Universidad del Valle de Guatemala
Electrónica Digital 2 – Sección 11
Pablo Mazariegos

Francisco Montúfar
Carné 19379
06/03/2021

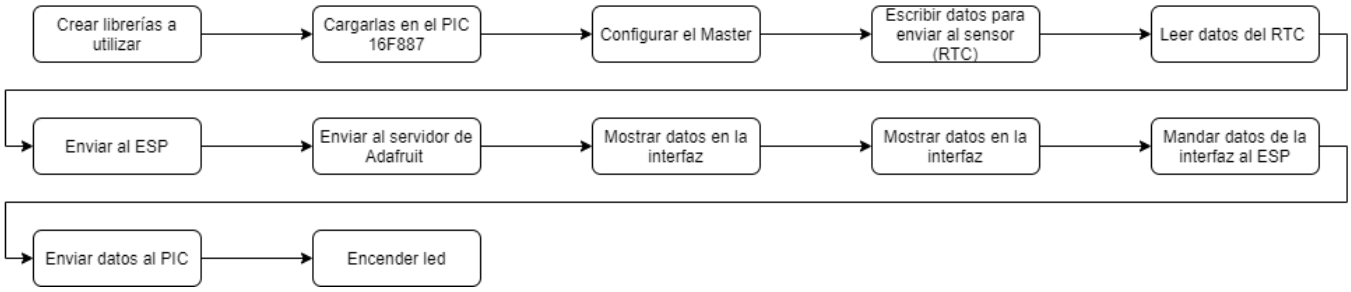
Mini Proyecto #2

I2C

Link de GitHub: https://github.com/mon19379/DIGITAL2.git

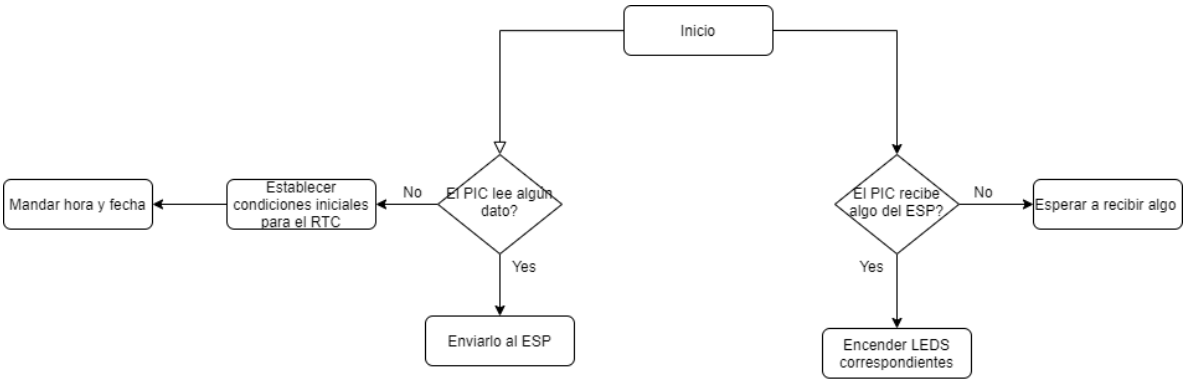
Link Video: https://youtu.be/qrefphhsG9k

Diagrama de flujo:

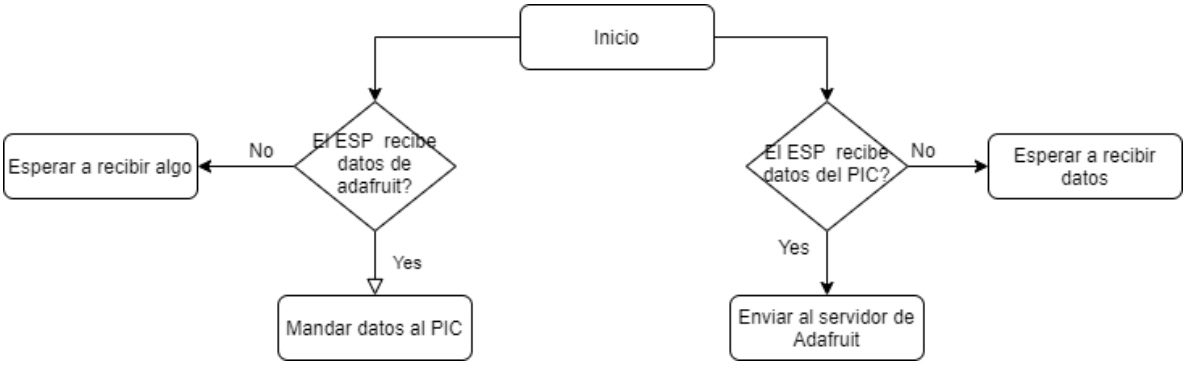


Pseudocódigo:

PIC:



ESP:



Código:

Master:

```
/*
 * File:   newmain.c
 * Author: franc
 *
 *
 */



#include <xc.h>
#include <stdint.h>
#include <pic16f887.h>
#include "I2C.h"
#include "OSC.h"
#include "usart.h"

//******************************************************************************
// Palabra de configuración
```

```
//*****************************************************************************

// CONFIG1

#pragma config FOSC = INTRC_NOCLKOUT        // Oscillator Selection bits (XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN)

#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)

#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)

#pragma config MCLRE = OFF      // RE3/MCLR pin function select bit (RE3/MCLR pin function is digital input, MCLR internally tied to VDD)

#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)

#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)

#pragma config BOREN = OFF      // Brown Out Reset Selection bits (BOR disabled)

#pragma config IESO = OFF       // Internal External Switchover bit (Internal/External Switchover mode is disabled)

#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)

#pragma config LVP = OFF        // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must be used for programming)


// CONFIG2

#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)

#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)

#define _XTAL_FREQ 8000000      //SE CONFIGURA EL OSCILADOR EXTERNO


//*****************************************************************************

//Variables

//*****************************************************************************

uint8_t CONT = 0;

uint8_t SEND = 0;

uint8_t SEC = 0;

uint8_t MIN = 0;

uint8_t H = 0;
```

```
uint8_t DAY = 0;
uint8_t MONTH = 0;
uint8_t YEAR = 0;
uint8_t NADA = 0;
uint8_t SECU = 0;
uint8_t SECD = 0;
uint8_t MINU = 0;
uint8_t MIND = 0;
uint8_t HORAU = 0;
uint8_t HORAD = 0;
uint8_t DAYU = 0;
uint8_t DAYD = 0;
uint8_t MONTHU = 0;
uint8_t MONTHD = 0;
uint8_t YEARU = 0;
uint8_t YEARD = 0;
uint8_t SU = 0;
uint8_t SD = 0;
uint8_t MU = 0;
uint8_t MD = 0;
uint8_t HU = 0;
uint8_t HD = 0;
uint8_t DU = 0;
uint8_t DD = 0;
uint8_t MOU = 0;
uint8_t MOD = 0;
uint8_t YU = 0;
uint8_t YD = 0;
uint8_t TOG = 0;
uint8_t FLAG = 0;

//*****************************************************************************
```

```
// Prototipos de funciones
//*****************************************************************************
void Setup(void);
void mandar(void);
void timeout(void);
void timein(void);
void conver(void);
void recibir(void);

//*****************************************************************************
//Interrupción
//*****************************************************************************

void __interrupt() ISR(void) {
   if (INTCONbits.T0IF == 1) { //INTERRUPCION DEL TIMER 0
      TMR0 = 236; //SE AGREGA VALOR AL TIMER 0
      CONT++; //SE INCREMENTA UN CONTADOR
      INTCONbits.T0IF = 0; //SE APAGA LA BANDERA
   }

   if (FLAG == 1) { //CONDICION CON BANDERA MANDADA POR EL ESP
      if (PIR1bits.TXIF == 1) { //INTERRUPCION DEL TX
         mandar(); //FUNCION DE ENVIAR DATOS
         SEND++; //SE INCREMENTA UN CONTADOR
         PIR1bits.TXIF = 0; //SE APAGA LA BANDERA
         PIE1bits.TXIE = 0; //SE APAGA LA INTERRUPCION
      }
   }
   if (PIR1bits.RCIF == 1) { //INTERRUPCION DEL RX
      TOG = RCREG; //SE METE EL VALOR DEL REGISTRO EN LA VARIABLE
      recibir(); // RUTINA DE RECIBIR DATOS
   }
```

```
}
//*****************************************************************************
//Ciclo pincipal
//*****************************************************************************

void main(void) {

    Setup();
    timeout(); //FUNCION DE ESCRITURA

    //*****************************************************************************
    // Loop principal
    //*****************************************************************************
    while (1) {

        timein(); //FUNCION DE LECTURA
        conver(); //CONVERSION DE LOS DATOS
        if (CONT > 30) { //FUNCION PARA QUE SE HABILITE EL TX CADA CIERTO TIEMPO
            CONT = 0;
            PIE1bits.TXIE = 1;
        }
    }

}
//*****************************************************************************
//Configuracion
//*****************************************************************************

void Setup(void) {
    initOsc(6); //CONFIGURACIÓN DEL OSCILADOR
    usart(); //CONFIGURACION DEL TX Y RX
    TRISA = 0; //PUERTO A SALIDAS
```

```
    TRISB = 0; //PUERTO B SALIDAS
    TRISD = 0; //PUERTO D SALIDAS
    TRISE = 0; //PUERTO E SALIDAS
    ANSEL = 0; // ENTRADAS DIGITALES Y BIT 0 ANALÓGICA
    ANSELH = 0;
    PORTA = 0; //PUERTO A EN 0
    PORTB = 0; //PUERTO B EN 0
    PORTC = 0; //PUERTO C EN 0
    PORTD = 0; //PUERTO D EN 0
    PORTE = 0; //PUERTO E EN 0
    //PINES RA0 Y RA2 COMO ENTRADAS, LOS DEMAS COMO SALIDAS
    OPTION_REG = 0b10000111; //SE APAGAN LAS PULLUPS DEL PUERTO B
    INTCONbits.GIE = 1; //SE HABILITAN LAS INTERRUPCIONES GLOBALES
    INTCONbits.PEIE = 1; //SE HABILITAN LAS INTERRUPCIONES PERIFERICAS
    PIR1bits.TXIF = 0; //SE LIMPIA LA BANDERA DE INTERRUPCION DEL TX
    PIE1bits.TXIE = 1; //SE HABILITA LA INTERRUPCION DEL TX
    INTCONbits.T0IF = 0; // SE LIMPIA LA BANDERA DE INTERRUPCION DEL TIMER 0
    INTCONbits.T0IE = 1; //SE HABILITA LA INTERRUPCION DEL TIMER0
    PIE1bits.RCIE = 1; //SE HABILITA LA INTERRUPCION DEL RX
    PIR1bits.RCIF = 0; //SE LIMPIA LA BANDERA DE INTERRUPCION DEL RX
    I2C_Master_Init(100000); //INICIALIZACION DEL MASTER


}

//*****************************************************************************
// Subrutinas
//*****************************************************************************

void mandar(void) {
    switch (SEND) {
        case 0:
```

```
    TXREG = 0x20; //ENVIAR UN ESPACIO
    break;
case 1:
    TXREG = HD; //ENVIAR DECENAS DE HORA
    break;
case 2:
    TXREG = HU; //ENVIAR UNIDADES DE HORA
    break;

case 3:
    TXREG = 0x3A; //ENVIAR :
    break;

case 4:
    TXREG = MD; //ENVIAR DECENAS DE MINUTO
    break;
case 5:
    TXREG = MU; //ENVIAR UNIDADES DE MINUTO
    break;
case 6:
    TXREG = 0x3A; //ENVIAR :
    break;
case 7:
    TXREG = SD; //ENVIAR DECENAS DE SEGUNDO
    break;

case 8:
    TXREG = SU; //ENVIAR UNIDADES DE SEGUNDO
    break;

case 9:
    TXREG = 0x20; //ENVIAR UN ESPACIO
```

```
    break;

case 10:
    TXREG = DD; //ENVIAR DECENAS DIA
    break;

case 11:
    TXREG = DU; //ENVIAR UNIDADES DIA
    break;

case 12:
    TXREG = 0x2F; //ENVIAR /
    break;
case 13:
    TXREG = MOD; //ENCIAR DECENAS MES
    break;

case 14:
    TXREG = MOU; //ENVIAR UNIDADES MES
    break;
case 15:
    TXREG = 0x2F; //ENVIAR /
    break;

case 16:
    TXREG = YD; //ENVIAR DECENAS AÑO
    break;

case 17:
    TXREG = YU; //ENVIAR UNIDADES AÑO
    break;
```

```
        case 18:
            TXREG = 0x0A; //ENVIAR ENTER
            SEND = 0;
            break;
    }
}

void timeout(void) {
    I2C_Master_Start(); //FUNCION PARA INICIAR LA ESCRITURA EN EL SENSOR
    I2C_Master_Write(0xD0); //SE INDICA QUE ESTA EN MODO ESCRITURA
    I2C_Master_Write(0); //SE LE DA UN VALOR AL PC DEL SENSOR
    I2C_Master_Write(0b00000000); //SE CONFIGURAN SEGUNDOS
    I2C_Master_Write(0b00000000); //SE CONFIGURAN MINUTOS
    I2C_Master_Write(0b00000000); // SE CONFIGURAN HORAS
    I2C_Master_Write(1); //SE OMITE
    I2C_Master_Write(0x20); //SE CONFIGURA DIA
    I2C_Master_Write(0x10); //SE CONFIGURA MES
    I2C_Master_Write(0x21); //SE CONFIGURA AÑO
    I2C_Master_Stop(); //FUNCION PARA DETENER LA ESCRITURA EN EL SENSOR

}

void timein(void) {
    I2C_Master_Start(); //FUNCION PARA INICIAR LA ESCRITURA EN EL SENSOR
    I2C_Master_Write(0xD0); //SE INDICA QUE ESTA EN MODO ESCTITURA
    I2C_Master_Write(0); //SE LE DA UN VALOR INICIAL AL PC DEL SENSOR

    I2C_Master_Start(); //FUNCION PARA INICIAR LA LECTURA EN EL SENSOR
    I2C_Master_Write(0xD1); //SE INDICA QUE ESTA EN MODO ESCRITURA
    SEC = I2C_Master_Read(1); //SE LEEN LOS SEGUNDOS
    MIN = I2C_Master_Read(1); // SE LEEN LOS MINUTOS
    H = I2C_Master_Read(1); //SE LEEN LAS HORAS
```

```
    NADA = I2C_Master_Read(1); //SE OMITE
    DAY = I2C_Master_Read(1); //SE LEEN DIAS
    MONTH = I2C_Master_Read(1); //SE LEEN MESES
    YEAR = I2C_Master_Read(0); //SE LEEN AÑOS
    I2C_Master_Stop(); //FUNCION PARA DETENER LA LECTURA EN EL SENSOR




}

void conver(void) {
    SECU = (SEC & 0b00001111); //SE MANIPULAN LOS SEGUNDOS PARA SEPARAR EN
    SECD = ((SEC & 0b11110000) >> 4); //DECENAS Y UNIDADES
    MINU = (MIN & 0b00001111); //SE MANIPULAN LOS MINUTOS PARA SEPARAR EN
    MIND = ((MIN & 0b11110000) >> 4); //DECENAS Y UNIDADES
    HORAU = (H & 0b00001111); //SE MANIPULAN LAS HORAS PARA SEPARAR EN
    HORAD = ((H & 0b00110000) >> 4); //DECEINAS Y UNIDADES
    DAYU = (DAY & 0b00001111); //SE MANIPULAN LOS DIAS PARA SEPARAR EN
    DAYD = ((DAY & 0b11110000) >> 4); //DECENAS Y UNIDADES
    MONTHU = (MONTH & 0b00001111); //SE MANIPULAN LOS MESES PARA SEPARAR EN
    MONTHD = ((MONTH & 0b11110000) >> 4); //DECENAS Y UNIDADES
    YEARU = (YEAR & 0b00001111); //SE MANIPULAN LOS AÑOS PARA SEPARAR EN
    YEARD = ((YEAR & 0b11110000) >> 4); //DECENAS Y UNIDADES



    SU = (SECU + 0x30);
    SD = (SECD + 0x30);
    MU = (MINU + 0x30);
    MD = (MIND + 0x30);
    HU = (HORAU + 0x30);
    HD = (HORAD + 0x30);
    DU = (DAYU + 0x30);
```

```
    DD = (DAYD + 0x30);
    MOU = (MONTHU + 0x30);
    MOD = (MONTHD + 0x30);
    YU = (YEARU + 0x30);
    YD = (YEARD + 0x30); //CONVERSION A ASCII DE TODAS LAS VARIABLES




}

void recibir(void) {
    if (TOG == 1) { //TOGGLE PARA LA ENVIADA DE DATOS
        FLAG = 1; //SE ACTIVA UNA BANDERA
        PORTAbits.RA0 = 0;
        PORTAbits.RA1 = 0; //SE APAGAN LEDS
    }

    if (TOG == 2) { //TOGGLE PARA ENCENDER UNA LED
        PORTAbits.RA0 = 1;
        PORTAbits.RA1 = 0;
    }
    if (TOG == 3) { //TOGGLE PARA ENCENDER UNA LED
        PORTAbits.RA1 = 1;
        PORTAbits.RA0 = 0;
    }

    if (TOG == 4) { //TOGGLE PARA ENCENDER AMBAS LEDS
        PORTAbits.RA0 = 1;
        PORTAbits.RA1 = 1;
    }
}
```

Librerías:

```
#include "I2C.h"
//*****************************************************************************
// Función para inicializar I2C Maestro
//*****************************************************************************
void I2C_Master_Init(const unsigned long c)
{
   SSPCON = 0b00101000;
   SSPCON2 = 0;
   SSPADD = (_XTAL_FREQ/(4*c))-1;
   SSPSTAT = 0;
   TRISCbits.TRISC3 = 1;
   TRISCbits.TRISC4 = 1;
}
//*****************************************************************************
// Función de espera: mientras se esté iniciada una comunicación,
// esté habilitado una recepción, esté habilitado una parada
// esté habilitado un reinicio de la comunicación, esté iniciada
// una comunicación o se este transmitiendo, el IC2 PIC se esperará
// antes de realizar algún trabajo
//*****************************************************************************
void I2C_Master_Wait()
{
   while ((SSPSTAT & 0x04) || (SSPCON2 & 0x1F));
}
//*****************************************************************************
// Función de inicio de la comunicación I2C PIC
//*****************************************************************************
void I2C_Master_Start()
{
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   SSPCON2bits.SEN = 1;    //inicia la comunicación i2c
```

```
}
//*****************************************************************************
// Función de reinicio de la comunicación I2C PIC
//*****************************************************************************
void I2C_Master_RepeatedStart()
{
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   SSPCON2bits.RSEN = 1;   //reinicia la comunicación i2c
}
//*****************************************************************************
// Función de parada de la comunicación I2C PIC
//*****************************************************************************
void I2C_Master_Stop()
{
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   SSPCON2bits.PEN = 1;    //detener la comunicación i2c
}
//*****************************************************************************
//Función de transmisión de datos del maestro al esclavo
//esta función devolverá un 0 si el esclavo a recibido
//el dato
//*****************************************************************************
void I2C_Master_Write(unsigned d)
{
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   SSPBUF = d;
}
//*****************************************************************************
//Función de recepción de datos enviados por el esclavo al maestro
//esta función es para leer los datos que están en el esclavo
//*****************************************************************************
unsigned char I2C_Master_Read(unsigned char a)
```

```
{
   unsigned short temp;
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   SSPCON2bits.RCEN = 1;
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   temp = SSPBUF;
   I2C_Master_Wait();      //espera que se cumplan las condiciones adecuadas
   if(a == 1){
      SSPCON2bits.ACKDT = 0;
   }else{
      SSPCON2bits.ACKDT = 1;
   }
   SSPCON2bits.ACKEN = 1;         // Iniciar sequencia de Acknowledge
   return temp;                   // Regresar valor del dato leído
}
//*****************************************************************************
// Función para inicializar I2C Esclavo
//*****************************************************************************
void I2C_Slave_Init(uint8_t address)
{
   SSPADD = address;
   SSPCON = 0x36;      // 0b00110110
   SSPSTAT = 0x80;     // 0b10000000
   SSPCON2 = 0x01;     // 0b00000001
   TRISC3 = 1;
   TRISC4 = 1;
   GIE = 1;
   PEIE = 1;
   SSPIF = 0;
   SSPIE = 1;
}
//*****************************************************************************
```

```
#include <pic16f887.h>
#include <xc.h>
#include "OSC.h"
//******************************************************************************
//Inicialización del oscilador interno pg. 62
//******************************************************************************
void initOsc(uint8_t IRCF){

   switch (IRCF){

      case 0: //OSCILADOR DE 31 kHz
         OSCCONbits.IRCF2 = 0;
         OSCCONbits.IRCF1 = 0;
         OSCCONbits.IRCF0 = 0;

         break;

      case 1: //OSCILADOR DE 125 kHz
         OSCCONbits.IRCF2 = 0;
         OSCCONbits.IRCF1 = 0;
         OSCCONbits.IRCF0 = 1;
         break;

      case 2: //OSCILADOR DE 250 kHz
         OSCCONbits.IRCF2 = 0;
         OSCCONbits.IRCF1 = 1;
         OSCCONbits.IRCF0 = 0;
         break;

      case 3: //OSCILADOR DE 500kHz
```

```
    OSCCONbits.IRCF2 = 0;
    OSCCONbits.IRCF1 = 1;
    OSCCONbits.IRCF0 = 1;
    break;

case 4: //OSCILADOR DE 1MHz
    OSCCONbits.IRCF2 = 1;
    OSCCONbits.IRCF1 = 0;
    OSCCONbits.IRCF0 = 0;
    break;

case 5: //OSCILADOR DE 2MHz
    OSCCONbits.IRCF2 = 1;
    OSCCONbits.IRCF1 = 0;
    OSCCONbits.IRCF0 = 1;
    break;


case 6: //OSCILADOR DE 4MHz
    OSCCONbits.IRCF2 = 1;
    OSCCONbits.IRCF1 = 1;
    OSCCONbits.IRCF0 = 0;
    break;

case 7: //OSCILADOR DE 8MHz
    OSCCONbits.IRCF2 = 1;
    OSCCONbits.IRCF1 = 1;
    OSCCONbits.IRCF0 = 1;
    break;
```

```
        default: //OSCILADOR DE 4MHz
            OSCCONbits.IRCF2 = 1;
            OSCCONbits.IRCF1 = 1;
            OSCCONbits.IRCF0 = 0;
            break;

    }

    OSCCONbits.SCS = 1; //SE VA A USAR EL OSCILADOR INTERNO


}

#include <pic16f887.h>

#include "usart.h"

void usart(void){

    //CONFIG TX
    TXSTAbits.TX9 = 0; //TRANSMISION DE 8 BITS
    TXSTAbits.SYNC = 0; //ASINCRONO
    TXSTAbits.BRGH = 1; //HIGH SPEED
    BAUDCTLbits.BRG16 = 0; //BAUD RATE DE 8 BITS
    SPBRGH = 0;
    SPBRG = 25;
    PIE1bits.TXIE = 1;
    TXSTAbits.TXEN = 1;

    //CONFIG RX
    RCSTAbits.SPEN = 1;
```

```
    RCSTAbits. RX9 = 0;
    RCSTAbits.CREN = 1;
}

/*
 * File            : I2C.h
 * Author          : Ligo George
 * Company         : electroSome
 * Project         : I2C Library for MPLAB XC8
 * Microcontroller : PIC 16F877A
 * Created on April 15, 2017, 5:59 PM
 * Link: https://electrosome.com/i2c-pic-microcontroller-mplab-xc8/
 * Modificada por: Pablo Mazariegos con la ayuda del auxiliar Gustavo Ordoñez
 * Basado en Link: http://microcontroladores-mrelberni.com/i2c-pic-comunicacion-serial/
 */

// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef __I2C_H
#define __I2C_H

#include <xc.h> // include processor files - each processor file is guarded.
#include <pic16f887.h>
#include <stdint.h>

#ifndef _XTAL_FREQ
#define _XTAL_FREQ 4000000
#endif

//*****************************************************************************
// Función para inicializar I2C Maestro
//*****************************************************************************
```

```
void I2C_Master_Init(const unsigned long c);
//*****************************************************************************
// Función de espera: mientras se esté iniciada una comunicación,
// esté habilitado una recepción, esté habilitado una parada
// esté habilitado un reinicio de la comunicación, esté iniciada
// una comunicación o se este transmitiendo, el IC2 PIC se esperará
// antes de realizar algún trabajo
//*****************************************************************************
void I2C_Master_Wait(void);
//*****************************************************************************
// Función de inicio de la comunicación I2C PIC
//*****************************************************************************
void I2C_Master_Start(void);
//*****************************************************************************
// Función de reinicio de la comunicación I2C PIC
//*****************************************************************************
void I2C_Master_RepeatedStart(void);
//*****************************************************************************
// Función de parada de la comunicación I2C PIC
//*****************************************************************************
void I2C_Master_Stop(void);
//*****************************************************************************
//Función de transmisión de datos del maestro al esclavo
//esta función devolverá un 0 si el esclavo a recibido
//el dato
//*****************************************************************************
void I2C_Master_Write(unsigned d);
//*****************************************************************************
//Función de recepción de datos enviados por el esclavo al maestro
//esta función es para leer los datos que están en el esclavo
//*****************************************************************************
unsigned char I2C_Master_Read(unsigned char a);
```

```
//******************************************************************************
// Función para inicializar I2C Esclavo
//******************************************************************************
void I2C_Slave_Init(uint8_t address);
//******************************************************************************
#endif   /* __I2C_H */



// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef OSC_H
#define OSC_H

#include <xc.h> // include processor files - each processor file is guarded.
#include <stdint.h>

void initOsc(uint8_t IRCF);



#endif   /* OSC_H */

#ifndef USART_H
#define USART_H

#include <xc.h> // include processor files - each processor file is guarded.
#include <stdint.h>
void usart(void);


#endif   /* USART_H */
```

```
#include "config.h"
char t[17];
int flag = 0;
int Bot1 = 0;
int Bot2 = 0;
int flagl = 0;
int flagl2 = 0;

AdafruitIO_Feed *Tiempo = io.feed("Tiempo");
AdafruitIO_Feed *Led1 = io.feed("Led1");
AdafruitIO_Feed *Led2 = io.feed("Led2");

void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
  io.connect(); //se conecta al servidor
  Led1->onMessage(handleMessage);
  Led2->onMessage(handleMessage2);

  Serial2.begin(9600, SERIAL_8N1, 16, 17);
  while (io.status() < AIO_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.println(io.statusText());
  Led1->get();
  Led2->get();
}
```

```
void loop() {
  io.run();
  // send data only when you receive data:
  while (Serial2.available()) {
    // read the incoming byte:
    // t = Serial.print(char(Serial2.read()));
    Serial2.write(1);

    Serial2.readBytesUntil(10, t, 17);

      if(flagl == 1 && flagl2 == 0){
      Serial2.write(2);
    }

    else if(flagl == 0 && flagl2 == 1){
      Serial2.write(3);
    }

    else if(flagl == 1 && flagl2 == 1){
      Serial2.write(4);
    }
  }


  Serial.print("sending -> ");
  Serial.println(t);
  Serial.println(flagl);
  Serial.println(flagl2);
  delay(3000);


  Tiempo->save(t);
```

```
}

void handleMessage(AdafruitIO_Data*data) {
  Bot1 = data->toInt();
  if (Bot1 == 0) {
    flagl = 0;
  }
  if (Bot1 == 1) {
    flagl = 1;
  }
}


void handleMessage2(AdafruitIO_Data*data) {
  Bot2 = data->toInt();
  if (Bot2 == 0) {
    flagl2 = 0;
  }
  if (Bot2 == 1) {
    flagl2 = 1;
  }
}
```