HACETTEPE UNIVERSITY COMPUTER ENGINEERING DEPARTMENT

Group Name:	Grup
Student Information:	Merve Müge Deliktaş - 21526896
	Onur Cankur – 21526791
Course Information:	BBM 434 - Embedded Systems Laboratory
	Spring – 2019
Report Information:	Lab-04 Experiment Report

Short Brief of Lab-04 and Function Explanations

In this lab, we implemented Approach Lighting System that contains LED lights for airports. It is important to make a safe take off. LEDs flashes from green, through yellow, to red at the beginning. However, when the wind direction changes, switch is pressed by the operator and LEDs are flashed from green, through yellow, to red.

```
//*****************************************************************************
// define GPIO registers (PORTE)
// define SYSCTL_RCGC2_R
// define variables for SysTick and interrupt
//*****************************************************************************
#define GPIO_PORTE_DATA_BITS_R  ((volatile unsigned long *)0x40024000)
#define GPIO_PORTE_DATA_R       (*((volatile unsigned long *)0x400243FC))
#define GPIO_PORTE_DIR_R        (*((volatile unsigned long *)0x40024400))
#define GPIO_PORTE_IS_R         (*((volatile unsigned long *)0x40024404))
#define GPIO_PORTE_IBE_R        (*((volatile unsigned long *)0x40024408))
#define GPIO_PORTE_IEV_R        (*((volatile unsigned long *)0x4002440C))
#define GPIO_PORTE_IM_R         (*((volatile unsigned long *)0x40024410))
#define GPIO_PORTE_RIS_R        (*((volatile unsigned long *)0x40024414))
#define GPIO_PORTE_MIS_R        (*((volatile unsigned long *)0x40024418))
#define GPIO_PORTE_ICR_R        (*((volatile unsigned long *)0x4002441C))
#define GPIO_PORTE_AFSEL_R      (*((volatile unsigned long *)0x40024420))
#define GPIO_PORTE_DEN_R        (*((volatile unsigned long *)0x4002451C))
#define GPIO_PORTE_LOCK_R       (*((volatile unsigned long *)0x40024520))
#define GPIO_PORTE_CR_R         (*((volatile unsigned long *)0x40024524))
#define GPIO_PORTE_AMSEL_R      (*((volatile unsigned long *)0x40024528))
#define GPIO_PORTE_PCTL_R       (*((volatile unsigned long *)0x4002452C))
#define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
#define NVIC_ST_CTRL_R          (*((volatile unsigned long *)0xE000E010))
#define NVIC_ST_RELOAD_R        (*((volatile unsigned long *)0xE000E014))
#define NVIC_ST_CURRENT_R       (*((volatile unsigned long *)0xE000E018))
#define NVIC_SYS_PRI3_R         (*((volatile unsigned long *)0xE000ED20))
#define NVIC_EN0_R              (*((volatile unsigned long *)0xE000E100))
#define NVIC_PRI1_R             (*((volatile unsigned long *)0xE000E41C))
```

Figure 1-defines

Figure 1 shows how we defined variables for Port E, SYSCTL_RCGC2_R and SysTick and interrupt.

```
//Global variables
unsigned long SW;
unsigned long color_arr[] = {0x02, 0x04, 0x08};
int pressed = 0;
int indx = -1;

//Function prototypes
void PortE_Init(void);
void SysTick_Init(unsigned long period);
void GPIOPortE_Handler(void);
void SysTick_Handler(unsigned int delay);
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
void WaitForInterrupt(void);  // low power mode
```

Figure 2-global variables and function prototypes

In Figure 2, you can see global variables and function prototypes that we used for our implementation.

```
int main(void){
  DisableInterrupts();
  PortE_Init();
  SysTick_Init(8000000);
  EnableInterrupts();   // Enable global Interrupt flag
  while(1){
    WaitForInterrupt();
  }
}
```

Figure 3-main function

Main function implementation is shown in Figure 3. First, we disabled interrupts before initializations. After all initializations are done, interrupts enabled and in an infinite while loop, it waits for an interrupt to occur.

```
void PortE_Init(void){ volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x00000010;          // activate clock for Port E
  delay = SYSCTL_RCGC2_R;                // allow time for clock to start
  GPIO_PORTE_LOCK_R = 0x4C4F434B;        // unlock GPIO Port E
  GPIO_PORTE_CR_R = 0x0E;                // allow changes to PE1-3
  GPIO_PORTE_AMSEL_R = 0x00;             // disable analog on PE
  GPIO_PORTE_PCTL_R &= ~0x0000FFFF;      // configure PE3-0 as GPIO
  GPIO_PORTE_DIR_R = 0x0E;               // PE0 in, PF3-1 out
  GPIO_PORTE_AFSEL_R = 0x00;             // disable alt funct on PE7-0
  GPIO_PORTE_DEN_R = 0x1F;               // enable digital I/O on PE4-0
  GPIO_PORTE_IS_R &= ~0x01;              // PE0 is edge-sensitive
  GPIO_PORTE_IBE_R &= ~0x01;             // PE0 is not both edges
  GPIO_PORTE_IEV_R &= ~0x01;             // PE0 falling edge event
  GPIO_PORTE_ICR_R = 0x01;               // clear flag0
  GPIO_PORTE_IM_R |= 0x01;               // arm interrupt on PE0
  NVIC_PRI1_R = (NVIC_PRI1_R&0xFFFFFF00)|0x00000020; // priority 1
  NVIC_EN0_R = 0x00000010;               // enable interrupt 30 in NVIC
}
```

Figure 4-PortE initialization

Like the last lab we did, we used port E again. From the Figure 4, you can see initializations. This time, we add interrupt initializations and we set priority of switch interrupt as 1. We used falling edge for our interrupt.

```
void SysTick_Init(unsigned long period){
  NVIC_ST_CTRL_R = 0;               // disable SysTick during setup
  NVIC_ST_RELOAD_R = period - 1;    // reload value
  NVIC_ST_CURRENT_R = 0;            // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;
  NVIC_ST_CTRL_R = 0x00000007;
}
```

Figure 5-SysTick initialization

From the figure above, SysTick initialization can be seen. We set its priority as 2 and interrupt bit is set.

```
void GPIOPortE_Handler(void){    // buton kontrol
  if(GPIO_PORTE_RIS_R & 0x01){   // PE0 touch
    GPIO_PORTE_ICR_R = 0x01;     // acknowledge flag0
    pressed = !pressed;          // pressed is toggled
  }
}
```

Figure 6-GPIOPortE_Handler()

Pressed variable is toggled when switch is pressed as you can see from the Figure 6.

```
void SysTick_Handler(unsigned int delay){
  if(!pressed){
    indx = (indx + 1) % 3;
  }
  else {
    indx = ((indx - 1) + 3) % 3;
  }
  GPIO_PORTE_DATA_R = color_arr[indx];
}
```

Figure 7-SysTick_Handler()

In the SysTick_Handler() function which can be seen from Figure 7, LED outputs are changing. If switch is not pressed, LEDs go from green, to yellow, to red and if switch is not pressed, it goes backwards. color_arr is an array that stores LED output pin values.

Rest of this report will be like the last report that we wrote for Lab 3.

PART A

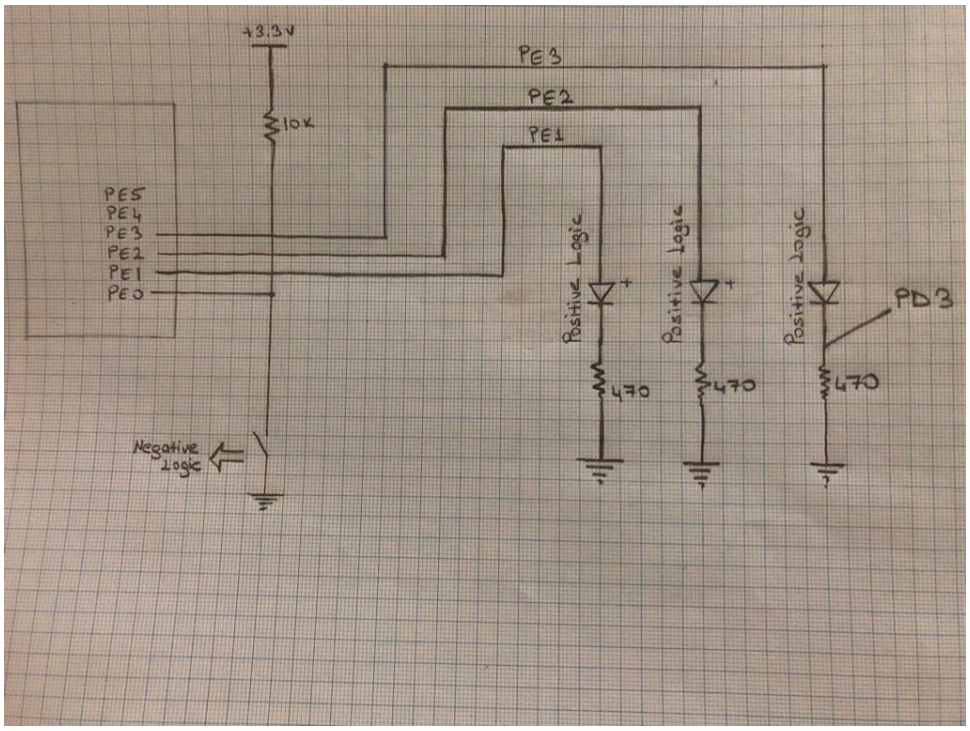

Figure 8-circuit

As we explained in lab 3, we used negative logic for switch and positive logic for LEDs. In addition to that, falling edge interrupt is used for the switch.

PART B

In this part, we wrote the software for our circuit. As we explained briefly before, Port E is chosen and PE0 is used as input port and PE1, PE2, PE3 are used as output ports.

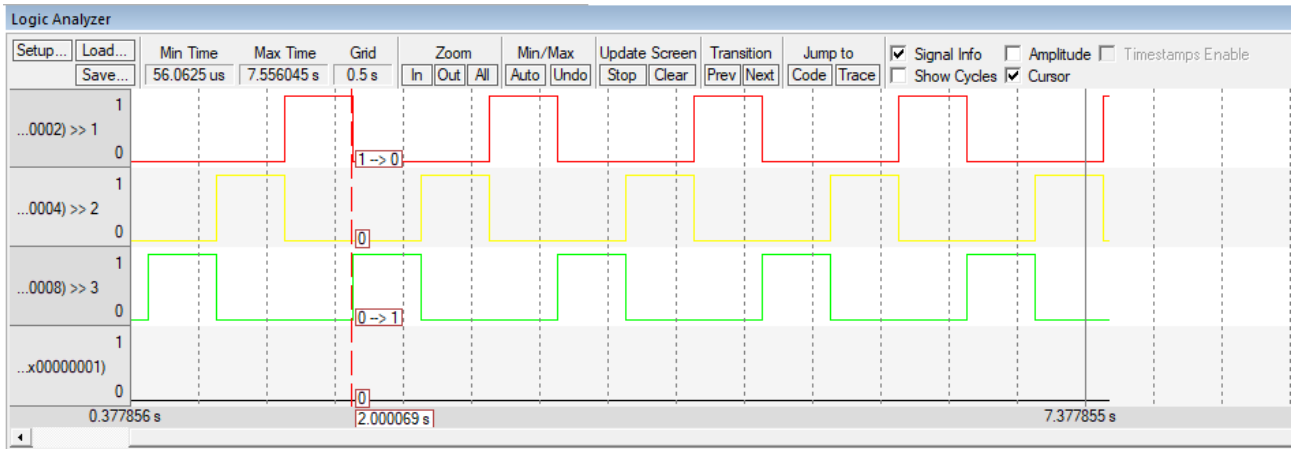

Figure 9-Logic Analizer

You can see from Figure 9, finishing a sequence lasts approximately 2 seconds. It means that each of our LEDs are interrupted every 0.66 seconds. Because of using negative logic on switch, the signal that you see at the bottom is 0 when the switch is pressed and 1 when the switch is not pressed. In this example, it is pressed.

PART C

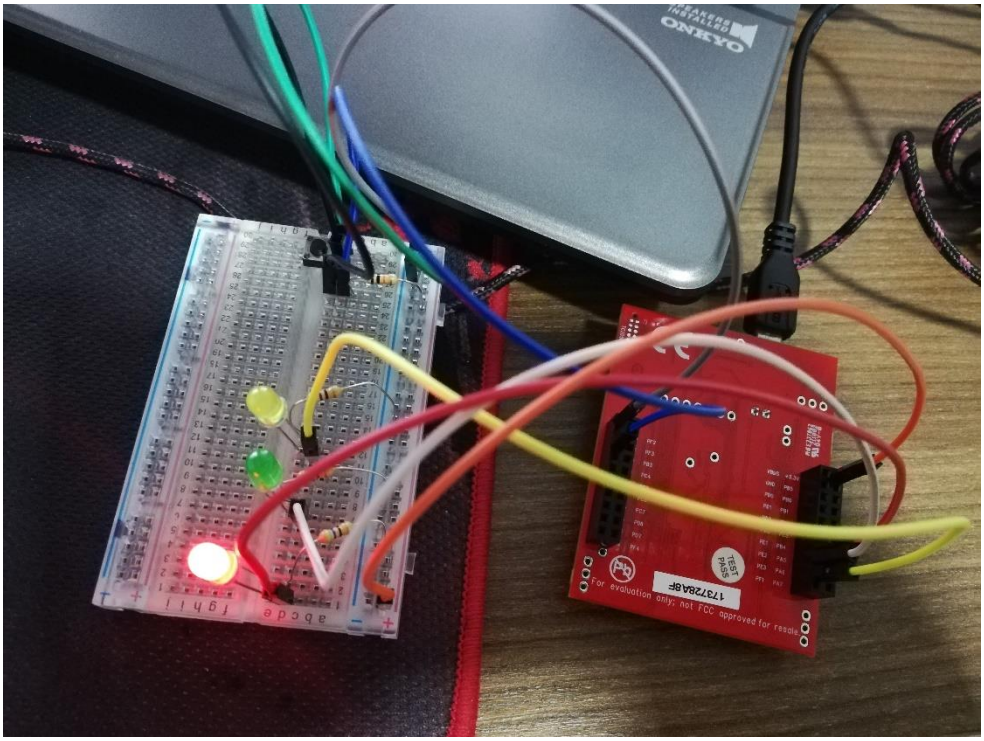

Figure 10-red LED on board

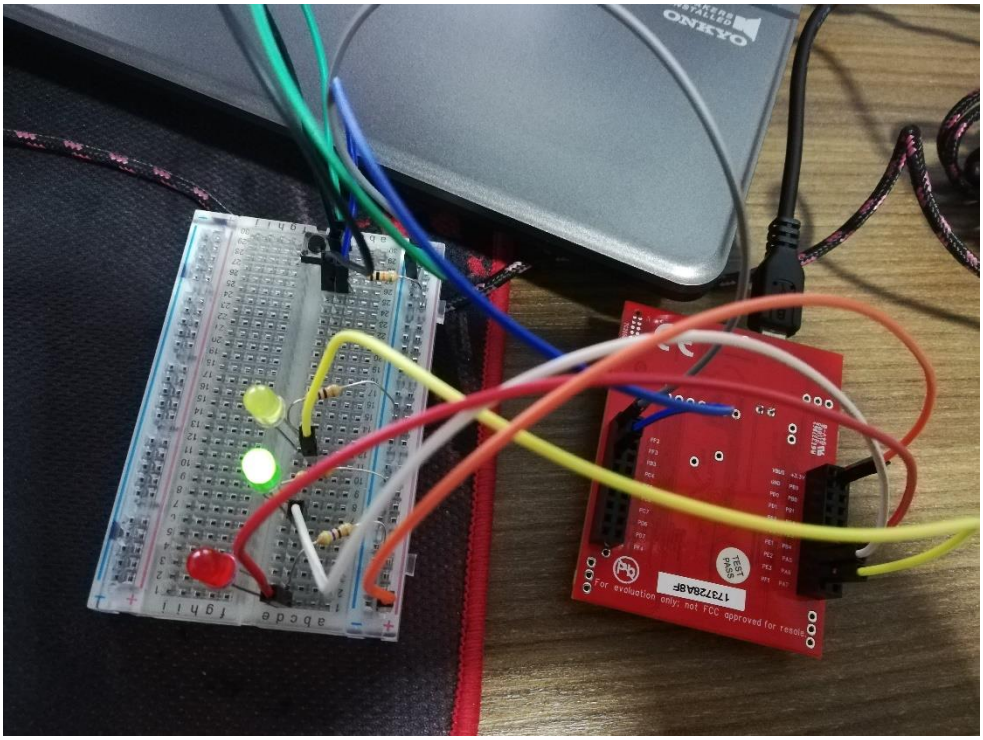

Figure 11-green LED on board

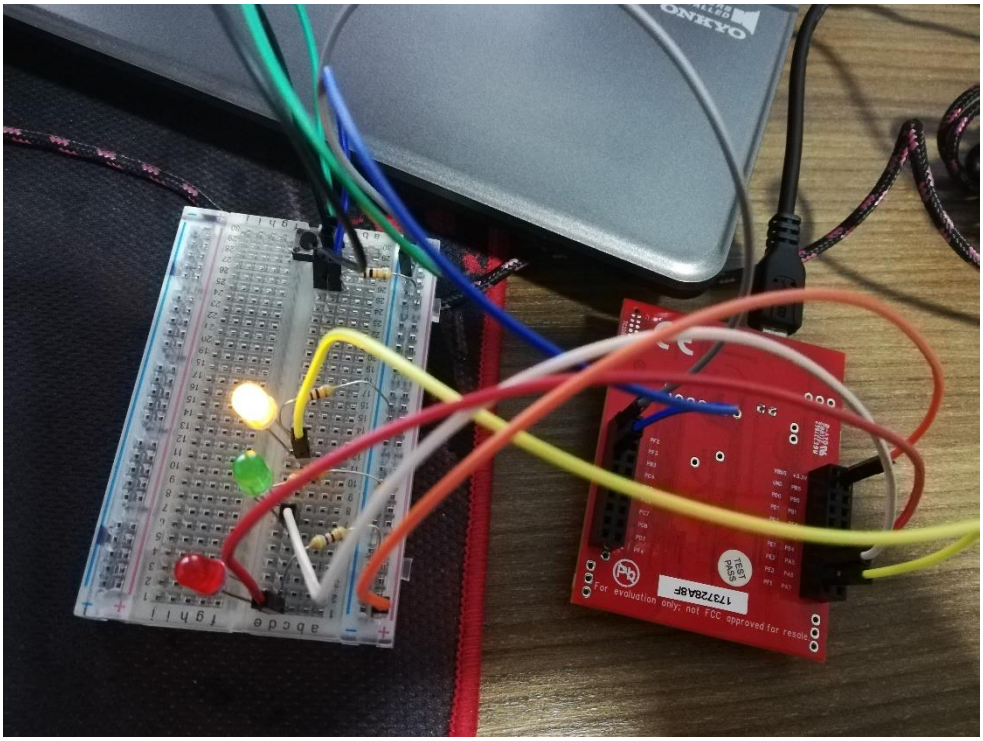

Figure 12-yellow LED on board

After debugged our software on debugger, we built the hardware on the real board. You can see our hardware in Figure 10, Figure 11, and Figure 12. We built it according to the circuit that we showed in Part A. We used 10kΩ resistor for the switch and 470Ω resistors for each LED. PE0 pin is used for the switch and if it is pressed signal on the port is 0, otherwise 1. PE3 pin used for red LED, PE2 is for yellow LED and PE1 is for green.

PART D

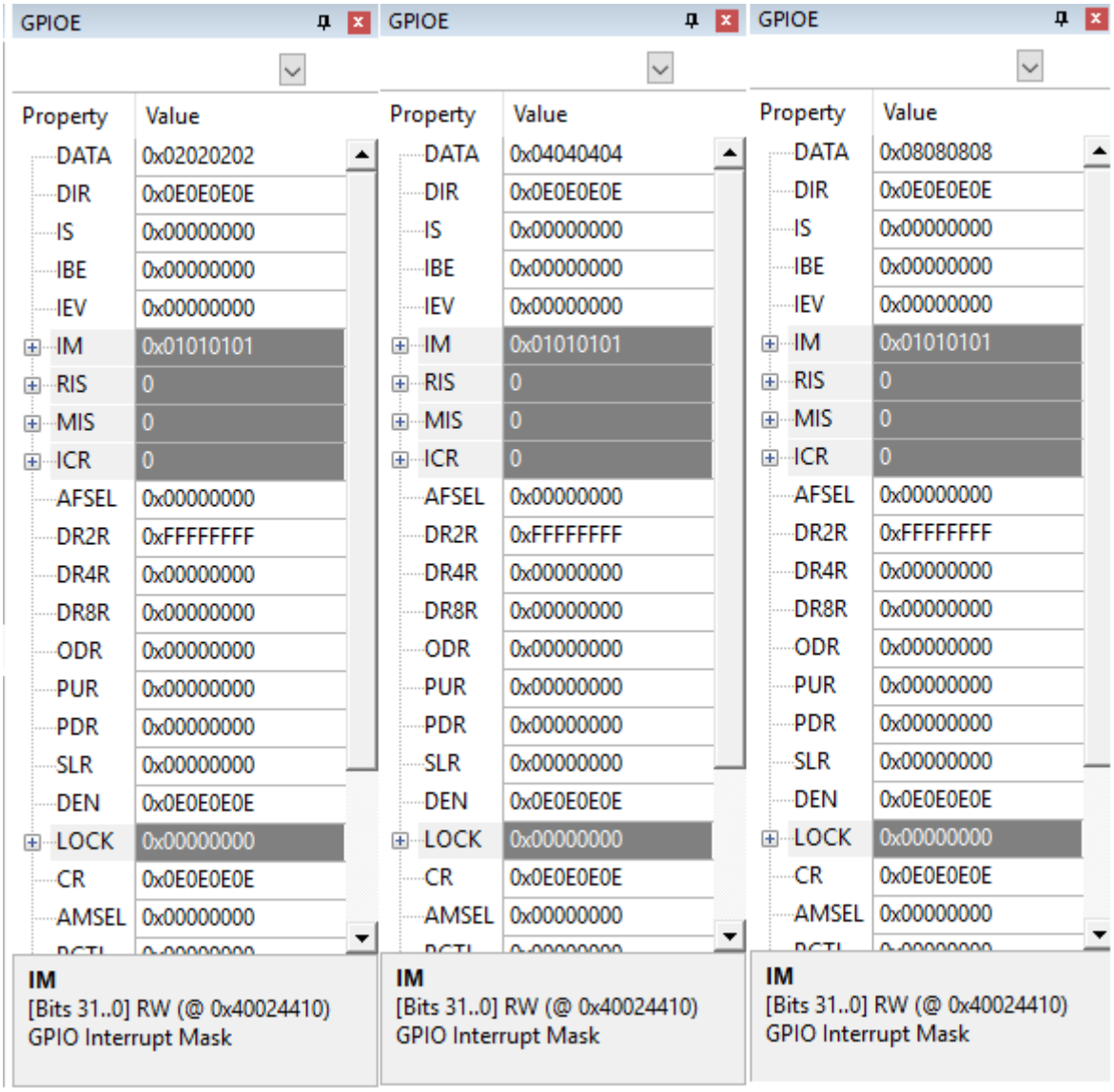

Figure 13-results on debugger

In this part, we debugged our combined hardware/software system on actual board. From Peripherals->SystemViewer->GPIO->GPIOE, we opened the debugger. As you can see from the Figure 13, values of DIR, DEN and CR is 0x0E because we assigned them like that in our PortE_Inıt() function which can be seen in Figure 4. In addition, you can see that DATA is changing from 0x02, to 0x04 and 0x08. This time, you can see that IM is 0x01 because we used interrupts.