EE308

MICROPROCESSORS LABORATORY

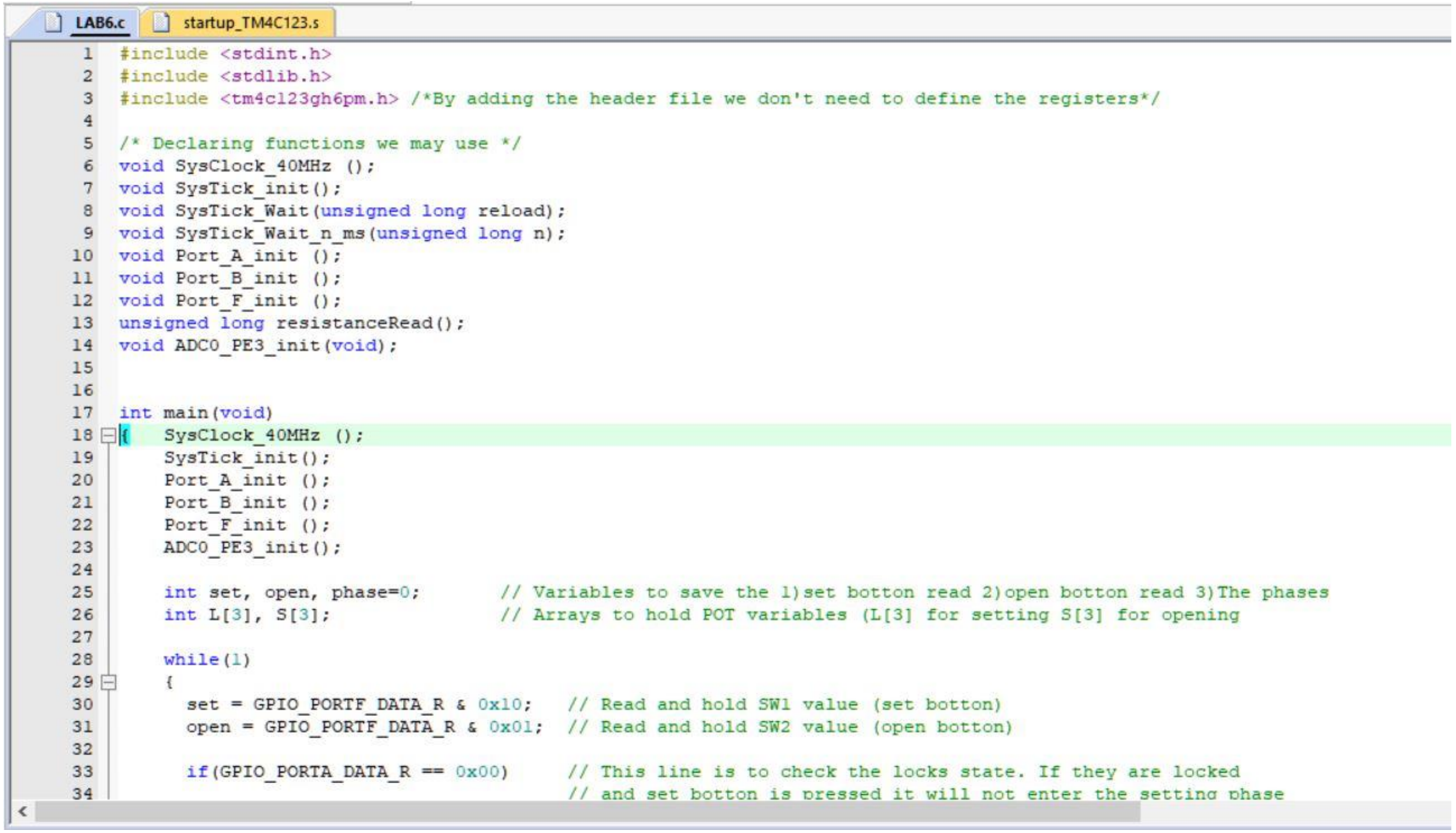
LAB -6-

Safe Lock

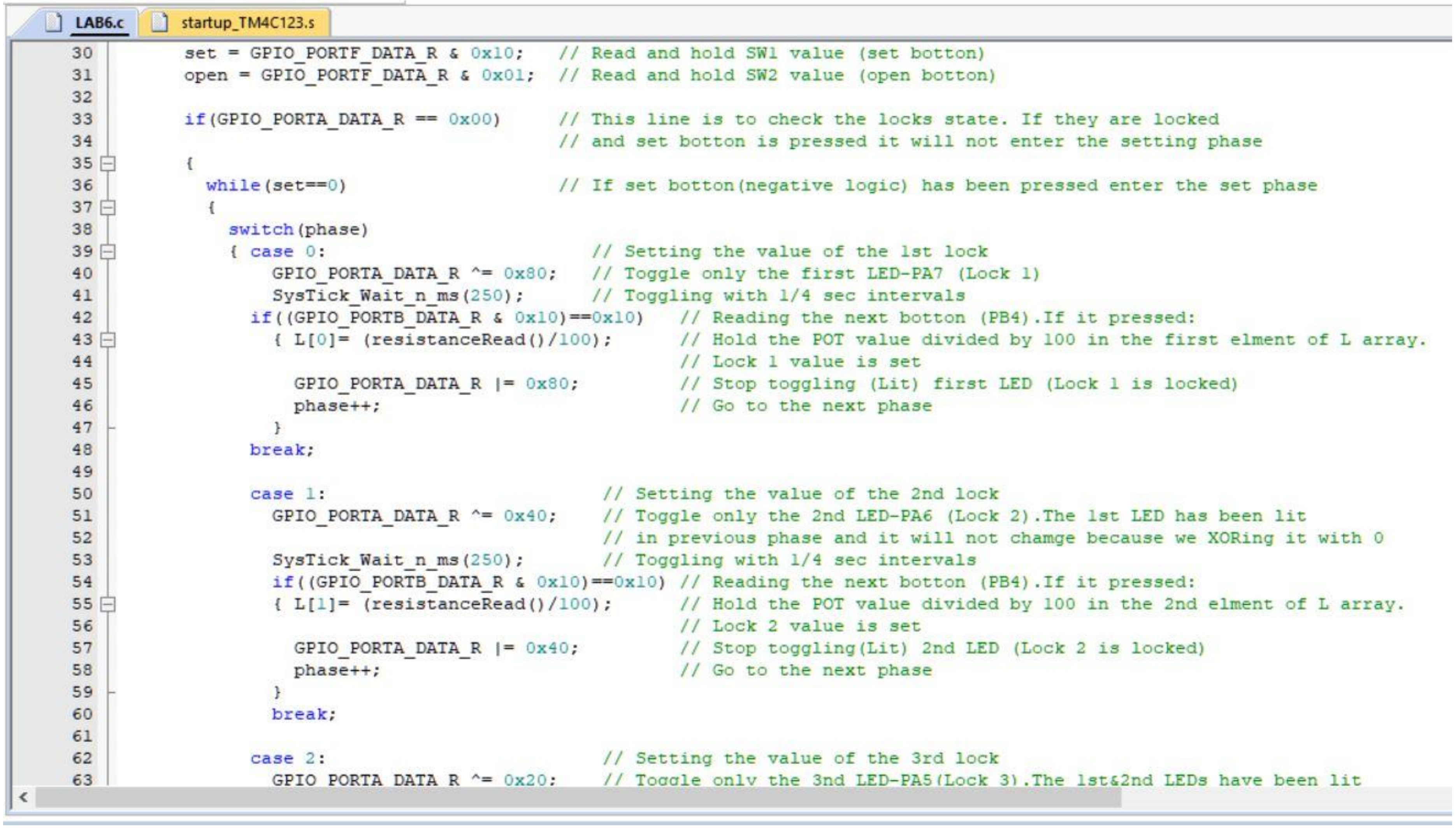
ZAID ALBU-AZAM
17050241001

MERVE DELİBAŞ
17050211032

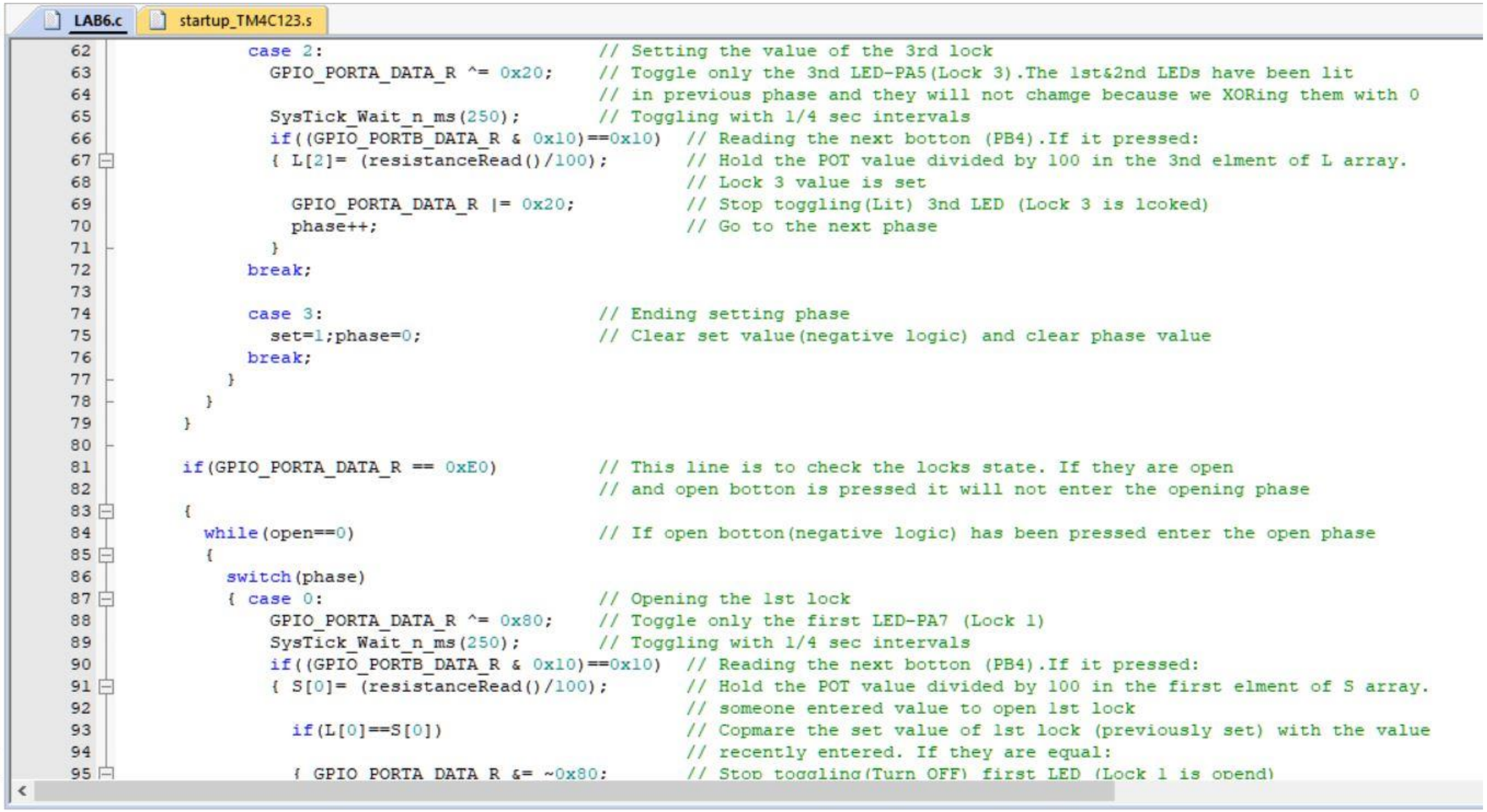
The main body



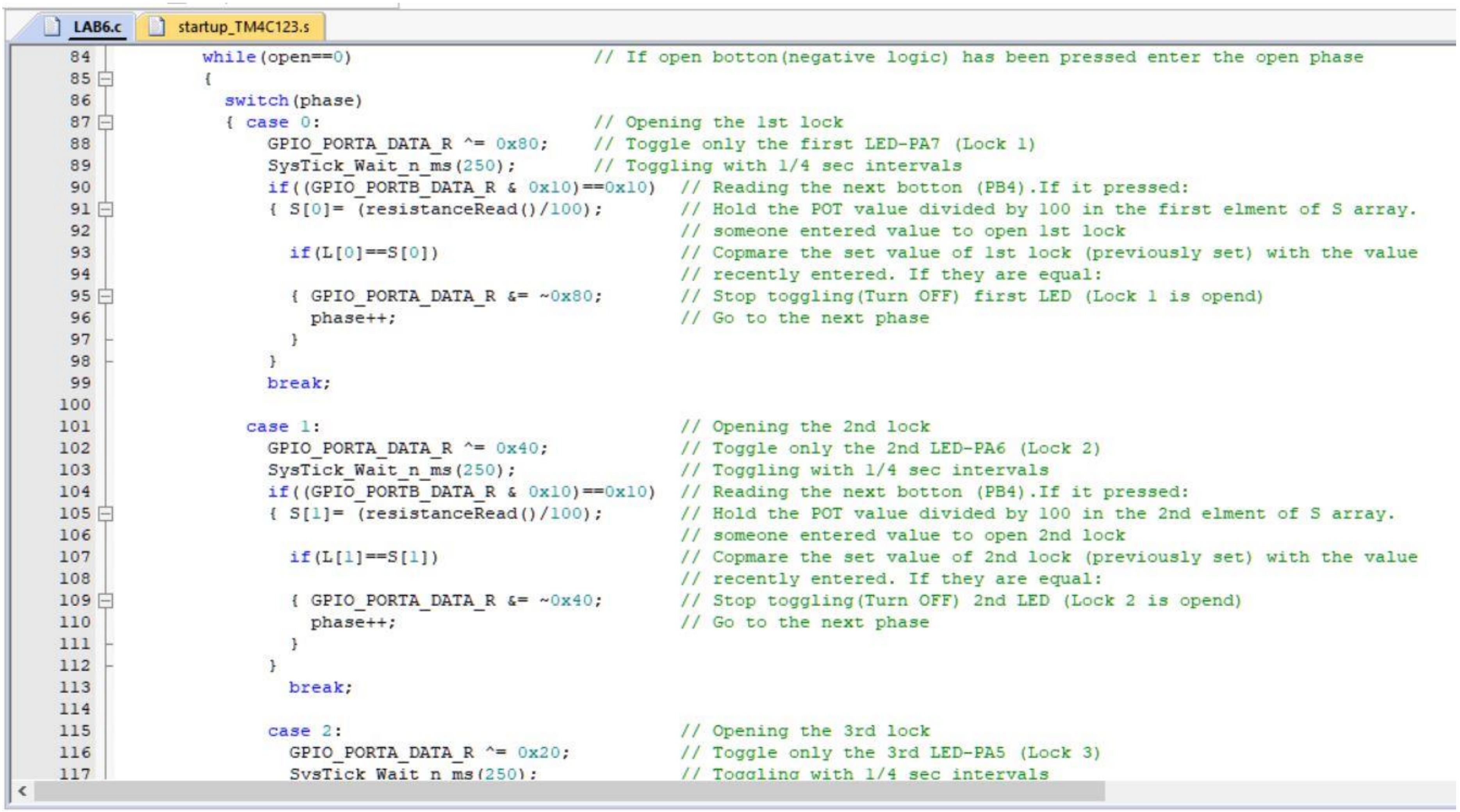
```c
#include <stdint.h>
#include <stdlib.h>
#include <tm4c123gh6pm.h> /*By adding the header file we don't need to define the registers*/

/* Declaring functions we may use */
void SysClock_40MHz ();
void SysTick_init();
void SysTick_Wait(unsigned long reload);
void SysTick_Wait_n_ms(unsigned long n);
void Port_A_init ();
void Port_B_init ();
void Port_F_init ();
unsigned long resistanceRead();
void ADC0_PE3_init(void);


int main(void)
{    SysClock_40MHz ();
     SysTick_init();
     Port_A_init ();
     Port_B_init ();
     Port_F_init ();
     ADC0_PE3_init();

     int set, open, phase=0;       // Variables to save the 1)set botton read 2)open botton read 3)The phases
     int L[3], S[3];               // Arrays to hold POT variables (L[3] for setting S[3] for opening

     while(1)
     {
       set = GPIO_PORTF_DATA_R & 0x10;   // Read and hold SW1 value (set botton)
       open = GPIO_PORTF_DATA_R & 0x01;  // Read and hold SW2 value (open botton)

       if(GPIO_PORTA_DATA_R == 0x00)     // This line is to check the locks state. If they are locked
                                         // and set botton is pressed it will not enter the setting phase
```



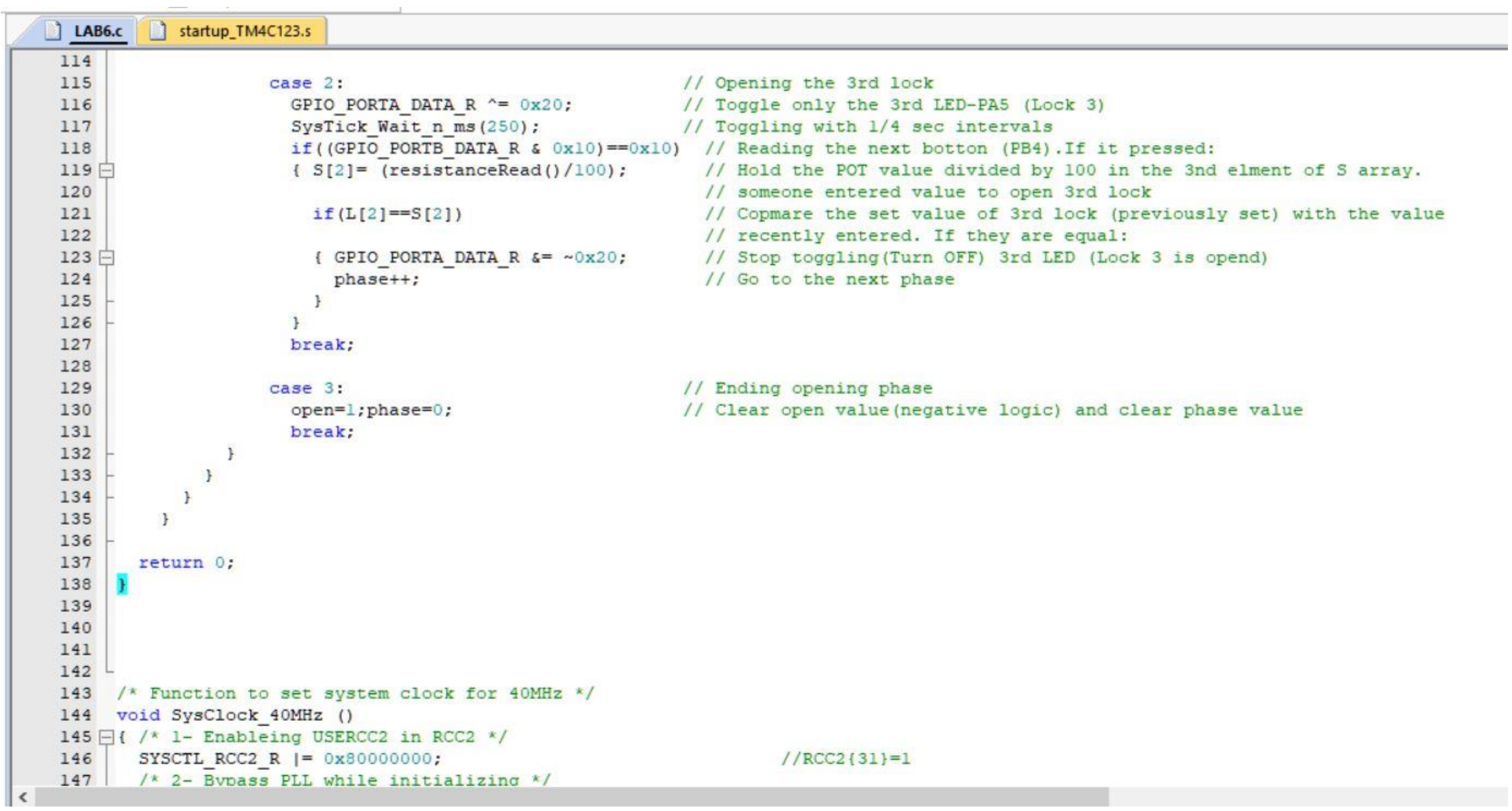
```c
set = GPIO_PORTF_DATA_R & 0x10;   // Read and hold SW1 value (set botton)
open = GPIO_PORTF_DATA_R & 0x01;  // Read and hold SW2 value (open botton)

if(GPIO_PORTA_DATA_R == 0x00)     // This line is to check the locks state. If they are locked
                                  // and set botton is pressed it will not enter the setting phase
{
  while(set==0)                   // If set botton(negative logic) has been pressed enter the set phase
  {
    switch(phase)
    { case 0:                        // Setting the value of the 1st lock
        GPIO_PORTA_DATA_R ^= 0x80;   // Toggle only the first LED-PA7 (Lock 1)
        SysTick_Wait_n_ms(250);      // Toggling with 1/4 sec intervals
      if((GPIO_PORTB_DATA_R & 0x10)==0x10)   // Reading the next botton (PB4).If it pressed:
        { L[0]= (resistanceRead()/100);      // Hold the POT value divided by 100 in the first elment of L array.
                                             // Lock 1 value is set
          GPIO_PORTA_DATA_R |= 0x80;         // Stop toggling (Lit) first LED (Lock 1 is locked)
          phase++;                           // Go to the next phase
        }
      break;

      case 1:                         // Setting the value of the 2nd lock
        GPIO_PORTA_DATA_R ^= 0x40;    // Toggle only the 2nd LED-PA6 (Lock 2).The 1st LED has been lit
                                      // in previous phase and it will not chamge because we XORing it with 0
        SysTick_Wait_n_ms(250);       // Toggling with 1/4 sec intervals
        if((GPIO_PORTB_DATA_R & 0x10)==0x10) // Reading the next botton (PB4).If it pressed:
        { L[1]= (resistanceRead()/100);      // Hold the POT value divided by 100 in the 2nd elment of L array.
                                             // Lock 2 value is set
          GPIO_PORTA_DATA_R |= 0x40;         // Stop toggling(Lit) 2nd LED (Lock 2 is locked)
          phase++;                           // Go to the next phase
        }
        break;

      case 2:                         // Setting the value of the 3rd lock
        GPIO_PORTA_DATA_R ^= 0x20;    // Toggle only the 3nd LED-PA5(Lock 3).The 1st&2nd LEDs have been lit
```



```c
      case 2:               // Setting the value of the 3rd lock
       GPIO_PORTA_DATA_R ^= 0x20;    // Toggle only the 3nd LED-PA5(Lock 3).The 1st&2nd LEDs have been lit
                            // in previous phase and they will not chamge because we XORing them with 0
       SysTick_Wait_n_ms(250);       // Toggling with 1/4 sec intervals
       if((GPIO_PORTB_DATA_R & 0x10)==0x10)  // Reading the next botton (PB4).If it pressed:
       { L[2]= (resistanceRead()/100);        // Hold the POT value divided by 100 in the 3nd elment of L array.
                                              // Lock 3 value is set
        GPIO_PORTA_DATA_R |= 0x20;            // Stop toggling(Lit) 3nd LED (Lock 3 is lcoked)
        phase++;                              // Go to the next phase
       }
      break;

      case 3:               // Ending setting phase
       set=1;phase=0;       // Clear set value(negative logic) and clear phase value
      break;
     }
    }
   }

   if(GPIO_PORTA_DATA_R == 0xE0)      // This line is to check the locks state. If they are open
                            // and open botton is pressed it will not enter the opening phase
   {
    while(open==0)          // If open botton(negative logic) has been pressed enter the open phase
    {
     switch(phase)
     { case 0:              // Opening the 1st lock
       GPIO_PORTA_DATA_R ^= 0x80;    // Toggle only the first LED-PA7 (Lock 1)
       SysTick_Wait_n_ms(250);       // Toggling with 1/4 sec intervals
       if((GPIO_PORTB_DATA_R & 0x10)==0x10)  // Reading the next botton (PB4).If it pressed:
       { S[0]= (resistanceRead()/100);        // Hold the POT value divided by 100 in the first elment of S array.
                                              // someone entered value to open 1st lock
        if(L[0]==S[0])                        // Copmare the set value of 1st lock (previously set) with the value
                                              // recently entered. If they are equal:
        { GPIO_PORTA_DATA_R &= ~0x80;         // Stop toggling(Turn OFF) first LED (Lock 1 is opend)
```



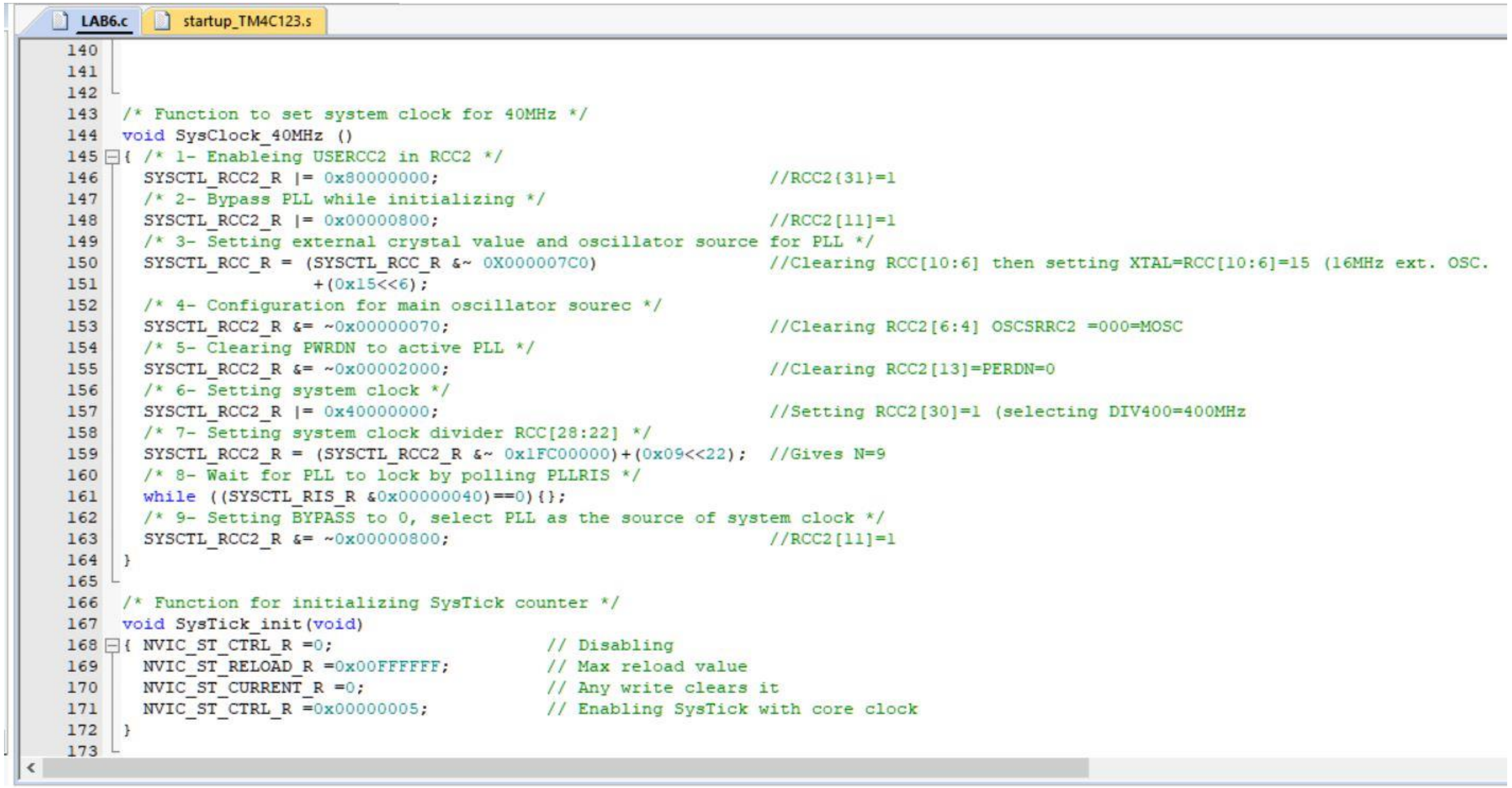
```c
        while(open==0)                       // If open botton(negative logic) has been pressed enter the open phase
        {
          switch(phase)
          { case 0:                          // Opening the 1st lock
              GPIO_PORTA_DATA_R ^= 0x80;     // Toggle only the first LED-PA7 (Lock 1)
              SysTick_Wait_n_ms(250);        // Toggling with 1/4 sec intervals
              if((GPIO_PORTB_DATA_R & 0x10)==0x10)   // Reading the next botton (PB4).If it pressed:
              { S[0]= (resistanceRead()/100);        // Hold the POT value divided by 100 in the first elment of S array.
                                                     // someone entered value to open 1st lock
                if(L[0]==S[0])                       // Copmare the set value of 1st lock (previously set) with the value
                                                     // recently entered. If they are equal:
                { GPIO_PORTA_DATA_R &= ~0x80;        // Stop toggling(Turn OFF) first LED (Lock 1 is opend)
                  phase++;                           // Go to the next phase
                }
              }
              break;

            case 1:                                  // Opening the 2nd lock
              GPIO_PORTA_DATA_R ^= 0x40;             // Toggle only the 2nd LED-PA6 (Lock 2)
              SysTick_Wait_n_ms(250);                // Toggling with 1/4 sec intervals
              if((GPIO_PORTB_DATA_R & 0x10)==0x10)   // Reading the next botton (PB4).If it pressed:
              { S[1]= (resistanceRead()/100);        // Hold the POT value divided by 100 in the 2nd elment of S array.
                                                     // someone entered value to open 2nd lock
                if(L[1]==S[1])                       // Copmare the set value of 2nd lock (previously set) with the value
                                                     // recently entered. If they are equal:
                { GPIO_PORTA_DATA_R &= ~0x40;        // Stop toggling(Turn OFF) 2nd LED (Lock 2 is opend)
                  phase++;                           // Go to the next phase
                }
              }
                break;

              case 2:                                // Opening the 3rd lock
                GPIO_PORTA_DATA_R ^= 0x20;           // Toggle only the 3rd LED-PA5 (Lock 3)
                SysTick_Wait_n_ms(250);              // Toggling with 1/4 sec intervals
```



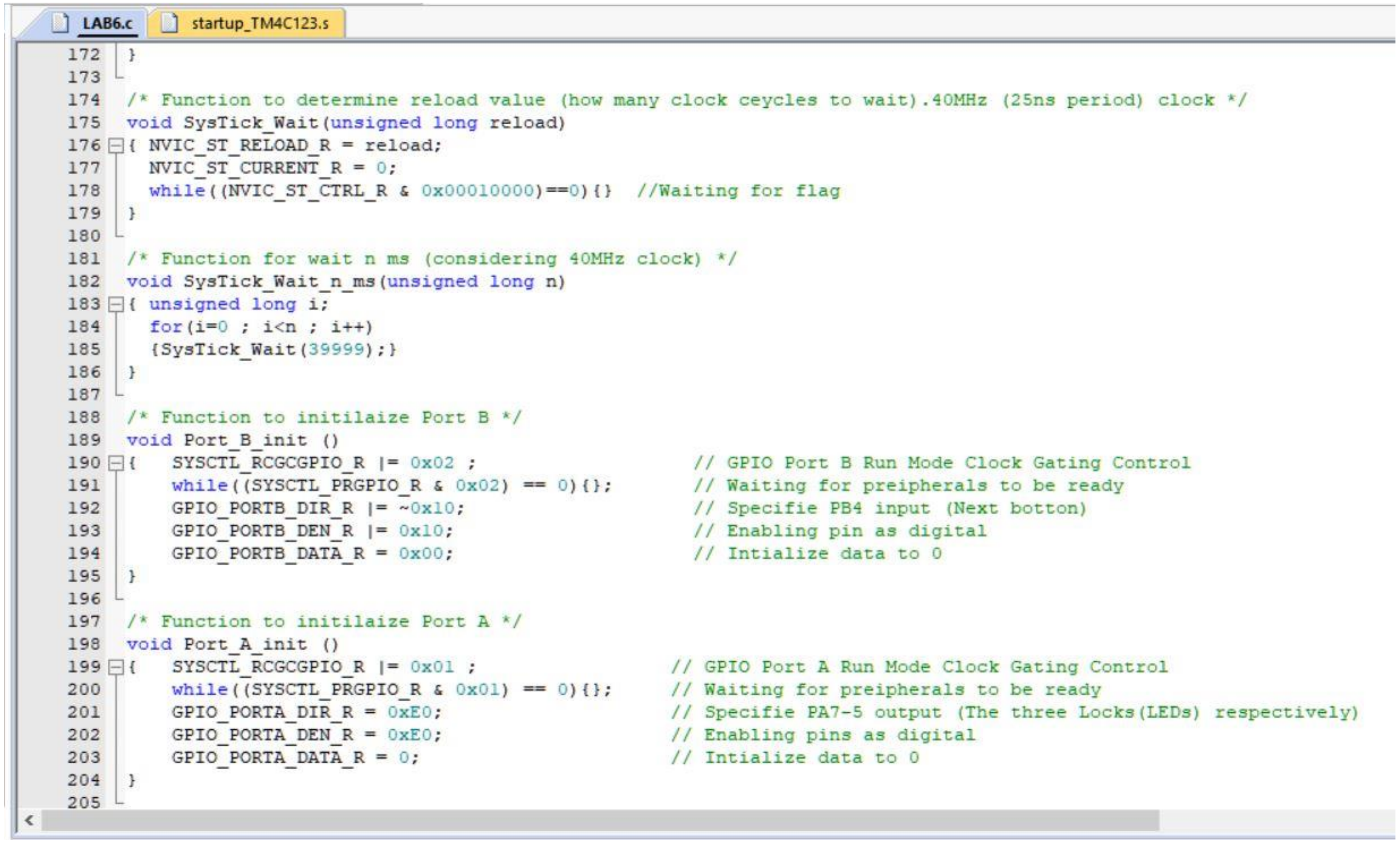
LAB6.c | startup_TM4C123.s

```

                  case 2:                                   // Opening the 3rd lock
                    GPIO_PORTA_DATA_R ^= 0x20;              // Toggle only the 3rd LED-PA5 (Lock 3)
                    SysTick_Wait_n_ms(250);                 // Toggling with 1/4 sec intervals
                    if((GPIO_PORTB_DATA_R & 0x10)==0x10)      // Reading the next botton (PB4).If it pressed:
                    { S[2]= (resistanceRead()/100);           // Hold the POT value divided by 100 in the 3nd elment of S array.
                                                              // someone entered value to open 3rd lock
                      if(L[2]==S[2])                          // Copmare the set value of 3rd lock (previously set) with the value
                                                              // recently entered. If they are equal:
                      { GPIO_PORTA_DATA_R &= ~0x20;           // Stop toggling(Turn OFF) 3rd LED (Lock 3 is opend)
                        phase++;                              // Go to the next phase
                      }
                    }
                    break;

                  case 3:                                   // Ending opening phase
                    open=1;phase=0;                         // Clear open value(negative logic) and clear phase value
                    break;
                }
              }
            }
          }

  return 0;
}




/* Function to set system clock for 40MHz */
void SysClock_40MHz ()
{ /* 1- Enableing USERCC2 in RCC2 */
  SYSCTL_RCC2_R |= 0x80000000;                                    //RCC2{31}=1
  /* 2- Bypass PLL while initializing */
```

The functions we used

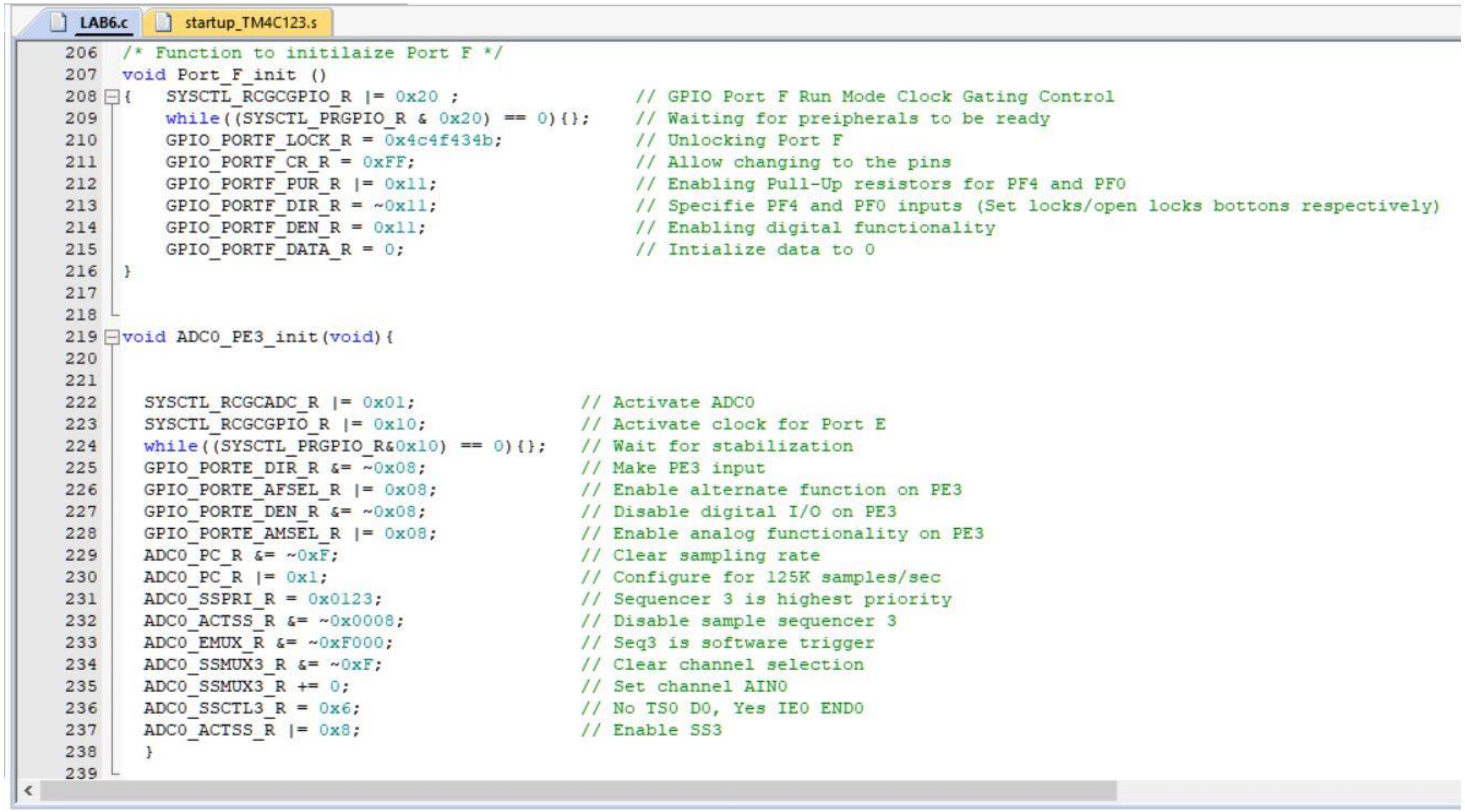


```c


/* Function to set system clock for 40MHz */
void SysClock_40MHz ()
{ /* 1- Enableing USERCC2 in RCC2 */
  SYSCTL_RCC2_R |= 0x80000000;                              //RCC2{31}=1
  /* 2- Bypass PLL while initializing */
  SYSCTL_RCC2_R |= 0x00000800;                              //RCC2[11]=1
  /* 3- Setting external crystal value and oscillator source for PLL */
  SYSCTL_RCC_R = (SYSCTL_RCC_R &~ 0X000007C0)               //Clearing RCC[10:6] then setting XTAL=RCC[10:6]=15 (16MHz ext. OSC.
                +(0x15<<6);
  /* 4- Configuration for main oscillator sourec */
  SYSCTL_RCC2_R &= ~0x00000070;                             //Clearing RCC2[6:4] OSCSRRC2 =000=MOSC
  /* 5- Clearing PWRDN to active PLL */
  SYSCTL_RCC2_R &= ~0x00002000;                             //Clearing RCC2[13]=PERDN=0
  /* 6- Setting system clock */
  SYSCTL_RCC2_R |= 0x40000000;                              //Setting RCC2[30]=1 (selecting DIV400=400MHz
  /* 7- Setting system clock divider RCC[28:22] */
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R &~ 0x1FC00000)+(0x09<<22);  //Gives N=9
  /* 8- Wait for PLL to lock by polling PLLRIS */
  while ((SYSCTL_RIS_R &0x00000040)==0){};
  /* 9- Setting BYPASS to 0, select PLL as the source of system clock */
  SYSCTL_RCC2_R &= ~0x00000800;                             //RCC2[11]=1
}

/* Function for initializing SysTick counter */
void SysTick_init(void)
{ NVIC_ST_CTRL_R =0;                    // Disabling
  NVIC_ST_RELOAD_R =0x00FFFFFF;         // Max reload value
  NVIC_ST_CURRENT_R =0;                 // Any write clears it
  NVIC_ST_CTRL_R =0x00000005;           // Enabling SysTick with core clock
}

```

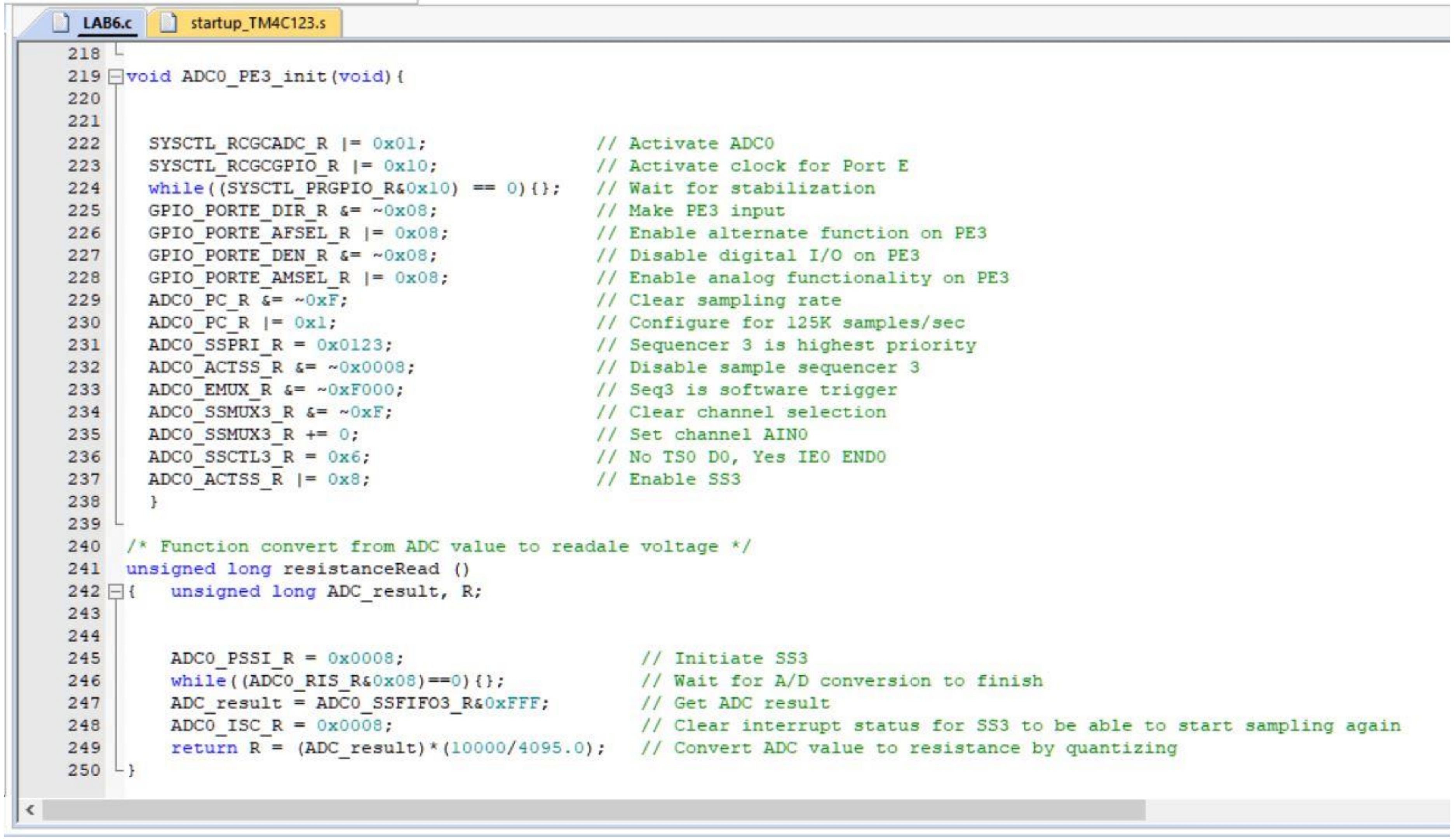


LAB6.c   startup_TM4C123.s

```c
}

/* Function to determine reload value (how many clock ceycles to wait).40MHz (25ns period) clock */
void SysTick_Wait(unsigned long reload)
{ NVIC_ST_RELOAD_R = reload;
  NVIC_ST_CURRENT_R = 0;
  while((NVIC_ST_CTRL_R & 0x00010000)==0){}  //Waiting for flag
}

/* Function for wait n ms (considering 40MHz clock) */
void SysTick_Wait_n_ms(unsigned long n)
{ unsigned long i;
  for(i=0 ; i<n ; i++)
  {SysTick_Wait(39999);}
}

/* Function to initilaize Port B */
void Port_B_init ()
{   SYSCTL_RCGCGPIO_R |= 0x02 ;                     // GPIO Port B Run Mode Clock Gating Control
    while((SYSCTL_PRGPIO_R & 0x02) == 0){};         // Waiting for preipherals to be ready
    GPIO_PORTB_DIR_R |= ~0x10;                      // Specifie PB4 input (Next botton)
    GPIO_PORTB_DEN_R |= 0x10;                       // Enabling pin as digital
    GPIO_PORTB_DATA_R = 0x00;                       // Intialize data to 0
}

/* Function to initilaize Port A */
void Port_A_init ()
{   SYSCTL_RCGCGPIO_R |= 0x01 ;                   // GPIO Port A Run Mode Clock Gating Control
    while((SYSCTL_PRGPIO_R & 0x01) == 0){};       // Waiting for preipherals to be ready
    GPIO_PORTA_DIR_R = 0xE0;                      // Specifie PA7-5 output (The three Locks(LEDs) respectively)
    GPIO_PORTA_DEN_R = 0xE0;                      // Enabling pins as digital
    GPIO_PORTA_DATA_R = 0;                        // Intialize data to 0
}
```



```c
/* Function to initilaize Port F */
void Port_F_init ()
{    SYSCTL_RCGCGPIO_R |= 0x20 ;                      // GPIO Port F Run Mode Clock Gating Control
     while((SYSCTL_PRGPIO_R & 0x20) == 0){};          // Waiting for preipherals to be ready
     GPIO_PORTF_LOCK_R = 0x4c4f434b;                  // Unlocking Port F
     GPIO_PORTF_CR_R = 0xFF;                          // Allow changing to the pins
     GPIO_PORTF_PUR_R |= 0x11;                        // Enabling Pull-Up resistors for PF4 and PF0
     GPIO_PORTF_DIR_R = ~0x11;                        // Specifie PF4 and PF0 inputs (Set locks/open locks bottons respectively)
     GPIO_PORTF_DEN_R = 0x11;                         // Enabling digital functionality
     GPIO_PORTF_DATA_R = 0;                           // Intialize data to 0
}


void ADC0_PE3_init(void){


   SYSCTL_RCGCADC_R |= 0x01;                     // Activate ADC0
   SYSCTL_RCGCGPIO_R |= 0x10;                    // Activate clock for Port E
   while((SYSCTL_PRGPIO_R&0x10) == 0){};         // Wait for stabilization
   GPIO_PORTE_DIR_R &= ~0x08;                    // Make PE3 input
   GPIO_PORTE_AFSEL_R |= 0x08;                   // Enable alternate function on PE3
   GPIO_PORTE_DEN_R &= ~0x08;                    // Disable digital I/O on PE3
   GPIO_PORTE_AMSEL_R |= 0x08;                   // Enable analog functionality on PE3
   ADC0_PC_R &= ~0xF;                            // Clear sampling rate
   ADC0_PC_R |= 0x1;                             // Configure for 125K samples/sec
   ADC0_SSPRI_R = 0x0123;                        // Sequencer 3 is highest priority
   ADC0_ACTSS_R &= ~0x0008;                      // Disable sample sequencer 3
   ADC0_EMUX_R &= ~0xF000;                       // Seq3 is software trigger
   ADC0_SSMUX3_R &= ~0xF;                        // Clear channel selection
   ADC0_SSMUX3_R += 0;                           // Set channel AIN0
   ADC0_SSCTL3_R = 0x6;                          // No TS0 D0, Yes IE0 END0
   ADC0_ACTSS_R |= 0x8;                          // Enable SS3
   }

```



```c
void ADC0_PE3_init(void){


    SYSCTL_RCGCADC_R |= 0x01;                     // Activate ADC0
    SYSCTL_RCGCGPIO_R |= 0x10;                    // Activate clock for Port E
    while((SYSCTL_PRGPIO_R&0x10) == 0){};         // Wait for stabilization
    GPIO_PORTE_DIR_R &= ~0x08;                    // Make PE3 input
    GPIO_PORTE_AFSEL_R |= 0x08;                   // Enable alternate function on PE3
    GPIO_PORTE_DEN_R &= ~0x08;                    // Disable digital I/O on PE3
    GPIO_PORTE_AMSEL_R |= 0x08;                   // Enable analog functionality on PE3
    ADC0_PC_R &= ~0xF;                            // Clear sampling rate
    ADC0_PC_R |= 0x1;                             // Configure for 125K samples/sec
    ADC0_SSPRI_R = 0x0123;                        // Sequencer 3 is highest priority
    ADC0_ACTSS_R &= ~0x0008;                      // Disable sample sequencer 3
    ADC0_EMUX_R &= ~0xF000;                       // Seq3 is software trigger
    ADC0_SSMUX3_R &= ~0xF;                        // Clear channel selection
    ADC0_SSMUX3_R += 0;                           // Set channel AIN0
    ADC0_SSCTL3_R = 0x6;                          // No TS0 D0, Yes IE0 END0
    ADC0_ACTSS_R |= 0x8;                          // Enable SS3
    }

/* Function convert from ADC value to readale voltage */
unsigned long resistanceRead ()
{   unsigned long ADC_result, R;


      ADC0_PSSI_R = 0x0008;                           // Initiate SS3
      while((ADC0_RIS_R&0x08)==0){};                  // Wait for A/D conversion to finish
      ADC_result = ADC0_SSFIFO3_R&0xFFF;              // Get ADC result
      ADC0_ISC_R = 0x0008;                            // Clear interrupt status for SS3 to be able to start sampling again
      return R = (ADC_result)*(10000/4095.0);       // Convert ADC value to resistance by quantizing
}
```

Video Link

https://youtu.be/vYyftYnwnLE

AIKEN questions

Digital Sample = (Analog Input x 4095) / max analog value.
A) TRUE
B) FALSE
ANSWER: A

Which of the following register is used to read the ADC0 result?
A) ADC0_SSPRI_R
B) ADC0_SSMUX3_R
C) ADC0_SSCTL3_R
D) ADC0_SSFIFO3
ANSWER: D

Which of the following is not one of the four limitations exist when sampling data?
A) Amplitude resolution
B) Amplitude range
C) Time quantization
D) ADC pins number
ANSWER: D

Which of the following enables sample sequencer 1?
A) ADC0_ACTSS_R |= 0x0008;
B) ADC0_ACTSS_R |= 0x0004;
C) ADC0_ACTSS_R |= 0x0002;
D) ADC0_ACTSS_R |= 0x0001;
ANSWER: C

Which of the following enables alternate function on PE3?
A) GPIO_PORTE_AMSEL_R |= 0x08;
B) GPIO_PORTE_AFSEL_R |= ~0x08;
C) GPIO_PORTE_AMSEL_R |= 0xF7;
D) GPIO_PORTE_AFSEL_R |= ~0xF7;
ANSWER: D

When using "switch" in C, we put "break;" after each case so the program go into the next case.
A) TRUE
B) FALSE
ANSWER: B

ADC0_PC_R register is used to specify the sampling rate.
A) TRUE
B) FALSE
ANSWER: A