ANEXOS

Anexo A - PRUEBAS CON CELDAS PELTIER

En las pruebas se utilizó una celda Peltier de 400W máximo y otra de 100W máximo (ver Figura 1). Para disipar el calor de la celda se utilizó un disipador cooler master hyper 212 con radiador, y para energizarlas se empleó la fuente del laboratorio de máquinas, la cual provee hasta 8A.

Características técnicas de los elementos utilizados

- ✓ **Celda Peltier de 400W**
  - Dimensiones 60mm x 60mm x 3.64mm.
  - Consumo máximo de potencia 400 Watts.
  - Opera desde 0-15.4 voltios DC y 0-26 amperios.
  - Opera desde -60º C a +180º C.
  - Equipada con 6 pulgadas de cable aislado.
  - Perímetro sellado para protección contra humedad.

- ✓ **Celda Peltier de 100W**
  - Dimensiones 40mm x 40mm x 3mm.
  - Consumo máximo de potencia 100 watts.
  - Opera desde 0-15 voltios DC y 0-10 amperios.
  - Opera desde -60ºC a + 180ºC.
  - Equipada con 4.5 pulgadas de cable aislado.
  - Perímetro sellado para protección contra humedad.

- ✓ **Disipador cooler master serie hyper 212 evo**
  - Tipo: ventilador disipador.
  - Tamaño del ventilador 120x120x25mm.
  - RPM 600 a 2000.

- Flujo de aire: 24.9 a 82.9 CFM ±10%.
- Nivel de ruido: 9 a 36 dB.
- Disipador de aluminio de dimensiones 120x80x159mm.
- Peso: 580g.

El montaje para hacer funcionar de manera adecuada las celdas; es relativamente sencillo. Como primer paso se identifica cual lado de la celda va a calentar y cuál va a enfriar según el flujo de la corriente, de esta manera se ubica el disipador de calor en el lado correcto de la celda, esto se hace identificando el cable rojo y ubicándolo al lado derecho, de tal forma que el lado que enfría va a ser el que quede hacia arriba y el que calienta hacia abajo siempre y cuando conectemos el cable rojo a positivo. Como el área del disipador que hace contacto con el elemento al que se le va a disipar el calor es más pequeña que el área de la celda, se optó por utilizar una lámina de aluminio de dimensiones 60x60x8mm. Esta lámina se ubica entre el lado caliente de la celda y el disipador de calor, así toda la celda hará contacto con la lámina de aluminio y se disipara el calor en toda el área de la celda. Para que el flujo térmico sea eficiente se aplicó una capa de grasa siliconada gel disipadora de calor entre las junturas celda-lamina y lamina-disipador, para un mejor intercambio de calor.

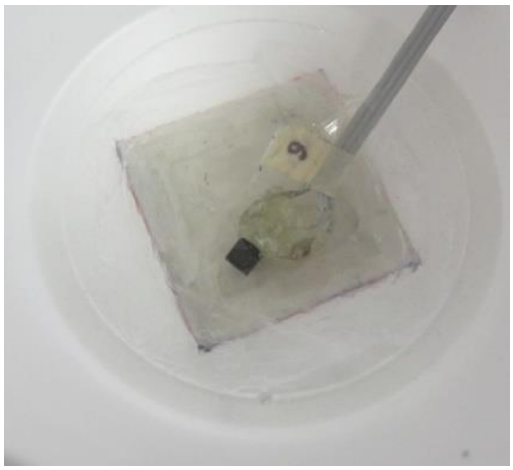
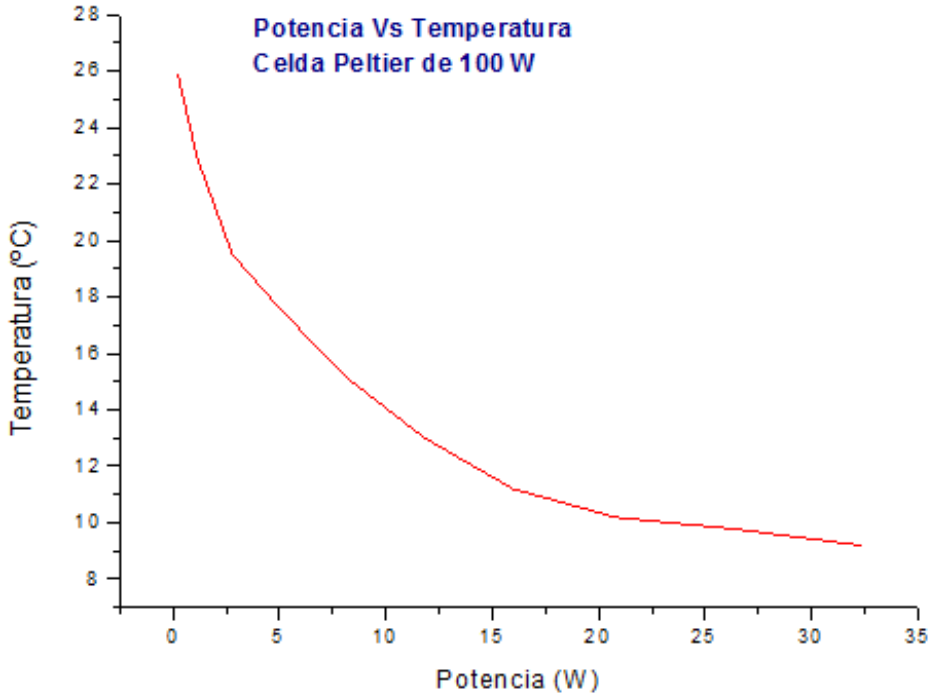


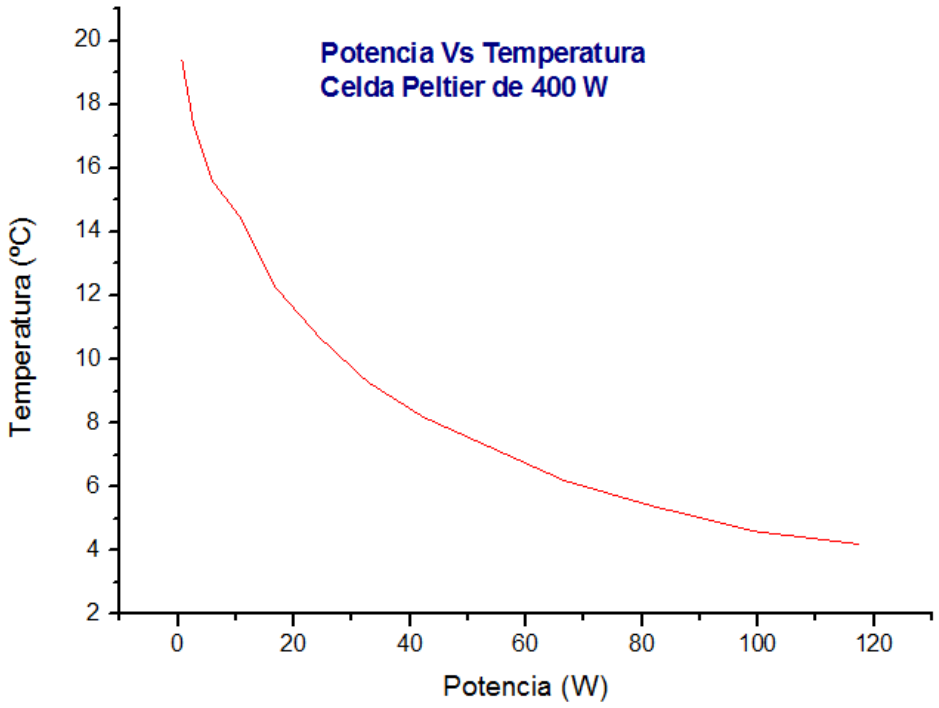
Figura 1. Celda Peltier

En las primeras pruebas se conectó la celda de 100w a 12V DC para llevarla al límite, y se midió una corriente de 4 A con la cual la superficie fría alcanzó una temperatura de 0˚C. Posteriormente se varió el voltaje incrementándolo desde 1v hasta 10V con incrementos de 1V entre cada prueba. En la Gráfica 1, se muestra la curva de potencia vs temperatura.



Gráfica 1. Potencia Vs Temperatura en la celda Peltier de 100 W

Con la celda de 400w se hizo un procedimiento similar, en donde se incrementó el voltaje que llegaba a la celda partiendo de 1V hasta llegar 13.16V, con incrementos cercanos tanto por arriba como por abajo a 1V entre cada prueba. Con los datos registrado se obtuvo la Gráfica 2, de Potencia vs Temperatura.



Gráfica 2. Potencia Vs Temperatura en la celda Peltier de 400 W

Las pruebas se hicieron a temperatura ambiente que estaba entre los 21°C y 22°C, aislando la cara fría de la celda mediante un recipiente de icopor para evitar corrientes de aire que actuaran como disturbios. En la Figura 2 y Figura 3, se ilustra el montaje de la prueba.

Figura 2. Montaje de prueba con la celda Peltier

Figura 3. Vista frontal de la prueba con la celda Peltier

El siguiente paso fue determinar qué tan viable es utilizar las celdas Peltier en el sistema de refrigeración; para lo cual se pensó en un diseño preliminar sencillo en donde se utilizaría una lámina de aluminio, a la cual se le adherirían las celdas por el lado frío, de esta manera se disminuiría la temperatura de la lámina, para después mediante extractores crear un flujo de aire frío desde la lámina hacia el medio a refrigerar. Con esta idea la siguiente prueba que se realizó fue montar una celda de 100W en un costado de la incubadora, en donde la cara fría de la celda queda dentro da la incubadora mientras que la cara caliente quedo por fuera haciendo contacto con el disipador de calor (ver Figura 4). A la cara fría se le agregó el disipador que aparece en la imagen, al que se le colocó un extractor para ayudar a la transferencia de temperatura.

Figura 4. Montaje de la celda Peltier en la planta

El objeto del experimento era observar que valor mínimo de temperatura era posible alcanzar en el disipador y en la incubadora. En la Figura 5 se pueden ver los elementos del montaje para el experimento.

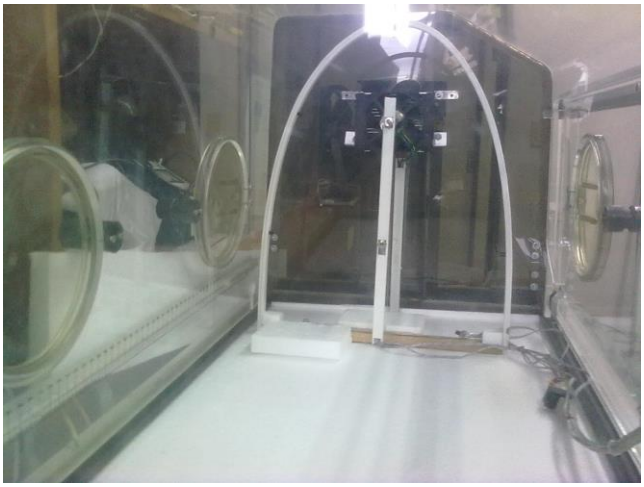


Figura 5. Planta lista para realizar las pruebas mediante el sistema Peltier

El experimento se realizó con una temperatura ambiente de 23°C, se supervisó la temperatura del disipador, y la temperatura del ambiente mediante un clúster de sensores de temperatura compuesto por dos termocuplas y 6 lm35. En principio se midió la temperatura del disipador sin poner en funcionamiento el extractor, al cabo de 12 minutos la temperatura de la superficie del disipador se encontraba a 8°C, con la celda trabajando a 12V y 4.5A. Al poner en marcha el extractor la temperatura del de disipador aumento rápidamente hasta alcanzar una temperatura de 18°C, esto debido a un gradiente de temperatura que el motor del extracto ingresa al sistema. Por otro lado la temperatura circundante alrededor de la celda no disminuyó significativamente con el extractor apagado; con el extractor funcionando la temperatura disminuyó solo 1°C por debajo de la temperatura ambiente que en ese momento era de 23°C.

Anexo B - CÓDIGO IMPLEMENTADO EN LA TARJETA PSoC3

A continuación se presenta el código implementado para la tarjeta PSoC3, con extensión .C. Éste es el código principal del programa, el cual de manera general se encarga de monitorear y leer las señales de los sensores, enviar los datos al sistema de supervisión, al mismo tiempo que recibe instrucciones para ejercer control sobre los actuadores.

Código principal implementado en la tarjeta PSoC3

```
CODIGO PSoC PARA EL SISTEMA PRELIMINAR DE
ADQUISICION Y PROCESAMIENTO DE DATOS
 *
 * ========================================
*/
//LIBRERIAS

#include <device.h>
#include <stdlib.h>
#include <stdio.h>
#include <Delay.h>
//========================================
//REGLAS CONTROL DIFUZZO SERVOMOTOR
//========================================
#define R0 3600
#define R1 3483
#define R2 3367
#define R3 3367
#define R4 3133
#define R5 3017
#define R6 2900
#define R7 3483
#define R8 3483
#define R9 3367
#define R10 3367
#define R11 3017
#define R12 2900
#define R13 2900
#define R14 3483
#define R15 3367
#define R16 3367
#define R17 3250
#define R18 3017
#define R19 2900
#define R20 2900
#define R21 3133
#define R22 3017
#define R23 2900
#define R24 2900
#define R25 2900
#define R26 2900
#define R27 2900
#define R28 2900
```

```
#define R29 2900
#define R30 2900
#define R31 2900
#define R32 2900
#define R33 2900
#define R34 2900
#define R35 2900
#define R36 2900
#define R37 2900
#define R38 2900
#define R39 2900
#define R40 2900
#define R41 2900
#define R42 2900
#define R43 2900
#define R44 2900
#define R45 2900
#define R46 2900
#define R47 2900
#define R48 2900
//================================
//REGLAS CONTROL DIFUZZO MOTOR DC
//================================
#define r0 200
#define r1 200
#define r2 200
#define r3 200
#define r4 267
#define r5 334
#define r6 401
#define r7 200
#define r8 200
#define r9 200
#define r10 267
#define r11 334
#define r12 401
#define r13 468
#define r14 200
#define r15 267
#define r16 334
#define r17 401
#define r18 468
#define r19 535
#define r20 602
#define r21 401
#define r22 468
#define r23 535
#define r24 602
#define r25 602
#define r26 602
#define r27 602
#define r28 602
#define r29 602
#define r30 602
#define r31 602
#define r32 602
#define r33 602
```

```
#define r34 602
#define r35 602
#define r36 602
#define r37 602
#define r38 602
#define r39 602
#define r40 602
#define r41 602
#define r42 602
#define r43 602
#define r44 602
#define r45 602
#define r46 602
#define r47 602
#define r48 602
//============================================================
//FUNCIONES
//Calculo valores de membresia.
void CMemebresia (char g,char m,float entra, float k);
//Calculo minimos
float minimo(float c,float y);
//Controlador Fuzzy
int32 Fuzzy (int32 error1,int32 Aerror1);
//============================================================
//Contadores Timer_1
extern unsigned int contimer;
extern unsigned int flagsample;
extern unsigned int timeenco;
extern unsigned int ControlF1;
extern unsigned int ControlF2;
extern unsigned int rxinterrump;
extern unsigned int countrxint;
extern unsigned int Time;
extern unsigned int Timerampa;
//variables envio de tato al serial.
unsigned int cont=0;
unsigned int dH=0,dL=0
//============================================================
//VARIABLES UTILIZADAS EN EL CONTROLADOR
//============================================================
char OutputString[32];
float entra=0,k=0;
int32 error1=0,Aerror1=0;
float c=0,y=0;
float comp=0;comp2=0;//Valores PWM
unsigned int st[8];
//Entrada, conjuntos fuzzy
float Input [8]={-0.2,-0.13333,-0.06666,0,0.06666,0.1333,0.2};
float Entrada[2],cal,Asigna[4],minimos[4],w,Asi;
struct {char D[2];float u[2];}asg[2];
char posicion[4],buffer2[4];
char  d=0,j=0;
int ControlFz=0;
```

```
//============================================================
//MATRIZ DE REGLAS CONTROLADOR FUZZY SERVOMOTOR
//============================================================
int Reglas1 [7][7]=
                {

R0,     R1,     R2,     R3,     R4,     R5,     R6,
R7,     R8,     R9,     R10,    R11,    R12,    R13,
R14,    R15,    R16,    R17,    R18,    R19,    R20,
R21,    R22,    R23,    R24,    R25,    R26,    R27,
R28,    R29,    R30,    R31,    R32,    R33,    R34,
R35,    R36,    R37,    R38,    R39,    R40,    R41,
R42,    R43,    R44,    R45,    R46,    R47,    R48

                };
//============================================================
//MATRIZ DE REGLAS CONTROLADOR FUZZY SERVOMOTOR
//============================================================

int Reglas2 [7][7]=
                {

r0,     r1,     r2,     r3,     r4,     r5,     r6,
r7,     r8,     r9,     r10,    r11,    r12,    r13,
r14,    r15,    r16,    r17,    r18,    r19,    r20,
r21,    r22,    r23,    r24,    r25,    r26,    r27,
r28,    r29,    r30,    r31,    r32,    r33,    r34,
r35,    r36,    r37,    r38,    r39,    r40,    r41,
r42,    r43,    r44,    r45,    r46,    r47,    r48,

};

//============================================================
//INICIO DEL PROGRAMA PRINCIPAL
//============================================================
void main()

{
//variables utilizadas para el error diferencial del error
int32 Aerror=0,control=0,Anter=0,Aerror2=0,control2=0,Anter2=0;
int32 Set_point=0,error=0,Set_point2=0,error2=0;
int con=0;
float entra,k,entra2,k2;
int32 PWM=0;
float b=0;
int contime=0,contrampa=0;
int32 ct=0;
//============================================================
//VARIABLES DE RECEPCION DE DATOS
 unsigned int Set[3];
 unsigned int SetPoint=0;
 unsigned int conmandos;
 int Pump;
```

```
 int ManualAutoControl;
 unsigned int PWMservo;
 unsigned int PWMmotor;
 unsigned int setpoint;
 int CONTROL=0;
//============================================================
//VARIABLES RAMPA
int32 TempPADC;//Temperatura interna del paciente valor ADC
float TempGC;//Temperatura en grados centigrados
float SetPointG;//Set Point en grados centigrados
int banr;//bandera.
int Timecontrol=0;

//============================================================
//INICIALIZACION DE LOS BLOQUE DEL ARCHIVO .cysch.
Envia_Start();
Ref_1_Start();
Mux1_Start();
Mux1_Select(0);
LCD_Start();
sensorT_Start();
sensorT_StartConvert();
Timer_1_Start();
interrup1_Start();
interrupenvia_Start();
Amp1_Start();
Amp1_SetGain(Amp1_GAIN_01);
PWM_1_Start();
PWM_2_Start();

//************************
PWM_1_WritePeriod(1199);
PWM_1_WriteCompare(600);
//************************
PWM_2_WritePeriod(59998);
PWM_2_WriteCompare(30000);
//************************
CYGlobalIntEnable;
cont=0;
      Pin_3_Write(0);

    for(;;)
    {
//============================================================
//TIMER EN MINUTOS
//============================================================
      if(Time==24000)
      {contime=contime+1;
      if(contime=24000)
      {contime=0;
      }
      Time=0;
      }


//============================================================
//RECEPCION DE DATOS POR SERIAL
```

```
//=============================================================
      if(rxinterrump)
         {
          Envia_ReadRxStatus();
          Set[countrxint]=Envia_ReadRxData();
          Envia_ClearRxBuffer();

          if(countrxint==3)
             {
            switch(Set[1])

              {
              //Encendido y apagado del sistema de refrigeración.
              case 1:{Pump=Set[2]+ 256*Set[3];
                  LCD_Position(0,1);
                  sprintf(OutputString, "on/off. %ud ",Pump );
                  LCD_PrintString(OutputString);break;
              } //Modo control manual o automatico.
              case 2:{ManualAutoControl=Set[2]+ 256*Set[3];
                  LCD_Position(0,1);
                  sprintf(OutputString, "2. % ud ",ManualAutoControl);
                  LCD_PrintString(OutputString);

                  if(ManualAutoControl==0)
                  {CONTROL=1;}
                  if(ManualAutoControl==1)
                  {CONTROL=2;}
                  break;
              } //PWM del Servomotor.
              case 3:{PWMservo=Set[2]+ 256*Set[3];
                  LCD_Position(0,1);
                  sprintf(OutputString, "3. %ud ",PWMservo );
                  LCD_PrintString(OutputString);
                  break;
              } //PWM del motor DC.
              case 4:{PWMmotor=Set[2]+ 256*Set[3];
                  LCD_Position(0,1);
                  sprintf(OutputString, "4. %ud ", PWMmotor);
                  LCD_PrintString(OutputString);break;
              }
               //Control mediante SetPoint.
              case 5:{setpoint=Set[2]+ 256*Set[3];
                  LCD_Position(0,1);
                  sprintf(OutputString, "5. %ud ",setpoint );
                  LCD_PrintString(OutputString);
                  CONTROL=3;break;
              }

              }

             countrxint=0;
            }
            rxinterrump=0;
         }
```

En el código de recepción de datos a través del serial se reciben paquetes de información codificada en tres bytes, en total se pueden recibir cinco paquetes diferentes. El primer byte es una etiqueta que referencia la información que viene en los dos bytes restantes. Estos datos se almacenan en el vector set y el swich se encarga de direccionar la información a cada case. Finalmente según el case se determina si se prende o apaga el sistema de refrigeración, si el control es manual o automático, o si el control se realiza mediante la recepción del Set Point. En el caso que el control se haga de forma manual se recibe el valor de la señal de PWM que va a cada actuador. En cada uno de los case se imprime en el LCD la orden correspondientes.

```
//=======================================================
//CONTROL ON/OFF COMPRESOR
//=======================================================
      if(Set[1]==1)
      { if(Pump==1)
        {
            LCD_Position(1,1);
      sprintf(OutputString, "star. %ud ",Pump );
            LCD_PrintString(OutputString);
            Pin_3_Write(1);
       }
        if(Pump==0)
        {
            LCD_Position(1,1);
            sprintf(OutputString, "stop. %ud ",Pump );
            LCD_PrintString(OutputString);
            Pin_3_Write(0);
            }
      }
```

Si desde el supervisorio llega un numero uno, el sistema de refrigeración se enciende colocando en alto un pin digital predefinido en el archivo .cysch. Si por el contrario se tiene la recepción de un cero, la orden será apagar el sistema de refrigeración.

```
//=====================================================
//MODOS DE CONTROL
//=====================================================
//CONTROL MANUAL
//=====================================================
      switch(CONTROL) {
      case(1):{
        if(Set[1]==3){
            control2=PWMservo*7+2900;
```

```
            PWM_2_WriteCompare(control2);
          LCD_Position(1,1);
            sprintf(OutputString, "%ld ",control2);
            LCD_PrintString(OutputString);

             }
        if(Set[1]==4){
          control=PWMmotor*4+200;
            PWM_1_WriteCompare(control);
          LCD_Position(1,1);
            sprintf(OutputString, "%ld ",control);
            LCD_PrintString(OutputString);

             }
            break;}
```

El control manual se ejecuta cuando la variable CONTROL toma el valor de uno, el case toma los valores enviados (valores enviados desde el supervisorio) en porcentaje de la señal se PWM, para luego enviar la señal por los pines predefinidos. También se imprime en el LCD el valor proporcional de la seña PWM.

```
//=======================================================
//CONTROL AUTOMATICO
//=======================================================
      case(2):{


      LCD_Position(1,1);
      sprintf(OutputString, "CA. %d ",ManualAutoControl);
      LCD_PrintString(OutputString);
//=======================================================
// FUNCION RAMPA CONTROL AUTOMATICO
//=======================================================
      /if(contime==5)
{banr=1; }

       if(Timerampa==24000)// 24000
       {
          if(banr==1)
      {
            TemLM=TemLM-(0.08333);

            conramp++;
            if(conramp==60)
            {
            banr=2;
            }
            }
        if(banr==2)
        {
            conramp2=conramp2+1;
            if(conramp2==20)
```

```
            {conramp2=0;
            banr=4;
            }

            }
          if(banr==4)
          {
          TemLM=TemLM+(0.08333);
          conramp3++;
              if(conramp3==60)
            {banr=3;
              }

           }
          if(banr==4)
          {
          TemLM;
                }

       dato=((TemLM+b2+m2*b1)/(m1*m2))*(ADC/v);
       SetPoint=dato;

       Timerampa=0;

        }
```

Cuando la variable CONTROL toma el valor de dos el sistema inicia el control automático. En este escenario de control del equipo inicia una rampa de descenso de la temperatura, en donde se reducen cinco grados centígrados la temperatura ambiente del domo en un tiempo de una hora, posteriormente se mantiene constante la temperatura durante veinte minutos. Finalmente se incrementa la temperatura mediante una rampa que tarda una hora en llevar la temperatura al valor de partida. Los datos de referencia de la temperatura que se desean según lo explicado, son almacenados en la variable SetPoint y enviados a los controladores Fuzzy para que se encarguen de realizar el control automático.

```
//=====================================================
//CONTROL CON SETPOINT
//=====================================================
      case(3):{
      if(Set[5]){
      SetPoint=setpoint;
      LCD_Position(1,1);
      sprintf(OutputString, "STP. %d ",SetPoint);
      LCD_PrintString(OutputString);

      }  break;}
```

```
}
```

En el momento en que la variable CONTROL toma el valor de tres, el sistema pasa a modo control por Set Point. En este escenario simplemente recibe el valor de referencia de la temperatura a la cual se desea llegue el ambiente al interior del domo. El valor de referencia enviado desde el supervisorio se almacena en la variable SetPoint, valor que utilizan los controladores fuzzy como referencia para ejercer el control sobre el sistema.

```
//=====================================================
// ENVIO DE DATOS AL PUERTO SERIAL
//=====================================================
         if(flagsample==4)
         {

           Mux1_Select(cont);
             CyDelay(1);
           if(sensorT_IsEndConversion(sensorT_RETURN_STATUS))
           st[cont]=sensorT_GetResult32();

             if(cont==7)
             {Envia_PutChar(cont);
             CyDelay(1);
             dH=SetPoint/256;
             dL=SetPoint-dH*256;
             Envia_PutChar(dL);
             CyDelay(1);
             Envia_PutChar(dH);
             CyDelay(1);}
             else
             {Envia_PutChar(cont);
             CyDelay(1);
             dH=st[cont]/256;
             dL=st[cont]-dH*256;
             Envia_PutChar(dL);
             CyDelay(1);
             Envia_PutChar(dH);
             CyDelay(1);}
//*************************************************
//SWITCH PARA CONMUTAR EL ELVIO DE DATOS AL SUPERVISORIO

             switch(cont)

              {
              case 0:cont=1;Amp1_SetGain(Amp1_GAIN_08);break;
              case 1:cont=2;Amp1_SetGain(Amp1_GAIN_08);break;
              case 2:cont=3;Amp1_SetGain(Amp1_GAIN_08);break;
              case 3:cont=4;Amp1_SetGain(Amp1_GAIN_08);break;
              case 4:cont=5;Amp1_SetGain(Amp1_GAIN_08);break;
              case 5:cont=6;Amp1_SetGain(Amp1_GAIN_08);break;
              case 6:cont=7;break;
```

```
            case 7:cont=0;Amp1_SetGain(Amp1_GAIN_08);break;
            }


          flagsample=0;
       }
    //*****************************************************
```

Para el envío de los datos leídos por los sensores se utiliza un swich que conmuta los pines del multiplexor leyendo un sensor a la vez. La variable cont es una etiqueta que distingue el dato de cada sensor en particular, excepto cuando la variable toma el valor 7, en este caso la etiqueta identifica el valor del SetPoint que recibe es sistema. El envío de los datos se hace por paquetes de tres bytes, siendo el primero de ellos la etiqueta mientras que el segundo y tercero llevan la información de los sensores o en el caso particular de cont igual a siete el valor de Set Point. El dato leído por los sensores se dividen en alto y bajo para enviarlo en dos bytes, ya que el ADC por tener una resolución de 16 bits no cabe en un solo byte.

```
//====================================================================
// CONTROL FUZZY SERVOMOTOR
//====================================================================

if(ControlF1==20)
{   ControlFz=1;
    Set_point=SetPoint;

    (int32)error=(int32)Set_point-(int32)st[2];

//*********************************************************************
      Aerror = error-Anter;
      (int32)control = Fuzzy(error,Aerror);
      Anter = error;
      if(control<2900)
      {control=2800;
    PWM_2_WriteCompare(control);
    }
      if(control>3559)
      {control=3600;
    PWM_2_WriteCompare(control);
      }
      else
      {PWM_2_WriteCompare(control);}
      ControlF1=0;
      ControlFz=0;

      }
```

```
//========================================================================
=
// CONTROL FUZZY MOTOR DC
//========================================================================
=

      if(ControlF2==25)
      {
      ControlFz=2;
    Set_point2=SetPoint;

    (int32)error2=Set_point2-st[2];

//**************************************************************************
      Aerror2 = error2-Anter2;
      (int32)control2 = Fuzzy(error2,Aerror2);

      Anter2 = error2;
      if(control2>590)
      {
      control2=100;
      PWM_1_WriteCompare(control2);
      }
      else{
      PWM_1_WriteCompare(control2);}
    ControlF2=0;
      ControlFz=0;
                  }

      }
}
```

Los controladores reciben el valor del Set Point (temperatura de referencia) almacenado en la variable SetPoint, a continuación se calcula error y se actualizan el cambio del error. La siguiente parte es el llamado a la función Fuzzy que recibe los valores del error y el cambio en el error. Esta función se encarga de calcular la señal de mando que ira hacia los actuadores. La función Fuzzy normaliza los valores de entrada del error y variación de error, calcula los valores de membresía, hace la evaluación de las reglas y finalmente hace la defucificación con la que se encuentra el valor de mando de la señal de cada actuador. Esta función distingue las reglas correspondientes a cada controlador.

```
//========================================================================
====
//CONTROLADOR FUZZY
//========================================================================
====
//DEFINICION DE REGLAS//
int32 Fuzzy (int32 error1,int32 Aerror1)
```

```
{
int32 dato1=0,dato2=0;

      dato1=error1;
      dato2=Aerror1;

(float)Entrada[0] = (float)(dato1)/(65380);

      (float)Entrada[1] =(float)dato2/65380;
LCD_Position(1,9);

if(Entrada[0]<-0.2)
{Entrada[0]=-0.2;}
if(Entrada[1]<0.2)
{Entrada[1]=-0.2;}

if(Entrada[0]>0.2)
{Entrada[0]=0.2;}
if(Entrada[1]>0.2)
{Entrada[1]=0.2;}

      for (j = 0 ; j<2; j++)
      {
            for (d = 0; d<6; d++)

                  {
                  if ((float)Entrada [j]<(float)Input[d] ) {break;}
                  }
      CMemebresia(d,j,Entrada [j],Input[d]);
      }
//*************************EVALUACIONREGLAS*****************************
**//

if(ControlFz==1)
{


Asigna[0]= Reglas1[asg[0].D[0]][asg[1].D[0]];//EL PUNTO INDICA LA
POSICION
                  minimos[0]=minimo(asg[0].u[0],asg[1].u[0]);
Asigna[1]= Reglas1[asg[0].D[1]][asg[1].D[0]];
                  minimos[1]=minimo(asg[0].u[1],asg[1].u[0]);
Asigna[2]= Reglas1[asg[0].D[0]][asg[1].D[1]];
                  minimos[2]=minimo(asg[0].u[0],asg[1].u[1]);
Asigna[3]= Reglas1[asg[0].D[1]][asg[1].D[1]];
                  minimos[3]=minimo(asg[0].u[1],asg[1].u[1]);
}
if(ControlFz==2)
{
Asigna[0]= Reglas2[asg[0].D[0]][asg[1].D[0]];//EL PUNTO INDICA LA
POSICION
                  minimos[0]=minimo(asg[0].u[0],asg[1].u[0]);
Asigna[1]= Reglas2[asg[0].D[1]][asg[1].D[0]];
                  minimos[1]=minimo(asg[0].u[1],asg[1].u[0]);
Asigna[2]= Reglas2[asg[0].D[0]][asg[1].D[1]];
                  minimos[2]=minimo(asg[0].u[0],asg[1].u[1]);
Asigna[3]= Reglas2[asg[0].D[1]][asg[1].D[1]];
```

```
                minimos[3]=minimo(asg[0].u[1],asg[1].u[1]);

}


w=
(minimos[0]*Asigna[0])+(minimos[1]*Asigna[1])+(minimos[2]*Asigna[2])+(min
imos[3]*Asigna[3]);
Asi = (minimos[0])+(minimos[1])+(minimos[2])+(minimos[3]);
(int32)PWM = (float)w/(float)Asi;


return PWM; //SALIDA DEL CONTROLADOR

}

//**********************MEMBRESIA
DIRECCION*******************************//
void CMemebresia (char g, char m, float entra, float k)
{
float cal;

float L1=1,L2=((0.2)/3.00005);
            (float)cal =(float)k - (float)entra;
            asg[m].u[0]=cal*(L1/L2);
            asg[m].u[1]= 1-(asg[m].u[0]);
            asg[m].D[0]=g-1;
            asg[m].D[1]=g;
return;
}


//*************************MINIMO*************************************
**//
float minimo(float c,float y)
{
if (c>y)
{c=y;}
return c;
}
```

Archivo .cysch de configuración de los elementos utilizados en el archivo principal.

En este archivo se agregan y configuran cada uno de los elementos que se utilizan el programa principal (ver Figura 1).

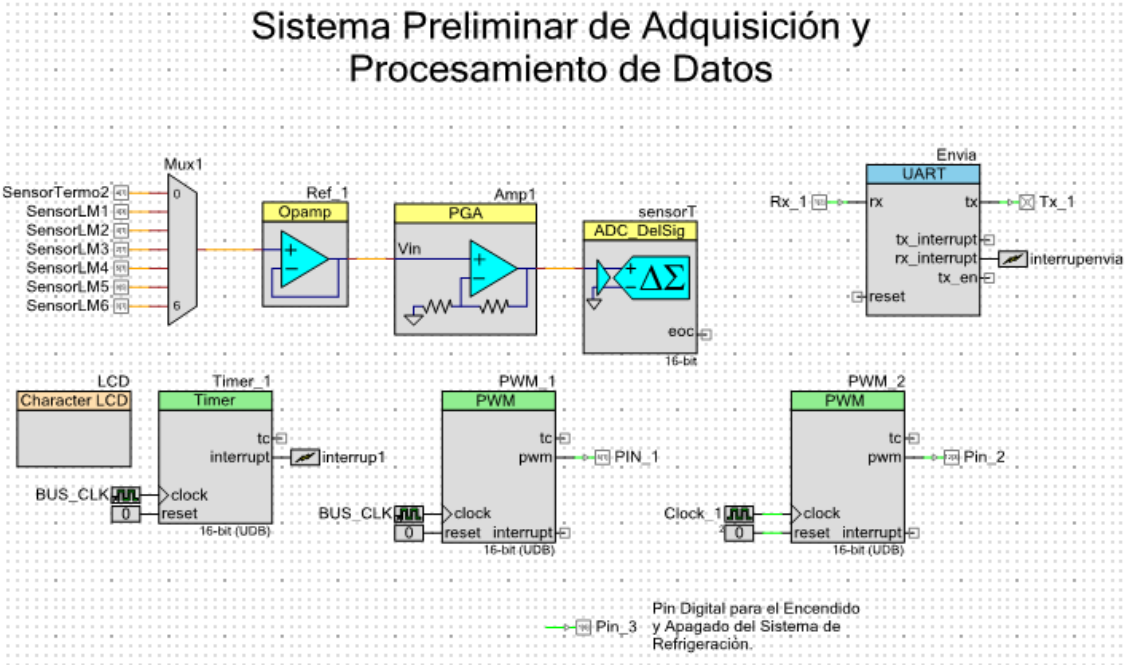
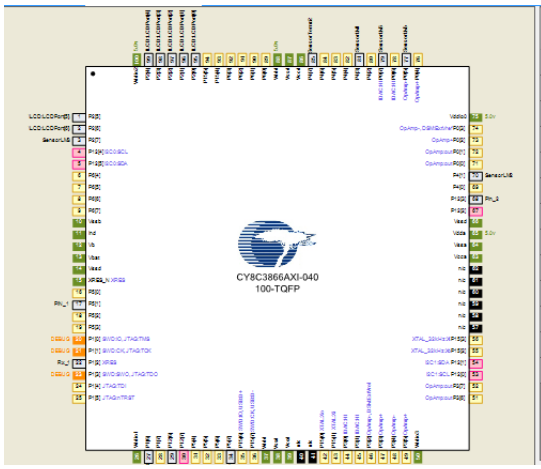


Figura 1. Configuración de los elementos del archivo principal

Cada uno de los bloques cuenta con un datasheet que explica y referencia las características y funciones propias del bloque. La configuración se hace desde este dando doble clik sobre el bloque, en tanto las funciones propias de cada bloque se utilizan en el archivo principal. Todos los bloques tiene por defecto la función Start() que permite inicializar el elemento desde el programa principal, sin esta función no se pueden utilizar las demás funciones, y se obtendrá un error de compilación. El pin digital 3 se configuró para generar un disparo que permita encender o apagar el sistema de refrigeración.

Archivo .cywr configuración de pines.

Este archivo permite configurar y elegir cada uno de los pines que se utilizan en el programa principal y que es están asociados a los bloques del archivo .cysch (ver Figura 2).



Alias	Name	Port	Pin	Lock
	\LCD:LCDPort[6:0]\	P2[6:0]	95..99,1..2	☑
	Pin_2	P12[3]	68	☑
	Pin_3	P1[6]	27	☑
	PIN_1	P5[1]	17	☑
	Rx_1	P1[2] XRES	22	☑
	SensorLM1	P4[3]	81	☑
	SensorLM2	P4[1]	70	☑
	SensorLM3	P2[7]	3	☑
	SensorLM4	P5[7]	34	☑
	SensorLM5	P0[5] OpAmp-	77	☑
	SensorLM6	P0[7] IDAC:HI	79	☑
	SensorTermo2	P4[7]	85	☑
	Tx_1	P12[6]	29	☑

Figura 2. Configuración de pines

Anexo C - CIRCUITOS DE ENCENDIDO DEL SISTEMA DE REFRIGERACIÓN

La circuitería externa de este sistema es mínima, ya que casi todo va conectado a la tarjeta PSoC3. Por lo tanto, sólo se usó unos circuitos bastante sencillos que sirven de apoyo al sistema. Estos circuitos son: Circuito para realizar el disparo que enciende o apaga el sistema de refrigeración y el otro, es un circuito para maneja la potencia del Motor DC y también realizar la conexión del servomotor.

Circuito de disparo para encender el sistema de refrigeración

El circuito de disparo para encender el sistema de refrigeración, como su nombre lo indica, es el encargado de encender o apagar el compresor (ver Figura 1), de acuerdo con la señal que recibe desde la PSoC3 a través del pin 3 del amplificador operacional LM358. Este circuito, a su vez protege la tarjeta PSoC3, ya que evita pulso o picos de voltaje que puedan ser enviados a través de este pin hacia la tarjeta. Esto se hace mediante el optoacoplador, el cual separa físicamente los dos circuitos tal como se puede ver en la siguiente figura.

Este circuito consta de:

1 Triac BT136

1 Optoacoplador MOC 3010

1 Amplificador Operacional LM358

2 Resistencia de 180 Ω

1 Resistencia de 2.7 K Ω

1 Capacitor de 0.1µF

1 Fusible

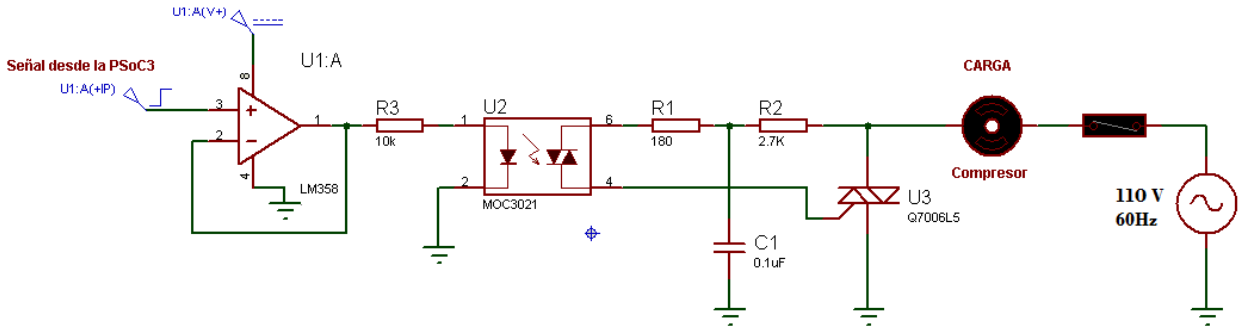


Figura 1. Circuito de disparo del triac

Anexo D - CIRCUITO DE POTENCIA DEL MOTOR DC

Este circuito se hizo para activar el motor DC del ventilador (ver Figura 1), el cual funciona a 12 Voltios con una corriente máxima de 6 Amperios, por lo tanto requiere una etapa de acople con el fin de proteger la tarjeta PSoC3 de posibles picos de voltaje generados en el motor, haciendo el acople de la señal PWM. Además este circuito incluye los conectores del servomotor.

Este circuito consta de:

1 Mosfet IRF540

2 diodos 1N4004

2 Transistores Tip 42

1 Transistor BD 135

1 Capacitor de 100nF

1 Resistencia de 15Ω

1 Resistencia de 33Ω

1Resistencia de 100Ω

1 Resistencia de 330Ω

2 Resistencia de 1KΩ

1 Resistencia de 10KΩ

1 Optoacoplador 4N35

1 Amplificador Operacional LM358

3 Conectores de dos entradas

2 Conectores de tres entradas

Además fue necesario adicionar un regulador de voltaje de 5 voltios para el optoacoplador.

1 Regulador de voltaje 7805

1 Capacitor de 0.1µF

1 Capacitor de 0.33µF

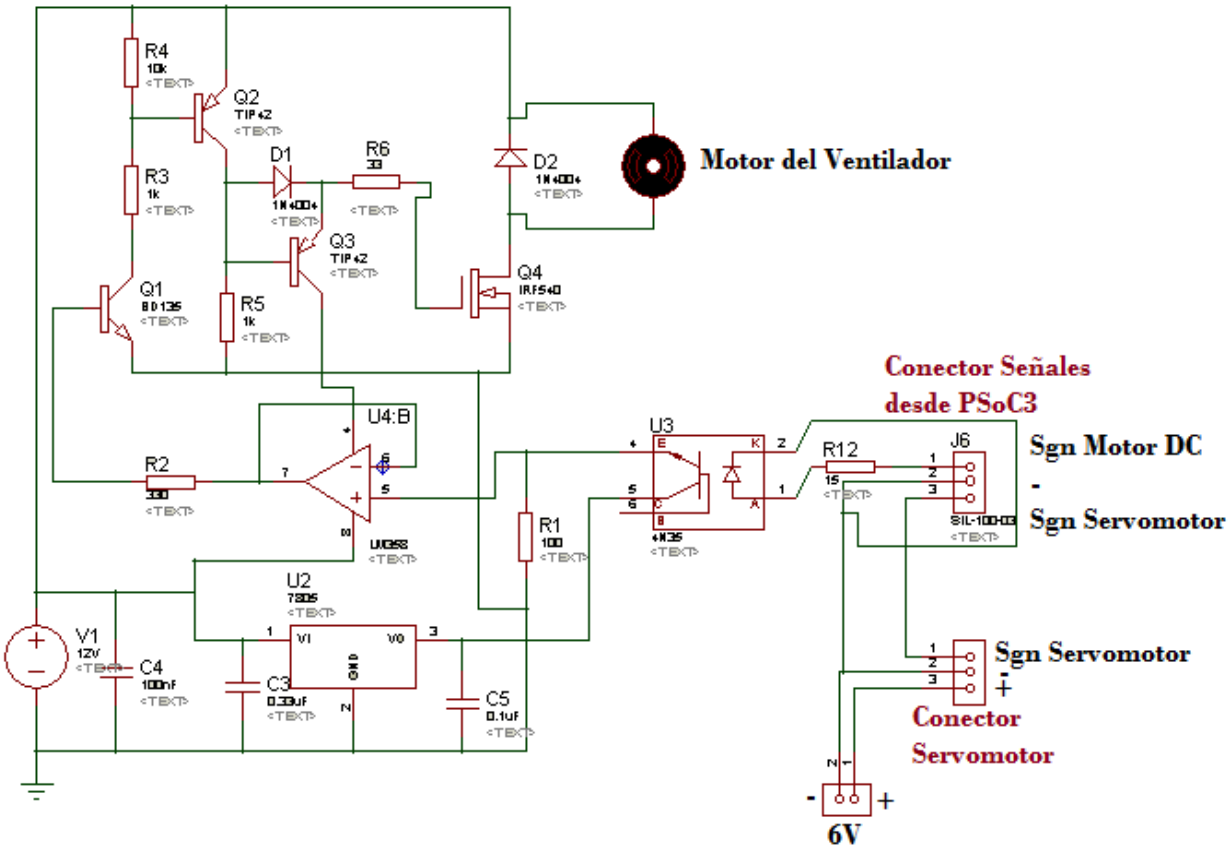


Figura 1. Circuito de potencia del motor DC

ANEXO E - INTERFACE PRELIMINAR EN LABVIEW

Se diseñó una interface en Labview 2010, la cual permite visualizar numérica y gráficamente las temperaturas leídas por los sensores. El programa también permite enviar un dato numérico hacia el serial, dicho dato se utilizó para enviar el valor de SetPoint en las pruebas. En otros casos también se usó para enviar el valor proporcional a las señales de PWM de prueba. Estos datos se guardan con periodos definidos de tiempo en un archivo .lvm. Esta interfaz sirvió para realizar las pruebas que permitieron visualizar el funcionamiento del sistema (ver Figura 1).

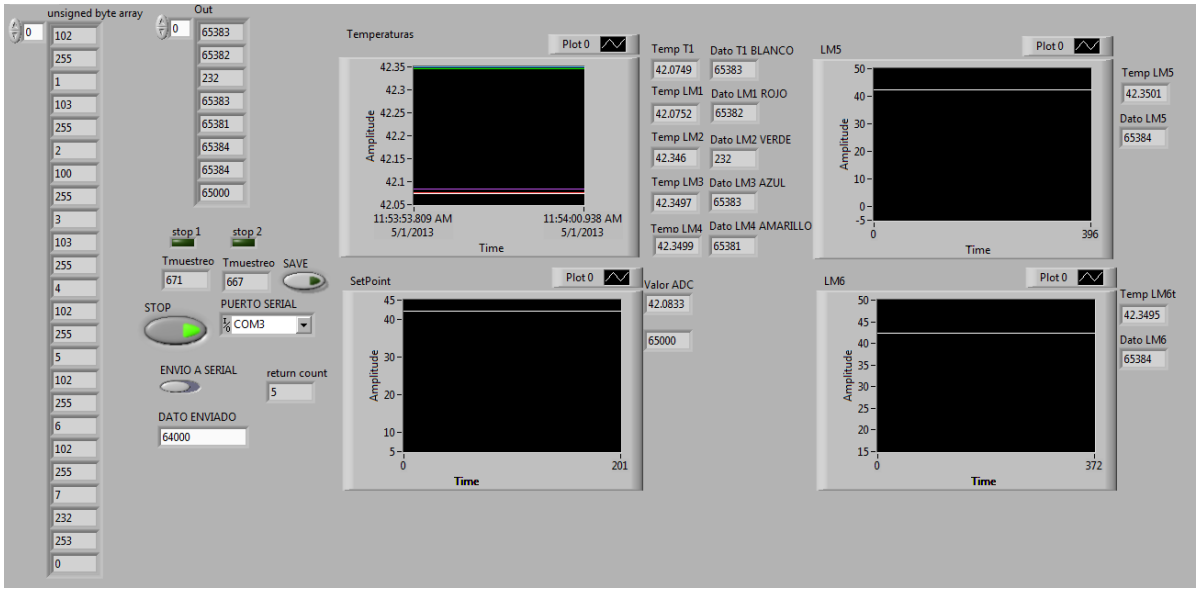


Figura 1. Interfaz montada en Labview

En la interfaz podemos ver una columna de 23 casillas en donde se puede observar como ingresan los datos. Se pueden distinguir las etiquetas que van de 0 a 7 de los datos enviados desde el la tarjeta PSoC. Los dos datos que se encuentran entre los identificadores corresponden al valor bajo y alto respetivamente de los valores de temperatura. El valor bajo y alto hacen referencia a los datos en que se divide le valor de 16 bits de la señal entregada por los sensores. La columna de 8 casillas muestra el dato armado de la señal entregada por los sensores en valor de ADC.

EL botón de STOP permite iniciar y detener el sistema de adquisición, mientras stop1 y stop2 permiten parar el tiempo de muestreo de adquisición de los datos.

Con el botón SAVE se puede guardar o detener la grabación de los datos recibidos. En la pestaña con el nombre PUERTO SERIAL se puede elegir el puerto al que se encuentra conectada la tarjeta PSoC. Por último la ventana nombrada DATO EVIADO deja ingresar el valor en ADC del Set Point o dado el caso el valor proporcional del PWM. Para enviar el dato se tiene que pulsar el botón llamado ENVIO SERIA, para que la sentencia sea válida.

Los cuatro recuadros permiten la visualización de las señales de los sensores, además de la temperatura de referencia o dado el caso la señal del PWM. El primer recuadro muestra las señales de los sensores: T1 (termocupla), LM1, LM2, LM3, LM4 y la señal de Referencia. El segundo recuadro al lado derecho permite visualizar la gráfica del sensor LM5 ubicado en el evaporador, mientras que en el tercer recuadro en la parte inferior se puede mirar la señal de referencia de temperatura o de PWM según corresponda. Por último al lado derecho en la parte inferior se puede observar la gráfica de la temperatura del LM6 ubicado en la parte externa del sistema para monitorear la temperatura ambiente.

ANEXO F - INTERFACE REALIZADA EN QT CREATOR

La interface se realizó en Qt creator, a continuación se muestran los códigos del programa.

Archivo principal extencion .cpp

```
#include "maingui.h"
#include <QApplication>
#include "serialthread.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainGUI w;
    SerialThread ThreadS;
    ThreadS.start();
    w.show();

    return a.exec();
}
```

Se inicia la aplicación de la interface de usuario, y se crea un objeto de la clase gui principal, también se crea el hilo para la comunicación serial.

Archivo maingui.h.

En el maingui.h se crea la case de la interface de usuario, las variables, signals y slots principales para generar le gui y chart de las gráficas.

```
#ifndef MAINGUI_H
#define MAINGUI_H

#include <QMainWindow>
#include <qwt.h>
#include <qwt_plot.h>
#include <qwt_plot_curve.h>
#include <qwt_symbol.h>
#include <qwt_math.h>
#include <qsize.h>
#include <time.h>
#include <string>

#define SIZEXAXIS 20
#define MAXTEMP 50


namespace Ui {
class MainGUI;
}

class MainGUI : public QMainWindow
```

```
{
    Q_OBJECT

public:
    explicit MainGUI(QWidget *parent = 0);
    ~MainGUI();
    void timerEvent(QTimerEvent *);
    long calc_sizefile(FILE *);


private slots:
    void on_StartButton_clicked();

    void on_StopButton_clicked();

    void on_InitCtrl_clicked();

    void on_PWMMotorSlider_sliderMoved(int position);

    void on_PWMServoSlider_sliderMoved(int position);

    void on_SetPointDial_dialMoved(int value);

    void on_tabWidget_currentChanged(int index);

    void on_commandLinkButton_clicked();

    void on_StopSaveDataButton_clicked();

    void on_SaveDataButton_clicked();

    void on_SampleSpin_valueChanged(const QString &arg1);

    void on_SampleSpin_editingFinished();

private:

    Ui::MainGUI *ui;

    /************************************/
    // Variables usadas para graficar

    QwtPlotCurve *Curve;
    QwtPlot *Plot1;

    unsigned int TimerCount=0;
    unsigned int TimerCountTemp=0;//Variable que guarda los ticks del
temporizador
    unsigned int TimerCountTemp2=0;
    unsigned char IndexTab=0;
    bool SaveData=false;
    unsigned int IDTimer;


    /************************************/
};
```

```
#endif // MAINGUI_H
```

Archivo serialthread.h

En este archivo se crea una clase de tipo thread para manejar un hilo de proceso con la comunicación serial.

```
#ifndef SERIALTHREAD_H
#define SERIALTHREAD_H
#include <QtCore>

extern float datos[3];
extern unsigned int counte;
extern unsigned char FlagGetSerialData;
extern double Temperature;
extern bool InitProcess;
extern unsigned int DataSendUART;
extern unsigned char HeaderUART;

class SerialThread:public QThread
{
public:
    SerialThread();
    void run();
};

#endif // SERIALTHREAD_H
```

Archivo serialthread. Cpp

Este archive tiene los métodos del hilo de la comunicación serial, el método run tiene el ciclo principal de hilo, en este ciclo principal se lee el Puerto serial y se guarda la información leída en un búfer, y se comunica con la clase de la interface de usuario principal.

```
#include "serialthread.h"
#include "configserial.cpp"


SerialThread::SerialThread()
{

}

void SerialThread::run()
{

        counte=0;

       ////////////////////////////////////////////
      HANDLE my;
      my=SerialInit(L"COM3",9600);
      uint8 dataS;
```

```
        char txchar;
        char txchar1;
        char txchar2;

        unsigned int DigitalTemp;


        DWORD iBytesRead;
        DWORD iBytesWritten;

       while(1)
        {
           while(!InitProcess);

           counte++;
          // qDebug()<<counte;

           //QThread::msleep(10);
           for(int i=0;i<3;i++)
           {
           while(!ReadFile(my,&dataS,1,&iBytesRead,NULL));
           datos[i]=dataS;
           }

               txchar=(char)HeaderUART;
               txchar2=DataSendUART/256;
               txchar1=DataSendUART-txchar2*256;

           if(FlagGetSerialData)
           { QThread::msleep(50);
             while(!WriteFile(my,&txchar,1,&iBytesWritten,NULL));
                qDebug()<<" Data1:"<<txchar;
             QThread::msleep(50);
            while(!WriteFile(my,&txchar1,1,&iBytesWritten,NULL));
               qDebug()<<" Data2:"<<txchar1;
            QThread::msleep(50);
             while(!WriteFile(my,&txchar2,1,&iBytesWritten,NULL));
    qDebug()<<" Data3:"<<txchar2;
             FlagGetSerialData=0;

           }

           // Evalua el Header de la trama del Serial
           switch((unsigned int)datos[0])
           {
              case 2:{
                      DigitalTemp=(double)(datos[1]+ 256*datos[2]);

Temperature=(((double)DigitalTemp*3.42/(65404))*13.2779-2.4695)*1.023-
0.5054;
                      break;
                      }

           }


           //qDebug()<<" ID:"<<datos[0];
```

```
    }
}
```

Archivo maingui.cpp

```
#include "maingui.h"
#include "ui_maingui.h"
```

Búfer en donde se almacenan los datos del sensor y los datos de tiempo.

```
double xval[100000];
double yval[100000];

QTextCursor CursorTempTxt;
FILE *DataLog;

unsigned int TIMER_PERIOD=200;
/***********************************/
```

Variables que comparten memoria los Threads las variables las usa el thread del Gui y el thread del serial.

```
float datos[3];
unsigned int counte;
unsigned char FlagGetSerialData;
double Temperature;
bool InitProcess;
unsigned int DataSendUART;
unsigned char HeaderUART;
/***********************************/
```

Constructor de la clase de la interface de usuario principal.

```
MainGUI::MainGUI(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainGUI)
{
    ui->setupUi(this);

    char CharTemp[30];
    QSize suze(200,500);
```

Instrucciones para crear los objetos necesarios para graficar los datos del sensor online. Se crea además un archivo plano LOG para grabar los datos leídos del sensor en todo el proceso de registro de la prueba.

```
    Plot1=new QwtPlot(this);
    Curve = new QwtPlotCurve("PLOTEO");
```

```
    Curve->setRenderHint(QwtPlotItem::RenderAntialiased);
    Curve->setPen(QPen(Qt::blue));
    Curve->setSymbol(new QwtSymbol(QwtSymbol::Ellipse, Qt::black,
QPen(Qt::black), QSize(2, 2)));
    Curve->setLegendAttribute(QwtPlotCurve::LegendShowLine);

    Plot1->setGeometry(390,100,500,300);
  //  Plot1->hide();

    sprintf(CharTemp,"     %d",ui->PWMMotorSlider->value());
    ui->PWMMotorTxt->setText(CharTemp);

    sprintf(CharTemp,"     %d",ui->PWMServoSlider->value());
    ui->PWMServoTxt->setText(CharTemp);

    sprintf(CharTemp,"     %d",ui->SetPointDial->value());
    ui->SetPointTxt->setText(CharTemp);

    DataLog=NULL;
    DataLog=fopen("I:\DataLog.txt","w");

    if(calc_sizefile(DataLog)>=400000000)
    DataLog=fopen("I:\DataLog.txt","w");

    fclose(DataLog);
}
```

Método que calcula el tamaño del archive LOG para borrar la memoria cuando supere la capacidad de 400000000 Bytes.

```
long MainGUI::calc_sizefile(FILE *arch)
{
    long nBytes;
    fseek(arch, 0, SEEK_END); // Colocar el cursor al final del fichero
    nBytes = ftell(arch); // Tamaño en bytes

    return nBytes;
}
```

Destructor de la clase de interface de usuario.

```
MainGUI::~MainGUI()
{
    delete ui;
}
```

Evento de Timer se ejecuta cada TIMER_PERIODO en milisegundos. En este método se actualizan en la interface las gráficas y los datos leídos de los sensores.

```
// Evento de Timer
// Ejecuta Esta Funcion Cada TIMER_PERIOD (valor) milisegundos
void MainGUI::timerEvent(QTimerEvent *t)
{
    char TempPrint[10];
    if(InitProcess)
    {
     char DataTemp[30];

    xval[TimerCount]=(double)TimerCountTemp*TIMER_PERIOD/1000;
    yval[TimerCount]=(float)Temperature;

    TimerCount++;
    TimerCountTemp++;

    if(SaveData)
    {

        DataLog=fopen("I:\DataLog.txt","a");
        fprintf(DataLog, "\n");
        sprintf(TempPrint,"%.3f",Temperature);
        fprintf(DataLog, TempPrint);
        fprintf(DataLog, "\t");

sprintf(TempPrint,"%.3f",(float)TimerCountTemp*TIMER_PERIOD/1000);
        fprintf(DataLog, TempPrint);
        fclose(DataLog);
    }
        sprintf(DataTemp,"TEMPERATURE DATA:          %.3f",Temperature);
        ui->TemperatureTxt->append(DataTemp);
        ui->TemperatureTxt->update();

    Curve->setRawSamples( xval, yval, TimerCount );
    Plot1->setAxisScale(QwtPlot::xBottom, TimerCountTemp2*SIZEXAXIS,
TimerCountTemp2*SIZEXAXIS+SIZEXAXIS);
    Plot1->setAxisScale(QwtPlot::yLeft, 0,MAXTEMP);
    Curve->attach(Plot1);
    Plot1->replot();

    if(IndexTab==1)
    Plot1->show();


    if(TimerCount==(unsigned int)SIZEXAXIS*1000/TIMER_PERIOD)
        {
        TimerCount=0;
        TimerCountTemp2++;
    }
  }

    CursorTempTxt = ui->TemperatureTxt->textCursor();
    CursorTempTxt.movePosition(QTextCursor::End);
    ui->TemperatureTxt->setTextCursor(CursorTempTxt);
}
```

En este método se inicia el proceso de registro de los sensores y se inicia el plot de los sensores.

```
void MainGUI::on_StartButton_clicked()
{

       InitProcess=true;

       killTimer(IDTimer);
       IDTimer=startTimer(TIMER_PERIOD);

       ui->StatusTxt->setText("               START");

       ui->TemperatureTxt->append("INIT ACQUISITION...");

       ui->SampleSpin->setDisabled(true);


}
```

Este evento detiene el proceso de registro de los sensores y el despliegue en la interface de usuario.

```
void MainGUI::on_StopButton_clicked()
{
      InitProcess=false;
      ui->StatusTxt->setText("             STOP");

      ui->TemperatureTxt->append("STOP ACQUISITION");

       ui->SampleSpin->setDisabled(false);

       TimerCount=0;
       TimerCountTemp=0;
       TimerCountTemp2=0;
}
```

Este método envía los comandos desde la interface hasta el Puerto serial para comunicarse con el sistema electrónico. Se arranca o apaga el motor dc, se envía el PWM para el SetPoint del servo motor, se arranca o para el compresor, se define el tipo de control manual o automático.

```
void MainGUI::on_InitCtrl_clicked()
{
       FlagGetSerialData=1;
       char DataTemp[40];
```

```
        char DataT[40];

        switch(ui->SelectCommandBox->currentIndex())
        {
        case 0:{HeaderUART=1;
             DataSendUART=1;
             ui->TemperatureTxt->append("Set Pump On");
             ui->TemperatureTxt->update();
             break;}
        case 1:{HeaderUART=1;
             DataSendUART=0;
             ui->TemperatureTxt->append("Set Pump Off");
             ui->TemperatureTxt->update();
             break;}
        case 2:{HeaderUART=2;
             DataSendUART=0;
             ui->TemperatureTxt->append("Set Manual Control");
             ui->TemperatureTxt->update();
             break;}
        case 3:{HeaderUART=2;
             DataSendUART=1;
             ui->TemperatureTxt->append("Set Automatic Control");
             ui->TemperatureTxt->update();
             break;}
        case 4:{HeaderUART=3;
             DataSendUART=ui->PWMServoSlider->value();
             sprintf(DataTemp,"%d",DataSendUART);
             strcpy(DataT,"SET PWM SERVO:                ");
             strcat(DataT,DataTemp);
             ui->TemperatureTxt->append(DataT);
             ui->TemperatureTxt->update();
             break;}
        case 5:{HeaderUART=4;
            DataSendUART=ui->PWMMotorSlider->value();
            sprintf(DataTemp,"%d",DataSendUART);
            strcpy(DataT,"SET PWM DC MOTOR:         ");
            strcat(DataT,DataTemp);
            ui->TemperatureTxt->append(DataT);
            ui->TemperatureTxt->update();
            break;}
        case 6:{HeaderUART=5;
            DataSendUART=ui->SetPointDial->value();
            sprintf(DataTemp,"%d",DataSendUART);
            strcpy(DataT,"SET POINT:                            ");
            strcat(DataT,DataTemp);
            ui->TemperatureTxt->append(DataT);
            ui->TemperatureTxt->update();
        break;}

        }
}
```

En este método se lee el valor del control Slider para tomar la referencia del motor DC.

```
void MainGUI::on_PWMMotorSlider_sliderMoved(int position)
{
    char PWMTxtTemp[10];

    sprintf(PWMTxtTemp,"     %d ",ui->PWMMotorSlider->value());
    ui->PWMMotorTxt->setText(PWMTxtTemp);
}
```

En este método se lee el valor del control Slider para tomar la referencia PWM del servomotor.

```
void MainGUI::on_PWMServoSlider_sliderMoved(int position)
{
    char PWMTxtTemp[10];

    sprintf(PWMTxtTemp,"     %d ",ui->PWMServoSlider->value());
    ui->PWMServoTxt->setText(PWMTxtTemp);

}

Se configura el SetPoint desde la interface
void MainGUI::on_SetPointDial_dialMoved(int value)
{
    char SetPointTemp[10];

    sprintf(SetPointTemp,"     %d ",ui->SetPointDial->value());
    ui->SetPointTxt->setText(SetPointTemp);


}


void MainGUI::on_tabWidget_currentChanged(int index)
{
    IndexTab=index;

    switch(IndexTab)
    {
    case 0:Plot1->hide();break;
    case 1:Plot1->show();break;
    }
}
```

Detiene el proceso de grabar los datos de los sensors en el archive LOG de registro.

```
void MainGUI::on_StopSaveDataButton_clicked()
{
    SaveData=false;

    ui->TemperatureTxt->append("STOP SAVE DATA");
    ui->TemperatureTxt->update();
}
```

Inicia el proceso de grabado de los datos de los sensores en el archive LOG de registro.

```
void MainGUI::on_SaveDataButton_clicked()
{
    SaveData=true;
    TimerCount=0;
    TimerCountTemp=0;
    TimerCountTemp2=0;

    ui->TemperatureTxt->append("INIT SAVE DATA");
    ui->TemperatureTxt->update();
}
```

Este método permite actualizar el valor de TIMER_PERIOD, equivalente al tiempo mínimo en el que la interface se actualice.

```
void MainGUI::on_SampleSpin_editingFinished()
{

    char DataTemp[40];
    char DataT[40];

    TIMER_PERIOD=ui->SampleSpin->value();


    sprintf(DataTemp,"%d",TIMER_PERIOD);
    strcpy(DataT,"CHANGE SAMPLE TIME:                ");
    strcat(DataT,DataTemp);
    ui->TemperatureTxt->append(DataT);
    ui->TemperatureTxt->update();
}
```