Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

Report week 12: LCD, Keyboard, Timers, Interrupt and ADC integration.

Laboratory of Microcontrollers

Gilberto Ochoa Ruiz

Perla Vanessa Jaime Gaytán ITE

A00344428

Nicol Gómez Tisnado ITE

A00227180

Instituto Tecnológico de Estudios Superiores de Monterrey Campus Guadalajara

Zapopan, Jalisco, México.

May 27, 2020

Part 1. ADC Part 1. In this part, you should integrate the ADC and an internal sensor. We saw that the ADC has an internal temperature sensor attached to one of the channels of the ADC. The idea in this part of the lab is to retrieve that value through a program running in your MCU. Instead of using the LEDs as it was the case of the example seen in class, you need to convert the integer obtained from the ADC to display the temperature in the LCD screen through polling.

Code:

```
#include "MKL25Z4.h"

#define RS 1    // BIT0 mask
#define RW 2    // BIT1 mask
#define EN 4    // BIT2 mask

void LCD_nibble_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);

void delayUs(int n);
void delayMs(int n);
void writeNumbers(char value);
void writeTemperature(int value);

void ADC0_init(void);


int main (void)
{
    short int result;
    short int temperature;
    LED_init();                     // Configure LEDs
    ADC0_init();                    // Configure ADC0
    LCD_init();                     // LCD configure
    while (1)
    {
        ADC0_SC1A = 26;                         // start conversion on channel 26 temperature
        while(!(ADC0_SC1A & 0x80)) { }          // wait for COCO
        result = ADC0_RA;                       // read conversion result and clear COCO flag
        temperature = ((result/10)-32)*5/9;     //Converter to Celsius
        writeTemperature(temperature);          //write the actual temperature on the LCD
        delayMs(250);                           //wait 250 ms
    }
}


void writeTemperature(int value)
{
    char buffer[3];                         //an array to save the temperature
    int i;
    sprintf(buffer, "%i", value);           //converts the temperature into an array
    LCD_command(1);                         // clear display
    LCD_command(0x80);                      // set cursor at the begging
    LCD_data('T');
    LCD_data('E');
    LCD_data('M');
    LCD_data('P');
    LCD_data('E');
    LCD_data('R');
    LCD_data('A');
    LCD_data('T');
    LCD_data('U');
    LCD_data('R');
    LCD_data('E');
    LCD_data(':');
    LCD_command(0xC0);                      //set cursos at the second line

    for(i=0; i<2; i++){
        LCD_data(buffer[i]);                //write the two numbers
    }
    LCD_data(0xDF);                         //write ° on the LCD
    LCD_data('C');                          //write C
    LCD_command(0x90);                      //mandar muy lejos lejitos al cursor

}
```

```
void ADC0_init(void)
{
    SIM_SCGC6 |= 0x8000000;                     // clock to ADC0
    ADC0_SC2 &= ~0x40;                          // software trigger

    /* clock div by 4, long sample time, single ended 12 bit, bus clock */
    ADC0_CFG1 = 0x40 | 0x10 | 0x04 | 0x00;
}


void LCD_init(void)
{
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO

    GPIOD_PDDR |= 0xF7;          // make PTD7-4, 2, 1, 0 as output pins
    delayMs(30);                 // initialization sequence
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x20, 0);  // use 4-bit data mode
    delayMs(1);
    LCD_command(0x28);           // set 4-bit data, 2-line, 5x7 font
    LCD_command(0x06);           // move cursor right
    LCD_command(0x01);           // clear screen, move cursor to home
    LCD_command(0x0F);           // turn on display, cursor blinking
}
```

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;                           // clear lower nibble for control
    control &= 0x0F;                        // clear upper nibble for data
    GPIOD_PDOR = data | control;            // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;       // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}


void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0);    // upper nibble first
    LCD_nibble_write(command << 4, 0);      // then lower nibble
    if (command < 4)
        delayMs(4);                         // commands 1 and 2 need up to 1.64ms
    else
        delayMs(1);                         // all others 40 us
}

void LCD_data(unsigned char data)
{
    LCD_nibble_write(data & 0xF0, RS);    // upper nibble first
    LCD_nibble_write(data << 4, RS);      // then lower nibble
    delayMs(1);
}


void delayMs(int n) {
    int i;
    int j;
    for(i = 0 ; i < n; i++){
        for(j = 0 ; j < 7000; j++) { }
    }
}
void delayUs(int n){
    int i,j;
    for(i = 0 ; i < n; i++) {
        for(j = 0; j < 5; j++) ;
    }
}
```

Registers:

SIM_SCGC5	0x00001180
SIM_SCGC6	0x08000001

```
SIM_SCGC5 = 1180

System Clock Gating Control Register 5

Bit Field Values:
          bits[ 31:20 ] = 0
          bits[ 19:19 ] = 0
          bits[ 18:14 ] = 0
    PORTE bits[ 13:13 ] = 0 Clock disabled
    PORTD bits[ 12:12 ] = 1 Clock enabled
    PORTC bits[ 11:11 ] = 0 Clock disabled
    PORTB bits[ 10:10 ] = 0 Clock disabled
    PORTA bits[  9:9  ] = 0 Clock disabled
          bits[  8:7  ] = 3
          bits[  6:6  ] = 0
    TSI   bits[  5:5  ] = 0 Access disabled
          bits[  4:2  ] = 0
          bits[  1:1  ] = 0
    LPTMR bits[  0:0  ] = 0 Access disabled
```

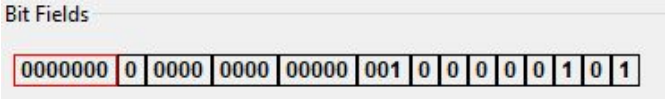
00000000000	0	00000	0	1	0	0	0	11	0	0	000	0	0

```
SIM_SCGC6 = 8000001

System Clock Gating Control Register 6

Bit Field Values:
    DAC0   bits[ 31:31 ] = 0    Clock disabled
           bits[ 30:30 ] = 0
    RTC    bits[ 29:29 ] = 0    Access and interrupts disabled
           bits[ 28:28 ] = 0
    ADC0   bits[ 27:27 ] = 1    Clock enabled
    TPM2   bits[ 26:26 ] = 0    Clock disabled
    TPM1   bits[ 25:25 ] = 0    Clock disabled
    TPM0   bits[ 24:24 ] = 0    Clock disabled
    PIT    bits[ 23:23 ] = 0    Clock disabled
           bits[ 22:16 ] = 0
           bits[ 15:15 ] = 0
           bits[ 14:2  ] = 0
    DMAMUX bits[  1:1  ] = 0    Clock disabled
    FTF    bits[  0:0  ] = 1    Clock enabled
```

0	0	0	0	1	0	0	0	0	0000000	0	00000000000000	0	1

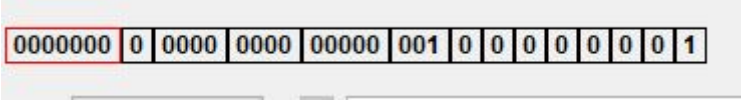
PORTD_PCR0	0x00000105
PORTD_PCR1	0x00000105
PORTD_PCR2	0x00000105
PORTD_PCR3	0x00000105
PORTD_PCR4	0x00000101
PORTD_PCR5	0x00000101
PORTD_PCR6	0x00000101
PORTD_PCR7	0x00000101

For PORTD_PCR0 to PORTD_PCR3 the configuration of the registers is the next.




```
PORTD_PCR3 = 105

Pin Control Register n

Bit Field Values:
        bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
        bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
        bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
        bits[  7:7   ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
        bits[  5:5   ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
        bits[  3:3   ] = 0
    SRE  bits[  2:2  ] = 1  Slow slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 0  Internal pullup or pulldown resistor is not enabled on the
corresponding pin.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

For PORTD_PCR4 to PORTD_PCR7 the configuration of the registers is the next.



```
PORTD_PCR4 = 101

Pin Control Register n

Bit Field Values:
         bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
         bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
         bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
         bits[  7:7  ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
         bits[  5:5  ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
         bits[  3:3  ] = 0
    SRE  bits[  2:2  ] = 0  Fast slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 0  Internal pullup or pulldown resistor is not enabled on the
corresponding pin.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

GPIOD_PDDR	0x000000f7

00000000000000000000000011110111

ADC0_CFG1	0x00000054

00000000000000000000000 | 0 | 10 | 1 | 01 | 00

```
ADC0_CFG1 = 54

ADC Configuration Register 1

Bit Field Values:
           bits[ 31:8  ] = 0
    ADLPC  bits[  7:7  ] = 0    Normal power configuration.
    ADIV   bits[  6:5  ] = 2    The divide ratio is 4 and the clock rate is (input
clock)/4.
    ADLSMP bits[  4:4  ] = 1    Long sample time.
    MODE   bits[  3:2  ] = 1    When DIFF=0:It is single-ended 12-bit conversion ;
when DIFF=1, it is differential 13-bit conversion with 2's complement output.
    ADICLK bits[  1:0  ] = 0    Bus clock
```

ADC0_SC1A	0x0000001a

```
ADC0_SC1A = 1a

ADC Status and Control Registers 1

Bit Field Values:
         bits[ 31:8  ] = 0
    COCO bits[  7:7  ] = 0  Conversion is not completed.
    AIEN bits[  6:6  ] = 0  Conversion complete interrupt is disabled.
    DIFF bits[  5:5  ] = 0  Single-ended conversions and input channels are selected.
    ADCH bits[  4:0  ] = 1a When DIFF=0, Temp Sensor (single-ended) is selected as
input; when DIFF=1, Temp Sensor (differential) is selected as input.
```

00000000000000000000000	0	0	0	11010

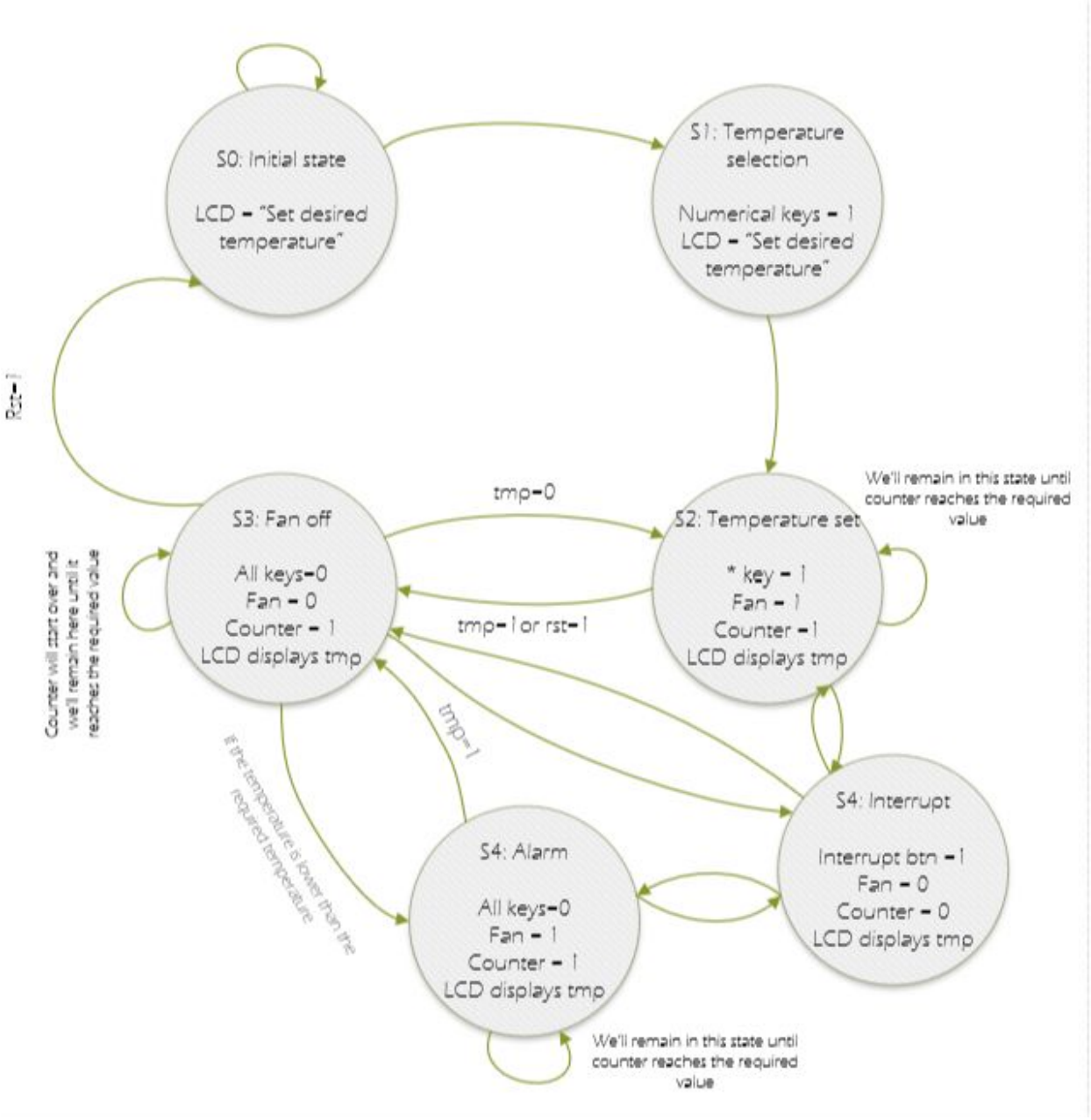
Part 2. ADC Part 2: Simple thermostat. As with the previous part, we will be using the internal ADC, but this time connecting a sensor. It will be optimal if you use something similar to the LM35 but any other sensor is fine (even a potentiometer). We will be using the keyboard, LCD, timers, interrupts and the UART. The code in this part should proceed as follows:

1. **When you start your application, the next message should be displayed. "Set the desired temperature:"**
2. **As with the previous code you should be able to read the temperature from the keyboard and display in the LCD. Then, "set" the "air conditioner" (a LED, but if you have a simple fan it would be nice). In order to start sensing the temperature (using the ADC) you should press the # or * button. The fan is activated for a predefined time (let say 10 seconds, for demonstration purposes) using a counter, and re-activated every minute to "maintain the temperature" (this is how these systems usually work)**
3. **The message should be displayed it for the normal execution of the program (only changing the values of the ADC), and the system should keep activating the fan (or LED). However, if the value surpasses the desired temperature, the system should activate another output ("alarm") and activate the fan for a longer time (2 mins). After this time, if the temperature is still higher than the set temperature, it should go again for two mints. Otherwise, it moves to normal mode.**
4. **The fan can be put in idle mode at any moment through a pushbutton via an interrupt, as in the previous lab. The idea is that you still show the temperature**

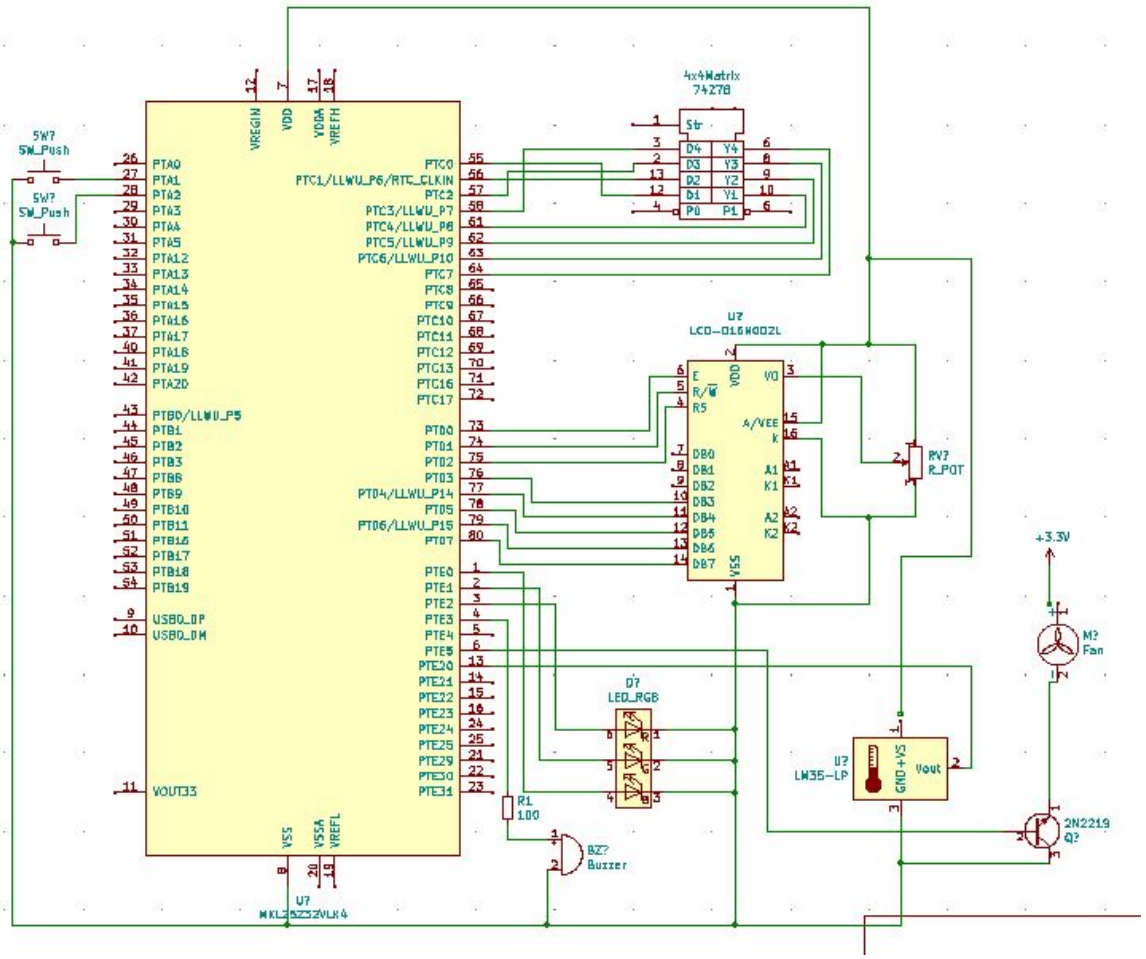
but you don’t activate the fan in this mode. Whenever you want, press the # button in the keyboard to resume.

5. **Reset the system with a second button and an interrupt, in which case the code should go to the first step and ask again for the temperature.**

State machine or a flow diagram:



Schematic:


4x4Matrix
74278
SW?
SW_Push
SW?
SW_Push
U?
MKL25Z32VLK4
U?
LCD-016N002L
RV?
R_POT
D?
LED_RGB
BZ?
Buzzer
R1
100
U?
LM35-LP
+3.3V
M?
Fan
2N2219
Q?

Code:

```
#include "MKL25Z4.h"

#define RS 1    // BIT0 mask
#define RW 2    // BIT1 mask
#define EN 4    // BIT2 mask

void ADC0_init(void);
void LED_set(int s);
void led_init(void);

void LCD_nibble_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);

void delayUs(int n);
void delayMs(int n);
void writeTemperature(int value);

void TPM_init(void);
void TPM_timer(void);

void keypad_init(void);
char keypad_getkey(void);
int read_key(int x);
int getNum(int key);



void TPM_init(void);
void TPM_timer(void);

void keypad_init(void);
char keypad_getkey(void);
int read_key(int x);
int getNum(int key);

void interupt_init(void);

void askTemperature(void);
void writeNumbers(int value);

void ledandbuzzerOn(void);
void ledandbuzzerOff(void);

void writeTempNorm(void);


int main (void)
{
    short int result;
    short int temperature;
    unsigned char key;
    int rkey;
    int m;
    int c=0;
    int n;
    int num;
    int num1 = 0;
    int num2 = 0;
    short int destemp;
    led_init();             // Configure LEDs
    ADC0_init();            // Configure ADC0
    LCD_init();             // LCD Configure
    keypad_init();          // Keyboard configure
    TPM_init();             // Timer Configure
    interupt_init();        // Interuption Configure
```

```
while (1)
{
    //Pruebas de sensor LM35

    /*ADC0_SC1A = 0;               //start conversion on channel 0
    while(!(ADC0_SC1A & 0x80)) { }  // wait for conversion complete
    result = ADC0_RA;               // read conversion result and clear COCO flag
    temperature = result* 50 / 1024;        //Converter to Celsius


    LCD_command(1);         // clear display
    LCD_command(0x80);      // set cursor at the middle
    LCD_data('T');
    LCD_data('E');
    LCD_data('M');
    LCD_data('P');
    LCD_data('E');
    LCD_data('R');
    LCD_data('A');
    LCD_data('T');
    LCD_data('U');
    LCD_data('R');
    LCD_data('E');
    LCD_data(':');
    LCD_command(0xC2);
    LCD_data(0xDF);
    LCD_data('C');
    writeTemperature(temperature);
    delayMs(500);*/


m = 1;
n = 0;                                  //números escritos

GPIOE_PCOR |= 0x20;                     //turn off fan
GPIOE_PCOR |= 0x08;                     //turn off red led
GPIOE_PCOR |= 0x02;                     //turn off green led
GPIOE_PCOR |= 0x01;                     //turn off blue led
GPIOE_PCOR |= 0x10;                     //turn off buzzer

askTemperature();


while(m == 1)
{
    key = keypad_getkey();              //get which key was pressed
    rkey  = read_key(key);              //read if the key was pressed
    num = getNum(key);
    if(rkey == 1)                       //if it was pressed then
    {
        if(key != 13 && key!=14 && key!=16 && key!= 0 && key!= 1 && key!= 9 && key!= 5)

        //si key no es igual a null,*,#,D,C,B,A
        {
            if (n==0 && num != 0)               //para escribir el primer numero
            {
                n++;
                writeNumbers(num);              //escribir el numero presionado
                num1 = num;
                while (rkey == 1)               //if still pressed stay here till is not as debouncer
                {
                    key = keypad_getkey();      //get the key pressed
                    rkey  = read_key(key);      //read if the key is still pressed
                }
            }
```

```
    else if(n==1)                       //para escribir el segundo numero
    {
        n++;
        writeNumbers(num);              //escribir el numero presionado
        num2 = num;
        while (rkey == 1)               //if still pressed stay here till is not as debouncer
        {
            key = keypad_getkey();      //get the key pressed
            rkey  = read_key(key);      //read if the key is still pressed
        }
        LCD_command(0x90);              // ocultar el cursor
    }
}
else if(key == 14)                      //if # is pressed
{
    destemp= num1*10 + num2;            //destemp is the desire temperature

    ADC0_SC1A = 0;                          //start conversion on channel 0 where LM45 is connected
    while(!(ADC0_SC1A & 0x80)) { }          // wait for conversion complete
    result = ADC0_RA;                       // read conversion result and clear COCO flag
    temperature = result* 50 / 1024;        //Converter to Celsius

    writeTempNorm();                        //Write Temperature:XX°C  Status normal.
```

```
   while(key != 16)                     //mientras * not pressed
   {

       while(temperature < destemp && key != 16 )  //mientras temperature do not reach desire temp
       {
           key = keypad_getkey();                  //get if the key was pressed
           writeTemperature(temperature);          //write the actual temp
           LCD_command(0x90);                      //ocultar cursor

           GPIOE_PCOR |= 0x20;          //turn off fan
           GPIOE_PCOR |= 0x08;          //turn off red led
           GPIOE_PSOR |= 0x02;          //turn green led
           GPIOE_PCOR |= 0x10;          //buzzer
           GPIOE_PCOR |= 0x01;          //turn off blue led

           TPM_timer();                 //wait 1 second
           c++;

           if (c==30){                          //every 30 sec

               GPIOE_PCOR |= 0x02;              //turn off green led
               GPIOE_PSOR |= 0x01;              //turn on blue led
               GPIOE_PSOR |= 0x20;              //set on fan
               int i;
               for(i=0; i<10; i++){                 //10 seconds

                   ADC0_SC1A = 0;                       //start conversion on channel 0
                   while(!(ADC0_SC1A & 0x80)) { }       // wait for conversion complete
```

```
                        ADC0_SC1A = 0;                      //start conversion on channel 0
                        while(!(ADC0_SC1A & 0x80)) { }      // wait for conversion complete
                        result = ADC0_RA;                   // read conversion result and clear COCO flag
                        temperature = result* 50 / 1024;    //Converter to Celsius

                    }
                    while(temperature >= destemp && key != 16)
                    {
                        key = keypad_getkey();              //get if the  * key pressed
                        writeTemperature(temperature);      //write the actual temperature

                        GPIOE_PSOR |= 0x08;                 //turn on red led
                        GPIOE_PSOR |= 0x10;                 //turn on buzzer
                        GPIOE_PCOR |= 0x02;                 //turn off green led
                        GPIOE_PSOR |= 0x20;                 //set on fan

                        ADC0_SC1A = 0;                      //start conversion on channel 0
                        while(!(ADC0_SC1A & 0x80)) { }      // wait for conversion complete
                        result = ADC0_RA;                   // read conversion result and clear COCO flag
                        temperature = result* 50 / 1024;    //Converter to Celsius
                        writeTemperature(temperature);      //write actual temperture
                        LCD_command(0x90);                  //hide the cursor
                        TPM_timer();                        //wait 1 second

                    }
                }
            }


            }
            else if(key == 16)                      //if * was pressed reset
            {
                m = 2;
            }
        }
    }

}
}
```

```
void writeTempIDLE(void) //write mode :IDLE
{
    LCD_command(1);          // clear display
    LCD_command(0x80);       // set cursor at the begging first line
    LCD_data('T');
    LCD_data('E');
    LCD_data('M');
    LCD_data('P');
    LCD_data('E');
    LCD_data('R');
    LCD_data('A');
    LCD_data('T');
    LCD_data('U');
    LCD_data('R');
    LCD_data('E');
    LCD_data(':');
    LCD_command(0xC2);       //set cursor at the 3rd position in the second line
    LCD_data(0xDF);          //write °
    LCD_data('C');           //write C
    LCD_command(0xCC);       //Set cursor almost at the end
    LCD_data('I');           //Mode: IDLE
    LCD_data('D');
    LCD_data('L');
    LCD_data('E');
}
```

```
void PORTA_IRQHandler(void)
{
    char key;
    short int result;
    short int temperature;
    GPIOE_PCOR |= 0x08;          //turn off red led
    GPIOE_PCOR |= 0x10;          //turn off buzzer

    writeTempIDLE();
    while(key != 16)
            {
                GPIOE_PSOR |= 0x20;             //turn on fan
                GPIOE_PSOR |= 0x01;             //turn blue led
                GPIOE_PSOR |= 0x02;             //turn green led

                key = keypad_getkey();          //just in case * was pressed

                ADC0_SC1A = 0;                      //start conversion on channel 0
                while(!(ADC0_SC1A & 0x80)) { }      // wait for conversion complete
                result = ADC0_RA;                   // read conversion result and clear COCO flag
                temperature = result* 50 / 1024;    //Converter to Celsius
                writeTemperature(temperature);      //Write the actual temperture
                LCD_command(0x90);                  //mandar a chsm el cursor :D
                TPM_timer();                        //wait 1 second
            }

    PORTA_ISFR = 0x00000002;    // clear interrupt flag
    GPIOE_PCOR |= 0x20;         // turn off fan
    GPIOE_PCOR |= 0x01;         // turn off blue led
    GPIOE_PCOR |= 0x02;         // turn off green led
    writeTempNorm();            // write mode: Normal

}
```

```
void writeTempNorm(void){
    LCD_command(1);          // clear display
    LCD_command(0x80);       // set cursor at the middle
    LCD_data('T');
    LCD_data('E');
    LCD_data('M');
    LCD_data('P');
    LCD_data('E');
    LCD_data('R');
    LCD_data('A');
    LCD_data('T');
    LCD_data('U');
    LCD_data('R');
    LCD_data('E');
    LCD_data(':');
    LCD_command(0xC2);       //set cursor 2nd line 3rd position
    LCD_data(0xDF);          //write °
    LCD_data('C');
    LCD_command(0xCC);       //set cursor almost at the end
    LCD_data('N');
    LCD_data('O');
    LCD_data('R');
    LCD_data('M');
}
```

```
void writeNumbers(int value)     //write numbers on the LCD
{
    if(value == 0)
        LCD_data('0');
    else if(value == 1)
        LCD_data('1');
    else if(value == 2)
        LCD_data('2');
    else if(value == 3)
        LCD_data('3');
    else if(value == 4)
        LCD_data('4');
    else if(value == 5)
        LCD_data('5');
    else if(value == 6)
        LCD_data('6');
    else if(value == 7)
        LCD_data('7');
    else if(value == 8)
        LCD_data('8');
    else if(value == 9)
        LCD_data('9');
}

void askTemperature(void)    //ask for the desire temperature
{
    LCD_command(1);          // clear display
    LCD_command(0x80);       // set cursor at the middle
    LCD_data('S');
    LCD_data('E');
    LCD_data('T');
    LCD_data(' ');
    LCD_data('T');
    LCD_data('H');
    LCD_data('E');
    LCD_data(' ');
    LCD_data('D');
    LCD_data('E');
    LCD_data('S');
    LCD_data('I');
    LCD_data('R');
    LCD_data('E');
    LCD_command(0xC0);       //set cursor at the second line
    LCD_data('T');
    LCD_data('E');
    LCD_data('M');
    LCD_data('P');
    LCD_data('E');
    LCD_data('R');
    LCD_data('A');
```

```
LCD_command(0xC0);      //set cursor at the second line
LCD_data('T');
LCD_data('E');
LCD_data('M');
LCD_data('P');
LCD_data('E');
LCD_data('R');
LCD_data('A');
LCD_data('T');
LCD_data('U');
LCD_data('R');
LCD_data('E');
delayMs(500);           // delay 500 ms
LCD_command(1);         // clear display
LCD_command(0x80);      //set cursor lst line lst position
LCD_data('T');
LCD_data('E');
LCD_data('M');
LCD_data('P');
LCD_data('E');
LCD_data('R');
LCD_data('A');
LCD_data('T');
LCD_data('U');
LCD_data('R');
LCD_data('E');
LCD_data(':');
LCD_command(0xC0);      //set cursor at the second line first position
LCD_data('P');
```

```
    LCD_command(0xC0);      //set cursor at the second line first position
    LCD_data('P');
    LCD_data('R');
    LCD_data('E');
    LCD_data('S');
    LCD_data('S');
    LCD_data(' ');
    LCD_data('#');
    LCD_data(' ');
    LCD_data('T');
    LCD_data('O');
    LCD_data(' ');
    LCD_data('S');
    LCD_data('E');
    LCD_data('T');
    LCD_command(0x8E);      //set cursor afet the numbers of the desire temperature that will be introduce
    LCD_data(0xDF);         //write °
    LCD_data('C');
    LCD_command(0x8C);      //set cursor to write the desire temp

}
```

```
void TPM_init(void)
{
    SIM_SCGC6|= SIM_SCGC6_TPM0_MASK;                                        //assign clk to TPM0

    SIM_SOPT2 |= (SIM_SOPT2 & ~SIM_SOPT2_TPMSRC_MASK)|SIM_SOPT2_TPMSRC(3); //SET CLK SOURCE TO BE 32.768 kHZ
    MCG_C1 |= MCG_C1_IRCLKEN_MASK;
    TPM0_SC = 0;

    TPM0_MOD = 32768-1;             //TPMx_MOD = value

    TPM0_SC |= TPM_SC_TOF_MASK;     //Clear TOF  flag

    TPM0_SC |= TPM_SC_CMOD(1);      //Enable timer
}

void TPM_timer(void){
    while (!(TPM0_SC & TPM_SC_TOF_MASK));   //wait until tof is set
    TPM0_SC |= TPM_SC_TOF_MASK;             //clear tof flag
}


void keypad_init(void)
{
     SIM_SCGC5 |= 0x0800;    //enable clk to port c

     PORTC_PCR0 = 0x103;     //PTC0 as GPIO and enable pullup
     PORTC_PCR1 = 0x103;     //PTC1 as GPIO and enable pullup
     PORTC_PCR2 = 0x103;     //PTC2 as GPIO and enable pullup
     PORTC_PCR3 = 0x103;     //PTC3 as GPIO and enable pullup
     PORTC_PCR4 = 0x103;     //PTC4 as GPIO and enable pullup
     PORTC_PCR5 = 0x103;     //PTC5 as GPIO and enable pullup
     PORTC_PCR6 = 0x103;     //PTC6 as GPIO and enable pullup
     PORTC_PCR7 = 0x103;     //PTC7 as GPIO and enable pullup
     GPIOC_PDDR = 0x0F;      //make PTC7-0 as input pins
}


char keypad_getkey(void){

    int col, row;
    const char row_select[] = {0x01, 0x02, 0x04, 0x08};

    GPIOC_PDDR = 0x0F;  //enable al rows
    GPIOC_PCOR = 0x0F;
    delayUs(2);         //wait for signal

    col = 0xF00 & GPIOC_PDIR;   //read all columns

    GPIOC_PDDR = 0;             //disable all rows

    if (col == 0xF0)            //no key pressed;
    return 0;

    for (row = 0; row<4; row++)         //finds out which key was pressed
    {
        GPIOC_PDDR = 0;                 //disable all rows
        GPIOC_PDDR |= row_select[row];  //enable one row
        GPIOC_PCOR = row_select[row];   //drive the active row low
        delayUs(2);                     //wait for signal to settle
        col = GPIOC_PDIR & 0xF0;        //read all columns
        if (col != 0xF0) break;         //if one of the inputs is low some key is pressed
    }
```

```
    for (row = 0; row<4; row++)          //finds out which key was pressed
    {
        GPIOC_PDDR = 0;                  //disable all rows
        GPIOC_PDDR |= row_select[row];   //enable one row
        GPIOC_PCOR = row_select[row];    //drive the active row low
        delayUs(2);                      //wait for signal to settle
        col = GPIOC_PDIR & 0xF0;         //read all columns
        if (col != 0xF0) break;          //if one of the inputs is low some key is pressed
    }

    GPIOC_PDDR = 0; //disable all rows

    if (row == 4)   //no key was pressed
    return 0;

    if (col == 0xE0) return row * 4 + 1; //key in column 1
    if (col == 0xD0) return row * 4 + 2; //key in column 2
    if (col == 0xB0) return row * 4 + 3; //key in column 3
    if (col == 0x70) return row * 4 + 4; //key in column 4
    return 0;                             //other information received
}


int read_key(int x)
{
    int key_press;

    if(x!=0)                 //if a key was pressed return 1
    return key_press = 1;
    else                     //if there is no key pressed return 0
    return key_press = 0;

}


int getNum(int value) //identify which number was pressed from the keyboard
{
    if(value == 15)
        return 0;
    else if(value == 4)
        return 1;
    else if(value == 3)
        return 2;
    else if(value == 2)
        return 3;
    else if(value == 8)
        return 4;
    else if(value == 7)
        return 5;
    else if(value == 6)
        return 6;
    else if(value == 12)
        return 7;
    else if(value == 11)
        return 8;
    else if(value == 10)
        return 9;
}
```

```
void writeTemperature(int value)     //write the actual temperature from the LM35
{
    char buffer[3];                  //array to save the temp
    int i;
    sprintf(buffer, "%i", value);    //convert into an array the value/temperature
    LCD_command(0xC0);               //set cursor at the second line 1st position
    for(i=0; i<2; i++){
        LCD_data(buffer[i]);         //write the temperature
    }
}
```

```
void LCD_init(void)
{
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO

    GPIOD_PDDR |= 0xF7;          // make PTD7-4, 2, 1, 0 as output pins
    delayMs(30);                 // initialization sequence
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x20, 0);   // use 4-bit data mode
    delayMs(1);
    LCD_command(0x28);           // set 4-bit data, 2-line, 5x7 font
    LCD_command(0x06);           // move cursor right
    LCD_command(0x01);           // clear screen, move cursor to home
    LCD_command(0x0F);           // turn on display, cursor blinking
}
```

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;                       // clear lower nibble for control
    control &= 0x0F;                    // clear upper nibble for data
    GPIOD_PDOR = data | control;        // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;   // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}


void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0);    // upper nibble first
    LCD_nibble_write(command << 4, 0);      // then lower nibble
    if (command < 4)
        delayMs(4);         // commands 1 and 2 need up to 1.64ms
    else
        delayMs(1);         // all others 40 us
}

void LCD_data(unsigned char data)
{
    LCD_nibble_write(data & 0xF0, RS);     // upper nibble first
    LCD_nibble_write(data << 4, RS);       // then lower nibble
    delayMs(1);
}


void delayMs(int n) {
    int i;
    int j;
    for(i = 0 ; i < n; i++){
        for(j = 0 ; j < 7000; j++) { }
    }
}

void delayUs(int n){
    int i,j;
    for(i = 0 ; i < n; i++) {
        for(j = 0; j < 5; j++) ;
    }
}
```

```
void ADC0_init(void)
{
    SIM_SCGC5 |= 0x2000;                // clock to PORTE
    PORTE_PCR20 = 0;                    // PTE20 analog input
    SIM_SCGC6 |= 0x8000000;             // clock to ADC0
    ADC0_SC2 &= ~0x40;                  // software trigger
    /* clock div by 4, long sample time, single ended 12 bit, bus clock */
    ADC0_CFG1 = 0x40 | 0x10 | 0x04 | 0x00;
}

void led_init(void){

    SIM_SCGC5 |= 0x2000;    //enable clk to port E

    //RED LED
    PORTE_PCR3 = 0x100;     //make PTE3 as GPIO
    GPIOE_PDDR |= 0x08;     //make PTE3 as output pin
    GPIOE_PCOR |= 0x08;     //turn off red LED
    //Fan
    PORTE_PCR5 = 0x100;     //make PTE5 as GPIO
    GPIOE_PDDR |= 0x20;     //make PTE5 as output pin
    GPIOE_PCOR |= 0x20;     //turn off fan
    //BUZZER
    PORTE_PCR4 = 0x100;     //make PTE4 as GPIO
    GPIOE_PDDR |= 0x10;     //make PTE4 as output pin
    GPIOE_PCOR |= 0x10;     //turn off buzzer
    //BLUE LED
    PORTE_PCR0 = 0x100;     //make PTE1 as GPIO
    GPIOE_PDDR |= 0x01;     //make PTE1 as output pin
    GPIOE_PCOR |= 0x01;     //turn off blue LED
    //GREEN LED
    PORTE_PCR1 = 0x100;     //make PTE0 as GPIO
    GPIOE_PDDR |= 0x02;     //make PTE0 as output pin
    GPIOE_PCOR |= 0x02;     //turn off green LED
}

void interupt_init(void)
{
    NVIC_ICPR |= 0x40000000;    // disable INT30 (bit 30 of ISER[0]) while configuring
    SIM_SCGC5 |= 0x200;         // enable clock to Port A

    /* configure PTA1 for interrupt */
    PORTA_PCR1 |=  0x00103;     // make it GPIO and enable pull-up
    GPIOA_PDDR &= ~0x0002;      // make pin input
    PORTA_PCR1 &= ~0xF0000;     // clear interrupt selection
    PORTA_PCR1 |=  0xA0000;     // enable falling edge interrupt

    NVIC_ISER |= 0x40000000;    // enable INT30 (bit 30 of ISER[0])
}
```

Registers:

SIM_SCGC5	0x00003b80
SIM_SCGC6	0x09000001

```
SIM_SCGC5 = 3b80

System Clock Gating Control Register 5

Bit Field Values:
          bits[ 31:20 ] = 0
          bits[ 19:19 ] = 0
          bits[ 18:14 ] = 0
    PORTE bits[ 13:13 ] = 1 Clock enabled
    PORTD bits[ 12:12 ] = 1 Clock enabled
    PORTC bits[ 11:11 ] = 1 Clock enabled
    PORTB bits[ 10:10 ] = 0 Clock disabled
    PORTA bits[  9:9  ] = 1 Clock enabled
          bits[  8:7  ] = 3
          bits[  6:6  ] = 0
    TSI   bits[  5:5  ] = 0 Access disabled
          bits[  4:2  ] = 0
          bits[  1:1  ] = 0
    LPTMR bits[  0:0  ] = 0 Access disabled
```

000000000000	0	00000	1	1	1	0	1	11	0	0	000	0	0

```
SIM_SCGC6 = 9000001

System Clock Gating Control Register 6

Bit Field Values:
    DAC0   bits[ 31:31 ] = 0    Clock disabled
           bits[ 30:30 ] = 0
    RTC    bits[ 29:29 ] = 0    Access and interrupts disabled
           bits[ 28:28 ] = 0
    ADC0   bits[ 27:27 ] = 1    Clock enabled
    TPM2   bits[ 26:26 ] = 0    Clock disabled
    TPM1   bits[ 25:25 ] = 0    Clock disabled
    TPM0   bits[ 24:24 ] = 1    Clock enabled
    PIT    bits[ 23:23 ] = 0    Clock disabled
           bits[ 22:16 ] = 0
           bits[ 15:15 ] = 0
           bits[ 14:2  ] = 0
    DMAMUX bits[  1:1  ] = 0    Clock disabled
    FTF    bits[  0:0  ] = 1    Clock enabled
```

0	0	0	0	1	0	0	1	0	0000000	0	0000000000000	0	1

SIM_SOPT2	0x03000000

```
SIM_SOPT2 = 3000000

System Options Register 2

Bit Field Values:
                bits[ 31:28 ] = 0
    UART0SRC    bits[ 27:26 ] = 0  Clock disabled
    TPMSRC      bits[ 25:24 ] = 3  MCGIRCLK clock
                bits[ 23:19 ] = 0
    USBSRC      bits[ 18:18 ] = 0  External bypass clock (USB_CLKIN).
                bits[ 17:17 ] = 0
    PLLFLLSEL   bits[ 16:16 ] = 0  MCGFLLCLK clock
                bits[ 15:8  ] = 0
    CLKOUTSEL   bits[  7:5  ] = 0  Reserved
    RTCCLKOUTSEL bits[  4:4  ] = 0  RTC 1 Hz clock is output on the RTC_CLKOUT pin.
                bits[  3:0  ] = 0
```

0000	00	11	00000	0	0	0	00000000	000	0	0000

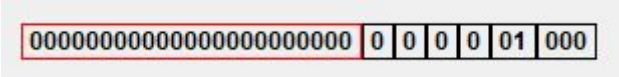
MCG_C1	0x06

```
MCG_C1 = 6

MCG Control 1 Register

Bit Field Values:
    CLKS     bits[  7:6  ] = 0  Encoding 0 - Output of FLL or PLL is selected (depends
on PLLS control bit).
    FRDIV    bits[  5:3  ] = 0  If RANGE 0 = 0 , Divide Factor is 1; for all other
RANGE 0 values, Divide Factor is 32.
    IREFS    bits[  2:2  ] = 1  The slow internal reference clock is selected.
    IRCLKEN  bits[  1:1  ] = 1  MCGIRCLK active.
    IREFSTEN bits[  0:0  ] = 0  Internal reference clock is disabled in Stop mode.
```

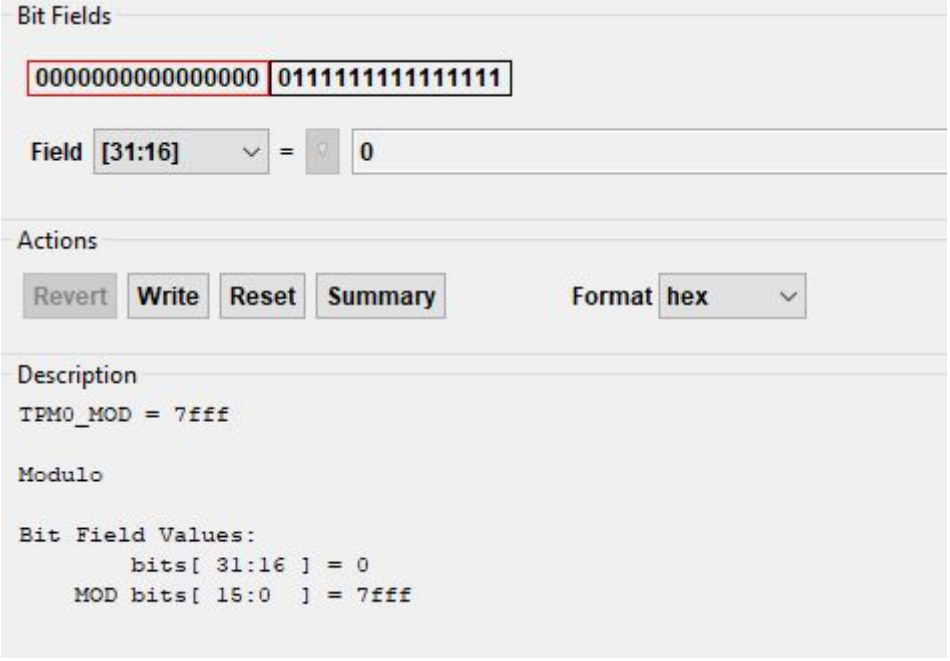
00	000	1	1	0

TPM0_SC	0x00000008

```
TPM0_SC = 8

Status and Control

Bit Field Values:
          bits[ 31:9  ] = 0
    DMA   bits[  8:8  ] = 0 Disables DMA transfers.
    TOF   bits[  7:7  ] = 0 LPTPM counter has not overflowed.
    TOIE  bits[  6:6  ] = 0 Disable TOF interrupts. Use software polling or DMA
request.
    CPWMS bits[  5:5  ] = 0 LPTPM counter operates in up counting mode.
    CMOD  bits[  4:3  ] = 1 LPTPM counter increments on every LPTPM counter clock
    PS    bits[  2:0  ] = 0 Divide by 1
```

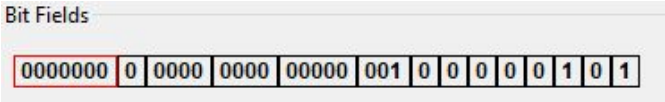


TPM0_MOD	0x00007fff



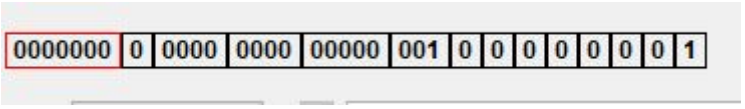
PORTD_PCR0	0x00000105
PORTD_PCR1	0x00000105
PORTD_PCR2	0x00000105
PORTD_PCR3	0x00000105
PORTD_PCR4	0x00000101
PORTD_PCR5	0x00000101
PORTD_PCR6	0x00000101
PORTD_PCR7	0x00000101

For PORTD_PCR0 to PORTD_PCR3 the configuration of the registers is the next.




```
PORTD_PCR3 = 105

Pin Control Register n

Bit Field Values:
        bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
        bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
        bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
        bits[  7:7   ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
        bits[  5:5   ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
        bits[  3:3   ] = 0
    SRE  bits[  2:2  ] = 1  Slow slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 0  Internal pullup or pulldown resistor is not enabled on the
corresponding pin.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

For PORTD_PCR4 to PORTD_PCR7 the configuration of the registers is the next.



```
PORTD_PCR4 = 101

Pin Control Register n

Bit Field Values:
         bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
         bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
         bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
         bits[  7:7  ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
         bits[  5:5  ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
         bits[  3:3  ] = 0
    SRE  bits[  2:2  ] = 0  Fast slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 0  Internal pullup or pulldown resistor is not enabled on the
corresponding pin.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

GPIOD_PDDR	0x000000f7

```
00000000000000000000000011110111
```

PORTC_PCR0	0x00000107
PORTC_PCR1	0x00000107
PORTC_PCR2	0x00000107
PORTC_PCR3	0x00000103
PORTC_PCR4	0x00000103
PORTC_PCR5	0x00000103
PORTC_PCR6	0x00000103
PORTC_PCR7	0x00000103

For PORTC_PCR0 to PORTC_PCR2 the configuration of the registers is the next.

0000000	0	0000	0000	00000	001	0	0	0	0	0	1	1	1

```
PORTC_PCR0 = 107

Pin Control Register n

Bit Field Values:
         bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
         bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
         bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
         bits[  7:7  ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
         bits[  5:5  ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
         bits[  3:3  ] = 0
    SRE  bits[  2:2  ] = 1  Slow slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 1  Internal pullup or pulldown resistor is enabled on the
corresponding pin, if the pin is configured as a digital input.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

For PORTC_PCR3 to PORTC_PCR7 the configuration of the registers is the next.

0000000	0	0000	0000	00000	001	0	0	0	0	0	0	1	1

```
PORTC_PCR7 = 103

Pin Control Register n

Bit Field Values:
         bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
         bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
         bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
         bits[  7:7  ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
         bits[  5:5  ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
         bits[  3:3  ] = 0
    SRE  bits[  2:2  ] = 0  Fast slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 1  Internal pullup or pulldown resistor is enabled on the
corresponding pin, if the pin is configured as a digital input.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
```

GPIOC_PDDR	0x0000000f

00000000000000000000000000001111

PORTE_PCR0	0x00000105
PORTE_PCR1	0x00000105

PORTE_PCR3	0x00000105
PORTE_PCR4	0x00000105
PORTE_PCR5	0x00000105

PORTE_PCR20	0x00000005

For PORTE_PCR0, PORTE_PCR1, PORTE_PCR3 to PORTE_PCR5 the configuration of the registers is the next.

```
PORTE_PCR0 = 105

Pin Control Register n

Bit Field Values:
         bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
         bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
         bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
         bits[  7:7  ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
         bits[  5:5  ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
         bits[  3:3  ] = 0
    SRE  bits[  2:2  ] = 1  Slow slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 0  Internal pullup or pulldown resistor is not enabled on the
corresponding pin.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

0000000	0	0000	0000	00000	001	0	0	0	0	1	0	1

For PORTE_PCR20 the configuration of the registers is the next.

```
PORTE_PCR20 = 5

Pin Control Register n

Bit Field Values:
         bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
         bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 0  Interrupt/DMA request disabled.
         bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 0  Pin disabled (analog).
         bits[  7:7  ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
         bits[  5:5  ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
         bits[  3:3  ] = 0
    SRE  bits[  2:2  ] = 1  Slow slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 0  Internal pullup or pulldown resistor is not enabled on the
corresponding pin.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
pin, if the corresponding Port Pull Enable field is set.
```

0000000	0	0000	0000	00000	000	0	0	0	0	0	1	0	1

ADC0_CFG1	0x00000054

```
ADC0_CFG1 = 54

ADC Configuration Register 1

Bit Field Values:
          bits[ 31:8  ] = 0
    ADLPC  bits[  7:7  ] = 0    Normal power configuration.
    ADIV   bits[  6:5  ] = 2    The divide ratio is 4 and the clock rate is (input
clock)/4.
    ADLSMP bits[  4:4  ] = 1    Long sample time.
    MODE   bits[  3:2  ] = 1    When DIFF=0:It is single-ended 12-bit conversion ;
when DIFF=1, it is differential 13-bit conversion with 2's complement output.
    ADICLK bits[  1:0  ] = 0    Bus clock
```

00000000000000000000000	0	10	1	01	00

NVIC_ISER	0x40000000

01000000000000000000000000000000

PORTA_PCR1	0x000a0107


```
PORTA_PCR1 = 655623

Pin Control Register n

Bit Field Values:
        bits[ 31:25 ] = 0
    ISF  bits[ 24:24 ] = 0  Configured interrupt is not detected.
        bits[ 23:20 ] = 0
    IRQC bits[ 19:16 ] = 10 Interrupt on falling edge.
        bits[ 15:11 ] = 0
    MUX  bits[ 10:8  ] = 1  Alternative 1 (GPIO).
        bits[  7:7   ] = 0
    DSE  bits[  6:6  ] = 0  Low drive strength is configured on the corresponding pin,
if pin is configured as a digital output.
        bits[  5:5   ] = 0
    PFE  bits[  4:4  ] = 0  Passive input filter is disabled on the corresponding pin.
        bits[  3:3   ] = 0
    SRE  bits[  2:2  ] = 1  Slow slew rate is configured on the corresponding pin, if
the pin is configured as a digital output.
    PE   bits[  1:1  ] = 1  Internal pullup or pulldown resistor is enabled on the
corresponding pin, if the pin is configured as a digital input.
    PS   bits[  0:0  ] = 1  Internal pullup resistor is enabled on the corresponding
```

GPIOA_PDDR	0x00000000

```
GPIOA_PDDR = 0

Port Data Direction Register

Bit Field Values:
    PDD bits[ 31:0  ] = 0   Pin is configured as general-purpose input, for the GPIO
function.
```

Video and Photos:

https://youtu.be/BUrXcrlOQwY