

1. INTRODUCTION:

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. You will use C language for the problems unless some parts require inline assembly. Use blinky project from stm32go repo as the starting point for your problems.

2. PROBLEMS:

2.1. Problem 1:

In this problem, you will work on implementing a simple tone generator utilizing Timer, PWM and External Interrupt modules and use a keypad, a speaker, and 7SDs.

- Connect a keypad to your microcontroller, pick a block of notes, and assign a tone for each key on the keypad.

- One of the keys should be silence (rest).

- Tones will be played using PWM by changing the period at 50% duty cycle.

- Design an amplifier, and connect a speaker with adjustable gain using a pot.

- When a key is pressed, the relevant tone should play indefinitely.

- Connect your 7SD to display the tone that is being played. You can display the frequency being played. (440, 480,...) OR you can display the tone being played (A4, B4, C4, ...).

Theoretical Research:

Pulse-width modulation:

Pulse width modulation (PWM), or pulse-duration modulation (PDM), is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load. Along with maximum power point tracking (MPPT), it is one of the primary methods of reducing the output of solar panels to that which can be utilized by a battery.[1] PWM is particularly suited for running inertial loads such as motors, which are not as easily affected by this discrete switching, because their inertia causes them to react slowly. The PWM switching frequency has to be high enough not to affect the load, which is to say that the resultant waveform perceived by the load must be as smooth as possible.

The rate (or frequency) at which the power supply must switch can vary greatly depending on load and application. For example, switching has to be done several times a minute in an electric stove; 120 Hz in a lamp dimmer; between a few kilohertz (kHz) and tens of kHz for a motor drive; and well into the tens or hundreds of kHz in audio amplifiers and computer power supplies. The main advantage of PWM is that power loss in the switching devices is very low. When a switch is off there is practically no current, and when it is on and power is being transferred to the load, there is almost no voltage drop across

the switch. Power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle. PWM has also been used in certain communication systems where its duty cycle has been used to convey information over a communications channel.

In electronics, many modern microcontrollers (MCUs) integrate PWM controllers exposed to external pins as peripheral devices under firmware control by means of internal programming interfaces. These are commonly used for direct current (DC) motor control in robotics and other applications.

Duty cycle

A **duty cycle** or **power cycle** is the fraction of one [period](#) in which a signal or system is active. Duty cycle is commonly expressed as a percentage or a ratio. A period is the time it takes for a signal to complete an on-and-off [cycle](#). As a formula, a duty cycle (%) may be expressed as:

$$D = \frac{PW}{T} \times 100\%^{[2]}$$

Equally, a duty cycle (ratio) may be expressed as:

$$D = \frac{PW}{T}$$

where D is the duty cycle, PW is the pulse width (pulse active time), and T is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.

Duty cycles can be used to describe the percent time of an active signal in an electrical device such as the power switch in a [switching power supply](#) or the firing of [action potentials](#) by a living system such as a [neuron](#).

The **duty factor** for periodic signal expresses the same notion, but is usually scaled to a maximum of one rather than 100%.^[1]

The duty cycle can also be notated as

50% duty cycle



75% duty cycle



25% duty cycle



2.2. Code:

```

1: #include "bsp.h"
2:
3:
4: void BSP_system_init(void){
5:
6:     __disable_irq();
7:
8:     init_timer1();
9:
10:    __enable_irq();
11:
12:
13: }
14:
15: void pwm_init(){
16:
17:     RCC->IOPENR |= (1U << 0);
18:
19:     GPIOA->MODER &= ~(3U << 2*6);
20:     GPIOA->MODER |= (2U << 2*6);
21:
22:     // choose AF2 from mux
23:     GPIOA->AFR[0] &= ~(0xFU << 2*6);
24:     GPIOA->AFR[0] |= (2U << 2*6);
25:
26:     TIM2->CCR3 = 800;
27:
28:     TIM2->CCMR1 |= (0x6U << 4); // mode 1 enable
29:     TIM2->CCMR1 |= (1U << 3);
30:     TIM2->CCER |= (1U << 8);
31:
32: }
33:
34:
35:
36: //void init_timer2(){
37: //
38: //
39: //     RCC->IOPENR |= (1U << 0);
40: //     RCC->APBENR1 |= (1U << 0); // enable time2 module clock
41: //
42: //     //setup PA6 as AF2
43: //     GPIOA->MODER &= ~(3U << 2*6);
44: //     GPIOA->MODER |= (2U << 2*6);
45: //
46: //     // choose AF2 from mux
47: //     GPIOA->AFR[0] &= ~(0xFU << 2*6);
48: //     GPIOA->AFR[0] |= (2U << 2*6);
49: //
50: //     TIM2->CR1=0; // zero out the control register just in case
51: //     TIM2->CR1 |= (1U << 7); // ARPE
52: //     TIM2->CNT=0; // zero out counter
53: //
54: //     /*1 second interrupt */

```

```

55: //
56: // TIM2->PSC = 9;
57: // TIM2->ARR = 1600;
58: // TIM2->CCR3 = 800;
59: //
60: // TIM2->CCMR1 |= (0x6U << 4); // mode 1 enable
61: // TIM2->CCMR1 |= (1U << 3);
62: // TIM2->CCER |= (1U << 8);
63: //
64: //
65: //
66: // TIM2->DIER |= (1 << 0); // update interrupt enable
67: // TIM2->CR1 |= (1 << 0); // tÄ±m1 enable
68: //
69: // NVIC_SetPriority(TIM2_IRQn,3);
70: // NVIC_EnableIRQ(TIM2_IRQn);
71: //
72: //
73: //
74: //}
75:
76: void init_timer1(){
77:
78:     RCC->APBENR1 |= (1U<< 0); // enable time2 module clock
79:
80:     TIM2->CR1=0; // zero out the control register just in case
81:     TIM2->CR1 |= (1U << 7); // ARPE
82:     TIM2->CNT=0; // zero out counter
83:
84:     /*1 second interrupt */
85:
86:     TIM2->PSC = 9;
87:     TIM2->ARR = 1600;
88:
89:     TIM2->DIER |= (1 << 0); // update interrupt enable
90:     TIM2->CR1 |= (1 << 0); // tÄ±m1 enable
91:
92:     NVIC_SetPriority(TIM2_IRQn,3);
93:     NVIC_EnableIRQ(TIM2_IRQn);
94:
95: }
96:
97: void TIM2_IRQHandler(void){
98:
99:     pwm_init();
100:     TIM2->SR &= ~(1U << 0); //clear update status register.
101:
102: }

```

```
1: #ifndef BSP_H_
2: #define BSP_H_
3:
4: #include "stm32g0xx.h"
5:
6: void delay(volatile unsigned int);
7:
8: void BSP_system_init();
9:
10: void pwm_init();
11: void init_timer1();
12: //void init_timer2();
13:
14: #endif
```

display.c

```
1 /*
2  * display.c
3  *
4  * Created on: Dec 19, 2020
5  * Author: Mehmet Akif/171024027
6  *
7  * description: In this section, necessary pins of
8  * the SSD have been activated in order for the lid ssd.
9  * Next, with the display function, the burning of the digits
10 * at the same time is provided by flashing the digits invisibly.
11 * and then we display the character from printdigit func
12 */
13
14 #include "display.h"
15
16 extern void main();
17
18
19
20 void init_SSD(){
21     GPIOB->MODER &= ~(3U << 2*1);
22     GPIOB->MODER |= (1U << 2*1); //PB1 is output
23
24     GPIOB->MODER &= ~(3U << 2*3);
25     GPIOB->MODER |= (1U << 2*3); //PB3 is output
26
27     GPIOB->MODER &= ~(3U << 2*6);
28     GPIOB->MODER |= (1U << 2*6); //PB6 is output
29
30     GPIOB->MODER &= ~(3U << 2*7);
31     GPIOB->MODER |= (1U << 2*7); //PB7 is output
32
33     GPIOA->MODER &= ~(3U << 2*0);
34     GPIOA->MODER |= (1U << 2*0); //PA0 is output
35
36     GPIOA->MODER &= ~(3U << 2*1);
37     GPIOA->MODER |= (1U << 2*1); //PA1 is output
38
39     GPIOA->MODER &= ~(3U << 2*4);
40     GPIOA->MODER |= (1U << 2*4); //PA4 is output
41
42     GPIOA->MODER &= ~(3U << 2*5);
43     GPIOA->MODER |= (1U << 2*5); //PA5 is output
44
45     GPIOA->MODER &= ~(3U << 2*6);
46     GPIOA->MODER |= (1U << 2*6); //PA6 is output
47
48     GPIOA->MODER &= ~(3U << 2*7);
49     GPIOA->MODER |= (1U << 2*7); //PA7 is output
50
51     GPIOA->MODER &= ~(3U << 2*11);
52     GPIOA->MODER |= (1U << 2*11); //PA11 is output
53
54     GPIOA->MODER &= ~(3U << 2*12);
55     GPIOA->MODER |= (1U << 2*12); //PA12 is output
56
57
58 }
59
60 void display_SSD(){
61
62     static int i = 0;
```

display.c

```

63     if(Display.Inv == 1){
64         invalid_SSD();
65     }
66 }
67 else if(Display.Oflw == 1){
68     overflow_SSD();
69 }
70 }
71
72 if(i == 1){
73     GPIOA->ODR |= (1U << 7); //PA7
74     GPIOB->ODR &= ~(1U << 6); //PB6
75     GPIOB->ODR &= ~(1U << 7); //PB7
76     GPIOB->ODR &= ~(1U << 1); //PB1
77     printDigit_SSD(Display.Digits[0]);
78     GPIOA->ODR |= ( 1U << 6); // PA6
79     if(Display.dot == 0) GPIOA->ODR &= ~( 1U << 6);
80 }
81 else if(i == 10){
82     GPIOA->ODR &= ~(1U << 7); //PA7
83     GPIOB->ODR |= (1U << 6); //PB6
84     GPIOB->ODR &= ~(1U << 7); //PB7
85     GPIOB->ODR &= ~(1U << 1); //PB1
86     printDigit_SSD(Display.Digits[1]);
87     GPIOA->ODR |= ( 1U << 6); // PA6
88     if(Display.dot == 1) GPIOA->ODR &= ~( 1U << 6);
89 }
90
91 else if(i == 20){
92     GPIOA->ODR &= ~(1U << 7); //PA7
93     GPIOB->ODR &= ~(1U << 6); //PB6
94     GPIOB->ODR |= (1U << 7); //PB7
95     GPIOB->ODR &= ~(1U << 1); //PB1
96     printDigit_SSD(Display.Digits[2]);
97     GPIOA->ODR |= ( 1U << 6); // PA6
98     if(Display.dot == 2) GPIOA->ODR &= ~( 1U << 6);
99 }
100 else if(i == 30){
101     GPIOA->ODR &= ~(1U << 7); //PA7
102     GPIOB->ODR &= ~(1U << 6); //PB6
103     GPIOB->ODR &= ~(1U << 7); //PB7
104     GPIOB->ODR |= (1U << 1); //PB1
105     printDigit_SSD(Display.Digits[3]);
106     GPIOA->ODR |= ( 1U << 6); // PA6
107     if(Display.dot == 3) GPIOA->ODR &= ~( 1U << 6);
108 }
109 else if(i == 40) i = 0;
110
111 i++;
112
113
114
115
116 }
117
118 void printDigit_SSD(uint8_t x){
119
120     switch(x){
121     case 0: //0
122
123         GPIOB->ODR &= ~( 1U << 3); // PB3
124         GPIOA->ODR &= ~( 1U << 0); // PA0

```


display.c

```

125     GPIOA->ODR &= ~( 1U << 1); // PA1
126     GPIOA->ODR &= ~( 1U << 4); // PA4
127     GPIOA->ODR &= ~( 1U << 5); // PA5
128     GPIOA->ODR &= ~( 1U << 12); // PA12
129     GPIOA->ODR |= ( 1U << 11); // PA11
130
131     break;
132
133     case 1: //1
134         GPIOB->ODR |= ( 1U << 3); // PB3
135         GPIOA->ODR &= ~( 1U << 0); // PA0
136         GPIOA->ODR &= ~( 1U << 1); // PA1
137         GPIOA->ODR |= ( 1U << 4); // PA4
138         GPIOA->ODR |= ( 1U << 5); // PA5
139         GPIOA->ODR |= ( 1U << 12); // PA12
140         GPIOA->ODR |= ( 1U << 11); // PA11
141
142     break;
143
144     case 2: //2
145         GPIOB->ODR &= ~( 1U << 3); // PB3
146         GPIOA->ODR &= ~( 1U << 0); // PA0
147         GPIOA->ODR |= ( 1U << 1); // PA1
148         GPIOA->ODR &= ~( 1U << 4); // PA4
149         GPIOA->ODR &= ~( 1U << 5); // PA5
150         GPIOA->ODR |= ( 1U << 12); // PA12
151         GPIOA->ODR &= ~( 1U << 11); // PA11
152
153     break;
154
155     case 3: //3
156
157         GPIOB->ODR &= ~( 1U << 3); // PB3
158         GPIOA->ODR &= ~( 1U << 0); // PA0
159         GPIOA->ODR &= ~( 1U << 1); // PA1
160         GPIOA->ODR &= ~( 1U << 4); // PA4
161         GPIOA->ODR |= ( 1U << 5); // PA5
162         GPIOA->ODR |= ( 1U << 12); // PA12
163         GPIOA->ODR &= ~( 1U << 11); // PA11
164
165     break;
166
167     case 4: //4
168         GPIOB->ODR |= ( 1U << 3); // PB3
169         GPIOA->ODR &= ~( 1U << 0); // PA0
170         GPIOA->ODR &= ~( 1U << 1); // PA1
171         GPIOA->ODR |= ( 1U << 4); // PA4
172         GPIOA->ODR |= ( 1U << 5); // PA5
173         GPIOA->ODR &= ~( 1U << 12); // PA12
174         GPIOA->ODR &= ~( 1U << 11); // PA11
175
176     break;
177
178     case 5: //5
179
180         GPIOB->ODR &= ~( 1U << 3); // PB3
181         GPIOA->ODR |= ( 1U << 0); // PA0
182         GPIOA->ODR &= ~( 1U << 1); // PA1
183         GPIOA->ODR &= ~( 1U << 4); // PA4
184         GPIOA->ODR |= ( 1U << 5); // PA5
185         GPIOA->ODR &= ~( 1U << 12); // PA12
186         GPIOA->ODR &= ~( 1U << 11); // PA11

```

display.c

```

187
188         break;
189
190     case 6:        //6
191         GPIOB->ODR &= ~( 1U << 3); // PB3
192         GPIOA->ODR |= ( 1U << 0); // PA0
193         GPIOA->ODR &= ~( 1U << 1); // PA1
194         GPIOA->ODR &= ~( 1U << 4); // PA4
195         GPIOA->ODR &= ~( 1U << 5); // PA5
196         GPIOA->ODR &= ~( 1U << 12); // PA12
197         GPIOA->ODR &= ~( 1U << 11); // PA11
198
199         break;
200
201     case 7:        //7
202
203         GPIOB->ODR &= ~( 1U << 3); // PB3
204         GPIOA->ODR &= ~( 1U << 0); // PA0
205         GPIOA->ODR &= ~( 1U << 1); // PA1
206         GPIOA->ODR |= ( 1U << 4); // PA4
207         GPIOA->ODR |= ( 1U << 5); // PA5
208         GPIOA->ODR |= ( 1U << 12); // PA12
209         GPIOA->ODR |= ( 1U << 11); // PA11
210
211         break;
212
213     case 8:        //8
214
215         GPIOB->ODR &= ~( 1U << 3); // PB3
216         GPIOA->ODR &= ~( 1U << 0); // PA0
217         GPIOA->ODR &= ~( 1U << 1); // PA1
218         GPIOA->ODR &= ~( 1U << 4); // PA4
219         GPIOA->ODR &= ~( 1U << 5); // PA5
220         GPIOA->ODR &= ~( 1U << 12); // PA12
221         GPIOA->ODR &= ~( 1U << 11); // PA11
222
223         break;
224
225     case 9:        //9
226         GPIOB->ODR &= ~( 1U << 3); // PB3
227         GPIOA->ODR &= ~( 1U << 0); // PA0
228         GPIOA->ODR &= ~( 1U << 1); // PA1
229         GPIOA->ODR &= ~( 1U << 4); // PA4
230         GPIOA->ODR |= ( 1U << 5); // PA5
231         GPIOA->ODR &= ~( 1U << 12); // PA12
232         GPIOA->ODR &= ~( 1U << 11); // PA11
233
234         break;
235
236     case 10://A
237
238         GPIOB->ODR &= ~( 1U << 3); // PB3
239         GPIOA->ODR &= ~( 1U << 0); // PA0
240         GPIOA->ODR &= ~( 1U << 1); // PA1
241         GPIOA->ODR &= ~( 1U << 4); // PA4
242         GPIOA->ODR &= ~( 1U << 5); // PA5
243         GPIOA->ODR |= ( 1U << 12); // PA12
244         GPIOA->ODR &= ~( 1U << 11); // PA11
245
246
247         break;
248

```

display.c

```

249
250     case 11://B
251
252         GPIOB->ODR |= ( 1U << 3); // PB3
253         GPIOA->ODR |= ( 1U << 0); // PA0
254         GPIOA->ODR &= ~( 1U << 1); // PA1
255         GPIOA->ODR &= ~( 1U << 4); // PA4
256         GPIOA->ODR &= ~( 1U << 5); // PA5
257         GPIOA->ODR &= ~( 1U << 12); // PA12
258         GPIOA->ODR &= ~( 1U << 11); // PA11
259
260
261         break;
262
263     case 12://C
264         GPIOB->ODR &= ~( 1U << 3); // PB3
265         GPIOA->ODR |= ( 1U << 0); // PA0
266         GPIOA->ODR |= ( 1U << 1); // PA1
267         GPIOA->ODR &= ~( 1U << 4); // PA4
268         GPIOA->ODR &= ~( 1U << 5); // PA5
269         GPIOA->ODR &= ~( 1U << 12); // PA12
270         GPIOA->ODR |= ( 1U << 11); // PA11
271
272         break;
273
274     case 13://D
275         GPIOB->ODR |= ( 1U << 3); // PB3
276         GPIOA->ODR &= ~( 1U << 0); // PA0
277         GPIOA->ODR &= ~( 1U << 1); // PA1
278         GPIOA->ODR &= ~( 1U << 4); // PA4
279         GPIOA->ODR &= ~( 1U << 5); // PA5
280         GPIOA->ODR |= ( 1U << 12); // PA12
281         GPIOA->ODR &= ~( 1U << 11); // PA11
282
283         break;
284
285     case 14://E
286         GPIOB->ODR &= ~( 1U << 3); // PB3
287         GPIOA->ODR |= ( 1U << 0); // PA0
288         GPIOA->ODR |= ( 1U << 1); // PA1
289         GPIOA->ODR &= ~( 1U << 4); // PA4
290         GPIOA->ODR &= ~( 1U << 5); // PA5
291         GPIOA->ODR &= ~( 1U << 12); // PA12
292         GPIOA->ODR &= ~( 1U << 11); // PA11
293
294
295         break;
296
297     case 15: //F
298         GPIOB->ODR &= ~( 1U << 3); // PB3
299         GPIOA->ODR |= ( 1U << 0); // PA0
300         GPIOA->ODR |= ( 1U << 1); // PA1
301         GPIOA->ODR |= ( 1U << 4); // PA4
302         GPIOA->ODR &= ~( 1U << 5); // PA5
303         GPIOA->ODR &= ~( 1U << 12); // PA12
304         GPIOA->ODR &= ~( 1U << 11); // PA11
305         break;
306
307
308     case 30: //u
309         GPIOB->ODR |= ( 1U << 3); // PB3
310         GPIOA->ODR |= ( 1U << 0); // PA0

```

display.c

```

311     GPIOA->ODR &= ~( 1U << 1); // PA1
312     GPIOA->ODR &= ~( 1U << 4); // PA4
313     GPIOA->ODR &= ~( 1U << 5); // PA5
314     GPIOA->ODR |= ( 1U << 12); // PA12
315     GPIOA->ODR |= ( 1U << 11); // PA11
316     break;
317
318     case 31: //L
319         GPIOB->ODR |= ( 1U << 3); // PB3
320         GPIOA->ODR |= ( 1U << 0); // PA0
321         GPIOA->ODR |= ( 1U << 1); // PA1
322         GPIOA->ODR &= ~( 1U << 4); // PA4
323         GPIOA->ODR &= ~( 1U << 5); // PA5
324         GPIOA->ODR &= ~( 1U << 12); // PA12
325         GPIOA->ODR |= ( 1U << 11); // PA11
326         break;
327
328     case 32: //n
329         GPIOB->ODR |= ( 1U << 3); // PB3
330         GPIOA->ODR |= ( 1U << 0); // PA0
331         GPIOA->ODR &= ~( 1U << 1); // PA1
332         GPIOA->ODR |= ( 1U << 4); // PA4
333         GPIOA->ODR &= ~( 1U << 5); // PA5
334         GPIOA->ODR |= ( 1U << 12); // PA12
335         GPIOA->ODR &= ~( 1U << 11); // PA11
336         break;
337
338     case 33: //D
339         GPIOB->ODR |= ( 1U << 3); // PB3
340         GPIOA->ODR &= ~( 1U << 0); // PA0
341         GPIOA->ODR &= ~( 1U << 1); // PA1
342         GPIOA->ODR &= ~( 1U << 4); // PA4
343         GPIOA->ODR &= ~( 1U << 5); // PA5
344         GPIOA->ODR |= ( 1U << 12); // PA12
345         GPIOA->ODR &= ~( 1U << 11); // PA11
346         break;
347
348     case 34: // negative sign
349         GPIOB->ODR |= ( 1U << 3); // PB3
350         GPIOA->ODR |= ( 1U << 0); // PA0
351         GPIOA->ODR |= ( 1U << 1); // PA1
352         GPIOA->ODR |= ( 1U << 4); // PA4
353         GPIOA->ODR |= ( 1U << 5); // PA5
354         GPIOA->ODR |= ( 1U << 12); // PA12
355         GPIOA->ODR &= ~( 1U << 11); // PA11
356         break;
357
358     case 35: // space
359         GPIOB->ODR |= ( 1U << 3); // PB3
360         GPIOA->ODR |= ( 1U << 0); // PA0
361         GPIOA->ODR |= ( 1U << 1); // PA1
362         GPIOA->ODR |= ( 1U << 4); // PA4
363         GPIOA->ODR |= ( 1U << 5); // PA5
364         GPIOA->ODR |= ( 1U << 12); // PA12
365         GPIOA->ODR |= ( 1U << 11); // PA11
366         break;
367 }
368 }
369
370
371 void displaychar_SSD(uint8_t x){
372     Display.dot = 5;

```

```

373     switch(x){
374     case 0: //-
375         Display.Digits[0] = 34;
376         Display.Digits[1] = 35;
377         Display.Digits[2] = 35;
378         Display.Digits[3] = 35;
379         break;
380     case 1: //'A'
381         Display.Digits[3] = 10;
382         Display.Digits[0] = 35;
383         Display.Digits[2] = 35;
384         Display.Digits[1] = 35;
385         break;
386     case 2: //'B'
387         Display.Digits[3] = 11;
388         Display.Digits[1] = 35;
389         Display.Digits[2] = 35;
390         Display.Digits[0] = 35;
391         break;
392     case 3: //'C'
393         Display.Digits[3] = 12;
394         Display.Digits[1] = 35;
395         Display.Digits[2] = 35;
396         Display.Digits[0] = 35;
397         break;
398     case 4: //'D'
399         Display.Digits[3] = 13;
400         Display.Digits[1] = 35;
401         Display.Digits[2] = 35;
402         Display.Digits[0] = 35;
403         break;
404     case 5: //'E'
405         Display.Digits[3] = 14;
406         Display.Digits[2] = 35;
407         Display.Digits[1] = 35;
408         Display.Digits[0] = 35;
409         break;
410     case 6: //'EA'
411         Display.Digits[3] = 14;
412         Display.Digits[2] = 10;
413         Display.Digits[1] = 35;
414         Display.Digits[0] = 35;
415         break;
416     case 7: //'EB'
417         Display.Digits[3] = 14;
418         Display.Digits[2] = 11;
419         Display.Digits[1] = 35;
420         Display.Digits[0] = 35;
421         break;
422     case 8: //'EC'
423         Display.Digits[3] = 14;
424         Display.Digits[2] = 12;
425         Display.Digits[1] = 35;
426         Display.Digits[0] = 35;
427         break;
428     case 9: //'ED'
429         Display.Digits[3] = 14;
430         Display.Digits[2] = 13;
431         Display.Digits[1] = 35;
432         Display.Digits[0] = 35;
433         break;
434     case 10:    //'EE'

```

display.c

```

435     Display.Digits[3] = 14;
436     Display.Digits[2] = 14;
437     Display.Digits[1] = 35;
438     Display.Digits[0] = 35;
439     break;
440 case 11:    //'EEA'
441     Display.Digits[3] = 14;
442     Display.Digits[2] = 14;
443     Display.Digits[1] = 10;
444     Display.Digits[0] = 35;
445     break;
446 case 12:    //'EEB'
447     Display.Digits[3] = 14;
448     Display.Digits[2] = 14;
449     Display.Digits[1] = 11;
450     Display.Digits[0] = 35;
451     break;
452 case 13:    //'EEC'
453     Display.Digits[3] = 14;
454     Display.Digits[2] = 14;
455     Display.Digits[1] = 12;
456     Display.Digits[0] = 35;
457     break;
458 case 14:    //'EED'
459     Display.Digits[3] = 14;
460     Display.Digits[2] = 14;
461     Display.Digits[1] = 13;
462     Display.Digits[0] = 35;
463     break;
464 case 20:    // 0uFL
465     Display.Digits[0] = 31;
466     Display.Digits[1] = 15;
467     Display.Digits[2] = 30;
468     Display.Digits[3] = 0;
469     break;
470 case 21:    // InuD
471     Display.Digits[0] = 33;
472     Display.Digits[1] = 30;
473     Display.Digits[2] = 32;
474     Display.Digits[3] = 1;
475
476     break;
477 }
478 }
479
480 void displayID_SSD(){
481     Display.Digits[0]= 7;
482     Display.Digits[1]= 2;
483     Display.Digits[2]= 7;
484     Display.Digits[3]= 1;
485
486
487 }
488
489 void overflow_SSD(){
490
491     displaychar_SSD(20);
492
493     Display.Oflw = 0;
494     calculation.current_process=0;
495     calculation.x=0;
496     calculation.y=0;

```

display.c

```

497     calculation.result=0;
498
499 }
500
501 void invalid_SSD(void){
502
503     displaychar_SSD(21);
504
505     Display.Inv = 0;
506     calculation.current_process=0;
507     calculation.x=0;
508     calculation.y=0;
509     calculation.result=0;
510
511 }
512
513 void utility_SSD(float var){
514
515     int number = (int)var;
516
517     float i = 0.0;
518
519     if((number < 0) & (number >= -999)){
520         Display.sign = 1;
521         i = -1.0;
522         Display.dot = 0;
523         if(number >= -99){
524             i = -10.0;
525             Display.dot = 1;
526             if(number >= -9){
527                 i = -100.0;
528                 Display.dot = 2;
529             }
530         }
531     }
532     else if((number >= 0) & (number <= 9999)){
533         Display.sign = 0;
534         i = 1.0;
535         Display.dot = 0;
536         if(number <= 999){
537             i = 10.0;
538             Display.dot = 1;
539             if(number <= 99){
540                 i = 100.0;
541                 Display.dot = 2;
542                 if(number <= 9){
543                     i = 1000.0;
544                     Display.dot = 3;
545                 }
546             }
547         }
548     }
549
550
551     number = (int)(var * i);
552
553     int temp = number / 10;
554     Display.Digits[0] = (uint8_t)(number - (temp*10));
555
556     temp = number / 100;
557     Display.Digits[1] = (uint8_t)((number - (temp * 100)) / 10);
558

```

display.c

```
559     temp = number / 1000;
560     Display.Digits[2] = (uint8_t)((number - (temp * 1000)) / 100);
561
562     temp = number / 10000;
563     Display.Digits[3] = (uint8_t)((number - (temp * 10000)) / 1000);
564
565     // negative sign
566     if (Display.sign) Display.Digits[3] = 34;
567
568 }
569
570
```


display.h

```
1 /*
2  * display.h
3  *
4  * Created on: Dec 19, 2020
5  * Author: Mehmet Akif/171024027
6  */
7
8 #ifndef DISPLAY_H_
9 #define DISPLAY_H_
10
11 #include "main.h"
12 #include "bsp.h"
13
14 typedef struct{
15     uint8_t Digits[4];
16     uint8_t Ofw:1;
17     uint8_t sign:1;
18     uint8_t dot:3;
19     uint8_t Inv:1;
20 }SSD;
21
22 /*
23  * Display struct keep the digits and
24  * overflow, sign, dot, invalid bits
25  */
26 SSD Display;
27
28 /*
29  * initiation for keypad pins
30  */
31 void init_SSD();
32
33 /*
34  * This function ensures that the digits on the display
35  * are lit by quickly flashing the digits.
36  */
37 void display_SSD();
38
39 /*
40  * the cases which are inside of this func show that
41  * how to display the character
42  */
43 void printDigit_SSD(uint8_t);
44
45 void displayID_SSD();
46
47 /*
48  * when the result bigger than 9999 or less than -999
49  * display shows that OuFL
50  */
51 void overflow_SSD();
52
53 /*
54  * when the operation is invalid i.e. 3/0 or sqrt(-2))
55  * display shows that Invd
56  */
57 void invalid_SSD();
58
59 /*
60  * separates the incoming result into digit
61  * and we can see that if number is negative or
62  * not through sign bit
```

display.h

```
63 */
64 void utility_SSD(float var);
65
66 /*
67 * It determines which character should be lit on which
68 * digit by assigning case values to digit.
69 */
70 void displaychar_SSD(uint8_t x);
71
72 #endif /* DISPLAY_H_ */
73
```

keypad.c

```
1 /*
2  * keypad.c
3  *
4  * Created on: Dec 19, 2020
5  * Author: Mehmet Akif/171024027
6  * description: In this section, necessary pins of
7  * the keypad have been activated in order for the keypad
8  * buttons to receive data. Next, the interrupt was created
9  * for the buttons. Thanks to this interrupt, when the button
10 * is pressed, it is processed according to priority.
11 * After determining which character the received data is from,
12 * it was sent to the display function for printing. It was sent to
13 * the calculation function for the necessary operations.
14 */
15 #include "keypad.h"
16
17 /*to the reach delay func*/
18 extern void delay_ms(volatile unsigned int);
19
20 void keypad_init(void){
21
22     /* Enable GPIOB and GPIOA clock */
23     RCC->IOPENR |= (1U << 1);
24     RCC->IOPENR |= (1U << 0);
25
26
27     /* Setup PA8,PB9,PB5 and PB4 as output (rows)*/
28     GPIOA->MODER &= ~(3U << 2*8);
29     GPIOA->MODER |= (1U << 2*8); //PA8 is output
30
31     GPIOB->MODER &= ~(3U << 2*9);
32     GPIOB->MODER |= (1U << 2*9); //PB9 is output
33
34     GPIOB->MODER &= ~(3U << 2*5);
35     GPIOB->MODER |= (1U << 2*5); //PB5 is output
36
37     GPIOB->MODER &= ~(3U << 2*4);
38     GPIOB->MODER |= (1U << 2*4); //PB4 is output
39
40
41
42     /* Setup PA9,PB0,PB2 and PB8 as input(columns) */
43     GPIOA->MODER &= ~(3U << 2*9); // PA9 is input
44     GPIOA->PUPDR |= (2U << 2*9); // Pull-down mode
45
46     GPIOB->MODER &= ~(3U << 2*0); //PB0 is input
47     GPIOB->PUPDR |= (2U << 2*0); // Pull-down mode
48
49     GPIOB->MODER &= ~(3U << 2*2); //PB2 is input
50     GPIOB->PUPDR |= (2U << 2*2); // Pull-down mode
51
52     GPIOB->MODER &= ~(3U << 2*8); //PB8 is input
53     GPIOB->PUPDR |= (2U << 2*8); // Pull-down mode
54
55
56     /*setup interrupts for inputs*/
57     EXTI->EXTICR[2] |= (0U << 8*1); //PA9
58     EXTI->EXTICR[0] |= (1U << 0); //PB0
59     EXTI->EXTICR[0] |= (1U << 2*8); //PB2
60     EXTI->EXTICR[2] |= (1U << 0); //PB8
61
62 }
```

keypad.c

```

63      /*rising edge*/
64      EXTI->RTSR1 |= (1U << 9); // 9th pin
65      EXTI->RTSR1 |= (1U << 0); // 0th pin
66      EXTI->RTSR1 |= (1U << 2); // 2th pin
67      EXTI->RTSR1 |= (1U << 8); // 8th pin
68
69
70      /* MASK*/
71      EXTI->IMR1 |= (1U << 9); // 9th pin
72      EXTI->IMR1 |= (1U << 0); // 0th pin
73      EXTI->IMR1 |= (1U << 2); // 2th pin
74      EXTI->IMR1 |= (1U << 8); // 8th pin
75
76
77      /*NVIC*/
78      NVIC_SetPriority(EXTI0_1_IRQn,0);
79      NVIC_EnableIRQ(EXTI0_1_IRQn);
80
81      NVIC_SetPriority(EXTI2_3_IRQn,0);
82      NVIC_EnableIRQ(EXTI2_3_IRQn);
83
84      NVIC_SetPriority(EXTI4_15_IRQn,0);
85      NVIC_EnableIRQ(EXTI4_15_IRQn);
86
87 }
88 /* interrut for PB0*/
89 void EXTI0_1_IRQHandler(void){
90     if (EXTI->RPR1 & (1U << 0)){ // check if pending register equal 1
91
92         clear_rows_keypad();
93         /* make PA8 enable*/
94         GPIOA->ODR ^=( 1U << 8);
95         if ((GPIOB->IDR >> 0) & 1){ //check if PB0 equal 1
96             /* #=(F) character*/
97             Keypad_data(15);
98
99         }
100         /*make PA8 disable*/
101         GPIOA->ODR ^=( 1U << 8); // PA8
102
103         /* make PB9 enable*/
104         GPIOB->ODR ^=( 1U << 9); // PB9
105         if ((GPIOB->IDR >> 0) & 1){
106             /* 9 character*/
107             Keypad_data(9);
108
109         }
110         /* make PB9 disable*/
111         GPIOB->ODR ^=( 1U << 9); // PB9
112
113         /* make PB5 enable*/
114         GPIOB->ODR ^=( 1U << 5); // PB5
115         if ((GPIOB->IDR >> 0) & 1){
116             /* 6 character*/
117             Keypad_data(6);
118
119         }
120         /* make PB5 disable*/
121         GPIOB->ODR ^=( 1U << 5); // PB5
122
123         /* make PB4 enable*/
124         GPIOB->ODR ^=( 1U << 4); // PB4

```

keypad.c

```

125     if ((GPIOB->IDR >> 0) & 1){
126         /* 3 character*/
127         Keypad_data(3);
128     }
129     /* make PB4 disable*/
130     GPIOB->ODR ^=( 1U << 4); // PB4
131
132
133     set_rows_keypad();
134     /*clear interrupt for clear pending register */
135     EXTI->RPR1 |= (1U << 0);
136 }
137 }
138 }
139
140 /* interrut for PB2*/
141 void EXTI2_3_IRQHandler(void){
142
143     if (EXTI->RPR1 & (1U << 2)){// check if pending register equal 1
144
145         clear_rows_keypad();
146         /*make PA8 enable*/
147         GPIOA->ODR ^=( 1U << 8); // PA8
148         if ((GPIOB ->IDR >> 2) & 1){//check if PB2 equal 1
149             /* D character*/
150             Keypad_data(13);
151
152         }
153         /*make PA8 disable*/
154         GPIOA->ODR ^=( 1U << 8); // PA8
155
156         /* make PB9 enable*/
157         GPIOB->ODR ^=( 1U << 9); // PB9
158         if ((GPIOB ->IDR >> 2) & 1){
159             /* C character*/
160             Keypad_data(12);
161
162         }
163         /* make PB9 disable*/
164         GPIOB->ODR ^=( 1U << 9); // PB9
165
166         /* make PB5 enable*/
167         GPIOB->ODR ^=( 1U << 5); // PB5
168         if ((GPIOB ->IDR >> 2) & 1){
169             /* B character*/
170             Keypad_data(11);
171
172         }
173         /* make PB5 disable*/
174         GPIOB->ODR ^=( 1U << 5); // PB5
175
176         /* make PB4 enable*/
177         GPIOB->ODR ^=( 1U << 4); // PB4
178         if ((GPIOB ->IDR >> 2) & 1){
179             /* A character*/
180             Keypad_data(10);
181
182         }
183         /* make PB4 disable*/
184         GPIOB->ODR ^=( 1U << 4); // PB4
185
186

```

keypad.c

```

187     set_rows_keypad();
188     /*clear interrupt for clear pending register */
189     EXTI->RPR1 |= (1U << 2);
190 }
191 }
192
193 /* interrut for PB8 and PA9*/
194 void EXTI4_15_IRQHandler(void){
195
196     /*interrut for PB8*/
197     if (EXTI->RPR1 & (1U << 8)){// check if pending register equal 1
198         clear_rows_keypad();
199         /*make PA8 enable*/
200         GPIOA->ODR ^=( 1U << 8); // PA8
201         if ((GPIOB ->IDR >> 8) & 1){//check if PB8 equal 1
202             /* *(E) character*/
203             Keypad_data(14);
204
205         }
206         /*make PA8 disable*/
207         GPIOA->ODR ^=( 1U << 8); // PA8
208
209         /* make PB9 enable*/
210         GPIOB->ODR ^=( 1U << 9); // PB9
211         if ((GPIOB ->IDR >> 8) & 1){
212             /* 7 character*/
213             Keypad_data(7);
214
215         }
216         /* make PB9 disable*/
217         GPIOB->ODR ^=( 1U << 9); // PB9
218
219         /* make PB5 enable*/
220         GPIOB->ODR ^=( 1U << 5); // PB5
221         if ((GPIOB ->IDR >> 8) & 1){
222             /* 4 character*/
223             Keypad_data(4);
224
225         }
226         /* make PB5 disable*/
227         GPIOB->ODR ^=( 1U << 5); // PB5
228
229         /* make PB4 enable*/
230         GPIOB->ODR ^=( 1U << 4); // PB4
231         if ((GPIOB ->IDR >> 8) & 1){
232             /* 1 character*/
233             Keypad_data(1);
234
235         }
236         /* make PB4 disable*/
237         GPIOB->ODR ^=( 1U << 4); // PB4
238
239
240         set_rows_keypad();
241         /*clear interrupt for clear pending register */
242         EXTI->RPR1 |= (1U << 8);
243     }
244
245     /*interrut for PA9*/
246     if (EXTI->RPR1 & (1U << 9)){// check if pending register equal 1
247         clear_rows_keypad();
248         /*make PA8 enable*/

```

keypad.c

```

249     GPIOA->ODR ^=( 1U << 8); //check if PA8 equal 1
250     if ((GPIOA ->IDR >> 9) & 1){
251         /* 0 character*/
252         Keypad_data(0);
253
254     }
255     /*make PA8 disable*/
256     GPIOA->ODR ^=( 1U << 8); // PA8
257
258     /* make PB9 enable*/
259     GPIOB->ODR ^=( 1U << 9); // PB9
260     if ((GPIOA ->IDR >> 9) & 1){
261         /* 8 character*/
262         Keypad_data(8);
263
264     }
265     /* make PB9 disable*/
266     GPIOB->ODR ^=( 1U << 9); // PB9
267
268     /* make PB5 enable*/
269     GPIOB->ODR ^=( 1U << 5); // PB5
270     if ((GPIOA ->IDR >> 9) & 1){
271         /* 5 character*/
272         Keypad_data(5);
273
274     }
275     /* make PB5 disable*/
276     GPIOB->ODR ^=( 1U << 5); // PB5
277
278     /* make PB4 enable*/
279     GPIOB->ODR ^=( 1U << 4); // PB4
280     if ((GPIOA ->IDR >> 9) & 1){
281         /* 2 character*/
282         Keypad_data(2);
283
284     }
285     /* make PB4 disable*/
286     GPIOB->ODR ^=( 1U << 4); // PB4
287
288
289     set_rows_keypad();
290
291     /*clear interrupt for clear pending register */
292     EXTI->RPR1 |= (1U << 9);
293 }
294
295 }
296
297
298 void clear_rows_keypad(void){
299     /*clearing the rows here*/
300     GPIOA->ODR &= ~(1U << 8); //PA8
301     GPIOB->ODR &= ~(1U << 9); //PB9
302     GPIOB->ODR &= ~(1U << 5); //PB5
303     GPIOB->ODR &= ~(1U << 4); //PB4
304 }
305
306 void set_rows_keypad(void){
307     /*setting the rows here*/
308     GPIOA->ODR |= (1U << 8); //PA8
309     GPIOB->ODR |= (1U << 9); //PB9
310     GPIOB->ODR |= (1U << 5); //PB5

```

```

311     GPIOB->ODR |= (1U << 4); //PB4
312
313 }
314
315 void Keypad_data(uint8_t a){
316
317     static int i = 0;
318
319     /*
320     *if the digits are already full,
321     *new number key presses are ignored
322     *by counter int i.
323     */
324     if ((a<10) & (i<4)){
325         calculation.x = (calculation.x * 10) + (float)a;
326         utility_SSD(calculation.x);
327         i++;
328     }
329     /*
330     *if ABCDE pressed,
331     *calculation mode will be selected
332     */
333     else if ((a>9) & (i<=4)){
334         i = 4;
335
336         if(a == 10){
337             calculation.current_process = Addition;
338             displaychar_SSD(Addition);
339             i = 7;
340         }
341         else if(a == 11){
342             calculation.current_process = Substraction;
343             displaychar_SSD(Substraction);
344             i = 7;
345         }
346         else if(a == 12){
347             calculation.current_process = Multiplacation;
348             displaychar_SSD(Multiplacation);
349             i = 7;
350         }
351         else if(a == 13){
352             calculation.current_process = Division;
353             displaychar_SSD(Division);
354             i = 7;
355         }
356         else if(a == 14){
357             calculation.current_process = E;
358             displaychar_SSD(E);
359             i = 5;
360         }
361     }
362     /*
363     * if E Key is pressed, scientific mode on
364     * and expect another keypress
365     */
366     else if (i == 5){
367
368         if(a == 10){
369             calculation.current_process = Log;
370             displaychar_SSD(Log);
371             i = 11;
372         }

```



```

373     else if(a == 11){
374         calculation.current_process = Ln;
375         displaychar_SSD(Ln);
376         i = 11;
377     }
378     else if(a == 12){
379         calculation.current_process = Sqrt;
380         displaychar_SSD(Sqrt);
381         i = 11;
382     }
383     else if(a == 13){
384         calculation.current_process = Pow2;
385         displaychar_SSD(Pow2);
386         i = 11;
387     }
388     else if(a == 14){
389         calculation.current_process = EE;
390         displaychar_SSD(EE);
391         i = 6;
392     }
393 }
394 /*
395  * if EE Key is pressed, trigonometric mode on
396  * and expect another keypress
397  */
398 else if (i == 6){
399
400     if(a == 10){
401         calculation.current_process = Sin;
402         displaychar_SSD(Sin);
403         i = 11;
404     }
405     else if(a == 11){
406         calculation.current_process = Cos;
407         displaychar_SSD(Cos);
408         i = 11;
409     }
410     else if(a == 12){
411         calculation.current_process = Tan;
412         displaychar_SSD(Tan);
413         i = 11;
414     }
415     else if(a == 13){
416         calculation.current_process = Cot;
417         displaychar_SSD(Cot);
418         i = 11;
419     }
420     else if(a == 14){
421         calculation.x = 3.141;
422         utility_SSD(calculation.x);
423         i = 4;
424     }
425 }
426 /*
427  * if ABCD pressed, requires another number
428  * for calculation
429  */
430 else if((a<10) & (i >= 7) & (i < 11)){
431     calculation.y = (calculation.y * 10) + (float)a;
432     utility_SSD(calculation.y);
433     i++;
434 }

```

```
435  /*
436   * F key is for equal
437   */
438  else if ((i == 11) | (a == 15)){
439
440      calculate();
441      i = 0;
442      calculation.x = calculation.result;
443      calculation.y = 0;
444      calculation.current_process = 0;
445      utility_SSD(calculation.x);
446  }
447 }
448
449
450
```

keypad.h

```
1 /*
2  * keypad.h
3  *
4  * Created on: Dec 19, 2020
5  * Author: Mehmet Akif/171024027
6  */
7
8 #ifndef KEYPAD_H_
9 #define KEYPAD_H_
10
11 #include "calc.h"
12 #include "main.h"
13
14 /*Keypad related function*/
15 void keypad_init(); //initiation for keypad pins
16 void clear_rows_keypad(); // set 0 keypad rows
17 void set_rows_keypad(); // set 1 keypad rows
18
19
20 /* taken data from button which is pressed
21 and figure out which button is this*/
22 void Keypad_data(uint8_t a);
23
24
25 #endif /* KEYPAD_H_ */
26
```

Problem 2:

In this problem, you will be working with reading and logging MPU6050 IMU sensor data utilizing Timer, I2C, and UART modules and use MPU6050, and 24LC512 EEPROM.

- Write your I2C routines to read / write multiple data. You should have four functions: single read, single write, multi read, multi write. Multi read and write deal with multiple bytes.
- To ensure the data is correct, send your results over UART to PC with You should read all sensor data and send them all as the example below:
1 AX: 0.12, AY: 0.53, ..., GX: 1.32, ...
- Sample the sensors every 10 ms, and write the data values to EEPROM. You should first start with writing and reading single bytes. Once the operation is completed successfully, work on your way to write and read multiple bytes.
- EEPROM and MPU6050 should be sharing the same I2C bus. Check the IMU board for pull-up resistors. If it includes pull-up resistors, you should not need to add another set of pull-up resistors.
- Once you press an external button, data collection should start, and once it collects 10 seconds of data, it should stop, and an LED should light up to display data is ready on EEPROM.
- When the LED is on (meaning there is data on the EEPROM) pressing the button will transmit all the data over UART to your PC.
- You can optionally save this to a file and/or plot your results using Python or Matlab.

Theoretical Research:

I2C tutorial:

This I2C tutorial shows you how the I2C protocol or more correctly written I²C (sometimes written as IIC) stands for Inter IC Communication and is intended for very short distance communication between ICs on a single PCB. It gives you a fully defined protocol for data transfer between multiple devices over two wires.

In this **I2C tutorial** you will learn all about the 2 wire I2C serial protocol; How easy it is to use, how it works and when to use it.

The I2C protocol is used in a huge range of chips - just a few examples from this site include the [DS1307](#) (RTC), [SSD1306](#) (OLED Display), [MCP23017](#) (Serial expander). The protocol allows you to connect many devices to a single set of two wires, and then communicate individually with each device.

This I2C tutorial shows you how the I2C protocol works at the physical bit level discussing single master mode (a single controlling device) which is the most common use for I2C in a small system.

512Kb I2C compatible 2-wire Serial EEPROM:

The Microchip Technology Inc. 24LC512 is a 512Kb (64K x 8) Serial Electrically Erasable PROM (EEPROM), capable of operation across a broad voltage range (2.5V to 5.5V). It has been developed for advanced, low-power applications such as personal communications and data acquisition. This device also has a page write capability of up to 128 bytes of data. This device is capable of both random and sequential reads up to the 512K boundary. Functional address lines allow up to eight devices on the same bus, for up to 4 Mbit address space. This device is available in the standard 8-pin plastic DIP, SOIJ and DFN packages.

Additional Features

- **Reliable EEPROM Memory**
 - 64K x 8 (512 Kbit)
 - 128-Byte Page Write Buffer
 - Page Write Time 5 ms Max.
 - Hardware Write-Protect Pin
 - Factory Programming Available
- **Low Power**
 - Operating voltage 1.7V to 5.5V
 - Read current 400 uA, max.
 - Standby current 1 uA, max.
- **2-Wire Serial Interface, I²C™ Compatible**
 - Cascadable up to Eight Devices
 - 100 kHz and 400 kHz Clock Compatible
- Pb-Free and RoHS Compliant

1.1. Code:

bsp.c

```
1 /*
2  * bsp.c
3  *
4  * Created on: 22 Ara 2020
5  * Author: Mehmet Akif/171024027
6  */
7 #include "bsp.h"
8
9 void BSP_init(){
10     __disable_irq();
11     init_I2C();
12     //SystemCoreClockUpdate(); //contains the system frequency
13     init_timer1();
14
15     __enable_irq();
16 }
17
18 void init_timer1(){
19
20     RCC->APBENR2 |= (1U<< 11); // enable time1 module clock
21
22     TIM1->CR1=0; // zero out the control register just in case
23     TIM1->CR1 |= (1<<7); // ARPE
24     TIM1->CNT=0; // zero out counter
25
26     /*0.1 ms interrupt */
27
28     TIM1->PSC=99;
29     TIM1->ARR=16;
30
31     TIM1->DIER |= (1 << 0); // update interrupt enable
32     TIM1->CR1 |= (1 << 0); // tim1 enable
33
34     NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn,3);
35     NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
36
37 }
38
39 void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
40 {
41     TIM1->SR &= ~(1U<<0); //clear update status register
42
43 }
44
45 void delay_ms(volatile unsigned int s){
46
47     for(int i=s; i>0; i--){
48         SysTick_Config(SystemCoreClock / 1000); // 16 MHz / 1000 ile 1 ms elde edildi.
49     }
50 }
51
52
```

bsp.h

```
1 /*
2  * bsp.h
3  *
4  * Created on: 22 Ara 2020
5  * Author: Mehmet Akif/171024027
6  */
7
8 #ifndef BSP_H_
9 #define BSP_H_
10
11 #include "main.h"
12
13 void BSP_init();
14 void init_timer1();
15
16 void delay_ms(volatile unsigned int);
17
18 #endif /* BSP_H_ */
19
```

eprom.c

```
1 /*
2  * eprom.c
3  *
4  * Created on: 29 Ara 2020
5  * Author: mehme
6  */
7
8 #include "eprom.h"
9
10 void read_eprom(uint16_t regAddr, uint8_t *data, uint32_t num){
11     uint8_t devAddr = 0x00;
12     uint8_t arr[2];
13     arr[0] = 0;
14     arr[1] = 0;
15     write_i2c(devAddr, arr, 1);
16     read_i2c(devAddr, data, num);
17 }
18
19 void write_eprom(uint16_t regAddr, uint8_t *data, uint32_t num){
20     uint8_t devAddr = 0x00;
21     uint8_t arr[2];
22     arr[0] = 0;
23     arr[1] = 0;
24     write_i2c(devAddr, arr, 1);
25     write_i2c(devAddr, data, num);
26 }
27
```


eprom.h

```
1 /*
2  * eprom.h
3  *
4  * Created on: 29 Ara 2020
5  * Author: mehme
6  */
7
8 #ifndef EEPROM_H_
9 #define EEPROM_H_
10
11 #include "main.h"
12
13 void read_eprom(uint16_t regAddr, uint8_t *data, uint32_t num);
14 void write_eprom(uint16_t regAddr, uint8_t *data, uint32_t num);
15
16 #endif /* EEPROM_H_ */
17
18
```

mpu6050.c

```
1 /*
2  * mpu6050.c
3  *
4  * Created on: 29 Ara 2020
5  * Author: mehme
6  */
7
8 #include "mpu6050.h"
9
10 void read_mpu(uint8_t regAddr, uint8_t *data, uint32_t num){
11     uint8_t devAddr = 0x00;
12     write_i2c(devAddr, &regAddr, 1);
13     read_i2c(devAddr, data, num);
14 }
15
16 void write_mpu(uint8_t regAddr, uint8_t *data, uint32_t num){
17     uint8_t devAddr = 0x00;
18     write_i2c(devAddr, &regAddr, 1);
19     write_i2c(devAddr, data, num);
20 }
21
```

mpu6050.h

```
1 /*
2  * mpu6050.h
3  *
4  * Created on: 29 Ara 2020
5  * Author: mehme
6  */
7
8 #ifndef MPU6050_H_
9 #define MPU6050_H_
10
11 #include "main.h"
12
13 void read_mpu(uint8_t regAddr, uint8_t *data, uint32_t num);
14 void write_mpu(uint8_t regAddr, uint8_t *data, uint32_t num);
15
16
17 #endif /* MPU6050_H_ */
18
19
20
```

I2C.c

```

1 /*
2  * I2C.c
3  *
4  * Created on: 27 Ara 2020
5  * Author: mehme
6  */
7
8 #include "I2C.h"
9
10
11 void I2C1_IRQHandler(void){
12
13 }
14
15
16 void init_I2C(void){
17
18     //Enable GPIOB
19     RCC->IOPENR |= (1U << 1);
20
21     //setup PB8 as AF6
22     GPIOB->MODER &= ~(3U << 2*8);
23     GPIOB->MODER |= (2 << 2*8);
24     GPIOB->OTYPER |= (1U << 8);
25
26     //choose AF from mux
27     GPIOB->AFR[1] &= ~(0xFU << 4*0);
28     GPIOB->AFR[1] |= (6 << 4*0);
29
30     //setup PB9 as AF6
31     GPIOB->MODER &= ~(3U << 2*9);
32     GPIOB->MODER |= (2 << 2*9);
33     GPIOB->OTYPER |= (1U << 9);
34
35     //choose AF from mux
36     GPIOB->AFR[1] &= ~(0xFU << 4*1);
37     GPIOB->AFR[1] |= (6 << 4*1);
38
39     //enable I2C1
40     RCC->APBENR1 |= (1U << 21);
41
42     I2C1->TIMINGR |= (3 << 28); //PRESC
43     I2C1->TIMINGR |= (0x13 << 0); //SCLL
44     I2C1->TIMINGR |= (0xF << 8); //SCLH
45     I2C1->TIMINGR |= (0x2 << 16); //SDADEL
46     I2C1->TIMINGR |= (0x4 << 20); //SCLDEL
47
48     I2C1->CR1 = (1U << 0); //PE
49
50     NVIC_SetPriority(I2C1_IRQn, 1);
51     NVIC_EnableIRQ(I2C1_IRQn);
52 }
53
54
55
56
57 void write_i2c(uint8_t devAddr, uint8_t *data, uint32_t num){
58
59     //Write operation (Send address and register to read)
60     I2C1->CR2 = 0;
61     I2C1->CR2 |= ((uint32_t)devAddr << 1); // slave address
62     I2C1->CR2 |= (num << 16); // Number of byte

```

I2C.c

```
63 I2C1->CR2 |= (1U << 25); // AUTOEND
64 I2C1->CR2 |= (1U << 13); // Generate Start
65
66 for (size_t i=0; i<num; ++i){
67     while(!(I2C1->ISR & (1 << 1))); //TXIS
68     I2C1->TXDR = data[i];
69 }
70 }
71
72
73 void read_i2c(uint8_t devAddr, uint8_t *data, uint32_t num){
74
75     I2C1->CR2 = 0;
76     I2C1->CR2 |= ((uint32_t)devAddr << 1);
77     I2C1->CR2 |= (1U << 10); //READ mode
78     I2C1->CR2 |= (num << 16); //Number of bytes
79     I2C1->CR2 |= (1U << 15); //NACK
80     I2C1->CR2 |= (1U << 25); //AUTOEND
81     I2C1->CR2 |= (1U << 13); //Generate Start
82
83     for(size_t i=0; i<num; i++){
84         while(!(I2C1->ISR & (1 << 2))); // wait until RXNE =1
85         data[i] = (uint8_t)I2C1->RXDR;
86     }
87 }
88
89
```

I2C.h

```
1 /*
2  * I2C.h
3  *
4  * Created on: 27 Ara 2020
5  * Author: mehme
6  */
7
8 #ifndef I2C_H_
9 #define I2C_H_
10
11 #include "main.h"
12
13
14 #define MPU6050_ADDRESS      0x68
15
16 #define MPU6050_WHO_AM_I    0x75
17 #define MPU6050_PWR_MGMT_1  0x6B
18
19 #define MPU6050_ACCEL_XOUT_H 0x3B
20 #define MPU6050_ACCEL_XOUT_L 0x3C
21 #define MPU6050_ACCEL_YOUT_H 0x3D
22 #define MPU6050_ACCEL_YOUT_L 0x3E
23
24 #define MPU6050_GYRO_XOUT_H   0x43
25 #define MPU6050_GYRO_XOUT_L   0x44
26
27 void init_I2C();
28 void read_i2c(uint8_t devAddr, uint8_t *data, uint32_t num);
29 void write_i2c(uint8_t devAddr, uint8_t *data, uint32_t num);
30
31
32 typedef struct {
33     uint8_t ax;
34     uint8_t ay;
35     uint8_t az;
36     uint8_t gx;
37     uint8_t gy;
38     uint8_t gz;
39 }values;
40
41
42 #endif /* I2C_H_ */
43
```

main.c

```
1 /*
2  * main.c
3  *
4
5 */
6
7 #include "main.h"
8
9
10
11
12
13
14 void delay(volatile uint32_t);
15
16 int main(void) {
17     BSP_init();
18
19     return 0;
20 }
21
22
23 void delay(volatile uint32_t s) {
24     for(; s>0; s--);
25 }
26
```

main.h

```
1 /*
2  * main.h
3  *
4  * Created on: 27 Ara 2020
5  * Author: mehme
6  */
7
8 #ifndef MAIN_H_
9 #define MAIN_H_
10
11 #include "stm32g0xx.h"
12 #include "stdlib.h"
13 #include "I2C.h"
14
15
16 #endif /* MAIN_H_ */
17
```


CONCLUSION

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. All functions required in Problem 1 are prepared etc. 7SDs, keypad and timer except Pwm module. While preparing the Pwm module, the required registers were assigned and the required pins were activated. But pwm was not observed to work. Where the error happened has not been found. So although everything was ready in problem 1, it could not be completely completed. In Problem 2, briefly requested from us; Reading the data from MPU6050 IMU sensor with I2C and writing to the memor with 24LC512 EEPROM I2C. Then transfer the values in EEPROM to PC via UART. We are expected to transfer data for 10 seconds at 10ms intervals. This corresponds to 1000 pieces of data in 10 seconds. Most of I2C has been completed, but multiple bit reading and writing could not be completed. Necessary research has been done. A struct was created to hold data in EEPROM. However, the systems created could not be successfully combined. Many problems were successfully concluded, but other problems could not be completed because there was not enough time and the self-teach technique was used in this lesson.

References

- <https://learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle>
- https://en.wikipedia.org/wiki/Pulse-width_modulation
- <https://www.robot-electronics.co.uk/i2c-tutorial#:~:text=I2C%20Device%20Addressing,be%20from%200%20to%20127.>
- https://elinux.org/Interfacing_with_I2C_Devices
- <https://www.best-microcontroller-projects.com/i2c-tutorial.html>
- <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>
- <https://www.microchip.com/wwwproducts/en/24LC512>