Proyecto#1: Monitoreo de Red de Sensores

Profesor: Pablo Mazariegos

Clase: Electrónica Digital 2

Estudiantes:

Rodrigo Figueroa 161678
Gonzalo Palarea 15559
Stefan Schwendener 151414

Sección: 20

Abstracto: **Proyecto donde se utilizan 7 PIC16F887 para controlar una red de 5 sensores y 5 motores. Se realiza en una banda movil el vaciado de parafina en un molde que luego sale de la banda. Los datos de los sensores se envian a una Raspberry Pi y se despliegan en un servidor de Adafruit IO.**

Introducción:

En este proyecto se pretende controlar 5 sensores independientes usando PIC16F887. Estos sensores deben comunicar valores a los microcontroladores, y estos en turno deben reportar el valor a un microcontrolador maestro utilizando el protocolo de I2C a través de 2 líneas de comunicación. El maestro luego debe de tomar estos valores y desplegarlos tanto en una LCD como en un portal en línea de Adafruit. Para manejar el portal se utiliza comunicacion SPI entre el microcontrolador maestro y una Raspberry Pi 3. Para este proyecto se tenía que idear un proceso donde se pudieran utilizar estos 5 sensores de una manera no trivial, para que se produzca un resultado medible y deseado. Para este propósito se escogió hacer una maquina que derrite parafina para la inclusión de muestras patológicas conocidas como 'microtomos'. Es importante resaltar que debido a la pandemia y el efecto que tuvo la misma en nuestro trabajo en grupo, hubieron limitantes que no nos permitieron completar todos los puntos de la evaluación. Hubieron varios problemas con el uso de la comunicación SPI de un PIC a la Raspberry Pi, por lo cual solo nos limitamos a programar el PIC que realiza la comunicación SPI y a simular sus salidas con luces LEDs

Propósito:

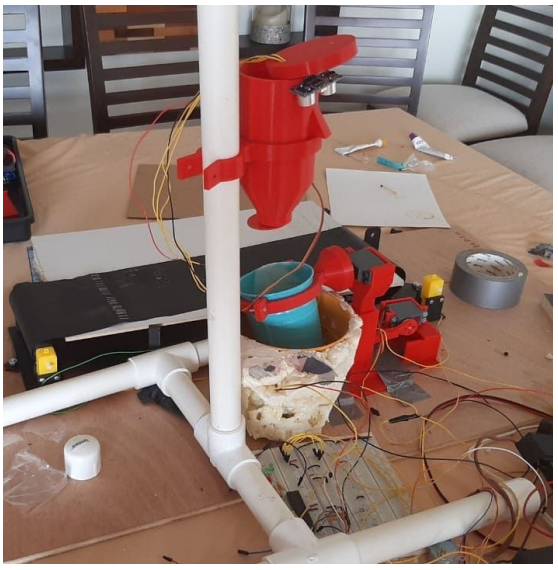
En la industria de la medicina, y específicamente en la patología, la velocidad de diagnostico se ve severamente limitada por el tiempo de procesamiento y empaquetado de muestras. Las muestras de tejido que se extraen para la detección de enfermedades tales como cáncer, pasan por un proceso donde se 'fosilizan' las muestras antes de ponerlas bajo un microscopio para poder hacer un diagnóstico. Esto se hace no sólo para preservar la muestra, sino que para obtener la capa más importante de la misma. Una capa tan fina que mide 4 nanometros de grosor. Las máquinas más modernas que realizan todo el proceso de manera automática valen de $30,000 hasta $50,000. Siendo una máquina que es más pequeña que una mesa de café, son precios extremadamente altos que van acompañados de la calidad que se espera de el equipo médico. Queríamos crear un equipo que no solo automatizará el proceso, sino que redujera los costos entregando resultados de la misma calidad. También se vería el valor de un equipo de sensores que sirvieran para crear un sistema de control.

Proceso:

Para encapsular las muestras se utiliza una tipo de parafina que se mezcla con otros polímeros para reducir el punto de fusión de la misma. Estas máquinas calientan la parafina en un tanque amplio y luego usando la gravedad y el peso de la parafina, hacen que esta salga como un chorro líquido que debe de caer en un molde con la muestra. Todo este proceso se hace de manera manual para controlar que el vaciado de la parafina sea parejo.

Nuestra máquina automatiza el proceso creando un sistema de control para derretir la parafina, agregando una cinta transportadora y una vertidora que siempre vierte la misma cantidad de parafina. El proceso enumerado sería el siguiente:

1. Encender la máquina, soltar una pequeña cantidad de parafina en el contenedor vertidor mientras que la resistencia térmica calienta el agua del contenedor más grande y derrite la parafina.
2. Esperar que una muestra cruce el sensor infrarrojo, el cual al activarse detiene el movimiento de la banda e inicia la cadena de servos
3. Se activa primero el servo que suelta parafina sólida en la cubeta para derretir la parafina que está por vertirse. Luego se activa el servo que levanta la cubeta vertidora por encima de la cubeta de calentamiento. Por último, se activa el servo que vuelca la cubeta y saca una volumen constante de parafina.
4. Por último el stepper se mueve 180 grados y distancia al molde del sensor infrarrojo, lo cual reactiva la banda y permite que el molde con parafina sea transportado hasta el final.



Sensores:

Profundidad:

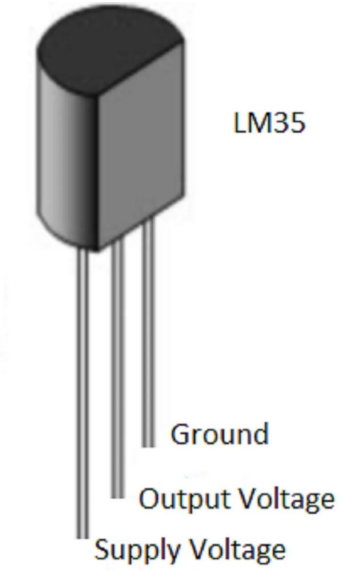
El sensor de Profundidad es un sensor que manda un valor analogico de 0-150 que indica la altura a la que se encuentra el agua. Este valor puede ser parametrizado para ofrecer una medida precisa

Incendios:

El sensor de incendios es un sensor que puede generar un valor analogico y un valor digital. El valor analogico permite medir la intensidad de la luz que detecta el sensor. El valor digital permite saber si el sensor detecta un incendio o no. El trigger de la detección se ajusta con el potenciómetro incluido. Al detectar un incendio, este manda un 1 lógico al microcontrolador.

Temperatura:

El sensor de temperatura LM35 es un sensor que incrementa en 1 mV su voltaje de salida por cada grado de temperatura que detecta. El valor analogico de este sensor se lee y se pasa por una ecuación dependiente de un voltaje de referencia (Vref) para poder representar la temperatura a la que está el sensor.



Infrarrojo:

El sensor infrarrojo es un sensor que utiliza una señal infrarroja para detectar movimiento. Si un objeto interrumpe la señal infrarroja, la señal regresa al sensor, activandolo. Este sensor procede a mandar una señal digital correspondiente a su estado de detección. Manda un 0 lógico si detecta un objeto, y manda un 1 lógico si no detecta un objeto. Se puede modificar la distancia de detección con el potenciómetro incluido.

Ultrasónico:

El sensor ultrasónico utiliza sonido para detectar movimiento o para medir distancias. Manda un pulso a través de la bocinas cuando se activa el pin de trigger (a una frecuencia muy alta, inaudible) y luego el pin de hecho se enciende cuando escucha el pulso reflejado. La salida de este sensor es una medida de cuán fuerte fue el pulso que se escuchó. Utilizando los timers del microcontrolador y tomando en consideración la velocidad del sonido, la salida de este sensor se puede convertir en una medida de distancia.

Motores:

Stepper:

Se utilizó un único motor stepper que se encarga de sacar la muestra rellena de parafina de la linea a un area de refrigeracion para permitir que la parafina se endurezca . El stepper mueve una pieza que mueve la muestra con sutileza, y luego con una rotación inversa, se regresa la pieza a su lugar .

DC:

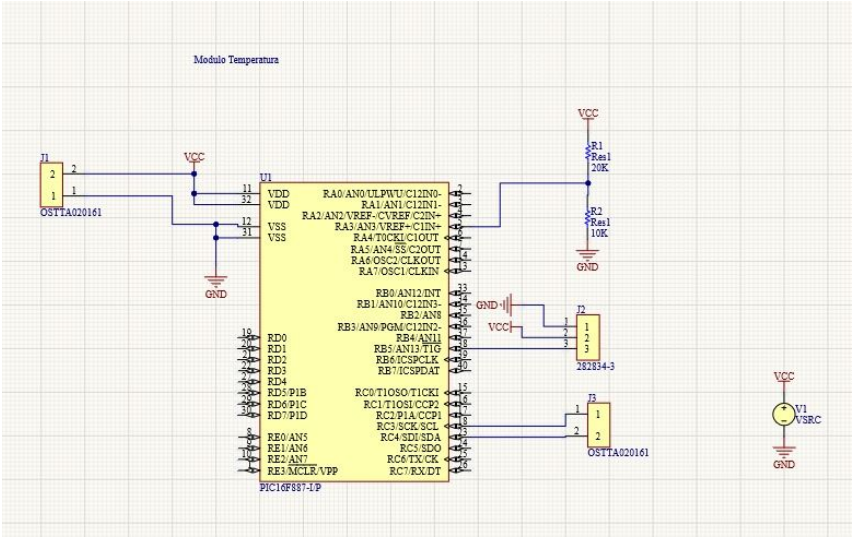
Se utilizaron un par de motores DC con caja reductora que controlan una banda transportadora en la que se mueven las muestras. Los motores se alimentan con una fuente de poder externa, pero se controla cuando están encendidos y apagados a través de la red de sensores.

Servo:

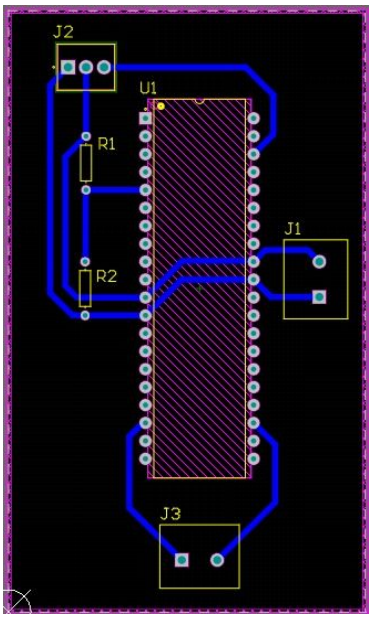
Se utilizaron 3 motores servo. Un motor servo que abre y cierra el cilindro que contiene las muestras de parafina en sólido. Un servo que que controla la altura a la que se encuentra la cubeta con la parafina derretida. Un último servo que controla el grado de inclinación de la cubeta con la parafina derretida para verterla.

Circuito:

Circuito de Sensor de Temperatura:

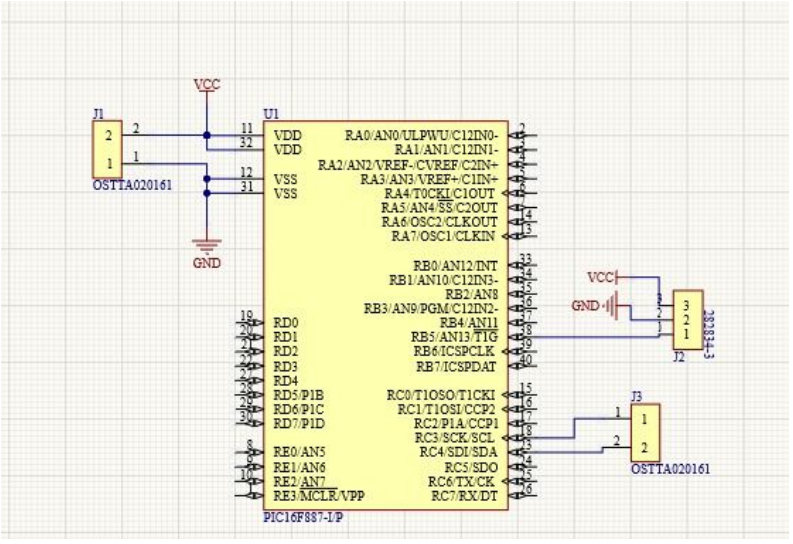


Figura#: Circuito de Sensor de Temperatura

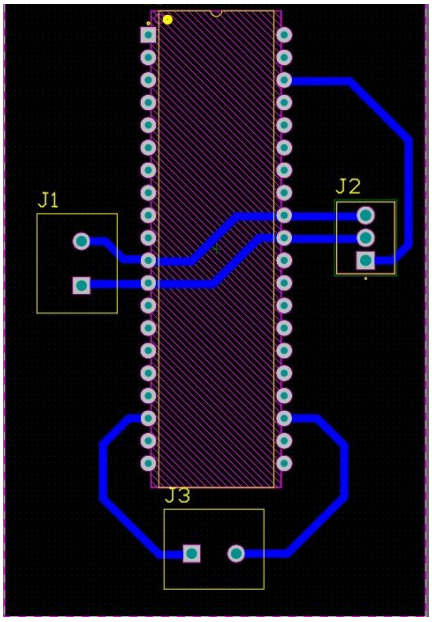


Figura#: Layout de Sensor de Temperatura

Circuito de Sensor de Profundidad:

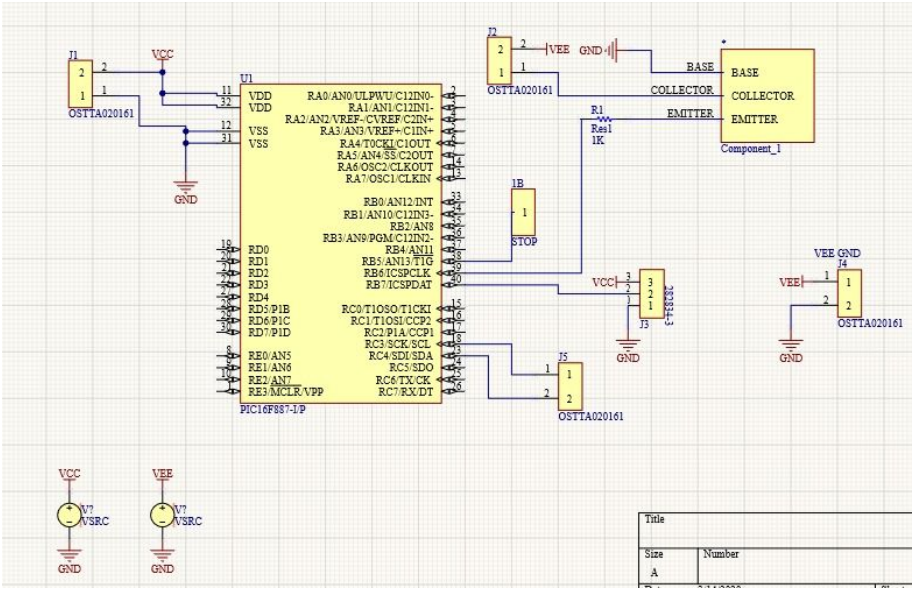


Figura#: Circuito de Sensor de Profundidad

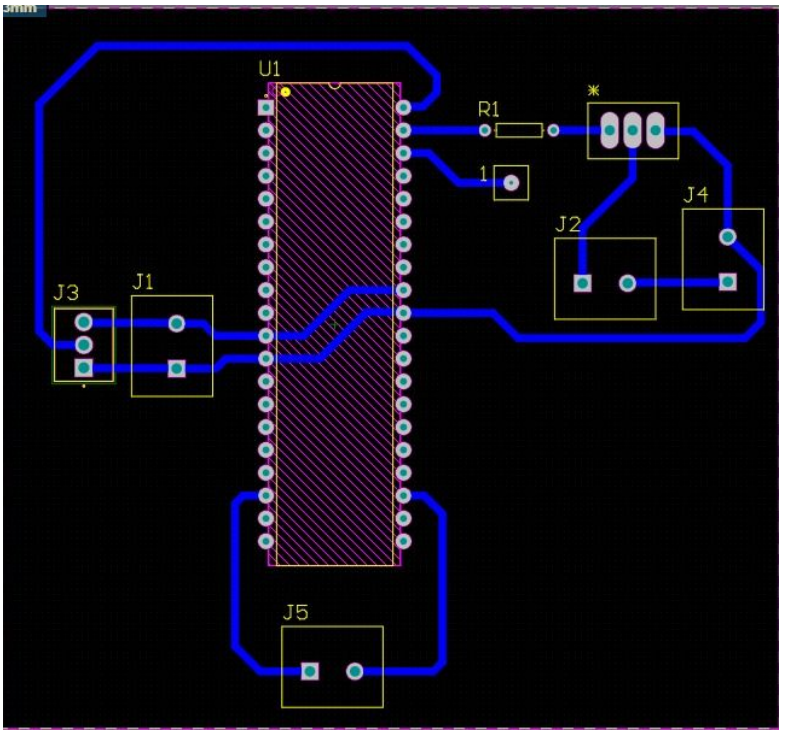


Figura#: Layout de Sensor de Profundidad

Circuito de Sensor Infrarrojo:

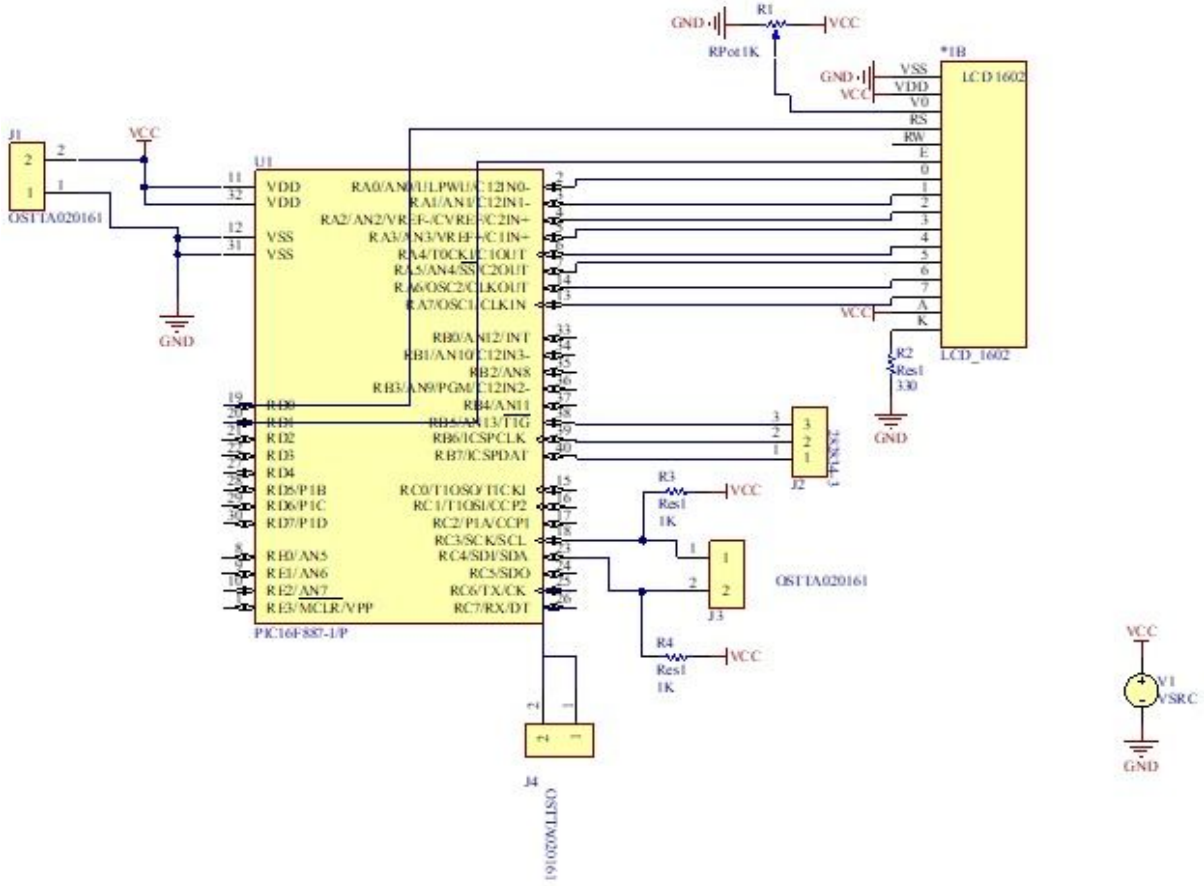


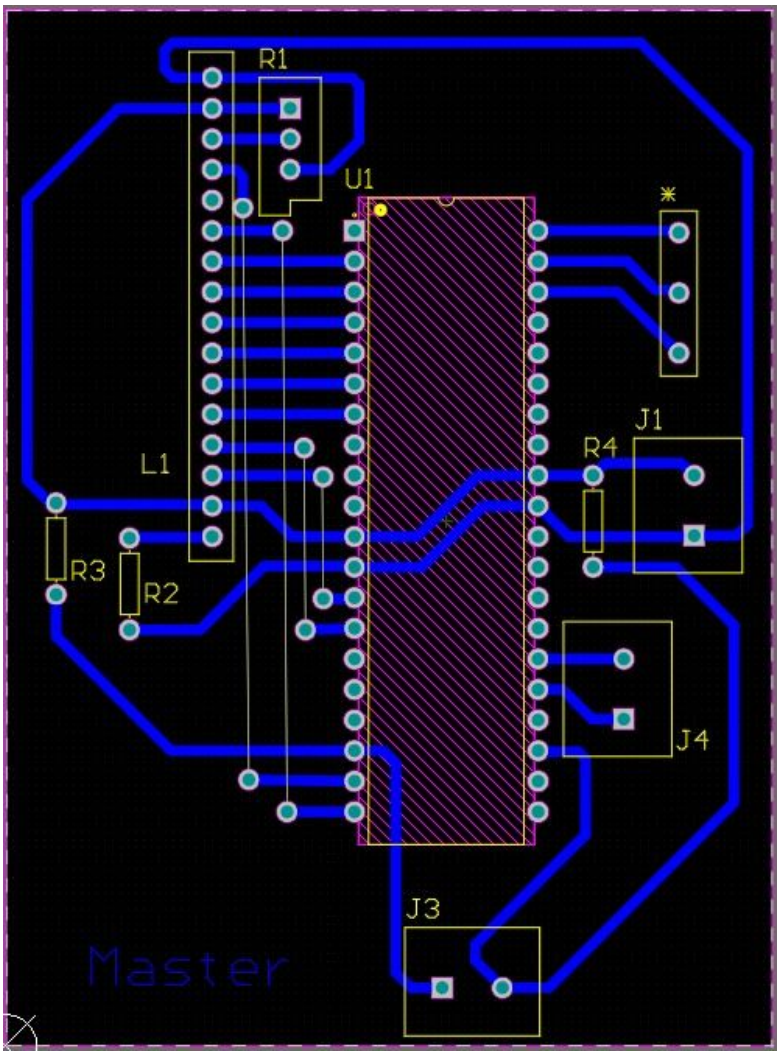
Figura#: Circuito de Sensor Infrarrojo



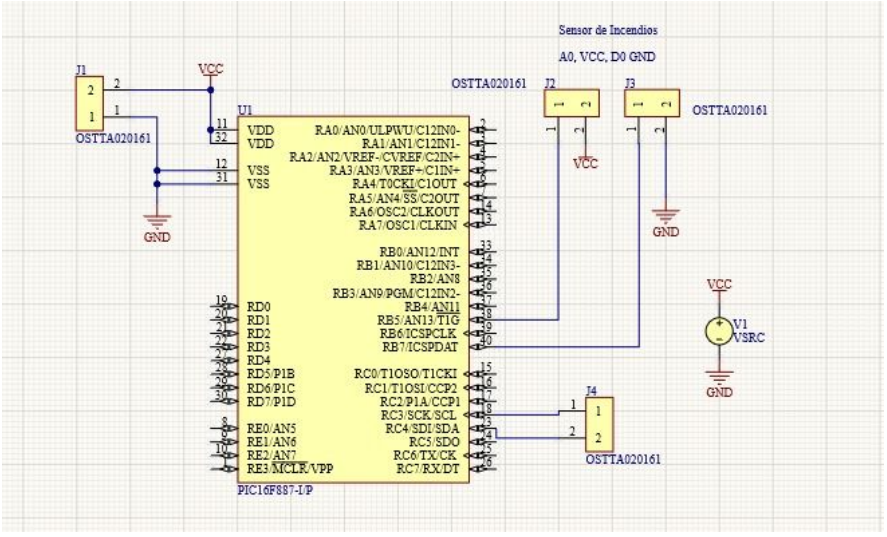
Figura#: Layout de Sensor Infrarrojo

Circuito de Máster:

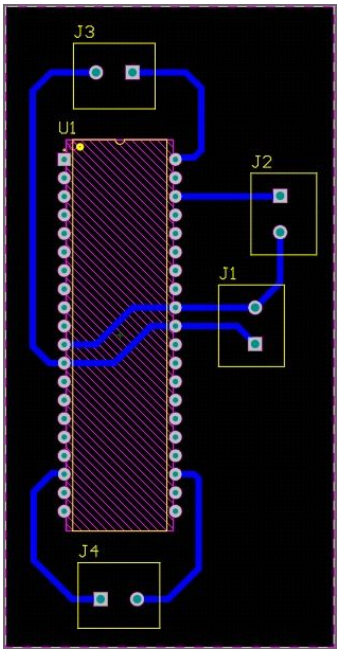




Circuito de Sensor de Incendios:

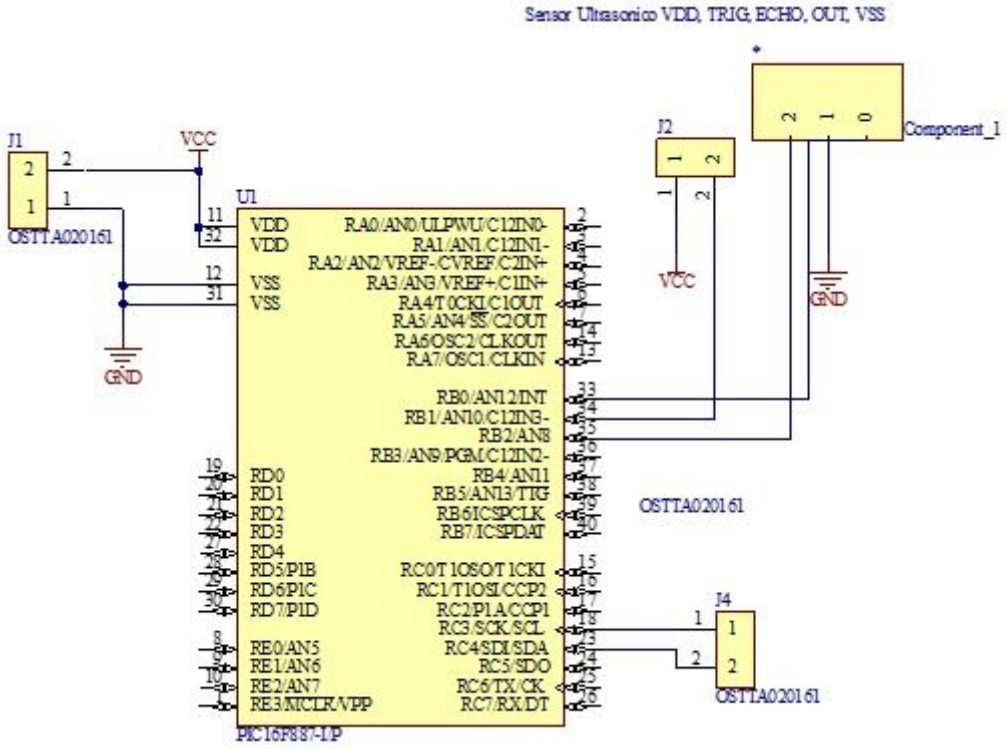


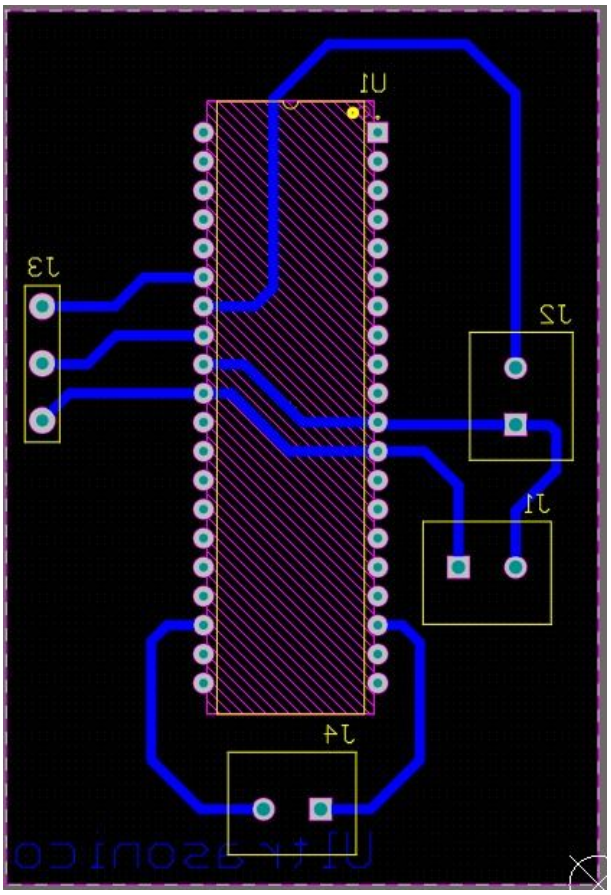
Figura#: Circuito de Sensor de Incendios



Figura#: Layout de Sensor de Incendios

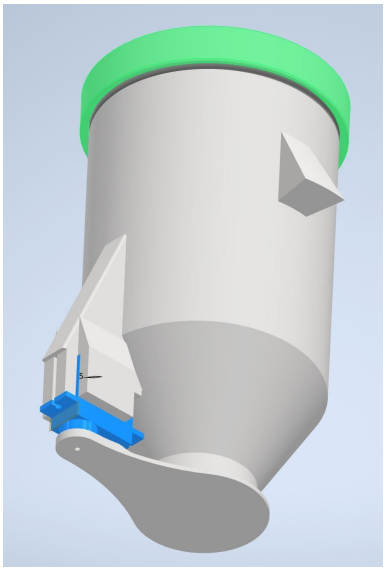
Circuito de Sensor Ultrasónico:



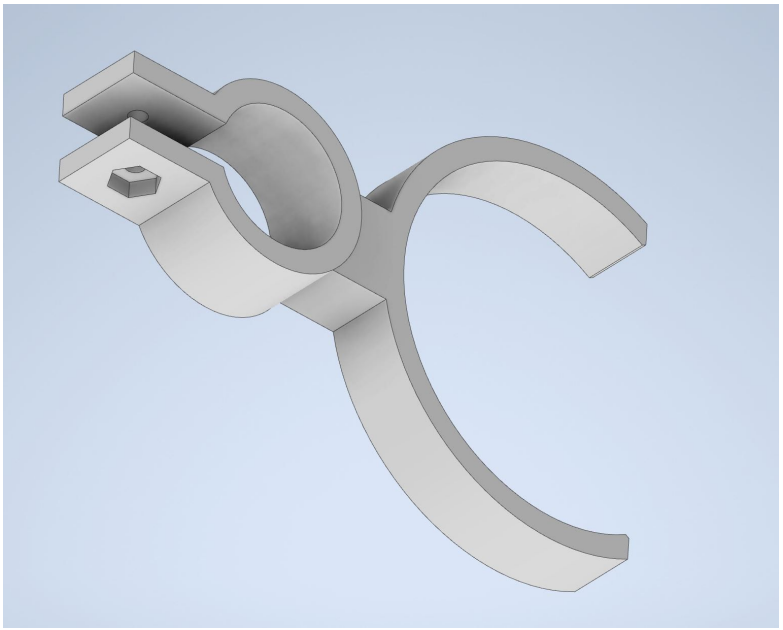

U1
J3
J2
J1
J4

Modelos CAD:

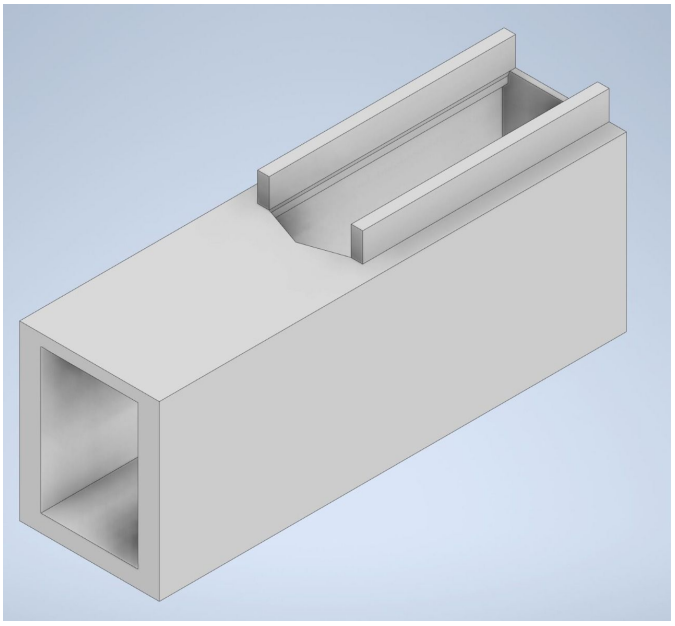
Vertidora de Parafina en Seco:



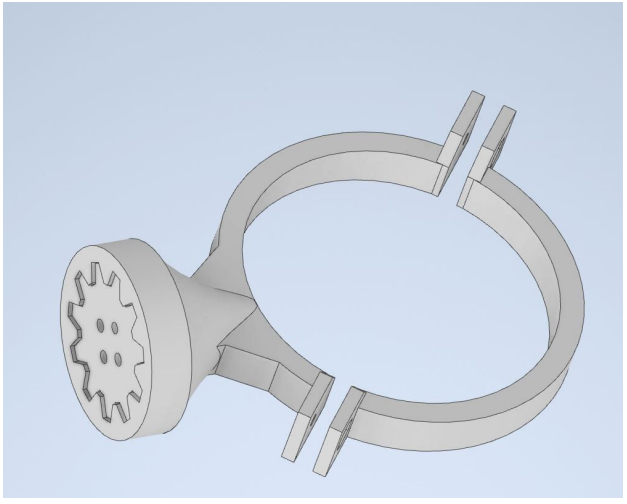
Agarradera para la vertedora en Seco:

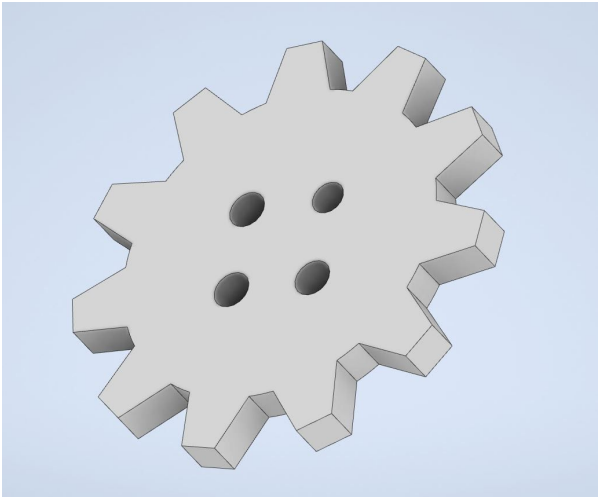


Servo que eleva el contenedor de parafina

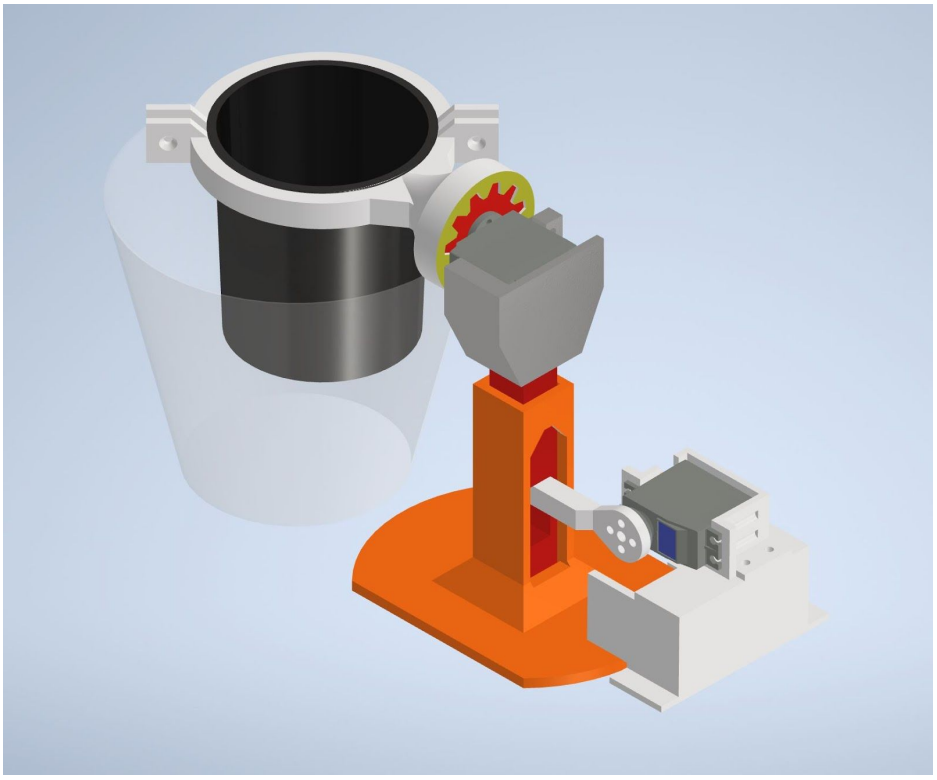


Adaptador para servo que vuelca el contenedor de parafina

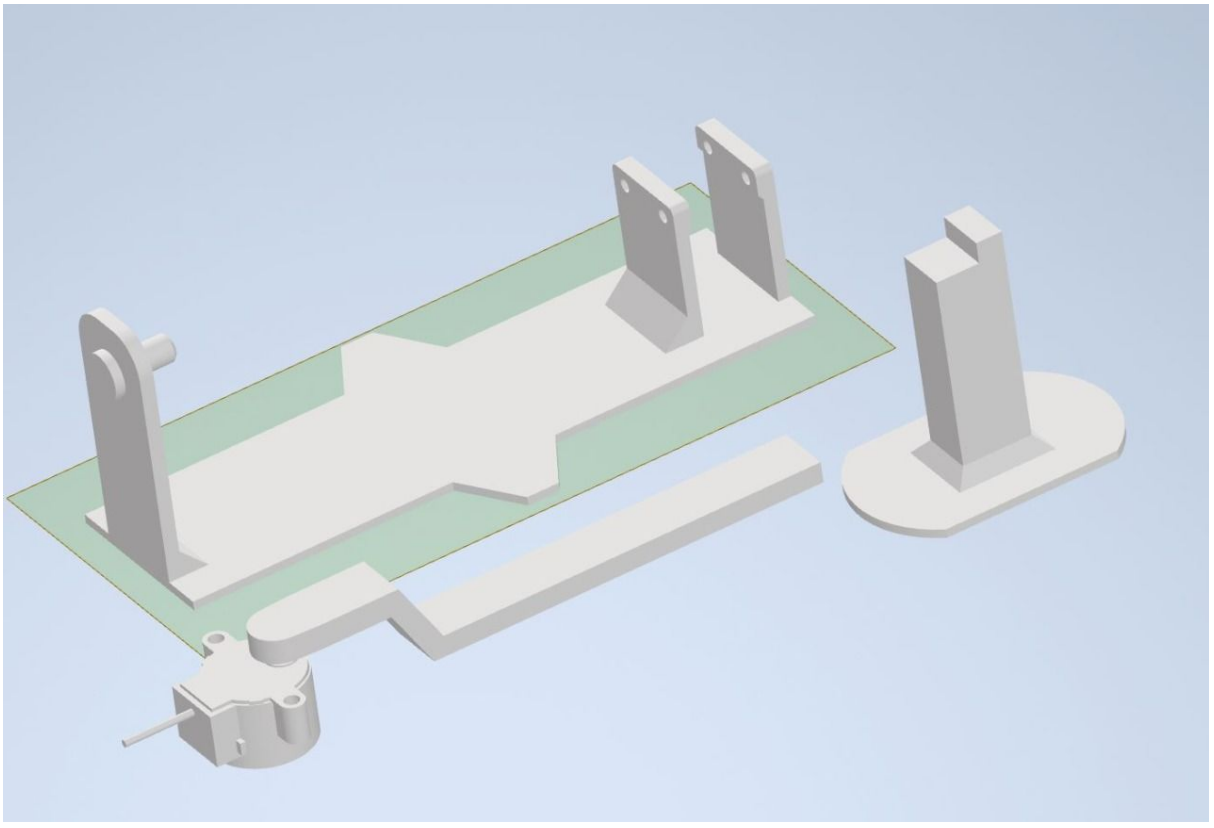




Ensamblaje de los componentes



Banda Transportadora y Brazo de Stepper



Códigos:

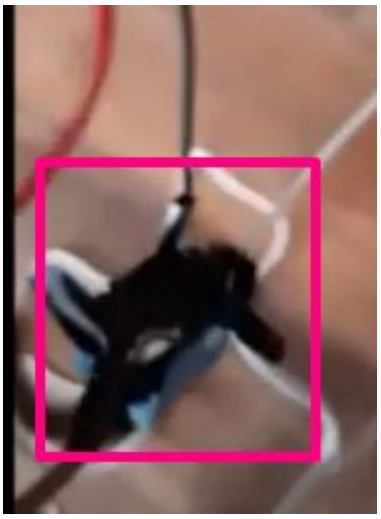
Código de Máster:

```
/*****************************************
   * File:    Projecto_1_Master.c               *
    * Date:    23/02/2020                  *
    * Autores:  Rodrigo Figueroa, Gonzalo Palarea, Stefan Schwendenner            *
    * Prof:     Pablo Mazariegos           *
    * Seccion:     20                      *
    * Clase:   Digital 2                   *
    * Compiler:    MPLAB XC8 v2.10         *
    * Arquitectura:    PIC16F887           *
    * Descripcion:  Projecto 1 de Digital, en donde este PIC atravez de I2C recibe
    * Datos de 5 sensores, los despliega en una LCD, y los Manda atravez de UART a otro PIC
    *
    *
    * Asignacion de Pins:
    * Puerto B = Pines de Control para la banda, RB7,RB6,RB5
    *
    * Puerto A = Salida de Pines D0-D7 de la LCD
    *
    * Puerto C = Salida de Clock y entrada de datos de los Slaves, Salida de UART
    *RC7,RC6,RC4,RC3
    *
    * Puerto D = Pines RS y E de la LCD
    *
    *
    *
    *
    * Link al Github: https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git
    *
    ****************************************/

//******************************************************************************
// Bits de Configuracion
//******************************************************************************
 #pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)
#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)

//******************************************************************************
// Declaracion de Librerias
//******************************************************************************
```

```
#include <xc.h>
#include "ProjectoMaster.h"
//*****************************************************************************
// Declaracion de Variables
//*****************************************************************************
uint8_t Profundidad; // Address 0x10
uint8_t Temperatura; // Address 0x20
uint8_t Fuego;       // Address 0x30
uint8_t Distancia;   // Address 0x40
uint8_t Luz;         // Address 0x50
char d1,d2,d3;
double Rtemp;
//*****************************************************************************
// Subrutina de Inicio
//*****************************************************************************
void Start(){
  TRISA = 0;//PORTA as output
  TRISD = 0;//PORTD as output
  PORTA = 0;//Inicializar Puerto A
  PORTD = 0;
  TRISB =  0;
  PORTB = 0;
  I2C_Master_Init(100000);      //Initialize I2C Master with 100KHz clock
  UARTInit(9600,1); //Iniciar UART en baudrate de 9600, en HS mode
  lcd_init();
  lcd_clear();

  __delay_ms(250);
}



void main(){
    Start();
    lcd_clear();
    while(1){

//*****************************************************************************
// 1-Declaramos Las ubicaciones para todas Las variables y nombres de los Sensores
//*****************************************************************************
      lcd_cursor(1,1);// Poner texto de cursor en posicion 1
      lcd_palabra("Agua");
      __delay_ms(10);
      lcd_cursor(1,8);// Poner texto de cursor en posicion 2
      lcd_palabra("Temp");
      __delay_ms(10);
      lcd_cursor(1,13);// Poner texto de cursor en posicion 3
      lcd_palabra("Fire");
      __delay_ms(10);
      lcd_cursor(1,19);// Poner texto de cursor en posicion 3
      lcd_palabra("Cera");
      __delay_ms(10);
      lcd_cursor(1,25);// Poner texto de cursor en posicion 3
      lcd_palabra("Banda");
      __delay_ms(10);


//*****************************************************************************
// 2-Sensor de Profundidad-Mandamos a Llamar por I2C al sensor de profundidad de
// Agua y recibimos el valor que manda el sensor, si el valor esta por debajo de
//un limite, le  pedimos al usuario que rellene la cubeta con agua. Desplegamos
```

```
//El estado del agua de la cubeta en la LCD. Por ultimo mandamos el dato por UART
//*****************************************************************************
   I2C_Master_Start();         //Condicion de inicio
   I2C_Master_Write(0x11);     //Address
   Profundidad = I2C_Master_Read(0); //Mandar valor leido a variable
   I2C_Master_Stop();          //Condicion de Fin
   itoa(buffer,Profundidad,10);     //Convertir variable en String
   if(Profundidad < 4){
   lcd_cursor(2,1);            //Desplegar en LCD
   lcd_palabra("Refill");
   }
   if(Profundidad >= 4){
   lcd_cursor(2,1);            //Desplegar en LCD
   lcd_palabra("OK");
   }
   UART_Write_Text(buffer);
//*****************************************************************************
// 3-Sensor de Temperatura-Mandamos a llamar por I2C al sensor de Temperatura y
//Recibimos el Valor analogico de Temperatura. Usando un factor de conversion,
//convertimos el valor a la temperatura actual. Desplegamos la Temperatura en Pantalla
//y luego revisamos una condicion de temperatura que controla el relay que enciende y apaga
//la resistencia termica.Luego Mandamos el Valor por UART de temperatura
//*****************************************************************************
   I2C_Master_Start();         //Condicion de inicio
   I2C_Master_Write(0x21);     //Address
   Temperatura = I2C_Master_Read(0); //Mandar valor leido a variable
   I2C_Master_Stop();          //Condicion de Fin
   Rtemp =Temperatura*150/255;
   itoa(buffer,Rtemp,10);      //Convertir en String
   lcd_cursor(2,9);            //Desplegar en LCD
   lcd_palabra(buffer);
   lcd_cursor(2,11);
   lcd_palabra("C");
   if(Rtemp< 45){
      PORTBbits.RB7= 1;
   }
   if(Rtemp>= 45){
      PORTBbits.RB7= 0;
      __delay_ms(10000)//delay de 10 segundos
   }
   UART_Write_Text(buffer);

//*****************************************************************************
// 4-Sensor de Incendios- Mandamos a llamar por I2C al sensor de Incendios y
//Recibimos el Valor Digital. Si el sensor detecta un incendio, desplegarlo en la
//Pantalla y encender una alarma. Luego mandar el estado del sistema por UART
//*****************************************************************************
   I2C_Master_Start();         //Condicion de inicio
   I2C_Master_Write(0x31);     //Address
   Fuego = I2C_Master_Read(0); //Mandar valor leido a variable
   I2C_Master_Stop();          //Condicion de Fin
   if(Fuego == 100){
      PORTBbits.RB6 = 0;       //Apagar la Alarma
      lcd_cursor(2,13);        //Desplegar en LCD
      lcd_palabra("Safe");
      UART_Write_Text("Safe");
   }
   else if(Fuego == 0){
      PORTBbits.RB6 = 1;       //Prender la Alarma
      lcd_cursor(2,13);        //Desplegar en LCD
      lcd_palabra("Fuego");
      UART_Write_Text("Fuego");
```

```
    }


//*****************************************************************************
// 5-Sensor Ultrasonico- Mandamos a llamar el valor de distancia por I2C. Recibimos
//el valor de 3 digitos y lo separamos en unidades y deceneas. Lo desplegamos en la
//LCD. Luego mandamos el valor de distancia por UART
//*****************************************************************************
    I2C_Master_Start();         //Condicion de inicio
    I2C_Master_Write(0x41);     //Address
    Distancia = I2C_Master_Read(0); //Mandar valor leido a variable
    I2C_Master_Stop();          //Condicion de Fin
     d1 = (Distancia/100)%10;
     d2 = (Distancia/10)%10;
     d3 = (Distancia/1)%10;
    itoa(buffer,Distancia,10);     //Convertir variable en String
    lcd_cursor(2,19);           //Desplegar en LCD
    lcd_char(d1+'0');
    lcd_char(d2+'0');
    lcd_char(d3+'0');
    lcd_cursor(2,22);           //Desplegar en LCD
    lcd_palabra("Cm");
    UART_Write_Text(buffer);

//*****************************************************************************
// 6-Sensor Infrarojo-Mandamos a llamar el estado del sensor por I2C y desplegamos
//En la LCD si la banda se esta moviendo o si esta detenida, dependiendo de el estado
//del sensor. Luego mandamos el estado de la banda por UART
//*****************************************************************************
    I2C_Master_Start();         //Condicion de inicio
    I2C_Master_Write(0x51);     //Address
    Luz = I2C_Master_Read(0); //Mandar valor leido a variable
    I2C_Master_Stop();          //Condicion de Fin
    if(Luz == 1){
    lcd_cursor(2,25);           //Desplegar en LCD
    lcd_palabra("Detener");
    UART_Write_Text("Detener");
    }
    if(Luz == 0){
    lcd_cursor(2,25);           //Desplegar en LCD
    lcd_palabra("Mover");
    UART_Write_Text("Mover");

    }


//*****************************************************************************
// Movemos la pantalla a la izquierda en cada ciclo
//*****************************************************************************
    LCD_SCREEN_SHIFT(1);//mover a la izquierda
    __delay_ms(200);
    }
    }
```



Que hace en resumen:

El Máster dirige toda la comunicación I2C, también controla lo que se despliega en la pantalla. El Master primero manda a llamar a cada sensor por su nombre designado, y espera a recibir un valor o una serie de valores del pic donde esta conectado el sensor. Con esta serie de valores y condicionales, manda un indicador numérico o de texto a la pantalla de lo que está sucediendo. Por último, el Máster también se comunica con un pic por comunicación UART, en donde le manda los datos de los sensores, para que este PIC pueda transmitirlos por SPI. Esto se hace debido a que no puede establecerse comunicación I2C y SPI en el mismo microcontrolador.

- Sensor de profundidad de agua: El PIC que recibe los datos del sensor manda un valor por I2C de 1-5, lo cual representa la profundidad en cm. Si la profundidad del agua es mayor a 4 cm, se indica en la pantalla que el nivel de llenura esta bien, si está por debajo de 4 cm, le pide al usuario que rellene el tanque de agua.
- Sensor de Incendios: El sensor de incendios manda una salida de 1 o 0 indicando la presencia de un fuego. Si se encuentra activo, el master manda a la LCD un mensaje que lee ‘fuego’. También el Master Prende un buzzer relativamente ruidoso que alerta al usuario de peligro.
- Sensor de Temperatura: El sensor de temperatura manda el valor analogico convertido a una temperatura de - 30 a 130 C. Conectando al master está la una de las patas que cierran el relé y permiten que se active la resistencia térmica. El máster está pendiente de que el valor de temperatura no supere nunca los 45 grados, ya que a partir de ese punto el agua se calienta demasiado rápido y al hervir burbujea lanzando agua por sobre toda la máquina. El máster se asegura de mantener la temperatura siempre por debajo de 50 grados, desactivando la resistencia térmica cuando se superan los 45 grados y encendiendola denuevo cuando esta baja de 45 grados. Este “switcheo” también se asegura que no haya un elemento térmico que consume mucha potencia encendido 24/7. Despliega en Pantalla la temperatura a la que esta el exterior de la cubeta con el agua.
- Sensor Infrarrojo: El master recibe el estado actual de la banda, todo gracias a la salida digital del sensor infrarrojo. Cuando el sensor manda un LOW, se

manda a decir en pantalla que la banda se está ‘Moviendo’. Cuando manda un HIGH porque hay algo enfrente del sensor, manda a pantalla que la banda está ‘Detenida’.

- Sensor Ultrasónico: El master recibe 3 bytes con las unidades y decenas de la función de distancia que lee el esclavo. El máster combina esta serie de bytes y la despliega como texto en la pantalla, seguido de ‘Cm’.

Código de Sensor Esclavo de Profundidad:

```
/*****************************************************************************************
   * File:    Projecto_1_Profundidad.c                                        *
    * Date:    23/02/2020                                                    *
    * Autores:  Rodrigo Figueroa, Gonzalo Palarea, Stefan Schwendenner       *
    * Prof:     Pablo Mazariegos                                            *
    * Seccion:     20                                                       *
    * Clase:   Digital 2                                                    *
    * Compiler:    MPLAB XC8 v2.10                                          *
    * Arquitectura:    PIC16F887                                            *
    * Descripcion:  Proyecto 1 de Digital, en donde este PIC atravez de I2C recibe  *
    * Datos de 5 sensores, los despliega en una LCD, y los Manda atravez de UART a otro PIC*
    *                                                                       *
    *                                                                       *
    * Asignacion de Pins:                                                   *
    * Puerto B = Entrada del ADC RB5                                        *
    *                                                                       *
    *                                                                       *
    *                                                                       *
    * Puerto C = Salida de Clock y Salida de datos de los Datos  RC4,RC3    *
    *                                                                       *
    *                                                                       *
    *                                                                       *
    *                                                                       *
    *                                                                       *
    *                                                                       *
    *                                                                       *
    * Link al Github: https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git     *
    *                                                                       *
    *****************************************************************************************/

//*****************************************************************************
// Bits de Configuracion
//*****************************************************************************

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on 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)
```

```
#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)
//******************************************************************************
// Librerias
//******************************************************************************
#include <xc.h>
#include <stdint.h>
#include "ProjectoProfundidad.h"
//******************************************************************************
// Variables
//******************************************************************************
short z;
uint8_t adcval;
uint8_t masterval;
uint8_t deep;
int adcsend;
//******************************************************************************
// Interrupcion de Esclavo- Por Ligo George
//******************************************************************************
void __interrupt() isr(void)
{
  if(SSPIF == 1)
  {
    SSPCONbits.CKP = 0;

    if ((SSPCONbits.SSPOV) || (SSPCONbits.WCOL)) //If overflow or collision
    {
      z = SSPBUF; // Read the previous value to clear the buffer
      SSPCONbits.SSPOV = 0; // Clear the overflow flag
      SSPCONbits.WCOL = 0; // Clear the collision bit
      SSPCONbits.CKP = 1;
    }

  if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW) //If last byte was Address + Write
  {
    z = SSPBUF;
    PIR1bits.SSPIF = 0;
    SSPCONbits.CKP = 1;
    while(!BF);
    masterval = SSPBUF;
     __delay_us(250);

  }
  else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW) //If last byte was Address + Read
  {
    z = SSPBUF;
    BF = 0;
    SSPBUF = deep;
    SSPCONbits.CKP = 1;
     __delay_us(250);
    while(SSPSTATbits.BF);
  }

  PIR1bits.SSPIF = 0;
  }
  return;
}


//******************************************************************************
// Funcion Principal- Leemos el valor del ADC y utilizando 5 rangos definimos
// la profundidad del sensor, luego mandamos el valor de profundidad en cm al
```

```
// master en la interrupcion
//*****************************************************************************
void main(void) {
    ADCinit(); //Iniciar ADC
    I2C_Slave_Init(0x10); //Iniciar PIC como Esclavo
    TRISB = 0b00100000;
    ANSELH = 0b00100000;
    PORTB = 0;
    ADCON0bits.CHS = 13;
    while(1){
        ADCread(); //leer valor de ADC y mandarlo en la  interrupcion
        adcsend = voltaje;
        if(adcsend<100){
            deep = 0;
        }
        if(adcsend<110 && adcsend>100){
            deep = 1;
        }
        if(adcsend<120 && adcsend>110){
            deep = 2;
        }
        if(adcsend<130 && adcsend>120){
            deep = 3;
        }
        if(adcsend<140 && adcsend>130){
            deep = 4;
        }
        if(adcsend>140){
            deep = 5;
        }
    }
}
```

Que hace en resumen:

El Sensor manda una senal analogica que debe ser convertida en una serie de valores lógicos por el ADC del PIC. Luego de convertir los valores analogicos del sensor a valores concretos, nos damos cuenta que al no estar sumergido, el sensor siempre entrega un valor de 100-109, y mientras incrementamos la profundidad hasta estar completamente sumergido, que es cuando manda valores superiores a 140. Entonces hacemos un mapeo de estos valores a una serie de valores lógicos de 1-5 cm que representan la altura real del agua conforme con la posición del sensor. Mandamos dichos valores al PIC maestro.

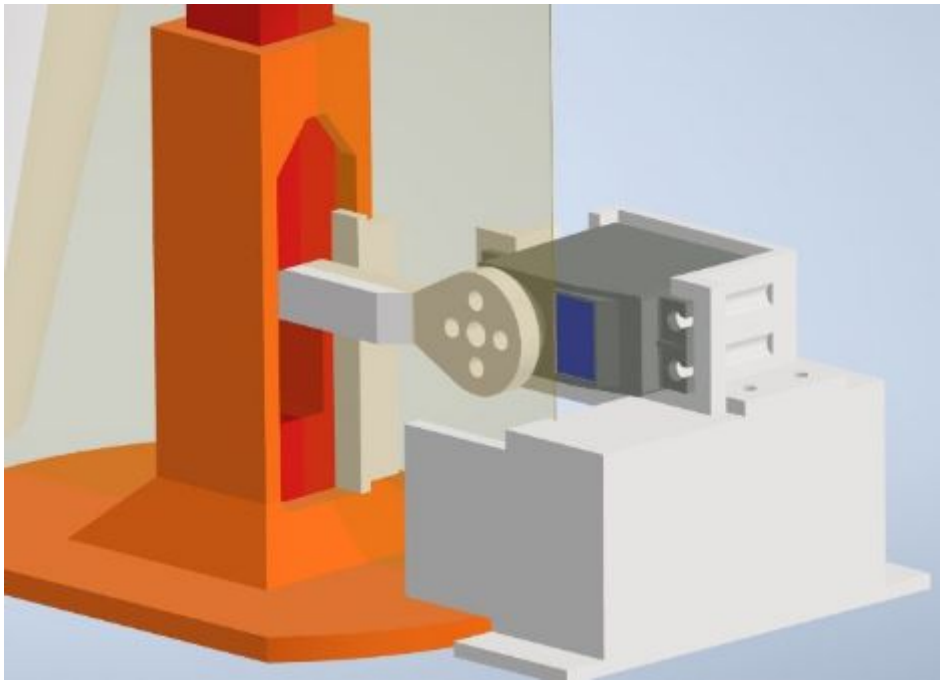
Código de Sensor Esclavo de Incendios:

```
*****************************************************************************************
  /*****************************************************************************************
  * File:    Projecto_1_Peso.c                                          *
   * Date:    23/02/2020                                               *
   * Autores:  Rodrigo Figueroa, Gonzalo Palarea, Stefan Schwendenner          *
   * Prof:     Pablo Mazariegos                                         *
   * Seccion:     20                                                  *
   * Clase:   Digital 2                                                *
   * Compiler:    MPLAB XC8 v2.10                                          *
   * Arquitectura:    PIC16F887                                         *
   * Descripcion:  Projecto 1 de Digital, en donde este PIC atravez de I2C recibe     *
   * Datos de 5 sensores, los despliega en una LCD, y los Manda atravez de UART a otro PIC*
   *                                                                   *
   *                                                                   *
   * Asignacion de Pins:                                               *
   * Puerto B = Entrada del ADC RB5                                        *
   *                                                                   *
   *                                                                   *
   *                                                                   *
   * Puerto C = Salida de Clock y Salida de datos de los Datos  RC4,RC3          *
   *                                                                   *
   *                                                                   *
   *                                                                   *
   *                                                                   *
   *                                                                   *
   *                                                                   *
   *                                                                   *
   * Link al Github: https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git     *
   *                                                                   *
   *****************************************************************************************/

//*****************************************************************************
// Bits de Configuracion
//*****************************************************************************

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)
//******************************************************************************
// Librerias
//******************************************************************************
#include <xc.h>
#include <stdint.h>
#include "ProjectoIncendio.h"
//******************************************************************************
// Variables
//******************************************************************************
short z;
uint8_t masterval;
int adcsend;
//******************************************************************************
// Interrupcion de Esclavo por Ligo George: Interrupcion de esclavo que activa el buffer para  recibir la senal de reloj del master
// Luego de recbir la senal de reloj del master espera para revisar si el master lo esta llamando por nombre
//Una vez que el maestro lo llama, le escribe al maestro con la informacion almacenada el la variable
//******************************************************************************
void __interrupt() isr(void)
{
 if(SSPIF == 1)
 {
  SSPCONbits.CKP = 0;

  if ((SSPCONbits.SSPOV) || (SSPCONbits.WCOL)) //If overflow or collision
  {
    z = SSPBUF; // Read the previous value to clear the buffer
    SSPCONbits.SSPOV = 0; // Clear the overflow flag
    SSPCONbits.WCOL = 0; // Clear the collision bit
    SSPCONbits.CKP = 1;
  }

 if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW) //If last byte was Address + Write
 {
  z = SSPBUF;
  PIR1bits.SSPIF = 0;
  SSPCONbits.CKP = 1;
  while(!BF);
  masterval = SSPBUF;
   __delay_us(250);

 }
 else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW) //If last byte was Address + Read
 {
  z = SSPBUF;
  BF = 0;
  SSPBUF = adcsend;
  SSPCONbits.CKP = 1;
   __delay_us(250);
  while(SSPSTATbits.BF);
 }
```

```
  PIR1bits.SSPIF = 0;
  }
  return;
}


//***********************************************************************
// Funcion Principal- Leemos el Digital del Sensor de incendios, Si manda un 1
//Le avisamos al master que hay un incendio, si mande un 0, le avisamos al master
//que esta a salvo. Tambien se tiene la opcion de encender una alarma. Tambien usamos
//este microcontrolador para operar el servo que eleva la cubeta con parafina. Se
//Mantiene leyendo una entrada que activa una funncion que crea 2 distintos PWM's.
//El PWM inicial matiene la barra horizontal, mientras que el PWM que manda cuando la
//Senal de control esta en alto
//***********************************************************************
#define ServoOut2 PORTDbits.RD2
#define Control PORTBbits.RB5
void ZeroGrados2(void) //0 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    ServoOut2 = 1;
    __delay_us(900);
    ServoOut2 = 0;
    __delay_us(19100);
  }
}

void NoventaGrados2(void) //90 Degree
{
  unsigned int i;
  for(i=0;i<30;i++)
  {
    ServoOut2 = 1;
    __delay_us(1500);
    ServoOut2 = 0;
    __delay_us(18500);
  }
}

void main(void) {
    I2C_Slave_Init(0x30); //Iniciar PIC como Esclavo
    TRISB = 0b10100000;
    TRISD = 0;
    ANSELH = 0;
    PORTB = 0;
    while(1){

        if(Control == 0){
            NoventaGrados2();
        }

        if(Control == 1){
            ZeroGrados2();
        }

        if(PORTBbits.RB7 == 1){
            adcsend = 100;
            PORTBbits.RB6 = 1;
        }
        else if (PORTBbits.RB7== 0){
```

```
            adcsend = 0;
            PORTBbits.RB6 = 0;
        }

}
}
```



Que hace en resumen:

Debido a la simplicidad del sensor, este PIC cumple 2 funciones. Hace la lectura digital de la salida del sensor, de acuerdo con su estado de activación, manda una variable que puede valer 0 o 100 para indicarle al maestro si debe de activar la alarma y mostrar si hay un fuego en pantalla. También aprovechamos que la operación de este sensor es completamente digital para poder conducir un servo. Específicamente el servo que levanta el contenedor de parafina para que este no colisione con el contenedor de agua a la hora de volcarlo. Este en una de sus salidas digitales genera entonces dos clases de PWM que determinan la posición del servo. Cuando se activa la función de noventa grados, el servo se mantiene en posición horizontal. Cuando se activa la función de zero grados, el servo mueve el brazo en la dirección de las agujas de reloj, levantando la cubeta y su contenido. La señal que activa y cambia la posición del servo sale del PIC que controla el sensor

infrarrojo. Esto asegura que la banda tiene que estar detenida antes de poder iniciar el proceso de vertido. Este servo se desactiva hasta que el stepper mueve el obstáculo de enfrente del sensor infrarrojo

Código de Sensor Esclavo de Temperatura:

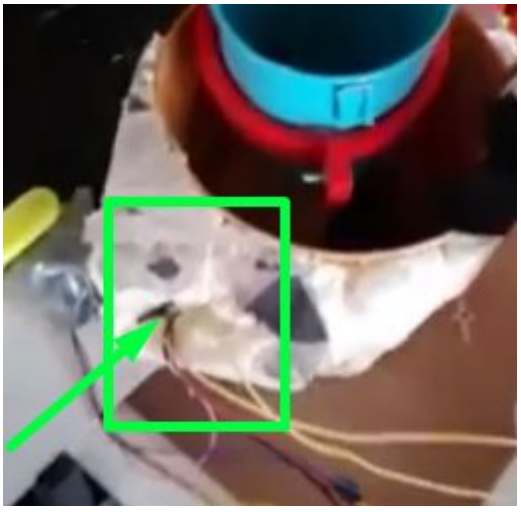
```
/*****************************************
   * File:    Projecto_1_Temperatura.c              *
    * Date:    23/02/2020                  *
    * Author:  Rodrigo Figueroa            *
    * Prof:    Pablo Mazariegos            *
    * Seccion:    20                       *
    * Clase:   Digital 2                   *
    * Compiler:    MPLAB XC8 v2.10         *
    * Arquitectura:    PIC16F887           *
    * Descripcion:  Projecto en donde un PIC maestro controla a otros 3 PICs
    * Utilizando Comunicacion I2C
    *
    *
    * Asignacion de Pins:
    * Puerto B = Entrada del Potenciometro
    *
    * Puerto A = Nada
    *
    * Puerto C = Salida de Clock y entrada de datos de los Slaves
    *
    * Puerto D = Nada
    *
    *
    *
    *
    * Link al Github: https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git
    *
    *****************************************/

//******************************************************************************
// Bits de Configuracion
//******************************************************************************

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)
//******************************************************************************
```

```
// Librerias
//*****************************************************************************
#include <xc.h>
#include <stdint.h>
#include "ProjectoTemperatura.h"
//*****************************************************************************
// Variables
//*****************************************************************************
short z;
uint8_t masterval;
uint8_t TempD;
uint8_t Temp;
//*****************************************************************************
// Interrupcion de Esclavo
//*****************************************************************************
void __interrupt() isr(void)
{
  if(SSPIF == 1)
  {
    SSPCONbits.CKP = 0;

    if ((SSPCONbits.SSPOV) || (SSPCONbits.WCOL)) //If overflow or collision
    {
      z = SSPBUF; // Read the previous value to clear the buffer
      SSPCONbits.SSPOV = 0; // Clear the overflow flag
      SSPCONbits.WCOL = 0; // Clear the collision bit
      SSPCONbits.CKP = 1;
    }

  if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW) //If last byte was Address + Write
  {
    z = SSPBUF;
    PIR1bits.SSPIF = 0;
    SSPCONbits.CKP = 1;
    while(!BF);
    masterval = SSPBUF;
     __delay_us(250);

  }
  else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW) //If last byte was Address + Read
  {
    z = SSPBUF;
    BF = 0;
    SSPBUF = Temp ;
    SSPCONbits.CKP = 1;
     __delay_us(250);
    while(SSPSTATbits.BF);
  }

  PIR1bits.SSPIF = 0;
  }
  return;
}
//*****************************************************************************
// Funcion Principal
//*****************************************************************************
void main(void) {
    ADCinit(); // Inicializar ADC
    I2C_Slave_Init(0x20); //Inicializar I2C como esclavo con Address 0x50
    ADCON0bits.CHS = 13; // Canal 13 del ADC
    TRISB = 0b10100000;//Inicializar Puerto
    ANSELH = 0b00100000;
```

```
    PORTB = 0;
    TRISA = 0b00001000;
    PORTA = 0;
    while(1){
        ADCread();// Leer Valor analogico del puerto y mandarlo al maestro
        Temp = voltaje;
        TempD = PORTBbits.RB7;
    }

}
```

Que hace en resumen:

En esencia el Sensor de temperatura saca un voltaje proporcional a la temperatura que siente. Para poder transformar este voltaje a algo logico, necesitamos poner un voltaje de referencia para el PIC, para que a la hora de hacer la conversion analogica, no nos de un resultado impreciso. Entonces ponemos un voltaje re referencia de 1.5 v para que asi el voltaje de salida del sensor sea de 1mV por grado. Luego de hacer la conversion, mandamos el valor de temperatura por I2C al maestro para que decida cuando prender y apagar el rele de la resistencia termica.



Codigo de Sensor Esclavo Infrarrojo:

```
/*****************************************
    * File:    Projecto_1_Laser.c              *
     * Date:    23/02/2020                  *
     * Author:  Rodrigo Figueroa            *
     * Prof:    Pablo Mazariegos            *
     * Seccion:    20                       *
     * Clase:   Digital 2                   *
     * Compiler:    MPLAB XC8 v2.10         *
     * Arquitectura:    PIC16F887           *
     * Descripcion:  Projecto en donde un PIC maestro controla a otros 3 PICs
     * Utilizando Comunicacion I2C
     *
     *
     * Asignacion de Pins:
     * Puerto B = Entrada del Potenciometro
```

```
    *
    * Puerto A = Nada
    *
    * Puerto C = Salida de Clock y entrada de datos de los Slaves
    *
    * Puerto D = Nada
    *
    *
    *
    *
    * Link al Github: https://github.com/Ricochetrj/Lab_5_I2C.git
    *
    ****************************************/

//******************************************************************************
// Bits de Configuracion
//******************************************************************************

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)
//******************************************************************************
// Librerias
//******************************************************************************
#include <xc.h>
#include <stdint.h>
#include "ProjectoLaser.h"
#define Motor RB6
#define SMotor1 RB4
#define SMotor2 RB3
#define SMotor3 RB2
#define Stop RB5
#define Laser RB7
//******************************************************************************
// Variables
//******************************************************************************
short z;
uint8_t adcval;
uint8_t masterval;
int adcsend;
uint8_t confirm;


//******************************************************************************
// Interrupcion de Esclavo: Interrupcion de esclavo que activa el buffer para  recibir la senal de reloj del master
// Luego de recbir la senal de reloj del master espera para revisar si el master lo esta llamando por nombre
//Una vez que el maestro lo llama, le escribe al maestro con la informacion almacenada el la variable
```

```
//*****************************************************************************
void __interrupt() isr(void)
{
  if(SSPIF == 1)
  {
    SSPCONbits.CKP = 0;

    if ((SSPCONbits.SSPOV) || (SSPCONbits.WCOL)) //If overflow or collision
    {
      z = SSPBUF; // Read the previous value to clear the buffer
      SSPCONbits.SSPOV = 0; // Clear the overflow flag
      SSPCONbits.WCOL = 0; // Clear the collision bit
      SSPCONbits.CKP = 1;
    }

  if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW) //If last byte was Address + Write
  {
    z = SSPBUF;
    PIR1bits.SSPIF = 0;
    SSPCONbits.CKP = 1;
    while(!BF);
    masterval = SSPBUF;
     __delay_us(250);

  }
  else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW) //If last byte was Address + Read
  {
    z = SSPBUF;
    BF = 0;
    SSPBUF = confirm;
    SSPCONbits.CKP = 1;
     __delay_us(250);
    while(SSPSTATbits.BF);
  }

  PIR1bits.SSPIF = 0;
  }
  return;
}


//*****************************************************************************
// Funcion Principal
//*****************************************************************************
void main(void) {
    I2C_Slave_Init(0x50); //Iniciar PIC como Esclavo
    TRISB = 0b10100000;
    TRISD = 0;
    ANSELH = 0;
    PORTB = 0;
    Motor =0;
    Stop  = 0;
//*****************************************************************************
// Loop infinito donde se lee el estado digital de la senal del sensor infrarojo
    // Si no pasa ningun objeto el sensor infrarojo manda una senal digital alta
    //Mientras la senal esta en alto, se manda el pulso al transistor que opera los
    //Motores de la banda, dejando asi que la banda se mueva. Cuando pasa un objeto
    //Se manda un 0 digital y se detiene la banda. Cuando se detiene la banda, se
    //manda por I2C una variable que indica que se detuvo
//*****************************************************************************
    while(1){
        if(Laser == 1){
```

```
            confirm = 0;
            Motor = 1;
            //SMotor1 = 1;
            //SMotor2 = 1;
            SMotor3 = 0;
        }
        if(Laser == 0){
            confirm = 1;
            Motor =0;
            //SMotor1 = 0;
            //SMotor2 = 0;
            SMotor3 = 1;
        }
    }
}
```

Que hace en resumen:

Este Sensor detecta la presencia de un objeto a una distancia de 3-4 cm del mismo sensor. Cuando un objeto pasa por enfrente del mismo, se apaga la salida digital del sensor. El programa controla el movimiento de la banda y de los servos de acuerdo con la salida del sensor. Luego de una lectura digital de la salida del sensor, si este se encuentra en HIGH, se mueve de manera continua la banda. En el momento que un objeto pasa en la cercanía del sensor, se corta la señal a la base de un transistor de potencia que opera los dos motores dc en serie. Esto detiene la banda y hace que se mande la señal que inicia la secuencia de vaciado para los tres servos y el stepper. Luego manda un valor de 1 o 0 por I2C para indicar si la banda se esta moviendo o si está detenida.

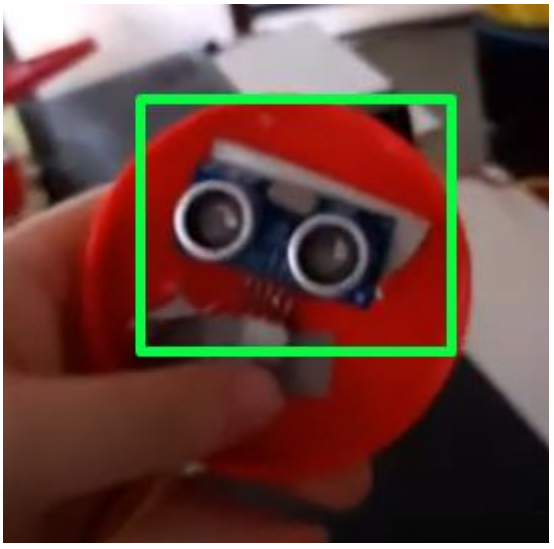
Código de Sensor Esclavo Ultrasónico:

```
/*****************************************
   * File:    ADC.c              *
    * Date:    23/02/2020              *
    * Author:  Rodrigo Figueroa          *
    * Prof:    Pablo Mazariegos          *
    * Seccion:    20                  *
    * Clase:   Digital 2               *
    * Compiler:    MPLAB XC8 v2.10        *
    * Arquitectura:    PIC16F887         *
    * Descripción:  Projecto en donde un PIC maestro controla a otros 3 PICs
    * Utilizando Comunicacion I2C
    *
    *
    * Asignacion de Pins:
```

```
    * Puerto B = Entrada del Potenciometro
    *
    * Puerto A = Nada
    *
    * Puerto C = Salida de Clock y entrada de datos de los Slaves
    *
    * Puerto D = Nada
    *
    *
    *
    *
    * Link al Github: https://github.com/Ricochetrj/Lab_5_I2C.git
    *
    ****************************************/

//******************************************************************************
// Bits de Configuracion
//******************************************************************************

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)
//******************************************************************************
// Librerias
//******************************************************************************
#include <xc.h>
#include <stdint.h>
#include "ProjectoUltrasonico.h"
//******************************************************************************
// Variables
//******************************************************************************
#define Trigger RB1 //34 is Trigger
#define Echo RB2//35 is Echo
short z;
uint8_t distancia;
uint8_t masterval;
uint8_t tiempo;
int adcsend;
char d1,d2,d3;
//******************************************************************************
// Interrupcion de Esclavo: Interrupcion de esclavo que activa el buffer para  recibir la senal de reloj del master
// Luego de recbir la senal de reloj del master espera para revisar si el master lo esta llamando por nombre
//Una vez que el maestro lo llama, le escribe al maestro con la informacion almacenada el la variable
//******************************************************************************
void __interrupt() isr(void)
{
  if(SSPIF == 1)
```

```
 {
  SSPCONbits.CKP = 0;

  if ((SSPCONbits.SSPOV) || (SSPCONbits.WCOL)) //If overflow or collision
  {
    z = SSPBUF; // Read the previous value to clear the buffer
    SSPCONbits.SSPOV = 0; // Clear the overflow flag
    SSPCONbits.WCOL = 0; // Clear the collision bit
    SSPCONbits.CKP = 1;
  }

 if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW) //If last byte was Address + Write
 {
  z = SSPBUF;
  PIR1bits.SSPIF = 0;
  SSPCONbits.CKP = 1;
  while(!BF);
  masterval = SSPBUF;
   __delay_us(250);

 }
 else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW) //If last byte was Address + Read
 {
  z = SSPBUF;
  BF = 0;
  SSPBUF = distancia;
  SSPCONbits.CKP = 1;
   __delay_us(250);
  while(SSPSTATbits.BF);
 }

 PIR1bits.SSPIF = 0;
 }
 return;
}


//*****************************************************************************
// Funcion Principal: Se manda un pulso al pin de trigger, luego de que se manda
//el pulso al trigger, este emite un un pulso de sonido de alta frecuecia que es
//inaudible. Luego el sensor espera a que regrese el echo de ese pulso, y cuando
//Recibe el echo y activa una senal en alto. Usando una funcion matematica que transforma
//El tiempo transcurrido (cuyo valor se guarda en el Timer 1) a una distancia en cm, mandamos dicha informacion por I2C al
Master.
//*****************************************************************************
void main(void) {
   I2C_Slave_Init(0x40); //Iniciar PIC como Esclavo
   TRISB = 0b00000101;
   ANSELH = 0;
   PORTB = 0;
   T1CON=0x20;
   while(1){
      TMR1H =0; TMR1L =0; //clear the timer bits
      Trigger = 1;
      __delay_us(10);
      Trigger = 0;
      while (Echo==0){

      }
         TMR1ON = 1;
      while (Echo==1){
```

```
        }
            TMR1ON = 0;

        tiempo = (TMR1L | (TMR1H<<8));
        distancia= (0.136*tiempo)/2;
    }
}
```



Que hace en resumen:

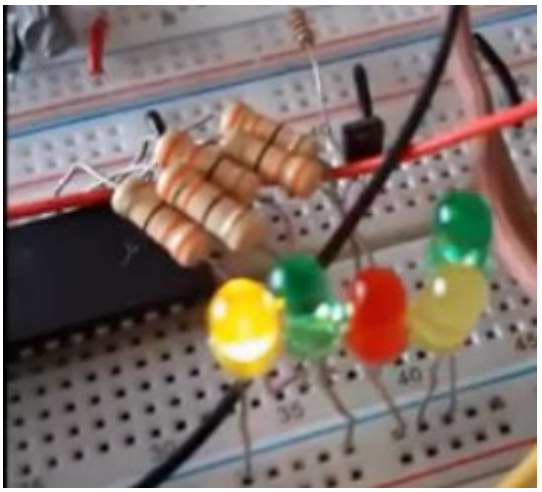
Al prender el sensor, este manda un pulso de alta frecuencia hacia lo que sea que esté enfrente del mismo. El código luego espera a escuchar este pulso de regreso y procede a activar el pin de echo. la diferencia de tiempo entre la salida inicial y la salida del pin echo nos sirve para medir el tiempo que tomó en rebotar el pulso. Este valor de tiempo se guarda en el Timer 1, donde luego pasa por una conversión matemática que lo divide en 2 para solo tomar en cuenta el tiempo de ida, y luego usando la velocidad del sonido calcula el valor de distancia a la que se encuentra el sensor de un obstáculo o pared. Este valor de distancia en cm se manda a través de la línea I2C para que el master determine la distancia que hay entre la parafina seca y el sensor para ver que tan vacio se encuentra el tanque de vaciado.

Código de UART a SPI:

```
/*****************************************
   * File:    Projecto_1_Master.c              *
    * Date:    04/03/2020                 *
    * Author:  Rodrigo Figueroa           *
    * Prof:    Pablo Mazariegos           *
    * Seccion:     20                     *
    * Clase:   Digital 2                  *
    * Compiler:    MPLAB XC8 v2.10        *
```

```
    * Arquitectura:    PIC16F887           *
    * Descripcion:
    *
    *
    * Asignacion de Pins:
    * Puerto B = Nada
    *
    * Puerto A = Salida de Pines D0-D7 de la LCD
    *
    * Puerto C = Salida de Clock y entrada de datos de los Slaves
    *
    * Puerto D = Pines RS y E de la LCD
    *
    *
    *
    *
    * Link al Github: https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git
    *
    ****************************************/

//******************************************************************************
// Bits de Configuracion
//******************************************************************************
 #pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)
#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)


#include <xc.h>
#include <stdint.h>
#include "ProjectoSPI.h"
unsigned char *Temp[3];
unsigned char *Luz[5];
unsigned char *Prof[2];
unsigned char *Fuego[5];
unsigned char *Dist[3];



void main(void) {
    UARTInit(9600,1); //Iniciar UART en baudrate de 9600, en HS mode
    spiInit(SPI_MASTER_OSC_DIV4, SPI_DATA_SAMPLE_MIDDLE, SPI_CLOCK_IDLE_LOW, SPI_IDLE_2_ACTIVE); //Modo
Spi Maestro
    //spiInit(SPI_MASTER_OSC_DIV4, SPI_DATA_SAMPLE_MIDDLE, SPI_CLOCK_IDLE_LOW, SPI_IDLE_2_ACTIVE);
//Modo Spi MaestrospiInit(SPI_SLAVE_SS_DIS, SPI_DATA_SAMPLE_MIDDLE, SPI_CLOCK_IDLE_LOW,
SPI_IDLE_2_ACTIVE);
    TRISA = 0;
    ANSEL = 0;
    PORTA = 0;
```

```
    TRISB = 0;
    while(1){
    ///////Profundidad
        if(UART_Data_Ready()){
        UART_Read_Text(Prof,2);
        PORTA = Prof;

    }
        PORTB = spiRead();
        spiWrite(Prof);


    //////Temperatura
        if(UART_Data_Ready()){
        UART_Read_Text(Temp,3);
        PORTA = Temp;

    }
        PORTB = spiRead();
        spiWrite(Temp);


    /////Incendio
        if(UART_Data_Ready()){
        UART_Read_Text(Fuego,5);
        PORTA = Fuego;
    }
        PORTB = spiRead();
        spiWrite(Fuego);


    ////////Distancia
        if(UART_Data_Ready()){
        UART_Read_Text(Dist,3);
        PORTA = Dist;
    }
        PORTB = spiRead();
        spiWrite(Dist);


    ///////Infrarojo
        if(UART_Data_Ready()){
        UART_Read_Text(Luz,5);
        PORTA = Luz;

    }
        PORTB = spiRead();
        spiWrite(Luz);


    }
}

.
```



Que hace en resumen:

Luego de recibir la información por UART que manda el PIC master, esta información se guarda en arrays discretos, que contienen la información de cada sensor. De acuerdo con la información que se almacena y se manda por SPI, se activan una serie de LEDs en el puerto A, que representan de manera binaria los datos que se están mandando por SPI. El codigo, como toda conexión SPI, espera que el dispositivo al que se le manda la información le conteste, por lo que tenemos conectada la entrada del pic a un generador de PWM para simular una entrada constante de datos. También mostramos lo que le está contestando el receptor en el puerto B, pero ya que solo es una señal de 0 a 5, decidimos no conectar los LEDs correspondientes.

Código de Controlador de 2 Servos:

```
/*****************************************
   * File:    Projecto_1_Controlador_de_Motores.c              *
    * Date:    23/02/2020                  *
    * Author:  Rodrigo Figueroa            *
    * Prof:    Pablo Mazariegos            *
    * Seccion:     20                      *
    * Clase:   Digital 2                   *
    * Compiler:    MPLAB XC8 v2.10         *
    * Arquitectura:    PIC16F887           *
    * Descripcion:  Projecto en donde un PIC maestro controla a otros 3 PICs
    * Utilizando Comunicacion I2C
    *
    *
    * Asignacion de Pins:
    * Puerto B = Entrada del Potenciometro
    *
    * Puerto A = Nada
    *
    * Puerto C = Salida de Clock y entrada de datos de los Slaves
    *
    * Puerto D = Nada
    *
    *
    *
    *
```

```
    * Link al Github: https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git
    *
    ****************************************/

//******************************************************************************
// Bits de Configuracion
//******************************************************************************

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on 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)


#include <xc.h>
#include <stdint.h>
#include "Servo.h"
#define Laser PORTBbits.RB7
void Start(void);

void main(void) {
   Start();



   while(1){
   CientoOchentaGrados();
   //ZeroGrados2();
   PORTBbits.RB4 = 0;

   if(Laser == 0 ){
   NoventaGrados();
   CientoOchentaGrados();
   __delay_ms(1000);
   CientoOchentaGrados();
   PORTBbits.RB4 = 1;
   //__delay_ms(1000);
   //CientoOchentaGrados2();
   __delay_ms(3000);
   CientoOchentaGrados();
   //ZeroGrados2();
   PORTBbits.RB4 = 0;
   }
   }
```

```
}

void Start(void){
   TRISB = 0b10000000;
   PORTB = 0x0F;
   ANSELH=0;

}
```

Que hace en resumen:

Lo que hace es que usamos dos salidas para generar dos PWMs que alimentan a dos servos, uno que abre el contenedor de parafina solida, y otro que vuelca el contenedor de parafina líquida. Lo que hacemos entonces, que usando la señal de salida del PIC con el sensor infrarrojo, iniciamos una secuencia de movimientos en un orden específico para evitar colisión entre partes. Variando el delay de activación y apagado del pulso en la función de Noventa, Zero y Ciento Ochenta, podemos cambiar el ángulo al que se encuentra el servo. Luego de esta rutina controlada, cuando se empieza a mover la banda de nuevo, regresamos a los servos a sus posiciones de descanso.

Repositorio de GitHub:

https://github.com/Ricochetrj/Projecto_1_De_Digital_Parafina.git

*Omitimos el codigo de Adafruit debido a que tuvimos problemas de coneccion de PIC a SPI de la Raspberry, solo simulamos las salidas con LED’s

Video de Youtube:

https://youtu.be/7K1EGIABoV8

Infinite loop that tells me to sleep

Interupt