1k was selected.

4.1.2 Pushbuttons

In accordance with our requirement specifications, we selected an interface that would use four buttons. The functionality of these four buttons is described below:

Table 1: Button functionality

Button	Function
1: Increment	Increases the current set time by a minute if in set mode, or increases the set alarm time if in alarm mode
2: Set	If pressed once, toggles set time mode to set the RTC time. If held, enters set alarm mode for setting the alarm time.
3: Reset	Snooze button for alarm
4: Decrement	Decreases the current set time by a minute if in set mode, or decreases the set alarm time if in alarm mode

The buttons were configured with pull down resistors for simplicity in the code (ie a high voltage level will correspond to a logic 1, meaning a pressed button).

4.1.3 LED indicators

If the clock was to be used in 12 hr mode, an indicator would be needed to show that the time displayed is in AM or PM. Further, a visual indicator was required for the alarm. To this effect, two LEDs were included in our design. The AM/PM indicator is a simple red LED, while the alarm LED is a RGB LED. An RGB LED was selected to make the most notable visual indication that the alarm is going off.

As with the 7 segment driver, current limiting resistors were required to safely operate the LEDs. The calculation for the values used is provided below:

The LED RGB can be used to indicate alarm time. The current limiting resistor for each LED is calculated. Since the forward voltage of Red led is 1.95V [4]

V resistor(r) = V stm - V red
V resistor(r) = 5 V - 1.95 V
V resistor(r) = 3.05V
Since the current limit is 20 mA for LED [4]

Resistor (r) = 3.05 V / 20mA
Resistor (r) = 152 Ω
150 Ω is used for the red LED

Since the forward voltage of Green and Blue LED is 3.3 V [4]
V resistor(b&g) = 5 - 3.3
V resistor(b&g) = 1.7 V
Since the current limit is 20 mA [4]

R resistor(b&g) = 1.7 / 0.02
R resistor(b&g) = 85 Ω
82 Ω is used for green and blue LED

4.2 Test procedures

The following tests of the system are detailed below:

- Display verification
- Real time clock verification
- Button verification
- Alarm verification
- Physical Inspection

All tests are to be carried out at 25 degrees celsius in a low moisture, anti-static environment. Failure to conform to any listed requirements constitutes a failed test.

4.2.1 Display verification

1. Power the clock using the micro-USB jack (5V supply)
2. The display will power all segments and the LEDs for 1 second at startup. Ensure that all 7 segments and both LED's turn on.
3. Verify that no segments are stuck on after the startup flash

4.2.2 Real Time Clock (RTC) verification

1. Power the clock using the micro-USB jack (5V supply)
2. Wait for the time to be displayed 1s after startup.
3. Use a calibrated clock (phone or internet) and ensure that the minute indicator is updating in exactly 60 seconds.
4. Power the device off, then power it again. Check that the time has not deviated.

4.2.3 Button Verification

1. Power the clock using the micro-USB jack (5V supply)
2. Press the set button.
3. Press increment. Verify that the set minute increments by one
4. Press decrement. Verify that the set minute decrements by one.
5. Hold the set button. Verify that the display starts flashing.
6. Press increment. Verify that the alarm minute increments by one
7. Press decrement. Verify that the alarm minute decrements by one.

4.2.4 Alarm verification

1. Set the alarm to one minute ahead of the current RTC time, and wait for it to go off.
2. Verify that the alarm LED begins to flash
3. Press RESET, verify that the alarm LED stops flashing
4. Wait one minute, verify that the snooze alarm was set and that the LED is flashing again.
5. Hold the SET button, verify that the LED stops and the snooze does not go off in a minute

4.2.5 Physical Inspection

1. Verify that the plastic chassis fits together properly. Ensure that the top and bottom sections make a proper compression fit
2. Ensure that the alarm PCB is securely attached to the STM32
3. Ensure that the LED elements and pushbuttons are properly aligned and accessible from the holes on the front of the case.
4. Plug in a micro USB cable and ensure that the jack fits.

4.3 Validation

In order to validate the prototype, it must be able to pass all of the test procedures. Failure of a single test in the procedures constitutes failure of the prototype and it cannot be validated.

The validation tests above were run on our prototype (with some exceptions due to the limitations imposed by the COVID-19 Pandemic) and were successful. A functional prototype was demonstrated in the lab environment (see figures 1 and 2):

Figure 1: Software clock emulator showing RTC display of time

Figure 2: Software clock emulator showing RTC time display after changing set time

5 Bill of Materials

The following bill of materials outlines the required components for the construction of the board:

Table 2: Bill of Materials

Schematic Label	Component Description	Part number	Cost (CAD)	#	Source
STM1	STM32_F4 microcontroller board	STM32F407 G-DISC1	26.81	1	https://www.digikey.ca/en/products/detail/stmicroelectronics/STM32F407G-DISC1/5824404
R6-R10, R18-R20	120Ω Current limiting resistor, 7 Segment driver	MFR-25FRF5 2-120R	0.13	8	https://www.digikey.ca/en/products/detail/yageo/MFR-25FRF52-120R/9138864
R5,R11,R15 ,R17	1kΩ MOSFET gate protection resistor	RNF14FTD1 K00	0.13	4	https://www.digikey.ca/en/products/detail/stackpole-electronics-inc/RNF14FTD1K00/1706678
R1-R4	10kΩ Button pull-down resistors	MFR-25FBF5 2-10K	0.13	4	https://www.digikey.ca/en/products/detail/yageo/MFR-25FBF52-10K/13219
R16,R14	150Ω LED current limiting resistors	MFR-25FBF5 2-150R	0.13	2	https://www.digikey.ca/en/products/detail/yageo/MFR-25FBF52-150R/12831
R12,R13	82 Ω LED current limiting resistors	RNMF14FTC 82R0	0.13	2	https://www.digikey.ca/en/products/detail/stackpole-electronics-inc/RNMF14FTC82R0/2617382
SW1-SW4	Pushbutton switches	MJTP1230	0.14	4	https://www.digikey.ca/en/products/detail/apem-inc/MJTP1230/1798037
Q1-Q4	MOSFET drivers for 7 segment multiplexing	2N7000	0.57	4	https://www.digikey.ca/en/products/detail/onsemi/2N7000/244278
U1	4x7 Segment LED display	LTC-4727JR	5.28	1	https://www.digikey.ca/en/products/detail/lite-on-inc/LTC-4727JR/408224
D1	AM/PM indicator LED	SLX-LX3054H D	0.03	1	https://www.mouser.com/ProductDetail/Lumex/SLX-LX3054HD?qs=eCH4ODkH9hGpYweCiylUkg==
D2	RGB alarm LED	L-154A4SUR KQBDZGW	1.15	1	https://www.newark.com/kingbright/l-154a4surkqbdzgw/led-5mm-red-green-blue-0-2-1-3/dp/66W1973
		TOTAL	38.71		

6 Circuit Schematic and Printed Circuit Board

The following section demonstrates the circuit schematic that was used for the Printed Circuit board, as well as an image of the final pcb layout and the printed board.

6.1 Schematic

Note: a full scale version of this schematic is attached with this report.

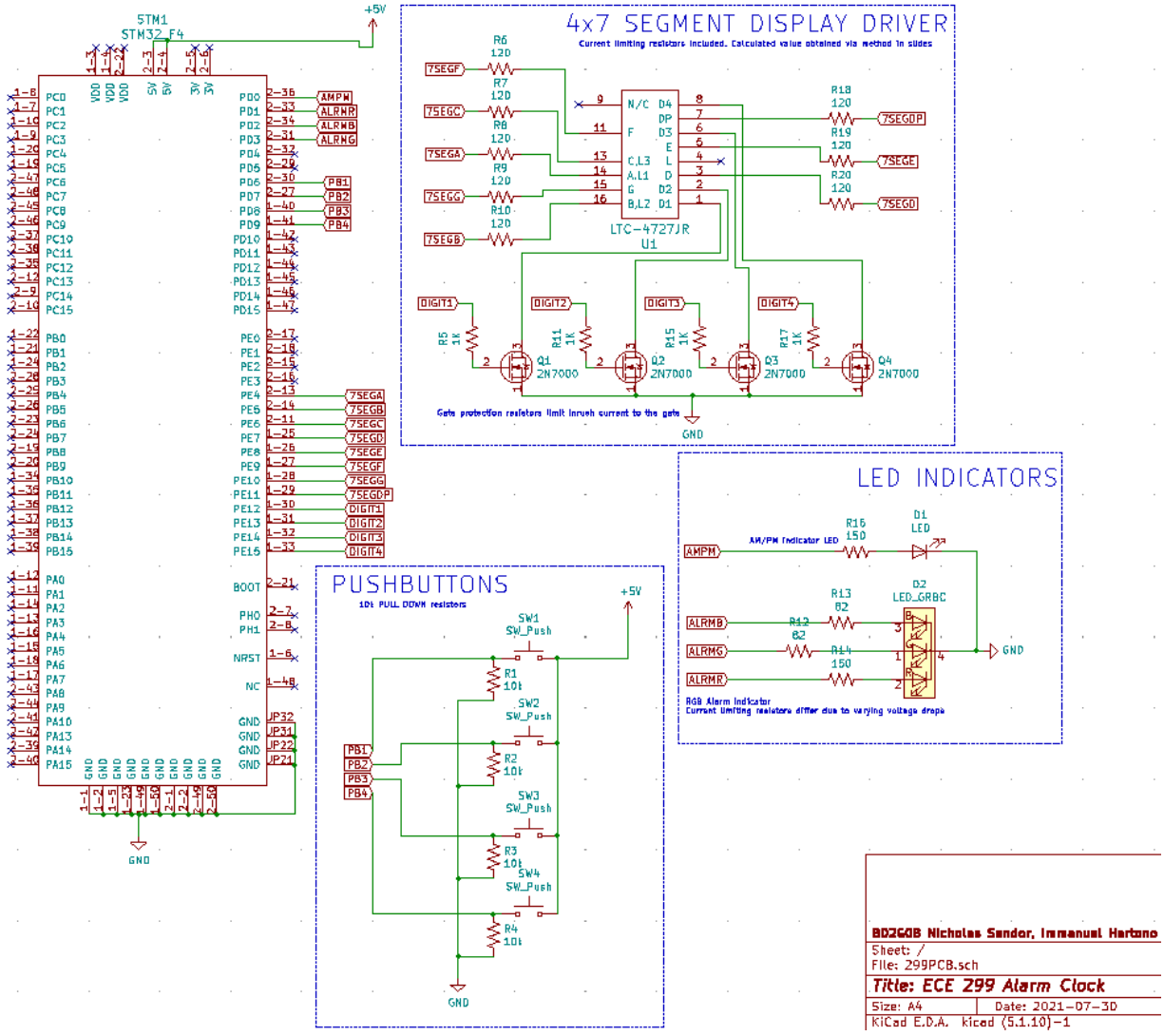

Figure 3: Circuit schematic with comments

6.2 PCB

The following figures demonstrate the gerber file which was sent for manufacture, and the completed PCB.

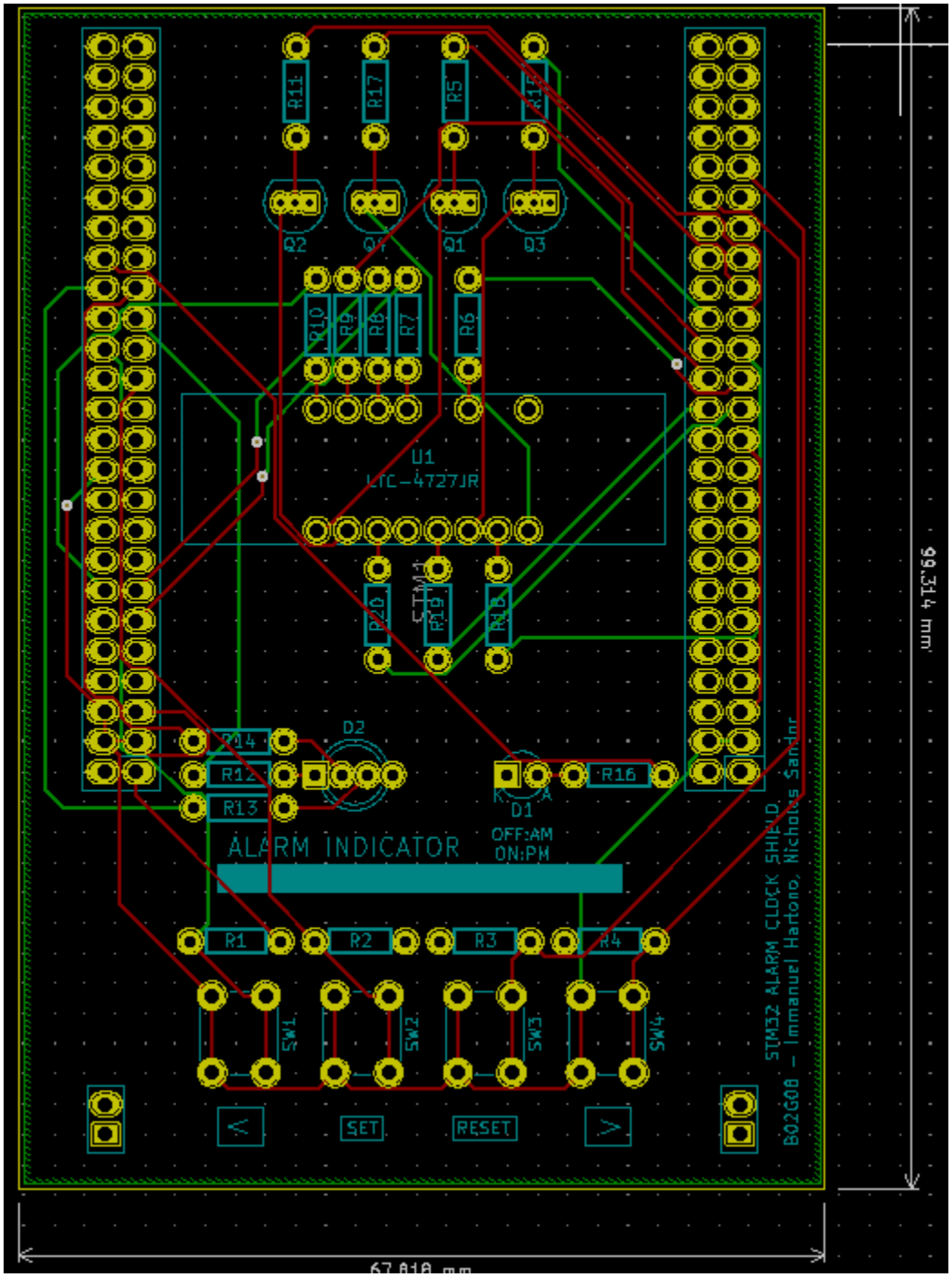

Figure 4: Final PCB layout with descriptive silkscreen and measurements shown

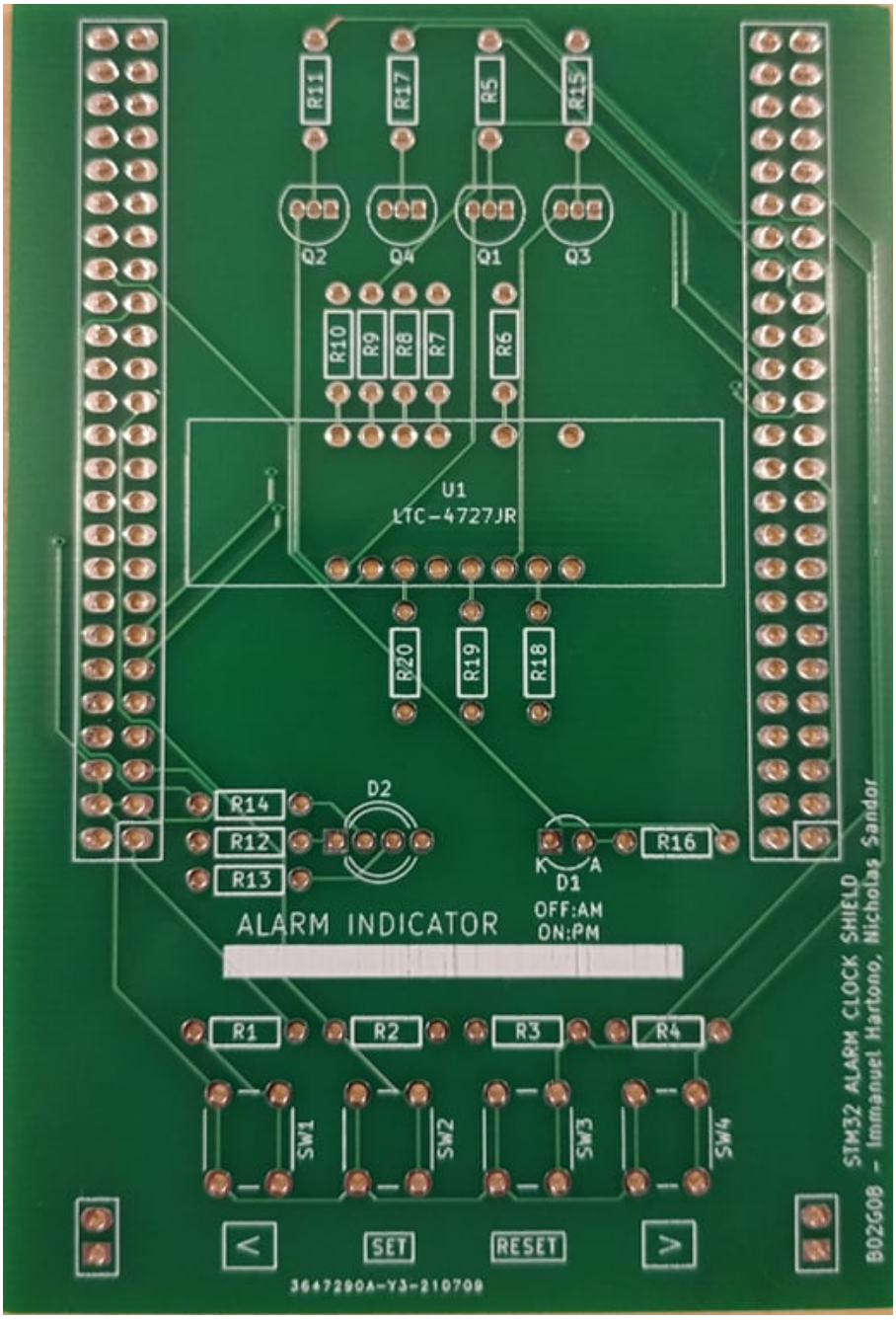

Figure 5 : Printed PCB from PCBWAY

7 Conclusion and Recommendations

Overall, our design for an STM32 based alarm clock was a success. We were able to meet our project requirement specifications to a satisfactory degree. The completed design was demonstrated in the lab environment and performed as expected.

We were able to meet our objectives for the electrical system. Our schematic design was implemented in a way that successfully limited the current through all onboard LEDs and multiplexed the LTC-4727JR. We were able to meet our software objectives, by getting the display to show the RTC time and have it modified by pushbuttons. We were able to set the RTC alarm and have it trigger a visible alarm, which could then be snoozed. Our enclosure design was successful at meeting the proper dimensions and containing all of the components while giving access to the pushbuttons and display.

There is, however, room for possible improvement with our design. A larger push button could be added to the snooze function, as it is often desirable to be able to press it easily. An auditory alarm would be a more effective method of alerting the user, so the addition of a speaker is desirable. The enclosure design can be improved and stylized, and further, the buttons can be enclosed more completely for environmental protection. Finally, the ability to run the device from a battery would allow the alarm clock to be used in a more diverse range of circumstances.

In conclusion, we were able to design an STM32 based alarm clock which met our design objectives.

References

[1] LiteOn Optoelectronics,"LTC-4727JR", DS30-2000-193, July 2009. Available: https://optoelectronics.liteon.com/upload/download/ds30-2000-193/c4727jr.pdf [Accessed Aug. 17,2021]

[2] OnSemi,"2N7000",NDS7002A/D, August 2021. Available: https://www.onsemi.com/pdf/datasheet/nds7002a-d.pdf [Accessed Aug. 17,2021]

[3] ST life.augmented,"STM32F407xx", DS8626, August 2020. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf [Accessed Aug. 17,2021]

[4] Kingbright, "FULL COLOR LED LAMP ", DSAF8941, October 2012. Available: https://www.farnell.com/datasheets/1683535.pdf [Accessed Aug. 17,2021]

Appendix A - Enclosure Details

The enclosure designed for the alarm clock is detailed in the following figure and drawing:

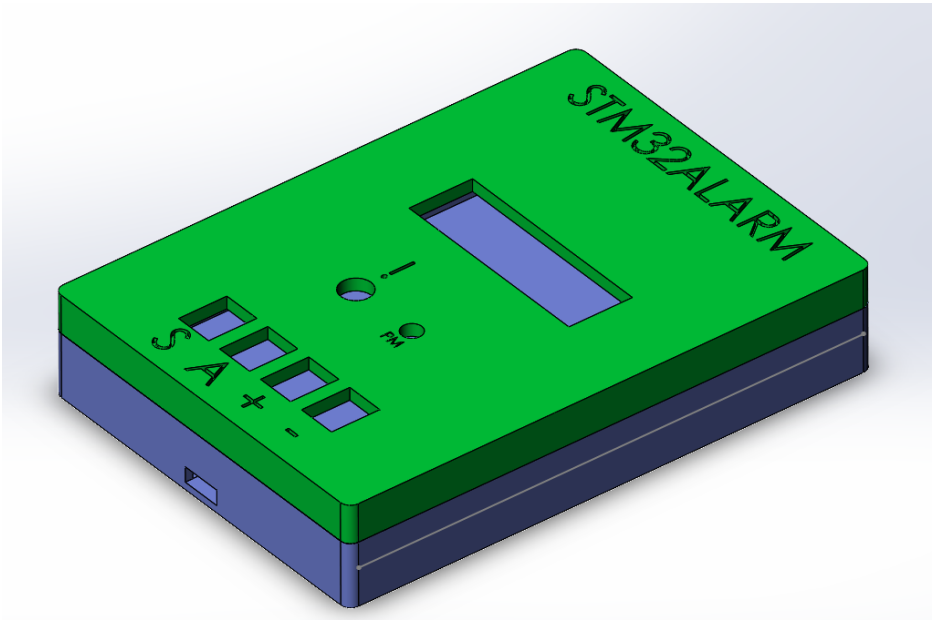

Figure 6 : Final enclosure assembly

Note: a full scale version of this drawing is attached with this report.

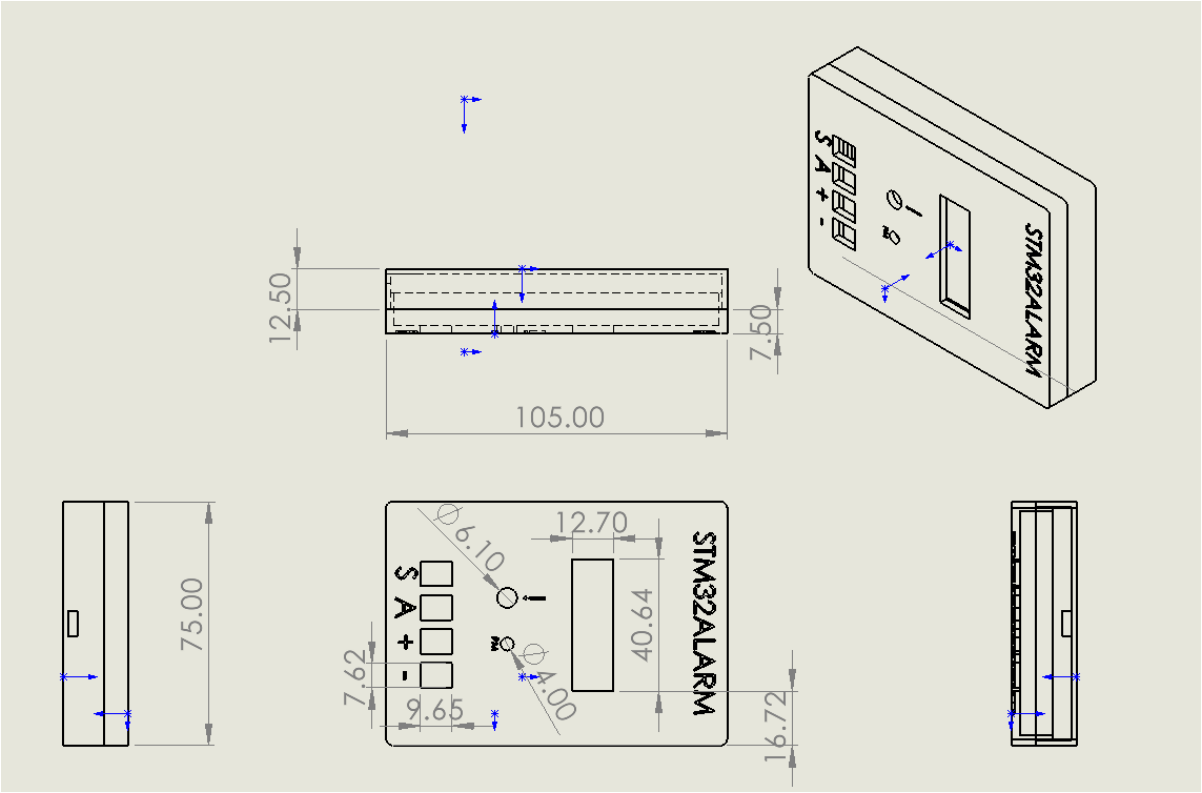

Figure 7: Final enclosure technical drawing

Appendix B - Commented Source Code

This source code expands on the provided template in brightspace.

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

// define each segment on 7 segment LED display
#define A GPIO_PIN_2
#define B GPIO_PIN_4
#define C GPIO_PIN_5
#define D GPIO_PIN_6
#define E GPIO_PIN_7
#define F GPIO_PIN_8
#define G GPIO_PIN_9

// enable real time clock support
#define USE_RTC

// declare all the functions
void
    Display7Segment( void ),
    SetTime( void ),
    SetAlarm( void ),
```

```
    Snooze( void ),
    ProcessButtons( void ),
    GetCurrentTime( void ),
    SystemClock_Config( void ),
   RTC_Alarm_IRQHandler(void);

// check button function
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
    AlarmTime;                // Structure to hold/store the current alarm
time

volatile uint32_t
    ClockAlarm;

#endif



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

```
subsystem



volatile int
    Alarm = FALSE,              // Flag indicating alarm
    DebounceCount = 0;          // Buttons debounce count

volatile uint16_t
    ButtonsPushed;              // 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

                                      // Current format for the displayed
time ( IE 12 or 24 hour format )
    ClockHourFormat = CLOCK_HOUR_FORMAT_24, // displaying 24 hour format
    AlarmPmFlag = 0,
    TimePmFlag = 0;

void RealTimeClockInit( void )
{
    RCC_OscInitTypeDef
       RCC_OscInitStruct;

    RCC_PeriphCLKInitTypeDef
       PeriphClkInitStruct;


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

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

```
// Assign the LSI clock to the RTC
    PeriphClkInitStruct.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
    PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_RTC;
    if(HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != 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_24;

    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);
}


void ConfigureDisplay( void )
{

    HAL_Init();
```

```
      __HAL_RCC_GPIOE_CLK_ENABLE();

      // Initializes GPIOE as low speed outputs without pulldown resistors.
      GPIO_InitTypeDef GPIO_Initstructure;
      // pin number 2, 4, 5, 6, 7, 8, 9 are the segment A, B, C, D, E, F, G
respectively
      // pin number 10, 11, 12, 13 ,14 are the digit selected digit 1, 2,
3, 4, 5 respectively
      // pin number 15 is the "ALARM" LED display

GPIO_Initstructure.Pin=GPIO_PIN_2|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN
_7|GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GP
IO_PIN_14|GPIO_PIN_15;
      GPIO_Initstructure.Mode=GPIO_MODE_OUTPUT_PP;
      GPIO_Initstructure.Speed=GPIO_SPEED_LOW;
      GPIO_Initstructure.Pull=GPIO_NOPULL;
      GPIO_Initstructure.Alternate = 0;
      HAL_GPIO_Init(GPIOE,&GPIO_Initstructure);

      // Initializes GPIOD as a fast speed input for the button
      __HAL_RCC_GPIOD_CLK_ENABLE();
      GPIO_InitTypeDef GPIO_Initstructure1;
      // pin number 0, 1, 2, 3, 6, are the button SET TIME, SET ALARM,
HOURS, MINUTES, SNOOZE respectively

GPIO_Initstructure1.Pin=GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PI
N_6|GPIO_PIN_7;
      GPIO_Initstructure1.Mode=GPIO_MODE_INPUT;
      GPIO_Initstructure1.Speed=GPIO_SPEED_FAST;
      GPIO_Initstructure1.Pull=GPIO_PULLUP;
      GPIO_Initstructure1.Alternate = 0;
      HAL_GPIO_Init(GPIOD,&GPIO_Initstructure1);

      // Initialize RCC TIM5 for the Timer
      __HAL_RCC_TIM5_CLK_ENABLE();
            DisplayTimer.Instance = TIM5;
            // to get 250 Hz, (period + 1)*(Prescaler + 1) needs to be
336000
            // this is because 84000000/336000 is 250
            // 399 is used for period and 839 is used for prescaler
      DisplayTimer.Init.Period = 399; //period & prescaler combination for
4 seconds count
       DisplayTimer.Init.Prescaler =839;
```

```
	DisplayTimer.Init.CounterMode = TIM_COUNTERMODE_UP;
		DisplayTimer.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
		HAL_TIM_Base_Init( &DisplayTimer );
	HAL_NVIC_SetPriority( TIM5_IRQn, 0, 0);//set priority for the
interrupt. Value 0 corresponds to highest priority
		HAL_NVIC_EnableIRQ( TIM5_IRQn );//Enable interrupt function
request of Timer5
	__HAL_TIM_ENABLE_IT( &DisplayTimer, TIM_IT_UPDATE );// Enable timer
interrupt flag to be set when timer count is reached
	__HAL_TIM_ENABLE( &DisplayTimer );//Enable timer to start

	//Enable clocks for Power clock (PWR_CLK)
	__HAL_RCC_PWR_CLK_ENABLE();
}
// Segon functions simplifies the syntax for switching segments.
void segon(uint16_t pin){
	HAL_GPIO_WritePin(GPIOE,pin,GPIO_PIN_SET);
}
// segoff functions simplifies the syntax for switching segments.
void segoff(uint16_t pin){
	HAL_GPIO_WritePin(GPIOE,pin,GPIO_PIN_RESET);
}

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();
    RealTimeClockInit();

// 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();

// Initialize RTC for real time clock.
// we set the time to 16:20 so that it is not 00:00
    ClockTime.Hours = 16;
       ClockTime.Minutes = 20;
       ClockTime.Seconds = 00;
       ClockTime.SubSeconds = 0;
       ClockTime.TimeFormat = RTC_HOURFORMAT_24;
       ClockTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
       ClockTime.StoreOperation = RTC_STOREOPERATION_RESET;

// Initialize the clock date for confirmation
    ClockDate.Date =     21;
       ClockDate.Month =     RTC_MONTH_JUNE;
       ClockDate.WeekDay = RTC_WEEKDAY_WEDNESDAY;
       ClockDate.Year =    17;

// set date in 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 );

// Initialize alarm in RTC
    AlarmTime.Alarm = RTC_ALARM_A;
       AlarmTime.AlarmTime.TimeFormat = RTC_HOURFORMAT_24;
       // initializing the alarm time to 03:03
       AlarmTime.AlarmTime.Hours = 0x03;
       AlarmTime.AlarmTime.Minutes = 0x03;
       AlarmTime.AlarmTime.Seconds = 0x00;
       AlarmTime.AlarmMask = RTC_ALARMMASK_DATEWEEKDAY;
       AlarmTime.AlarmDateWeekDay = 1;
       HAL_RTC_SetAlarm_IT( &RealTimeClock, &AlarmTime, RTC_FORMAT_BIN );

// 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;

// Start the display timer (TIM5)
    while ( TRUE )
    {

// Wait for an interrupt to occur

       // if alarm is true turn on the alarm led display
       if(Alarm ==TRUE){
             segon(GPIO_PIN_15);

       //else turn it off
       }else{
             segoff(GPIO_PIN_15);
       }
       __asm__ volatile ( "wfi" );

    }
}

/*
 *
 *  System Clock Configuration
 *
 */

void SystemClock_Config(void)
{
    //initialize system clock configuration
    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 )
    {

       // call function GetCurrentTime();
       GetCurrentTime();

// if button of set alarm is pressed, change alarm to FALSE (turn off the
alarm)
//and get alarm time and put it on BcdTime
if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_1)==0){
    Alarm=FALSE;
    BcdTime[0]=
(AlarmTime.AlarmTime.Hours-(AlarmTime.AlarmTime.Hours%10))/10;
    BcdTime[1]=AlarmTime.AlarmTime.Hours%10;

BcdTime[2]=(AlarmTime.AlarmTime.Minutes-(AlarmTime.AlarmTime.Minutes%10))/1
0;
       BcdTime[3]=AlarmTime.AlarmTime.Minutes%10;
}

// call function Display7Segment();
    Display7Segment();

// Process any button events
       if ( TRUE == CheckButtons())
       {
             // Debug code
             ProcessButtons();
             SetTime();
            SetAlarm();
            Snooze();
       }

// 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 );

// turn on the alarm led display
       Alarm = TRUE;

    }
}

#endif

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

void Display7Segment(void)
```

```
{

// clear digit selection bits
segoff(GPIO_PIN_10);
segoff(GPIO_PIN_11);
segoff(GPIO_PIN_12);
segoff(GPIO_PIN_13);
segoff(GPIO_PIN_14);

// clear segment selection bits
segoff(A);
segoff(B);
segoff(C);
segoff(D);
segoff(E);
segoff(F);
segoff(G);

// Select current digit
// it will set the LED to display number depends on what digit and Bcd time

//if digit 1 is selected
if (DisplayedDigit ==0){

    // select digit 1 of GPIO
    segon(GPIO_PIN_10);

    // switch the first array of BcdTime
    switch (BcdTime[0]){

    // if first array is 0
    case 0:

       // the segon and segoff of the segment will create number 0
                    segon(A);
                    segon(B);
                    segon(C);
                    segon(D);
                    segon(E);
                    segon(F);
                    segoff(G);
                    break;
```

```
    // if first array is 1
    case 1:

       // the segon and segoff of the segment will create number 1
       // this will continue until number 9
                    segoff(A);
                    segon(B);
                    segon(C);
                    segoff(D);
                    segoff(E);
                    segoff(F);
                    segoff(G);
                    break;
        case 2:
                    segon(A);
                    segon(B);
                    segoff(C);
                    segon(D);
                    segon(E);
                    segoff(F);
                    segon(G);
                    break;
        case 3:
                    segon(A);
                    segon(B);
                    segon(C);
                    segon(D);
                    segoff(E);
                    segoff(F);
                    segon(G);
                    break;
        case 4:
                    segoff(A);
                    segon(B);
                    segon(C);
                    segoff(D);
                    segoff(E);
                    segon(F);
                    segon(G);
                    break;
        case 5:
                    segon(A);
                    segoff(B);
```

```
                segon(C);
                segon(D);
                segoff(E);
                segon(F);
                segon(G);
                break;
        case 6:
                segon(A);
                segoff(B);
                segon(C);
                segon(D);
                segon(E);
                segon(F);
                segon(G);
                break;
       case 7:
                segon(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segoff(F);
                segoff(G);
                break;
        case 8:
                segon(A);
                segon(B);
                segon(C);
                segon(D);
                segon(E);
                segon(F);
                segon(G);
                break;
       case 9:
                segon(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segon(F);
                segon(G);
                break;
    }
```

```
//if digit 2 is selected
}else if (DisplayedDigit==1){

    // this will repeat the same process of digit 1 but the 2nd digit GPIO
will be selected
    // and the second array of BcdTime will be selected
    segon(GPIO_PIN_11);
    switch (BcdTime[1]){
       case 0:
                    segon(A);
                    segon(B);
                    segon(C);
                    segon(D);
                    segon(E);
                    segon(F);
                    segoff(G);
                    break;
       case 1:
                    segoff(A);
                    segon(B);
                    segon(C);
                    segoff(D);
                    segoff(E);
                    segoff(F);
                    segoff(G);
                    break;
             case 2:
                    segon(A);
                    segon(B);
                    segoff(C);
                    segon(D);
                    segon(E);
                    segoff(F);
                    segon(G);
                    break;
             case 3:
                    segon(A);
                    segon(B);
                    segon(C);
                    segon(D);
                    segoff(E);
                    segoff(F);
```

```
            segon(G);
            break;
        case 4:
            segoff(A);
            segon(B);
            segon(C);
            segoff(D);
            segoff(E);
            segon(F);
            segon(G);
            break;
        case 5:
            segon(A);
            segoff(B);
            segon(C);
            segon(D);
            segoff(E);
            segon(F);
            segon(G);
            break;
        case 6:
            segon(A);
            segoff(B);
            segon(C);
            segon(D);
            segon(E);
            segon(F);
            segon(G);
            break;
        case 7:
            segon(A);
            segon(B);
            segon(C);
            segoff(D);
            segoff(E);
            segoff(F);
            segoff(G);
            break;
        case 8:
            segon(A);
            segon(B);
            segon(C);
            segon(D);
```

```
                    segon(E);
                    segon(F);
                    segon(G);
                    break;
             case 9:
                    segon(A);
                    segon(B);
                    segon(C);
                    segoff(D);
                    segoff(E);
                    segon(F);
                    segon(G);
                    break;
       }

// if Displayed digit is 2 is referring to the colon of the time **:**
which is the 3rd digit
}else if (DisplayedDigit==2){

    // select GPIOE pin 12 for the colon
       segon(GPIO_PIN_12);

    // turn on segment A and B to turn on the colon
        segon(A);
        segon(B);

// repeat the same process like the 1st digit and 2nd digit for 4th digit
}else if (DisplayedDigit ==3){
    segon(GPIO_PIN_13);
    switch (BcdTime[2]){
       case 0:
                    segon(A);
                    segon(B);
                    segon(C);
                    segon(D);
                    segon(E);
                    segon(F);
                    segoff(G);
                    break;
       case 1:
                    segoff(A);
                    segon(B);
                    segon(C);
```

```
                segoff(D);
                segoff(E);
                segoff(F);
                segoff(G);
                break;
        case 2:
                segon(A);
                segon(B);
                segoff(C);
                segon(D);
                segon(E);
                segoff(F);
                segon(G);
                break;
        case 3:
                segon(A);
                segon(B);
                segon(C);
                segon(D);
                segoff(E);
                segoff(F);
                segon(G);
                break;
        case 4:
                segoff(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segon(F);
                segon(G);
                break;
        case 5:
                segon(A);
                segoff(B);
                segon(C);
                segon(D);
                segoff(E);
                segon(F);
                segon(G);
                break;
        case 6:
                segon(A);
```

```
                segoff(B);
                segon(C);
                segon(D);
                segon(E);
                segon(F);
                segon(G);
                break;
            case 7:
                segon(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segoff(F);
                segoff(G);
                break;
            case 8:
                segon(A);
                segon(B);
                segon(C);
                segon(D);
                segon(E);
                segon(F);
                segon(G);
                break;
            case 9:
                segon(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segon(F);
                segon(G);
                break;
    }

// repeat the same process like the 1st digit and 2nd digit for 5th digit
}else if (DisplayedDigit==4){
    segon(GPIO_PIN_14);
    switch (BcdTime[3]){
       case 0:
                segon(A);
                segon(B);
```

```
                segon(C);
                segon(D);
                segon(E);
                segon(F);
                segoff(G);
                break;
        case 1:
                segoff(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segoff(F);
                segoff(G);
                break;
            case 2:
                segon(A);
                segon(B);
                segoff(C);
                segon(D);
                segon(E);
                segoff(F);
                segon(G);
                break;
            case 3:
                segon(A);
                segon(B);
                segon(C);
                segon(D);
                segoff(E);
                segoff(F);
                segon(G);
                break;
            case 4:
                segoff(A);
                segon(B);
                segon(C);
                segoff(D);
                segoff(E);
                segon(F);
                segon(G);
                break;
            case 5:
```

```
            segon(A);
            segoff(B);
            segon(C);
            segon(D);
            segoff(E);
            segon(F);
            segon(G);
            break;
        case 6:
            segon(A);
            segoff(B);
            segon(C);
            segon(D);
            segon(E);
            segon(F);
            segon(G);
            break;
        case 7:
            segon(A);
            segon(B);
            segon(C);
            segoff(D);
            segoff(E);
            segoff(F);
            segoff(G);
            break;
        case 8:
            segon(A);
            segon(B);
            segon(C);
            segon(D);
            segon(E);
            segon(F);
            segon(G);
            break;
        case 9:
            segon(A);
            segon(B);
            segon(C);
            segoff(D);
            segoff(E);
            segon(F);
            segon(G);
```

```
                    break;
    }
}

// Advance to the next digit to be display on next interrupt
DisplayedDigit=DisplayedDigit +1 ;
    if(DisplayedDigit ==5){
       DisplayedDigit=0;
    }
}


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

void SetTime(void)
{
    // if the SET TIME button is pressed
    if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_0)==0){

       // if the HOURS button pressed while the SET TIME pressed
       if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_2)==0 ){

             //Get both time and date of RTC
             HAL_RTC_GetDate(&RealTimeClock, &ClockDate, RTC_FORMAT_BIN);
             HAL_RTC_GetTime(&RealTimeClock, &ClockTime, RTC_FORMAT_BIN);

             // if hours is 23 make it 00
             if (ClockTime.Hours==23){
                   ClockTime.Hours =00;

             // else plus 1 to the hours
             }else{
             ClockTime.Hours = ClockTime.Hours+1;

             }
```

```
            // set the date and time again to the RTC
                  HAL_RTC_SetDate(&RealTimeClock, &ClockDate,
RTC_FORMAT_BIN);
                  HAL_RTC_SetTime(&RealTimeClock, &ClockTime,
RTC_FORMAT_BIN);
       }

       //but if the MINUTES button is pressed
       if( HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_3)==0 ){

             //Get both time and date of RTC
             HAL_RTC_GetDate(&RealTimeClock, &ClockDate, RTC_FORMAT_BIN);
             HAL_RTC_GetTime(&RealTimeClock, &ClockTime, RTC_FORMAT_BIN);

             // if minutes is 59 make it 00
             if(ClockTime.Minutes==59){
                   ClockTime.Minutes=00;

             // else plus 1 to the minutes
             }else{
                   ClockTime.Minutes=ClockTime.Minutes+1;
             }

             // set the date and time to the RTC
             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 the SET ALARM button is pressed
  if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_1)==0){

      // if the HOURS button is pressed
      if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_2)==0){

         // if the hours of alarm time is 23 then make it to 00
         if (AlarmTime.AlarmTime.Hours==23){
               AlarmTime.AlarmTime.Hours =00;

       //else plus 1 to the alarm time
         }else{
               AlarmTime.AlarmTime.Hours = AlarmTime.AlarmTime.Hours+1;
       }

         // set the alarm time to the RTC
         HAL_RTC_SetAlarm_IT( &RealTimeClock, &AlarmTime, RTC_FORMAT_BIN);

      }

    // if the MINUTES button is pressed
    if( HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_3)==0 ){

       // if the minutes is 59 then make it to 00
            if( AlarmTime.AlarmTime.Minutes==59){
                  AlarmTime.AlarmTime.Minutes=00;

       // else plus 1 to the alarm minutes
            }else{
                  AlarmTime.AlarmTime.Minutes=
AlarmTime.AlarmTime.Minutes+1;
            }

       // set the alarm time to the RTC
        HAL_RTC_SetAlarm_IT( &RealTimeClock, &AlarmTime, RTC_FORMAT_BIN);
    }
  }
}
```

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

void Snooze(void)
{
    // if the SNOOZE button is pressed
    if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_6)==0&&Alarm==TRUE){

       // turn of the ALARM LED display
       Alarm =FALSE;

       // add alarm minutes to the desired alarm time
       // in this case, we snooze it for 10 minutes
        AlarmTime.AlarmTime.Minutes= AlarmTime.AlarmTime.Minutes+10;

        // set alarm time to the RTC
        HAL_RTC_SetAlarm_IT( &RealTimeClock, &AlarmTime, RTC_FORMAT_BIN);
    }
}


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

void GetCurrentTime(void)
{
    // get time and date of RTC
    HAL_RTC_GetDate(&RealTimeClock, &ClockDate, RTC_FORMAT_BIN);
```

```
    HAL_RTC_GetTime(&RealTimeClock, &ClockTime, RTC_FORMAT_BIN);

    // put the left digit for hours to BcdTime array number 1
    BcdTime[0]= (ClockTime.Hours-(ClockTime.Hours%10))/10;

    //put the right digit for hours to BcdTime array number 2
    BcdTime[1]=ClockTime.Hours%10;

    // put the left digit for minutes to BcdTime array number 3
    BcdTime[2]=(ClockTime.Minutes-(ClockTime.Minutes%10))/10;

    // put the right digit for minutes to BcdTime array number 4
    BcdTime[3]=ClockTime.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 any of the 5 button is pressed
    if(HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_0)==0||HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_1)==0||HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_2)==0||HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_3)==0||HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_6)==0){

       // add the DebounceCount by 1
       DebounceCount = DebounceCount + 1;

            // if DebounceCount is equal to 20, make it 0 and return TRUE
           if (DebounceCount == 20){
                  DebounceCount=0;
                return TRUE;
```

```
            }
      }

       // else if the buttons are not pushed, the ButtonPushed equal to
0x00
       // and DebounceCount equal to 0, RETURN FALSE
      else {
       ButtonsPushed = 0x00;
            DebounceCount = 0;
      }

    return( FALSE );
}


#pragma GCC diagnostic pop
```