GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC334

MICROPROCESSORS

HW #4

Prepared by
1801022022 – Alperen Karataş

Problem 1. Board Support Package:

A board support package (BSP) is essential code for a given computer hardware device that will make that device work with the computer's OS (operating system). The contents of the BSP depend on the particular hardware and OS.

BSP can host a drive specifically for each supported graphics card. Depending on the graphics driver supported by the hardware system used, the desired drive can be included via BSP.

BSPs can be found as a package with ready-made libraries, ready-made functions, code sequences. In addition, BSPs can be specially prepared by the person or played inside a library that is available. This can be decided based on the need for the code to be used or the work to be done.

For example, as can be seen in the BSP prepared by me, a number of functions have been created to be used in need and where necessary. Examples of these functions are various functions such as LED toggle operation, external interrupt creation process with button, code function that determines numbers on SSD. These functions mentioned and the files of the prepared BSP (bsp.h, bsp.c) can be seen in the section where the codes are at the end. The following is a block diagram of a BSP in terms of hardware and software ability.

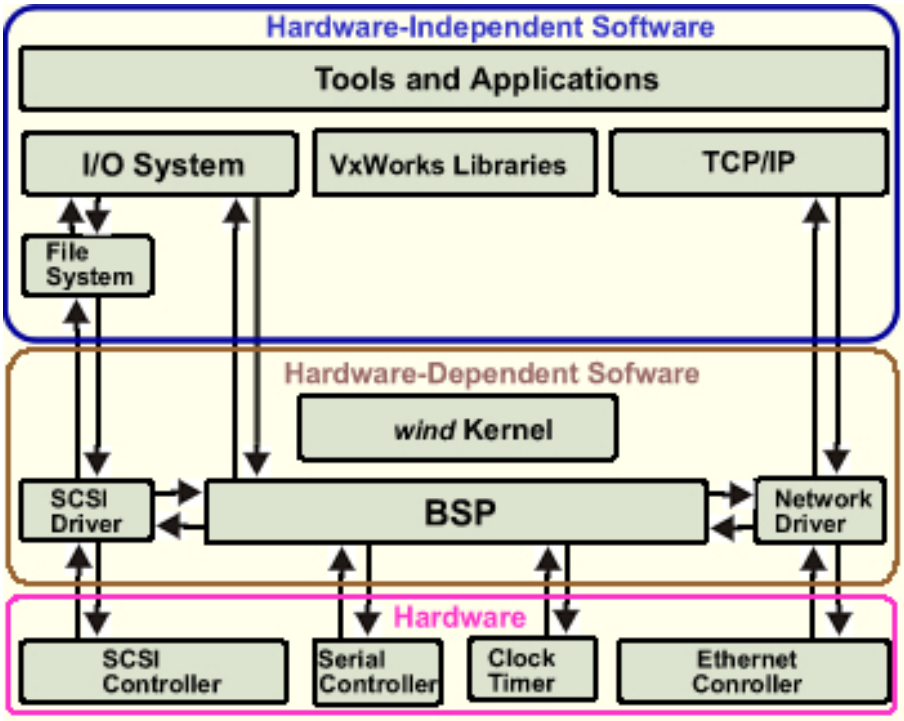

Figure 1. BSP is an interface between kernel and hardware

Problem 2. Software Faults in the wild:

Syntax Error:

This error is an error found in the program's source code. Compilers must follow a syntax within the framework of certain rules in order to compile the program according to the programming language used. Syntax Error will occur at a point that does not conform to the syntax of this programming language.

It can also be caused by simple grammar errors, or Syntax Error may be encountered due to some characters that have been forgotten or missed. This type of error is not a type that can be encountered because of an error in the program's flow or logic. As can be seen in the following piece of code, for example, in C language, not added to the end of the line **;** character will cause the compiler to return Syntax Error because the C language does not fit the syntax.

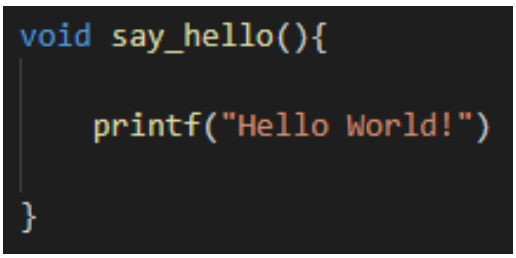

Figure 2. Bunch of C code about Syntax Error

Logic Error:

The situation with Logic Error may occur when incorrect or unexpected behavior is encountered on the source code. This error may cause the program to produce an out-of-expected or incorrect output. Logic Error is a type of error that can be noticed during operation. Because of this type of error is usually hidden in source code, these errors are more difficult to find and extract than Syntax Error. For example, the use of missing brackets on a function performing a calculation can draw an out-of-expected result. As shown in the following example, when summing the numbers a and b, different results will be seen in the output if this operation bracketed or not received.

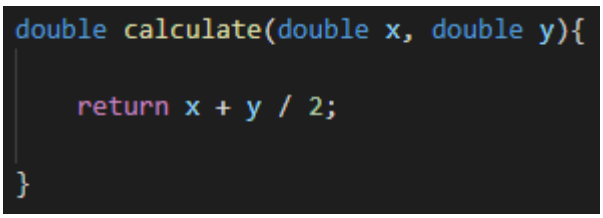

Figure 3. Bunch of C code about Logic Error

Segmentation Fault:

Segmentation Fault can be called an access violation. This error may be encountered when trying to access the memory space of the operating system that is protected by hardware. The operating system will take a re-correct action in response to resolve this issue. This action is about sending a signal to the process and taking it into a process to solve the part that poses a problem. In this process, a special signal handler is sometimes used, which allows recovery on their own, while sometimes the operating system uses its own signal handler. Segmentation Faults is a type of error commonly seen in languages with low-level memory access, such as C. The following is an example of this type of error in C Programming Language.

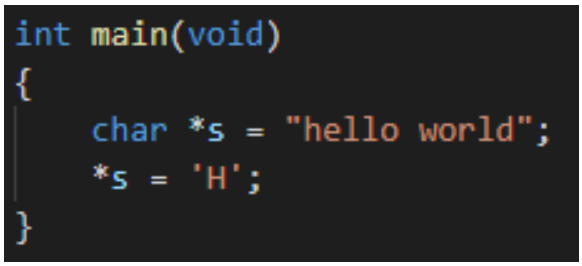

Figure 4. Bunch of C code about Segmentation Fault

Error Handling Errors:

Kullanıcının işlem sırasında bir hata aldığı zaman bu hatanın sebebini açıkça ekranda görebilmesi gereklidir. Hatanın gerçekte ne olduğuna dair bir takım açıklamaların yapması ve bunların bilinmesi gereklidir. İzlenmesi gereken adımlar, açıkça yazılmalıdır. Yetersiz ve kısa açıklamalar bu hata sınıfına girer.

Different examples from the World:

F35 fighter jets were found to have made some inaccuracies in determining the number of targets during an exercise. Due to a software error based on glitch formation, it was noticed that planes confused and mis-determined the number of these targets when trying to detect targets from different angles.

Knight, one of america's largest market makers, lost $440 million in 30 minutes due to a bug in the software. Knight’s trading algorithms reportedly started pushing erratic trades through on nearly 150 different stocks, sending them into spasms. This software error caused the company's shares to lose 75% in two days.

In February 1991, a software failure caused serious consequences when Patriot missiles stationed at an American base in Saudi Arabia were cannot locate incoming attacks. While the software error in the missile continued, it was found out that the more the missile worked, the more serious errors it made in locating. The system, which has been running for more than 100 hours, killed 28 American soldiers in the attacks, as it repeatedly showed the wrong locations as it tried to locate the attack. Even if the software error was resolved 1 day before the attacks, delivering this solution to the region could only happen one day after the attack.

In 1994, a bug was discovered in Intel's Pentium processors by a math professor. The company acknowledged the error, but said the flaw would go unnoticed by the vast majority of people. However, it has been said that this gap can be closed for users who want it. This rhetoric cost Intel $ 475 million to lose.

Reference articles and video links:

https://www.defensenews.com/air/2019/06/12/the-pentagon-is-battling-the-clock-to-fix-serious-unreported-f-35-problems/

http://www.cs.unc.edu/~smp/COMP205/LECTURES/ERROR/lec23/node4.html

https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

https://apnews.com/article/853864f8f1a74457adf39ab272181e08

https://www.youtube.com/watch?v=xCVq1KYcM_E

https://www.youtube.com/watch?v=bfWxAG1vUM4

Problem 3. Bouncing:

According to the logic of working button, two metal parts come together as soon as the button is pressed. This process may appear to have taken place directly at the time of contact by the person pressing the button, but this is not true. There are metal parts inside the button. As soon as the button is pressed, these metal fragments only make contact in a short part of a microsecond. Then this time is a little longer and it grows a little longer again, and eventually the switch closes. In this case, once the button is pressed, it considers that the hardware button is pressed more than once, since the contact in the button occurs several times and indecisively. Hardware and software methods are included to solve this bouncing.

Hardware methods:

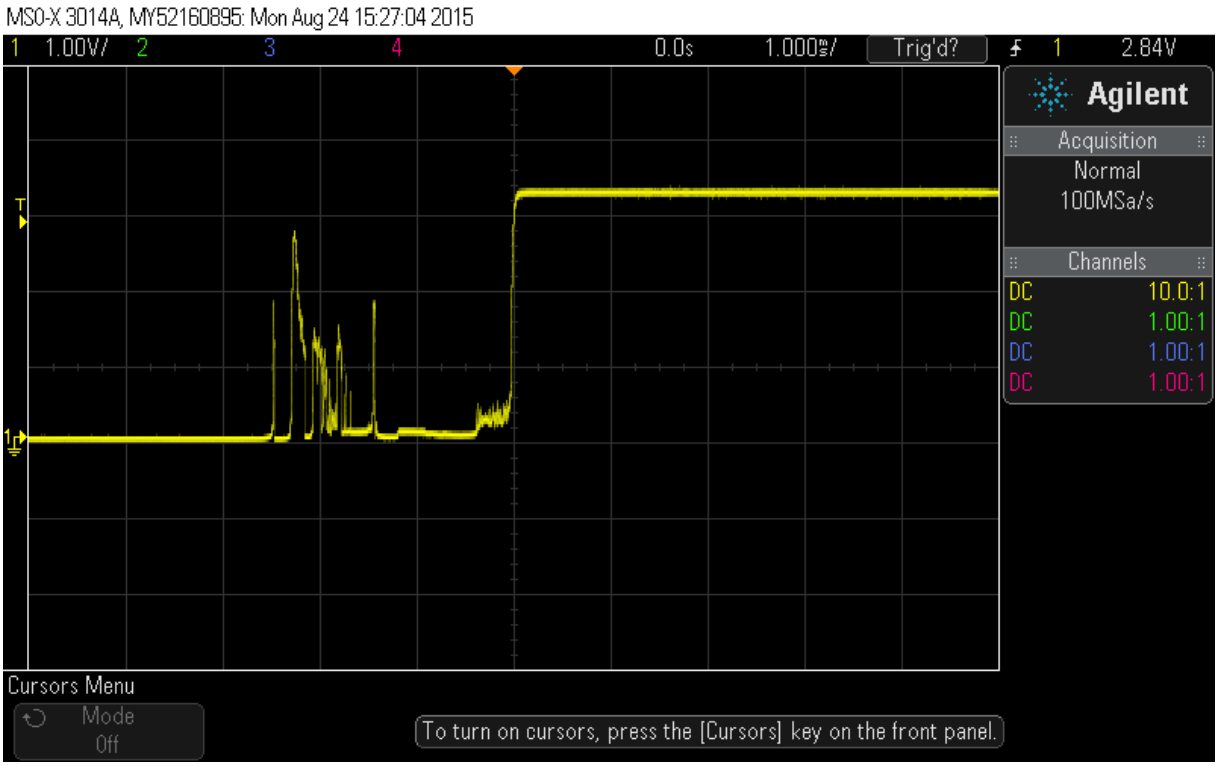

Figure 5. Button debouncing oscilloscope screen

In the oscilloscope output on Figure 5, the related bouncing status can be easily seen with the interval consisting of unstable state. To resolve this situation, a capacitor can be added to the circuit. The corresponding circuit connection can be seen in Figure 6.

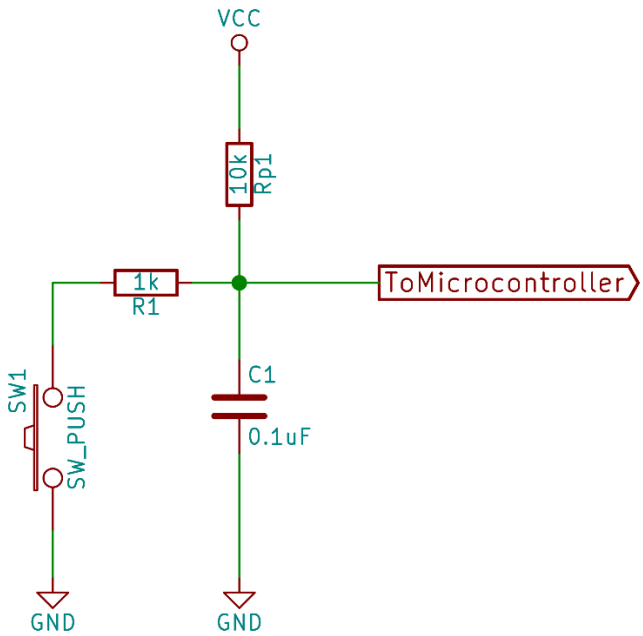

Figure 6. Circuit schematic about button with capacitor

The capacitor will act as a buffer for the unstable region in between, as the voltage in the button will merge with the incoming signal and flow into the microcontroller. This will solve the problem.

Because of the capacitor solved the problem at a certain level, there was a need to look for a more precise solution. For this purpose, integrated circuits are used to solve the problem. Examples of these integrated circuits include HCF4017BE, MAX6816, MAX6817, and MAX6818. Figure 7 features the MAX6816 integrated circuit and the circuit installed for button bouncing.

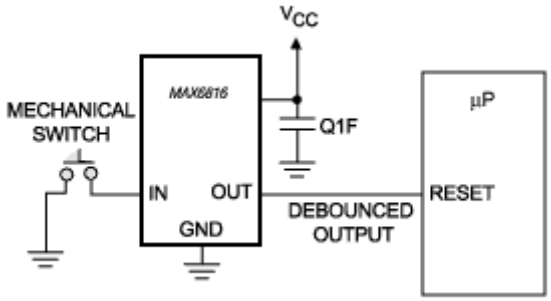

Figure 7. Circuit schematic about button with IC

Software Methods:

One of the software methods used to ignore and pass unstable state generated by the button is to add a delay. However, this means that the processor delays other work in the same sense, and as predicted, this is not very desired. As another solution, necessary actions can be taken based on the state of a defined variable. The necessary actions will be taken according to the variable receiving a value of 1 and 0. Situations where the variable is 1 are the desired states and mean that the button is pressed. However, for variable debounce states, it should take a value of 0 in certain places and this unstable state period should be expected to pass. Figure 8 has a piece of C code for the relevant solution.

```c
/* external interrupt function for button */
void EXTI2_3_IRQHandler(void){

    if ((GPIOB->IDR & (1 << 3)) && button_state == 1) {
        mode_type_cnt++;
        if (mode_type_cnt == 5)
            mode_type_cnt = 0;
        choose_mode();
        button_state = 0;           // for debouncing
    }
    else
        delay_ms(1000000);

    EXTI->RPR1 |= (1U << 3);    // rising edge pending register

}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void){
    BSP_led_toggle();
    button_state = 1;
    TIM1->SR &= ~(1U << 0);
}
```

Figure 8. Bunch of C code about preventing the debouncing

Problem 4. State Machines:

When writing this code, an enum structure was used for the desired mode states. This enum structure contains 5 different variables for 5 different modes. Pressing the button created a function to change the mode. To solve the debouncing problem of the button, the methods described in the software methods mentioned in Problem 4 are used. Toggle times for LED were made by changing the ARR register value linked to TIM1. The relevant code content can be found in the section where the codes are located at the end. In addition, the relevant video link for the code and circuit can be found in the "**video_link.txt**" file in the assignment file.

Problem 5. Reading: (The article abstracts are written in Turkish because of the Turkish is allowed for abstracts.)

Trends in Embedded Software Engineering

Son yıllarda, yeni ürünlerin geliştirilmesi, kısa sürede birden çok işlem yapabilme yeteneği, donanım-yazılım güvenliği gibi sebeplerle yazılım gömülü sistemlerin üzerindeki etkisi oldukça arttı; böylelikle bu alandaki talepte de bir yükseliş meydana geldi..

Gömülü sistemlerde, en az maliyet, en az seviyede enerji, en az seviyede boyutlarla en yüksek performans, en efektif seviyede verim, birçok fonksiyona sahip bir algoritma geliştirmek en temel yaklaşımlardan birisi olmalıdır. Aynı zamanda gömülü sistemlerde, ürün odaklı çalışmak göz önünde bulundurulmalıdır. Bütün bu etmenler göz önünde bulundurulduğuna, aslında gömülü sistemlerde birden çok mühendislik disiplinine ihtiyaç duyulacağı gerçeğiyle karşılaşılmaktadır. Bundan kaynaklı olarak da, bu disiplinler arasında ürün odaklı bir sistem geliştirilmesi gerektiği ana fikri göz önünde bulundurularak, senkronize çalışma durumu benimsenmelidir.

Gömülü sistemlerde karışıklığı azaltmak, pazara sunma süresini kısaltmak ve ürünün kalitesini artırmak ve maliyeti minimum seviyede tutmak gibi bazı temel kaygılar vardır. Bu kaygıların çözümünü için son yıllarda ortaya çıkan bir yaklaşım olan **model-driven development (MDD)** yaklaşımı, oldukça umut vericidir. Bu yaklaşımda, programlama dilleriyle doğrudan yazılımı kodlamak yerine, sezgisel ve soyutsal temelde bazı modellemeler ve grafikler kullanılarak yazılım sistemleri modellenmektedir.

Gömülü sistemler için kalite güvencesi de oldukça önemli bir unsurdur. Kalite güvencesi, sistemin güvenilirliği ile ölçülebilir. Güvenliği sağlamada statik ve resmi doğrulama teknikleri uygulanabilir. Ancak her tekniğin tam bir doğruluk sağlayacağı garantisi verilemez. Resmi doğrulama tekniklerine örnek olarak dinamik testler söylenebilir.

Güvenliği sağlamada kullanılan tekniklerde, standart tekniklere kıyasla, güvenlik analizi tekniklerinin kullanımı çok daha doğru uygun olacaktır. Hata ağaç yapısını temel alarak kullanılan bu modele talep çok fazladır.

Güvenlik mühendisleri, geliştirme sürecinde bazı kritik tasarım hatalarını önlemek ve bu süreci daha yapıcı bir şekilde kullanmak için yukarılarda bahsedilen modelleme tekniklerini kullanmaktadırlar. Ancak sonuçların doğruluğu güvenlik ve güvenilirlik modellerinin testlerine bağlıdır.

Problem 6. Reading: (The article abstracts are written in Turkish because of the Turkish is allowed for abstracts.)

Applying Test Driven Development to Embedded Software

Testi geçmek için bir kod yazılır, derleme yapılır ve testten geçilip geçilmediği görülür. Yinelemenin kaldırılması için yeniden düzenleme işlemi yapılır ve bu adımlar tekrar edilir.

TDD (Test Driven Development) döngüsü için temel bazı adımlar vardır; üzerinde çalışılacak modül ve ortak çalışma alanları belirlenir, modül hakkında ara yüz keşfi ve alt küme seçimi yapılır. Bu adımlar için testler ve daha karışık testler için TDD döngüsü tekrardan uygulanır. İşlemin tamam olup olmadığı kontrol edilir. TDD uygulamaları, öngörülebilirlik, tekrarlanabilirlik ve hız açısından oldukça önemli faydalar sağlar.

Test çeşitlerinden birim testi, yazılan kodun beklenen şeyi gerçekleştirip gerçekleştirmediği konusunda geliştiriciye bir geri bildirim sağlar. Gömülü yazılımda çoğunluk C/C++ dilleri kullanılır. Birim testleri C/C++ dillerinin içerdiği modüller üzerinde işlem yapar ve geliştiriciye bu şekilde bir dönüt sağlar.

TDD konusunda, gömülü sistem mühendislerinin bir takım şüpheleri vardır. Bu şüpheler TDD uygulamasının, bazı çeşitli sebeplerden dolayı gömülü sistemlerde uygulanmasının zor olacağı hakkındadır. Lakin TDD, bazı gömülü sistem uygulamalarında da işe yarayabilmektedir. Gömülü sistem projelerindeki bazı büyük sorunlardan biri eşzamanlı donanım/yazılım geliştirme konusudur. Geliştirilen yazılım, Test edilmek için güvenli bir donanımı bekler.

Gömülü TDD döngüsü bazı aşamalardan oluşmaktadır. İlk aşama en az kusurla kodu yazmaktır. İkinci aşamada derleme yapılırken bağlantı sorunlarının olup olmadığını kontrol edilir. Üç ve dördüncü aşamalarda yazılan kod hedef donanım üzerinde çalıştırılır. Son olarak ise test aşaması gerçekleştirilir.

TDD yapılırken, donanımın bellek boyutu, derleyici ile geliştirme sistemi arasındaki iletişimin uyumlu bir şekilde kullanılması gibi bazı şartlar bulunmaktadır. Gömülü sistem, donanıma bağımlı hale gelmeden test edilebilir duruma getirilmelidir. Test aşamalarına, otomatik test, kısmen otomatikleştirilmiş donanım testleri ve harici otomatik donanım testleri örnek verilebilir.

TDD, C/C++ hatta bazen Java dillerinde kullanılabilecek şekilde daha kaliteli ürünler sunma konusunda oldukça önemli bir yazılım geliştirme uygulamasıdır.

References:

1. https://en.wikipedia.org/wiki/Board_support_package
2. https://whatis.techtarget.com/definition/board-support-package
3. FAQ: What is a Board Support Package? By Ron Fredericks, Wind River, August 20, 2001, updated November 20, 2002
4. https://techterms.com/definition/syntax_error#:~:text=A%20syntax%20error%20is%20an,will%20produce%20a%20syntax%20error.
5. https://techterms.com/definition/logic_error
6. https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/
7. https://hackaday.com/2015/12/09/embed-with-elliot-debounce-your-noisy-buttons-part-i/
8. https://www.maximintegrated.com/en/design/technical-documents/app-notes/2/287.html
9. https://www.quora.com/

and **Problem 2.Software Faults in the wild** references which has been added to the end.

!!!There are codes for Problem 1 and Problem 4 after this page!!!

Problem 1:

```c
/* bsp.h
*
* Prepared by:
* Alperen Karataş - 1801022022
*
*/

#ifndef BSP_H_
#define BSP_H_

void BSP_led_init(void);
void BSP_led_toggle(void);
void BSP_led_set();
void BSP_led_clear();
void BSP_button_interrupt_init(void);
void BSP_clearSSD(void);
void BSP_setSSD(int x);

#endif
```

```c
/* bsp.c
*
* Prepared by:
* Alperen Karataş - 1801022022
*
*/

#include "stm32g0xx.h"
#include "bsp.h"

void BSP_led_init(void){
     /* Enable GPIOA clock */
     RCC->IOPENR |= (1U << 0);

     /* Setup PA6 as output */
     GPIOA->MODER &= ~(3U << 2*6);
     GPIOA->MODER |= (1U << 2*6);

     /* clear LED */
     GPIOA->BRR |= (1U << 6);
}

void BSP_led_toggle(void){
     GPIOA->ODR ^= (1U << 6);
}

void BSP_led_set(){
     /* Turn on LED on PA6 */
     GPIOA->ODR |= (1U << 6);
}
void BSP_led_clear(){
     /* Turn on LED on PA6 */
     GPIOA->BRR |= (1U << 6);
}
```

```c
void BSP_button_interrupt_init(void){
    /* Enable GPIOB clock */
    RCC->IOPENR |= (1U << 1);

    /* Setup PB3 as input */
    GPIOB->MODER &= ~(3U << 2*3);

    /* rising edge trigger selection, selection register
and mask register */
    EXTI->RTSR1 |= (1U << 3);
    EXTI->EXTICR[0] |= (1U << 8*3);
    EXTI->IMR1 |= (1U << 3);

    /* enable NVIC and set interrupt priority */
    NVIC_SetPriority(EXTI2_3_IRQn, 0);
    NVIC_EnableIRQ(EXTI2_3_IRQn);

}

void BSP_clearSSD(void){

    GPIOB ->ODR |= (1U << 0);  //PB0 A
    GPIOB ->ODR |= (1U << 1);  //PB1 B
    GPIOB ->ODR |= (1U << 2);  //PB2 C
    GPIOB ->ODR |= (1U << 3);  //PB3 D
    GPIOB ->ODR |= (1U << 4);  //PB4 E
    GPIOB ->ODR |= (1U << 5);  //PB5 F
    GPIOB ->ODR |= (1U << 6);  //PB6 G
}
void BSP_setSSD(int x){
    BSP_clearSSD();
    switch(x){
        case 0:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 4); //PB4 E
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            break;
```

```
        case 1:
             GPIOB->ODR &= ~(1U << 1); //PB1 B
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             break;

        case 2:
             GPIOB->ODR &= ~(1U << 0);  //PB0 A
             GPIOB->ODR &= ~(1U << 1);  //PB1 B
             GPIOB->ODR &= ~(1U << 3);  //PB3 D
             GPIOB->ODR &= ~(1U << 4); //PB4 E
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;

        case 3:
             GPIOB->ODR &= ~(1U << 0); //PB0 A
             GPIOB->ODR &= ~(1U << 1); //PB1 B
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             GPIOB->ODR &= ~(1U << 3); //PB3 D
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;

        case 4:
             GPIOB->ODR &= ~(1U << 1); //PB1 B
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             GPIOB->ODR &= ~(1U << 5); //PB5 F
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;
        case 5:
             GPIOB->ODR &= ~(1U << 0); //PB0 A
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             GPIOB->ODR &= ~(1U << 3); //PB3 D
             GPIOB->ODR &= ~(1U << 5); //PB5 F
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;
```

```
        case 6:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 4); //PB4 E
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            GPIOB->ODR &= ~(1U << 6);  //PB6 G
            break;

        case 7:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            break;

        case 8:
            GPIOB->ODR &= ~(1U << 0); //PA0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 4); //PB4 E
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            GPIOB->ODR &= ~(1U << 6);  //PB6 G
            break;

        case 9:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            GPIOB->ODR &= ~(1U << 6);  //PB6 G
            break;
    }
}
```

Problem 4:

```
/* main.c
*
* Prepared by:
* Alperen Karataş - 1801022022
*
* Notes:
* ELEC334 2020 Fall HW#4 - Problem 4
* State Machines
*
*/

#include "stm32g0xx.h"
#include "bsp.h"

enum modes{
     mode0, mode1, mode2, mode3, mode4
};

enum modes mode;

_Bool button_state=1;
volatile uint32_t sys_count = 0;
int mode_type_cnt=1;

/* interrupt function */
void SysTick_Handler(void) {
     if(sys_count > 0)
      sys_count--;
}

/* external interrupt function for button */
void EXTI2_3_IRQHandler(void){

     if ((GPIOB->IDR & (1 << 3)) && button_state == 1) {
          mode_type_cnt++;
          if (mode_type_cnt == 5)
               mode_type_cnt = 0;
          choose_mode();
          button_state = 0;           // for debouncing
     }
```

```
else
          delay_ms(1000000);

     EXTI->RPR1 |= (1U << 3);    // rising edge pending
register

}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void){
     BSP_led_toggle();
     button_state = 1;
     TIM1->SR &= ~(1U << 0);
}

/* this function chooses the state */
void choose_mode(){

     if (mode_type_cnt == mode0)
          TIM1->ARR = 0;
     else if (mode_type_cnt == mode1)
          TIM1->ARR = 1600*10;
     else if (mode_type_cnt == mode2)
          TIM1->ARR = 1600*5;
     else if (mode_type_cnt == mode3)
          TIM1->ARR = 1600*1;
     else if (mode_type_cnt == mode4)
          TIM1->ARR = 1600*39;

     button_state = 1;
}

void delay_ms(volatile uint32_t s) {
     for (int i = 0; i < s; ++i);
}
```

```c
void init_timer1(){

    /* enable TIM1 module clock */
    RCC->APBENR2 |= (1U << 11);

    TIM1->CR1 = 0;          // make zero the control
register just in case
    TIM1->CR1 |= (1 << 7); // ARPE register
    TIM1->CNT = 0;             // make zero the counter

    /* 1 second interrupt -for first toggle obviously-
*/
    TIM1->PSC = 999;
    TIM1->ARR = 1600*10;

    TIM1->DIER |= (1 << 0);          // update interrupt
enable
    TIM1->CR1 |= (1 << 0);      // TIM1 enable

    /* enable NVIC and set interrupt priority  */
    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 3);
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}

int main(void) {

    SysTick_Config(16000);
    BSP_led_init();
    init_timer1();
   BSP_button_interrupt_init();
   choose_mode();

   while(1) {

   }

   return 0;
}
```

```c
/* bsp.h
*
* Prepared by:
* Alperen Karataş - 1801022022
*
*/

#ifndef BSP_H_
#define BSP_H_

void BSP_led_init(void);
void BSP_led_toggle(void);
void BSP_led_set();
void BSP_led_clear();
void BSP_button_interrupt_init(void);
void BSP_clearSSD(void);
void BSP_setSSD(int x);

#endif
```

```c
/* bsp.c
*
* Prepared by:
* Alperen Karataş - 1801022022
*
*/

#include "stm32g0xx.h"
#include "bsp.h"

void BSP_led_init(void){
    /* Enable GPIOA clock */
    RCC->IOPENR |= (1U << 0);

    /* Setup PA6 as output */
    GPIOA->MODER &= ~(3U << 2*6);
    GPIOA->MODER |= (1U << 2*6);

    /* clear LED */
    GPIOA->BRR |= (1U << 6);
}

void BSP_led_toggle(void){
    GPIOA->ODR ^= (1U << 6);
}

void BSP_led_set(){
    /* Turn on LED on PA6 */
    GPIOA->ODR |= (1U << 6);
}
void BSP_led_clear(){
    /* Turn on LED on PA6 */
    GPIOA->BRR |= (1U << 6);
}
```

```c
void BSP_button_interrupt_init(void){
    /* Enable GPIOB clock */
    RCC->IOPENR |= (1U << 1);

    /* Setup PB3 as input */
    GPIOB->MODER &= ~(3U << 2*3);

    /* rising edge trigger selection, selection register
and mask register */
    EXTI->RTSR1 |= (1U << 3);
    EXTI->EXTICR[0] |= (1U << 8*3);
    EXTI->IMR1 |= (1U << 3);

    /* enable NVIC and set interrupt priority */
    NVIC_SetPriority(EXTI2_3_IRQn, 0);
    NVIC_EnableIRQ(EXTI2_3_IRQn);

}

void BSP_clearSSD(void){

    GPIOB ->ODR |= (1U << 0);  //PB0 A
    GPIOB ->ODR |= (1U << 1);  //PB1 B
    GPIOB ->ODR |= (1U << 2);  //PB2 C
    GPIOB ->ODR |= (1U << 3);  //PB3 D
    GPIOB ->ODR |= (1U << 4);  //PB4 E
    GPIOB ->ODR |= (1U << 5);  //PB5 F
    GPIOB ->ODR |= (1U << 6);  //PB6 G
}
void BSP_setSSD(int x){
    BSP_clearSSD();
    switch(x){
        case 0:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 4); //PB4 E
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            break;
```

```
        case 1:
             GPIOB->ODR &= ~(1U << 1); //PB1 B
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             break;

        case 2:
             GPIOB->ODR &= ~(1U << 0);  //PB0 A
             GPIOB->ODR &= ~(1U << 1);  //PB1 B
             GPIOB->ODR &= ~(1U << 3);  //PB3 D
             GPIOB->ODR &= ~(1U << 4); //PB4 E
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;

        case 3:
             GPIOB->ODR &= ~(1U << 0); //PB0 A
             GPIOB->ODR &= ~(1U << 1); //PB1 B
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             GPIOB->ODR &= ~(1U << 3); //PB3 D
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;

        case 4:
             GPIOB->ODR &= ~(1U << 1); //PB1 B
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             GPIOB->ODR &= ~(1U << 5); //PB5 F
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;

        case 5:
             GPIOB->ODR &= ~(1U << 0); //PB0 A
             GPIOB->ODR &= ~(1U << 2); //PB2 C
             GPIOB->ODR &= ~(1U << 3); //PB3 D
             GPIOB->ODR &= ~(1U << 5); //PB5 F
             GPIOB->ODR &= ~(1U << 6);  //PB6 G
             break;
```

```
        case 6:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 4); //PB4 E
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            GPIOB->ODR &= ~(1U << 6);  //PB6 G
            break;

        case 7:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            break;

        case 8:
            GPIOB->ODR &= ~(1U << 0); //PA0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 4); //PB4 E
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            GPIOB->ODR &= ~(1U << 6);  //PB6 G
            break;

        case 9:
            GPIOB->ODR &= ~(1U << 0); //PB0 A
            GPIOB->ODR &= ~(1U << 1); //PB1 B
            GPIOB->ODR &= ~(1U << 2); //PB2 C
            GPIOB->ODR &= ~(1U << 3); //PB3 D
            GPIOB->ODR &= ~(1U << 5); //PB5 F
            GPIOB->ODR &= ~(1U << 6);  //PB6 G
            break;
    }
}
```