industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Diseño y construcción mediante Arduino de un sistema de medición de temperaturas en tiempo real

TRABAJO FIN DE GRADO

GRADO EN TECNOLOGÍAS INDUSTRIALES

Autor: **Miguel Aparicio Albaladejo**
Director: Jose Ramón Navarro Andreu
Codirector: Andrés Cabrera Lozoya

Cartagena, 22/09/2016

Universidad Politécnica de Cartagena

Agradecimientos

A mis padres, Miguel Aparicio Pérez y Loli Albaladejo Conesa, por la oportunidad que me dieron, y por facilitarme su ayuda moral y económica durante estos años de carrera, lo que ha supuesto finalizarla con éxito.

Agradecer también al resto de familia y amigos que me han apoyado a lo largo de este tiempo, pero en especial a mi tío Antonio por su constante interés en mi progreso, y a mi querida abuela Juana por su cariño y fe.

Al director de este TFE, Jose Ramón Navarro Andreu, por haberme dado la oportunidad de realizar este proyecto, y al codirector del mismo, Andrés Cabrera Lozoya, por su ayuda, disponibilidad, y dedicación mostrada en todo momento.

Agradecer las colaboraciones de profesores del Departamento de Tecnología Electrónica de la UPCT, en concreto a Manuel Jiménez Buendía, por la ayuda recibida en LabVIEW, y a Juan Suardíaz Muro, artífice de la PCB.

A las personas que me ayudaron directa e indirectamente en este periodo de aprendizaje, desde profesores hasta los propios compañeros.

De igual modo, la elaboración de este trabajo no hubiese sido posible sin recurrir a las fuentes de conocimiento y a la experiencia de numerosas personas que de manera libre o adscrita a una entidad pública o privada han trabajado en el desarrollo de aplicaciones tanto hardware como Software para la Plataforma Open Hardware Arduino.

Agradecer también la valiosa labor de NI (National Instruments) al atender esta demanda de la comunidad académica en el desarrollo de LIFA (LabVIEW Interface for Arduino). De ellos he utilizado información que me ha permitido desarrollar este proyecto.

Miguel Aparicio Albaladejo

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

CAPÍTULO 1. Antecedentes y motivación. Objetivos

Los antecedentes son los que han motivado la elaboración de este TFE; es primordial conocer lo que tenemos para mejorar sobre ello. A continuación realizamos un **análisis** minucioso del **equipo vigente**:

- Obsolescencia del sistema de medición actual, motivada no por un mal funcionamiento del mismo, sino por un insuficiente desempeño de sus funciones en comparación con los equipos y tecnologías existentes en el mercado.

- Los resultados obtenidos durante la hora que perdura la monitorización de temperaturas solo son visibles una vez que ésta ha finalizado. De este modo el alumnado, mientras que realiza la práctica, no es capaz de visualizar con claridad el comportamiento térmico de los diferentes materiales. A continuación adjuntamos dos capturas que muestran la forma en la que este equipo recoge los resultados obtenidos al finalizar la monitorización.

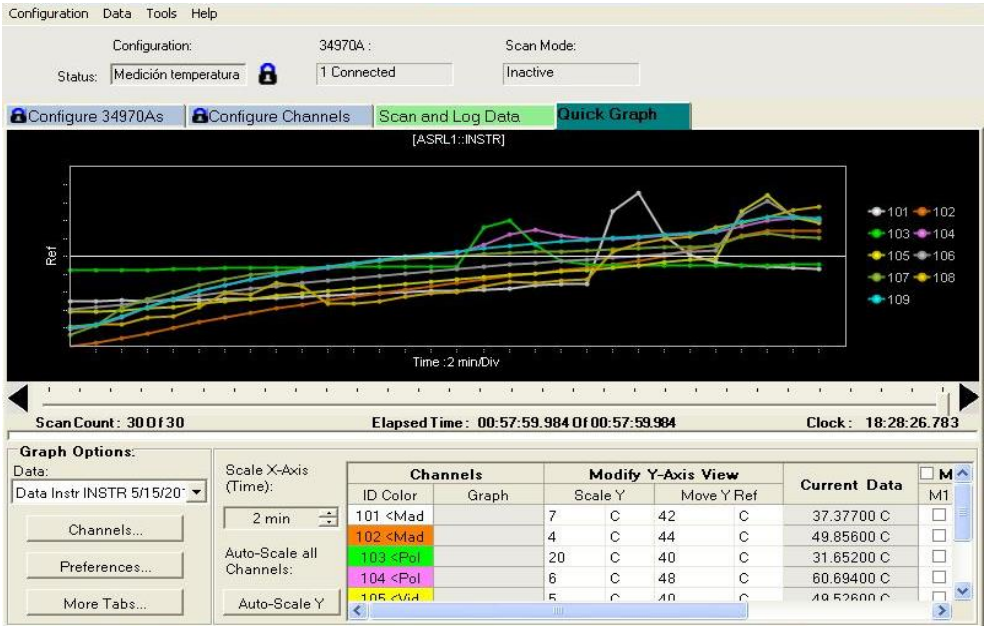


Figura 1.1- Graficación de los resultados obtenidos

Acquisition Date:	05/02/2016 17:54								
Scan	101 <Mad ext> (C)	102 <Mad int> (C)	103 <Pol ext> (C)	104 <Pol int> (C)	105 <Vid ext> (C)	106 <Vid int> (C)	107 <Aire int> (C)	108 <Fibra ext> (C)	109 <Fibra int> (C)
1	21,674	21,682	21,797	21,908	22,307	22,404	22,265	22,259	22,225
2	21,906	22,281	22,078	22,935	22,323	22,809	25,938	22,346	23,336
3	22,095	23,462	22,533	25,514	22,724	23,686	32,375	23,536	25,807
4	22,303	24,657	23,038	28,529	23,204	24,702	37,611	24,367	28,689
5	22,578	25,771	23,561	31,214	23,775	25,698	41,398	24,961	31,355

Figura 1.2- Datos exportados a Excel

- La unidad de adquisición de datos empleada es un Agilent 34970A de 64 canales, cuyo mantenimiento es tedioso y costoso; se trata de tecnología anticuada donde la disponibilidad de repuestos es baja. Existe una necesidad de actualizarnos a los tiempos de ahora.

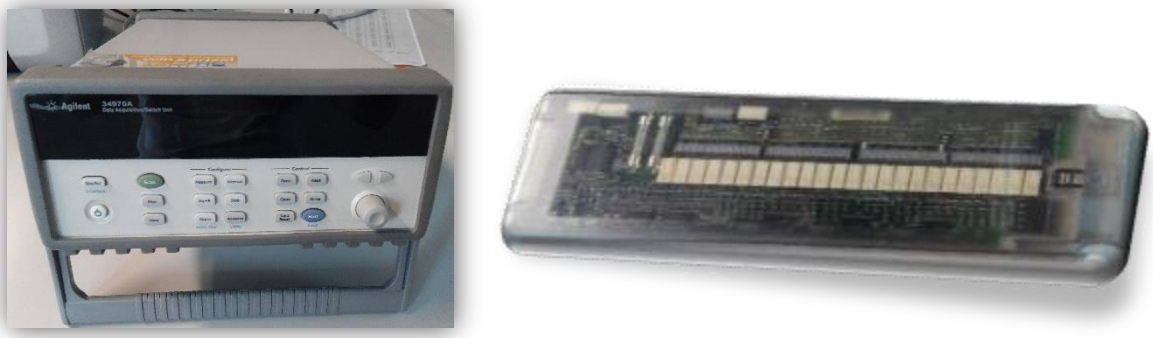


Figura 1.3- Equipo Agilent 34970A de 64 canales

- La interfaz de salida de dicho multiplexor se realiza mediante el estándar de comunicación RS-232. Se puede considerar obsoleto, teniendo en cuenta que en la actualidad la mayoría de equipos no implementan este tipo de conexión.

- Éste incluye un software propietario, por lo que estamos limitados a las capacidades que nos pueda ofrecer. Este hecho produce un aumento en el precio del equipo que lo incorpora; el software también está fijado en el precio del instrumento de medición.

- En cuanto a los sensores, la tolerancia es de $\pm$ 1,5 °C.

Figura 1.4- Termopares empleados

Una vez presentadas las limitaciones del equipo vigente, podemos establecer los **objetivos a satisfacer**; motivo principal del TFE.

- Este proyecto tiene como objetivo diseñar y construir mediante el microcontrolador Arduino un sistema económico para la adquisición de temperaturas en tiempo real. Dicha magnitud será registrada a partir de sensores de temperatura instalados en una maqueta ubicada en el laboratorio del departamento de Física Aplicada de la UPCT. La maqueta cúbica, formada por diferentes materiales, nos servirá para conocer la respuesta transitoria ante procesos de calentamiento de los mismos.

- Los datos tomados por los sensores pasan a formar parte de un entorno informático que nos permita el análisis y comprensión, así como una gestión eficiente de los mismos. La aplicación informática a desarrollar debe permitir que un usuario de la misma, tenga información en tiempo real de las diferentes temperaturas que se registren en nuestro sistema. Además será necesario que la aplicación tenga un funcionamiento fácil e intuitivo.

- Uno de los pasos más determinantes del proyecto será la elección del sensor de temperatura que forme parte de nuestro sistema, éste deberá ajustarse perfectamente a todos los requerimientos que exige nuestro proyecto. Una premisa de diseño importante a la hora de decidir, será la resolución en las medidas de los sensores y su sensibilidad, debiendo llegar a un equilibrio que permita tener un sistema de medición lo más preciso posible.

Fijados los objetivos a cumplir, finalmente el equipo desarrollado para sustituir el actual consta de las siguientes **características:**

- Al emplear el microcontrolador Arduino como DAQ, éste nos permite construir un equipo económico, y donde se reduce drásticamente el espacio ocupado por el mismo.

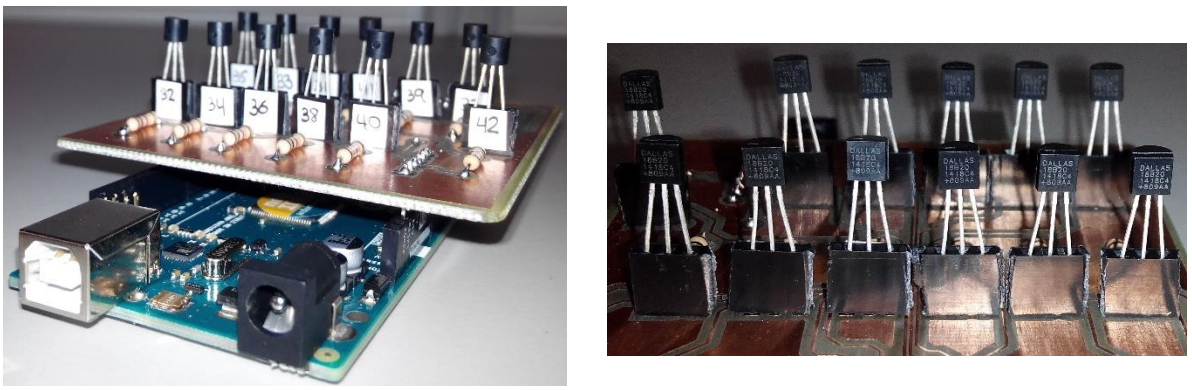


Figura 1.5- DAQ y sensores empleados

- El entorno informático utilizado para realizar dicho análisis y comprensión es LabVIEW. Destacar que aunque se trata de un software de pago, está disponible para utilizarlo dentro de la red interna de la UPCT.

- El alumno ahora es capaz de comprender y visualizar los resultados obtenidos por medio de una interfaz con gráficos en tiempo real; innovación docente.

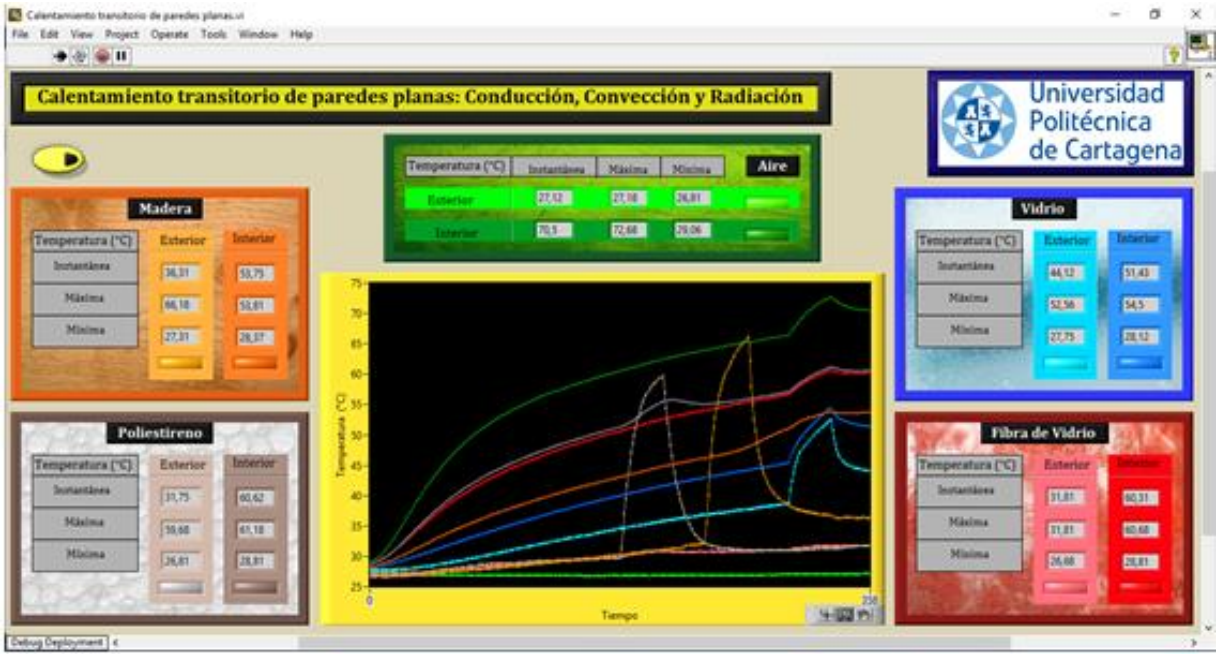


Figura 1.6- Interfaz de usuario

- La interfaz de salida de dicha placa se realiza mediante cable USB AM-BM, cuya conexión al ordenador es mediante un puerto USB. Al contrario del multiplexor mencionado con anterioridad, Arduino está actualizado a la demanda actual.

- Arduino al ser open-hardware, tanto su diseño como su distribución son libres, es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

- Los sensores empleados en este proyecto son digitales, tienen una resolución de 0,06 °C, y una tolerancia de $\pm$ 0,5 °C (en el rango de temperaturas en el que vamos a trabajar).

CAPÍTULO 2. Dimensionado de los componentes y elección de los mismos dentro de la oferta de mercado.

En este capítulo se ha realizado una búsqueda de los componentes, tanto de adquisición de datos y comunicación con la computadora, como de los sensores de temperatura, con el objetivo de elegir correctamente los que mejor se adapten a nuestros requisitos básicos.

2.1. Adquisición de datos y comunicación

Lo primero que se estudió, fue la manera con la cual comunicaríamos los sensores con la computadora, para la adquisición de los datos.

Para ello existen numerosos sistemas, como PLC (Programmable Logic Controller), DAQ (Data Acquisition Modules) y Arduino.

2.1.1 Arduino

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (boot loader) que corre en la placa.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado al software del ordenador. Al ser open-hardware, tanto su diseño como su distribución son libres, es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

2.1.2 Selección de la placa Arduino

Tras observar las alternativas anteriormente expuestas, se tomó la decisión de utilizar en la realización de este proyecto las placas Arduino, debido a su bajo coste, su gran flexibilidad, la posibilidad de funcionamiento autónomo (sin necesidad de conexión a pc) y la amplitud de componentes existentes, así como ser un open-hardware.

Una vez decidido que se iba a utilizar Arduino para la realización de este proyecto, pasamos a elegir la placa que más se adapte a nuestras necesidades.

Para la elección de nuestro hardware Arduino, existen una serie de datos importantes que condicionan la elección de la placa, según el uso que le vayamos a dar:

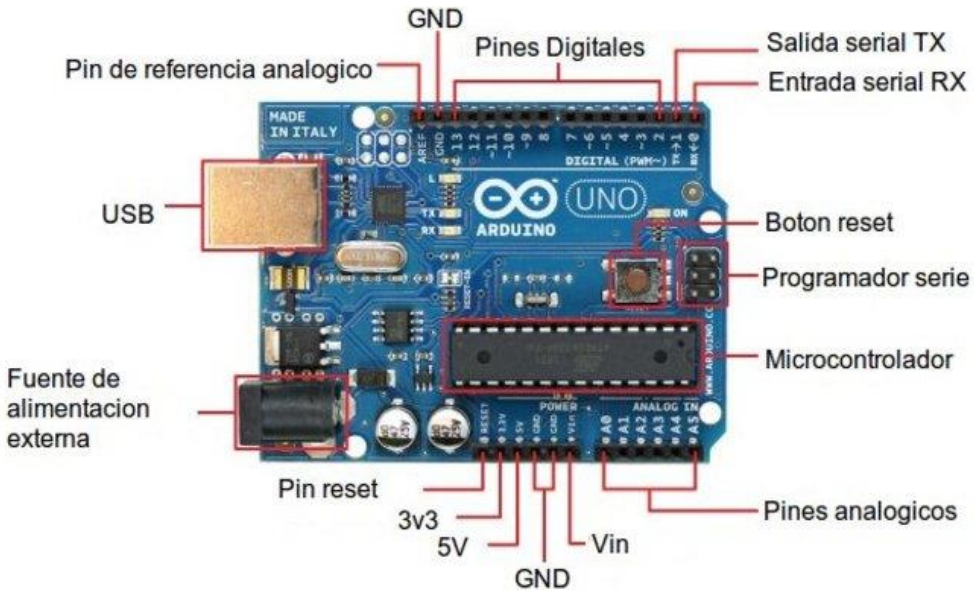


Figura 2.1- Composición de la placa Arduino

Lo primero que debemos conocer es el tipo de proyecto que vamos a implementar. Esto nos da una idea de la **cantidad de pines analógicos y digitales** (normales y de tipo PWM o modulados por ancho de pulso para simular una salida analógica) que necesitamos para nuestro trabajo. Este primer escrutinio nos permite descartar algunas placas más simples que no tengan suficientes pines o, al contrario, descartar las de mayor número de ellos para reducir los costes, puesto que con menos pines nos podríamos conformar.

También podemos deducir el **tamaño de código** que vamos a generar para nuestros sketchs. Un programa muy largo, con muchas constantes y variables demandará una cantidad mayor de memoria flash para su almacenamiento, por lo que se debe elegir una placa adecuada para no quedarnos cortos.

La **RAM** será la encargada de cargar los datos para su inmediato procesamiento, pero no es uno de los mayores escollos, puesto que esto solo afectaría a la velocidad de procesamiento. La RAM va ligada al microcontrolador, puesto que ambos afectan a la agilidad de procesamiento de Arduino.

Por último, en cuanto al **voltaje**, no importa demasiado a nivel electrónico, excepto en algunos casos, para tener en cuenta la cantidad de tensión que la placa puede manejar a la hora de montar el circuito. Cuando queremos prescindir de una fuente de alimentación externa, hay que tener en cuenta que este es el voltaje que se puede manejar, para no destruir la placa con sobretensiones no soportadas.

A continuación se pasa a presentar las alternativas existentes en el mercado:

- Arduino UNO

Arduino UNO es una placa microcontrolador basada en ATmega328. Tiene 14 entradas/salidas digitales (de las cuales 6 proporcionan salida PWM), 6 entradas analógicas, un cristal oscilador a 16Mhz, conexión USB, entrada de alimentación, una cabecera ISCP, y un botón de reset. Contiene todo lo necesario para hacer funcionar el microcontrolador; simplemente hay que conectarlo al ordenador con el cable USB o alimentarlo con un trasformador o batería.

Características:

- ✓ **Microcontrolador:** ATmega368
- ✓ **Tensión de alimentación:** 5V
- ✓ **Tensión de entrada recomendada:** 7-12V
- ✓ **Límite de entrada:** 6-20V
- ✓ **Pines digitales:** 14 (6 proporcionan salida PWM)
- ✓ **Entradas analógicas:** 6
- ✓ **Corriente máxima por pin:** 40 mA
- ✓ **Corriente máxima para el pin 3.3V:** 50 mA
- ✓ **Memoria flash:** 32 KB de las cuales 2 KB las usa el gestor de arranque.
- ✓ **SRAM:** 2 KB
- ✓ **EEPROM:** 1 KB
- ✓ **Velocidad de reloj:** 16 MHz

Figura 2.2- Arduino UNO

- Arduino Mega 2560

Arduino Mega es una placa microcontrolador basada en ATmega2560. Tiene 54 entradas/salidas digitales (de las cuales 14 proporcionan salida PWM), 16 entradas digitales, 4 UARTS (puertos serie por hardware), un cristal oscilador de 16MHz, conexión USB, entrada de corriente, conector ICSP y botón de reset. Contiene todo lo necesario para hacer funcionar el microcontrolador; simplemente hay que conectarlo al ordenador con el cable USB o alimentarlo con un trasformador o batería.

Características:

- ✓ **Microcontrolador:** ATmega2560
- ✓ **Tensión de alimentación:** 5V
- ✓ **Tensión de entrada recomendada:** 7-12V
- ✓ **Límite de entrada:** 6-20V
- ✓ **Pines digitales:** 54 (14 proporcionan salida PWM)
- ✓ **Entradas analógicas:** 16
- ✓ **Corriente máxima por pin:** 40 mA
- ✓ **Corriente máxima para el pin 3.3V:** 50 mA
- ✓ **Memoria flash:** 256 KB
- ✓ **SRAM:** 8 KB
- ✓ **EEPROM:** 4 KB
- ✓ **Velocidad de reloj:** 16 MHz

c

Figura 2.3- Arduino Mega 2560

Existen otras placas Arduino como la Diecimila, Nano, Lilypad, Fio, Mini, etc. Éstas o tienen unas aplicaciones diferentes a las que nos interesan, como son Lilypad y Fio, o son versiones antiguas, como Diecimila, o versiones reducidas, Mini o Nano.

Para este proyecto, la placa elegida es **Arduino Mega 2560**. Esto se puede justificar debido a que en nuestro proyecto se va a emplear un gran número de sensores de temperatura, que deberán ir conectados a la misma; esta placa es la que nos garantiza el mayor número de entradas.

2.2 Sensores de temperatura

2.2.1 Introducción

Los sensores de temperatura son dispositivos que transforman los cambios de temperatura en cambios en señales eléctricas que son procesados por equipo eléctrico o electrónico. Los sensores pueden ser de indicación directa (como el termómetro de mercurio, que aprovecha la propiedad que posee el mercurio de dilatarse o contraerse por la acción de la temperatura) o pueden estar conectados a un indicador (convertidor analógico-digital, un ordenador…) de forma que los valores puedan ser leídos por un operador y/o almacenados de forma digital.

Hay tres tipos de sensores de temperatura, los termistores, los RTD y los termopares.

El sensor de temperatura típicamente suele estar formado por el elemento sensor (de cualquiera de los tipos anteriores), la vaina que lo envuelve y que está rellena de un material muy conductor de la temperatura (para que los cambios se transmitan rápidamente al elemento sensor), y del cable al que se conectarán el equipo electrónico.

A continuación van a ser expuestas las principales características de cada tipo.

- Termopar: Son transductores de temperatura constituidos por dos alambres conductores hechos de metales diferentes y soldados por uno de sus extremos formando una unión. Al calentar esta última (unión de medida), se produce entre los extremos de la termocupla (uniones frías) un voltaje proporcional a la diferencia de temperaturas entre la unión caliente y cualquiera de las uniones frías, las cuales deben estar a una misma temperatura de referencia, generalmente 0°C. Este fenómeno se conoce como efecto termoeléctrico o Seebeck (Ver Figura 2.4).

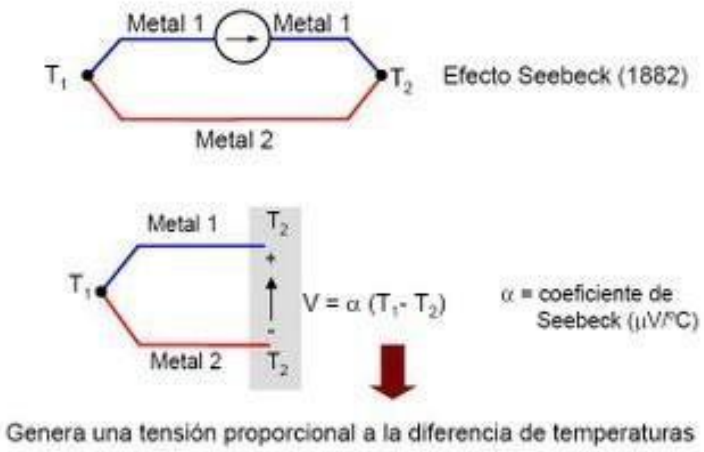


Figura 2.4- Termopar y efecto Seebeck

Sus principales ventajas son su precio económico, que son intercambiables, que sus conectores son estándar, y su capacidad para medir un amplio rango de temperaturas, lo que les hace ampliamente utilizados en la industria. Por el contrario, el principal inconveniente estriba en su precisión, ésta es pequeña en comparación con los sensores de temperatura RTD o termistores; difícilmente son capaces de obtener errores del sistema inferiores a un grado Celsius.

- Termoresistencia o RTD (Resistive Temperature Detector): Son sensores de temperatura cuyo principio físico se basa en la resistividad de los metales, es decir, en la variación de la resistencia de un conductor con la temperatura. Esto se debe a que al incrementar la temperatura, los iones vibran con mayor amplitud y así se dificulta el paso de los electrones a través del conductor.

  Entre las características que deben tener los metales, cabe destacar un alto coeficiente de resistencia y alta resistividad, para que tenga mayor sensibilidad y que haya una relación lineal entre la resistencia y la temperatura (Ver Figura 2.5). La máxima calidad de los RTD la dan los detectores de platino ya que permiten realizar medidas más exactas y estables, hasta una temperatura de aproximadamente 500 °C. Los RTD más económicos utilizan el níquel o aleaciones de níquel pero éstos no son tan estables ni lineales como los que emplean platino.

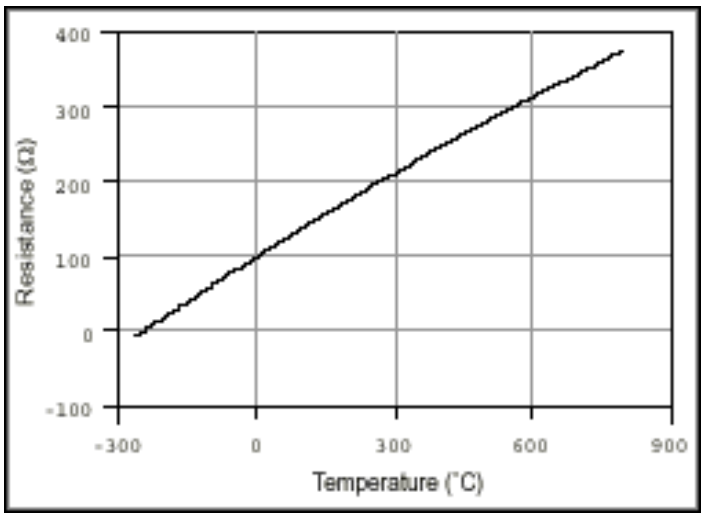


Figura 2.5- Curva Resistencia-Temperatura RTD de Platino 100Ω

Sus principales ventajas son su elevada linealidad, elevada precisión, buena estabilidad a largo plazo, son elementos pasivos, tiempos de respuesta pequeños, tienen elevados márgenes de temperatura. Por el contrario, sus principales puntos negativos son su elevado coste, su poca robustez, sensible autocalentamiento y sensibilidad menor que los termistores.

- Termistor: Son dispositivos basados en óxidos metálicos semiconductores, que exhiben un gran cambio en su resistencia eléctrica cuando se someten a cambios relativamente pequeños de temperatura. Existen dos tipos de termistores, los PTC (Positive Temperature Coefficient) y los NTC (Negative Temperature Coefficient), siendo estos últimos los más utilizados. Ambos tipos presentan una respuesta no lineal y decreciente con el aumento de la temperatura, en el caso de los NTC, y creciente, en el caso de los PTC.

  Los termistores pueden adoptar una gran variedad de formas y tamaños, llegando incluso a ser tan diminutos como la cabeza de un alfiler. El tipo de óxidos, las proporciones usadas, y el tamaño físico, determinan los rangos de resistencia y temperatura deseados para el dispositivo.

  Los termistores ofrecen varias ventajas con respecto a los RTD y los termopares. Por ejemplo, son más económicos, estables y confiables, proveen un alto grado de intercambiabilidad, pueden hacerse lo suficientemente pequeños para permitir la medición puntual, facilitan la medición remota a través de cables largos y tienen una mayor sensibilidad o respuesta de señal. La principal desventaja es su falta de linealidad (Ver Figura 2.6).

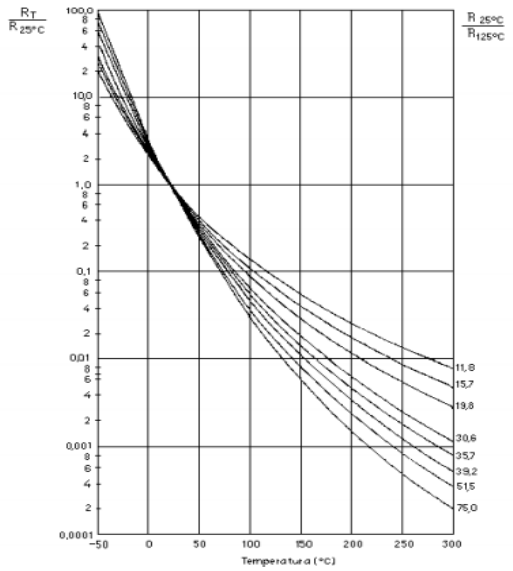


Figura 2.6- Variación de la resistencia de diversos termistores NTC con la temperatura

A modo de resumen, exponemos las ventajas y desventajas de todos ellos:

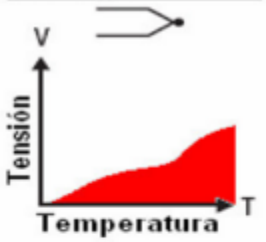
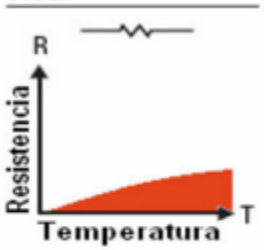
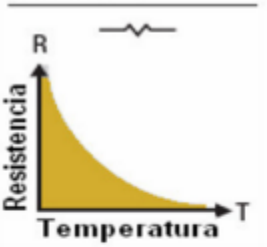
Termopar	RTD	Termistor
		
VENTAJAS Rango -270…+1 800 °C • Sencillo • Robusto, resistencia a vibraciones y golpes • Económico • Amplia variedad de formas físicas • Gran rango de temperatura	Rango -260…+850 °C • Muy estable • Amplio alcance de temperatura • Buena exactitud • Mejor linealidad que el termopar • Mejor deriva que el termopar	Rango -80…+150 °C • Tiempo de respuesta rápida • Medición a 2 hilos • Cambios grandes de resistencia vs. Temperatura • Pequeños • Baratos • Buena estabilidad
DESVENTAJAS • No lineal • Baja tensión • Requiere compensación en la unión fria • Baja sensibilidad • Baja estabilidad	• Caro • Frágil, sensible a vibraciones y golpes • Tiempo de respuesta lento • Requiere fuente de corriente • Cambios pequeños de resistencia • Para precisión requiere medir a 4 hilos	• No lineal • Requiere fuente de corriente • Alcance de temperatura limitado • Frágil

Figura 2.7- Comparativa de los distintos tipos de sensores de temperatura

2.2.2 Selección del sensor de temperatura.

Una vez expuestos los distintos tipos de sensores de temperatura, seleccionaremos aquel sensor que mejor se adapte a las premisas que impone nuestro sistema.

El mercado de los sensores de temperatura es muy amplio, en cuanto a la elección del mismo, prestaremos especial atención en:

- El precio. Para la realización de dicho proyecto necesitamos emplear 10 sensores del mismo tipo; debemos de ajustarnos al presupuesto de nuestro proyecto.
- El rango: Deben de registrar mediciones precisas, cuyo margen de error sea muy pequeño. Como mínimo, esta precisión debe de ser reflejada dentro del siguiente rango de temperaturas: 15°C < T < 85°C.
- La tolerancia. El error en nuestras mediciones debe de ser el mínimo posible, al menos en el rango de temperaturas especificado.
- La linealidad en la respuesta. Para este proyecto se emplearán exclusivamente, sensores que presenten relación lineal entre la tensión generada por dicho dispositivo y la temperatura.
- Las dimensiones. Serán instalados en las distintas superficies que conforman nuestro cubo. Por lo tanto, el tamaño de los mismos debe de ser reducido, prestando especial importancia en el espesor.

A continuación presentamos los sensores que pueden satisfacer nuestras necesidades, y que nos permiten una comunicación sencilla con Arduino. Todos ellos presentan linealidad en su respuesta.

- LM35: Este sensor es analógico y también el más económico de todos, se puede encontrar por menos de 1 €. Tiene un rango de medición de -55 a 150 °C, con una precisión de 0.25 °C a temperatura ambiente o de 0.75 °C entre 55 y 150 °C. Al ser un sensor analógico el cable que une el sensor y el Arduino puede influenciar en las lecturas.

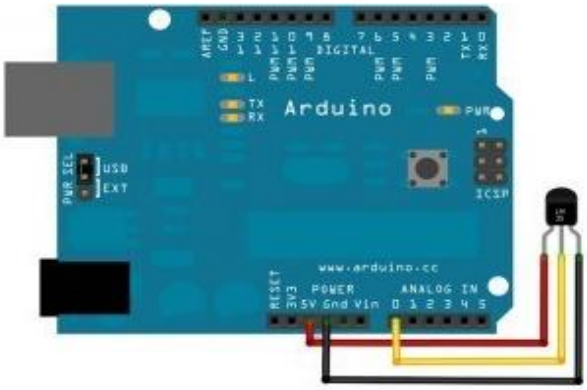


Figura 2.8- Conexión del sensor LM35 al microcontrolador Arduino

- DHT22: Éste sensor se caracteriza por proporcionar a la misma vez, la temperatura y humedad relativa. Usan un encapsulado de 4 pines, aunque solo usan 3, y emplean para su funcionamiento un pin digital. El hecho de estar encapsulado implica que el espesor será mayor al resto de sensores. Su rango de medición es de -40 a 125 °C con una tolerancia de 0,2 °C, y la resolución es de 0,1°C. Se puede encontrar por 4 €.

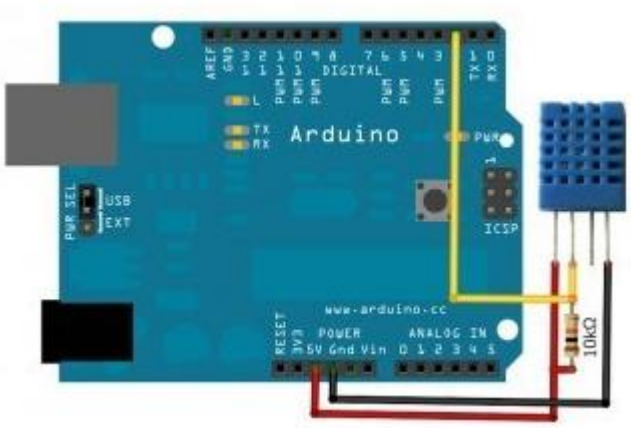


Figura 2.9- Conexión del sensor DHT22 al microcontrolador Arduino

- DS18B20: Éste usa un pin digital y su precio ronda los dos euros. Tiene un rango de medición de -55 a 125 °C con una tolerancia de 0,5 °C cuando se encuentra entre un rango de -10 a 85 °C, y con una resolución de 0,06 °C. A diferencia del resto, este nos permite conectar varios de ellos usando el mismo pin; mediante el protocolo de comunicación One Wire.

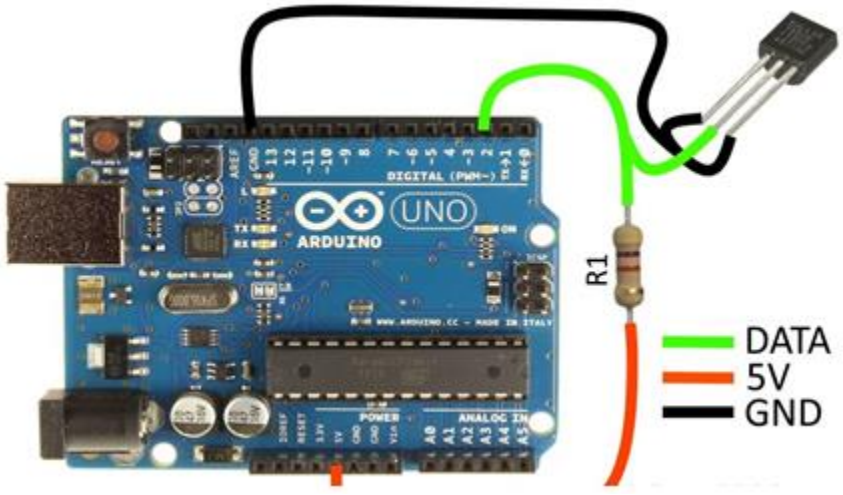


Figura 2.10- Conexión del sensor DS18B20 al microcontrolador Arduino

Tras analizar los diferentes sensores de temperatura, se ha tomado la decisión de utilizar el **DS18B20**, debido a que el sistema de medición requiere una gran precisión, es una buena alternativa económica, e incorpora el protocolo de comunicación *One Wire*.

Este protocolo de comunicación tiene sus ventajas e inconvenientes, pero no deja de ser una opción muy a tener en cuenta cuando vamos justos de conexiones, ya que nos ahorramos un cable en cada sensor.

☑ *Ventajas*

Con el protocolo *One Wire* conseguimos enviar y recibir datos por una sola línea de datos, es decir, empleamos únicamente un pin de nuestro Arduino. Es un bus donde disponemos de un maestro y varios esclavos en una sola línea de datos. Por supuesto, necesita igualmente una referencia a tierra común a todos los dispositivos. También debemos conocer que la línea de datos/alimentación debe de disponer de una resistencia de '*pull-up*' conectada a alimentación, para cuando no se produzca la transmisión de datos.

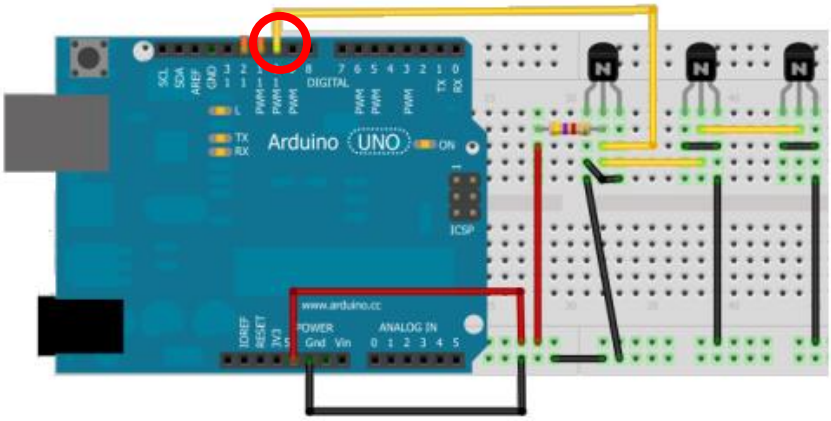


Figura 2.11- Conexión del sensor DS18B20 a Arduino mediante protocolo One Wire

☒ *Inconvenientes*

Para realizar la comunicación One Wire, necesitamos conocer la dirección de cada sensor para leer independientemente cada uno de ellos. Esto dificulta el código a implementar al realizar la monitorización de los diferentes sensores; es más sencillo referirnos a un determinado sensor en función del pin al que esté conectado a Arduino, que si lo hacemos utilizando la dirección de cada uno de ellos.

⊙ *Conclusión*

El hecho de solo emplear 10 sensores en nuestro proyecto, y el de disponer suficientes pines digitales en nuestro Arduino como para conectar cada sensor directamente con el microcontrolador, nos lleva a obviar la comunicación One Wire**.** En el siguiente capítulo se aborda con mayor detalle el conexionado de todos los componentes.

CAPÍTULO 3. Conexionado de los componentes.

En el presente capítulo se detalla el conexionado de los sensores para una correcta monitorización. Tras la aparición de errores en la lectura de temperaturas, derivados por fallos en la conexión, se propone la fabricación de una placa PCB.

3.1. Conexionado en la protoboard

Cada sensor requiere la conexión de una resistencia de '*pull-up*' para una correcta medición. Un ejemplo de conexión del sensor DS18B20 puede ser el mostrado en la siguiente imagen:

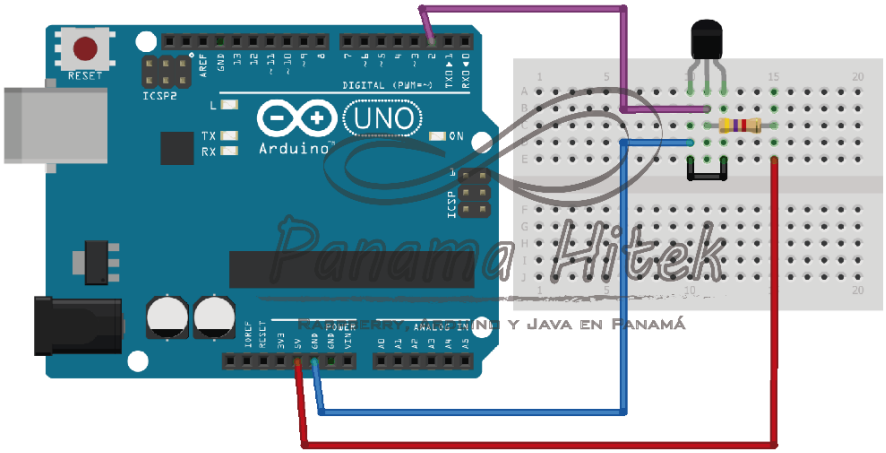


Figura 3.1- Ejemplo de conexión de un sensor digital DS18B20 a los pines de una tarjeta Arduino Uno

Tras repetir este montaje para los 10 sensores empleados en el proyecto, y asignando a cada uno un pin de entrada diferente (Ver Código 6.8), el resultado obtenido es el siguiente:

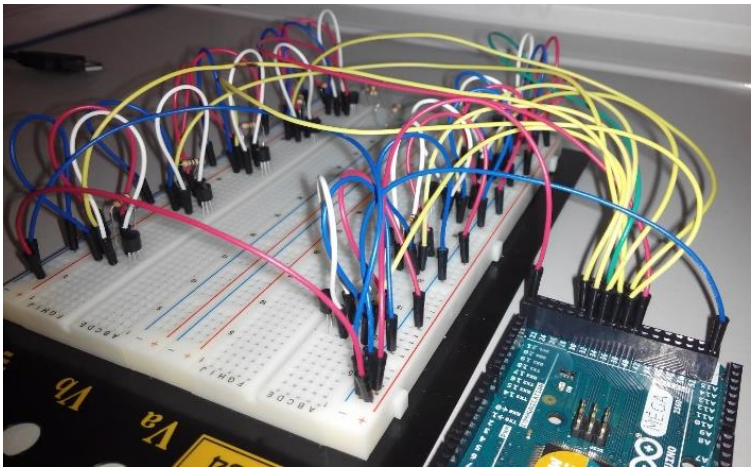


Figura 3.2- Conexionado de todos los sensores en la protoboard

Tal y como puede apreciarse en la anterior imagen, se requiere una gran cantidad de cableado, lo que conlleva a la aparición de numerosos fallos en el conexionado (cables y resistencias que se sueltan de su posición o no hacen un buen contacto en los orificios del tablero).

La finalidad de nuestro proyecto es la de construir un sistema de medición y monitorización de temperaturas, para que el alumnado pueda realizar numerosas pruebas con él. La eficacia será requisito básico en el sistema de medición empleado; no podemos permitirnos fallos en el conexionado de los diferentes componentes.

Se pensó en trasladar el montaje de la protoboard a un circuito impreso, placa PCB. De este modo, podríamos eliminar fallos debidos al conexionado, y conseguir un sistema de medición mucho más seguro y reducido, en comparación al realizado en la protoboard.

3.2. Diseño de PCB de Shield para Arduino Mega 2560

3.2.1. Especificaciones

Se requiere diseñar una placa PCB, con funcionalidad de "*shield*" para una tarjeta Arduino Mega 2560, con posibilidad de monitorización de hasta 12 sensores digitales de temperatura, del tipo DS18B20. Recordar que en nuestro proyecto solo se van a emplear 10 sensores de temperatura, el hecho de que a la larga pueda fallar el conexionado de algún sensor, nos ha llevado a incluir dos más en el diseño de la PCB.

Se indica como especificación adicional, que en la actualidad se están utilizando los pines 31 al 40 para los pin data de los sensores: 10 en total, necesitando en la nueva versión a desarrollar de PCB el uso de 12, que se acuerda asignar a los pines del 31 al 42 de la tarjeta Arduino Mega. Como ejemplo de asignación de pines, en el *Código 6.8* podemos comprobar como al sensor 1 (madera exterior) se le asigna el pin de entrada 31 para el pin data del mismo.

3.2.2. Procedimiento de diseño

Para el diseño se toma como referencia la distribución hardware de una tarjeta Arduino Mega 2560.

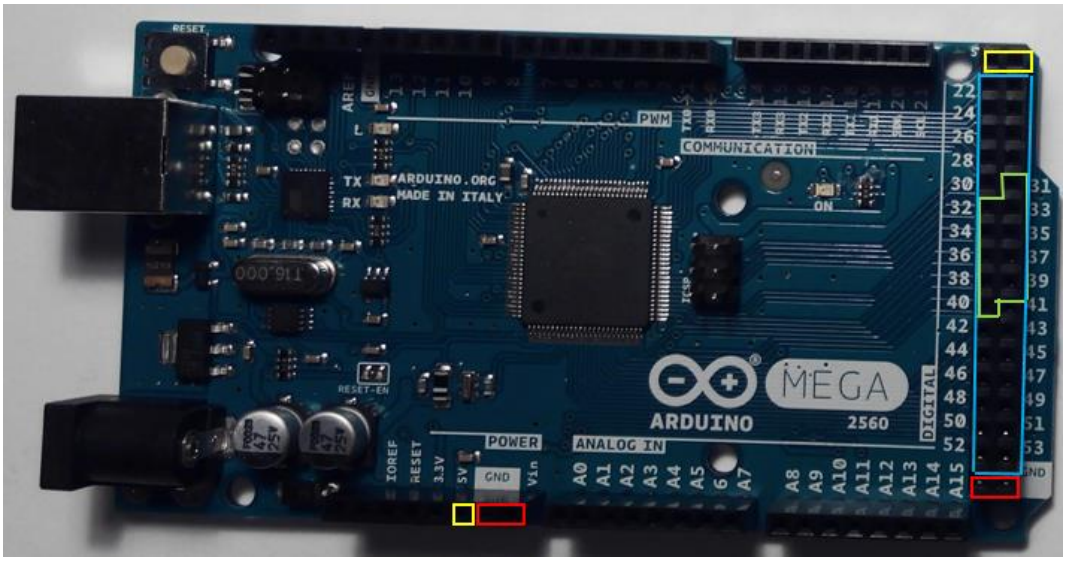


Figura 3.3- Distribución de zócalos de expansión en una tarjeta Arduino Mega.

Sobre la anterior imagen, cabe resaltar los siguientes pines de conexión:

- AMARILLO: pin de 5V. (se necesitará sólo 1).
- ROJO: pin GND. (se necesitará sólo 1).
- VERDE: pines digitales utilizados actualmente (10). La nueva versión deberá utilizar 12, asignados a los pines 31 al 42.
- AZUL: pines digitales que sería posible utilizar como entrada de sensores.

Aunque se podría obtener tanto alimentación, como conexiones a los pines digitales del zócalo dedicado a extensión 'DIGITAL' indicado en la Figura 3.3, se opta por utilizar el zócalo de extensión 'POWER' para dar una mayor estabilidad física al '*shield*' una vez montado. De este modo, se obtiene de este zócalo las líneas de alimentación (5V) y tierra (GND), asociadas a la alimentación; y dejando para las conexiones con los sensores el patillaje localizado en el zócalo 'DIGITAL'.

Con estas especificaciones, se procede a desarrollar el PCB resumido en la Figura 3.4 (a y b).

a)

b)

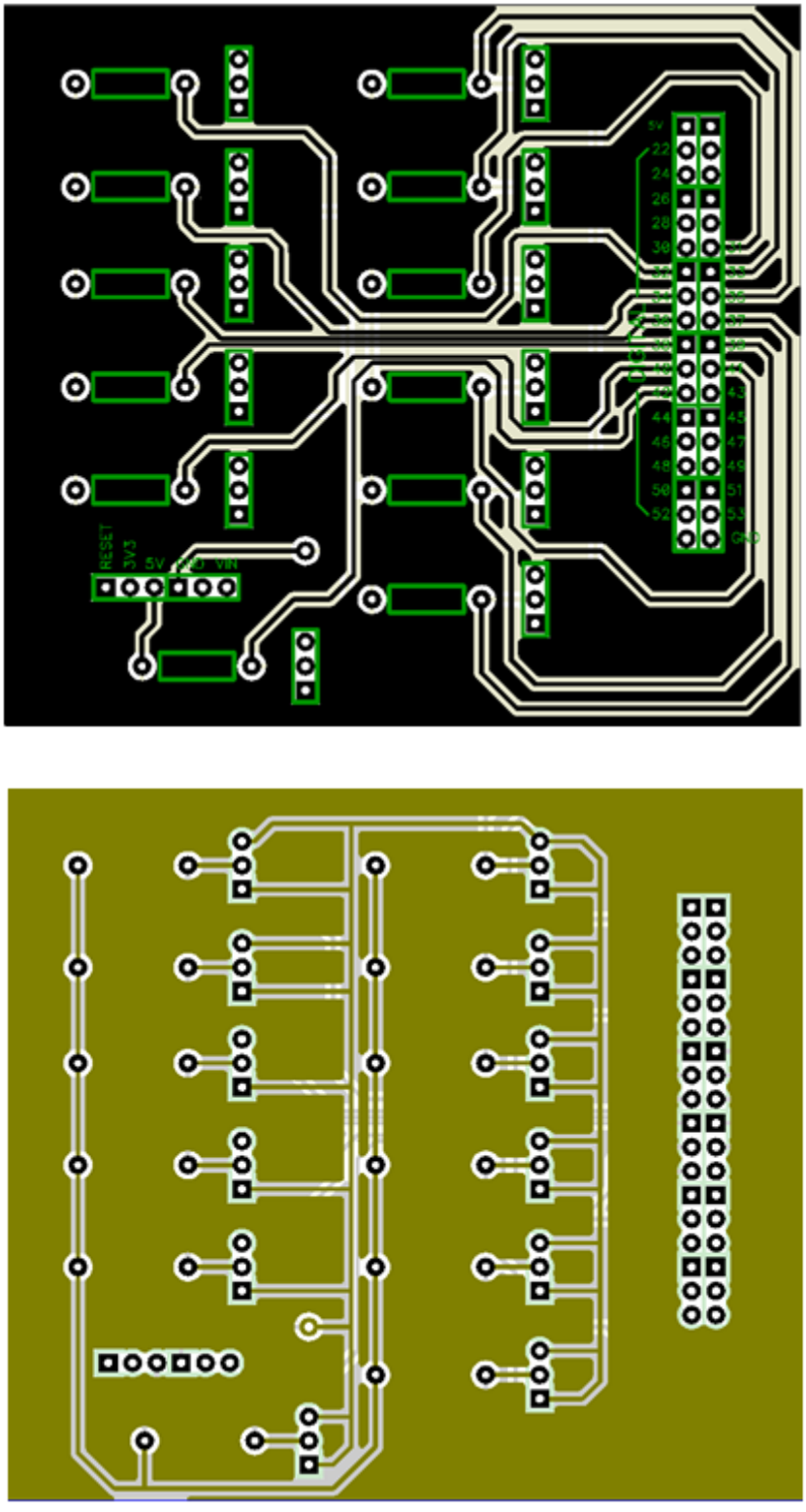


Figura 3.4- Diseño de PCB: a) Capa superior o Top. b) Capa inferior o Bottom

En la Figura 3.5, asociado a la cara de componentes, se puede apreciar la asignación de pines asociada a cada zócalo sensor. Se ha llevado a cabo un procedimiento de diseño en el que se ha optado por mantener los pines asignados para los primeros 10 solicitados y extendiendo hasta 12 el diseño final. Por lo tanto, el rango de pines asignado se extiende del 31 al 42, asignados tal y como se especifica en la cara de componentes reflejada en la siguiente imagen. El pin correspondiente, hace referencia al pin central del zócalo asociado al sensor, ya que los dos extremos por especificación de diseño van conectados a masa (GND).

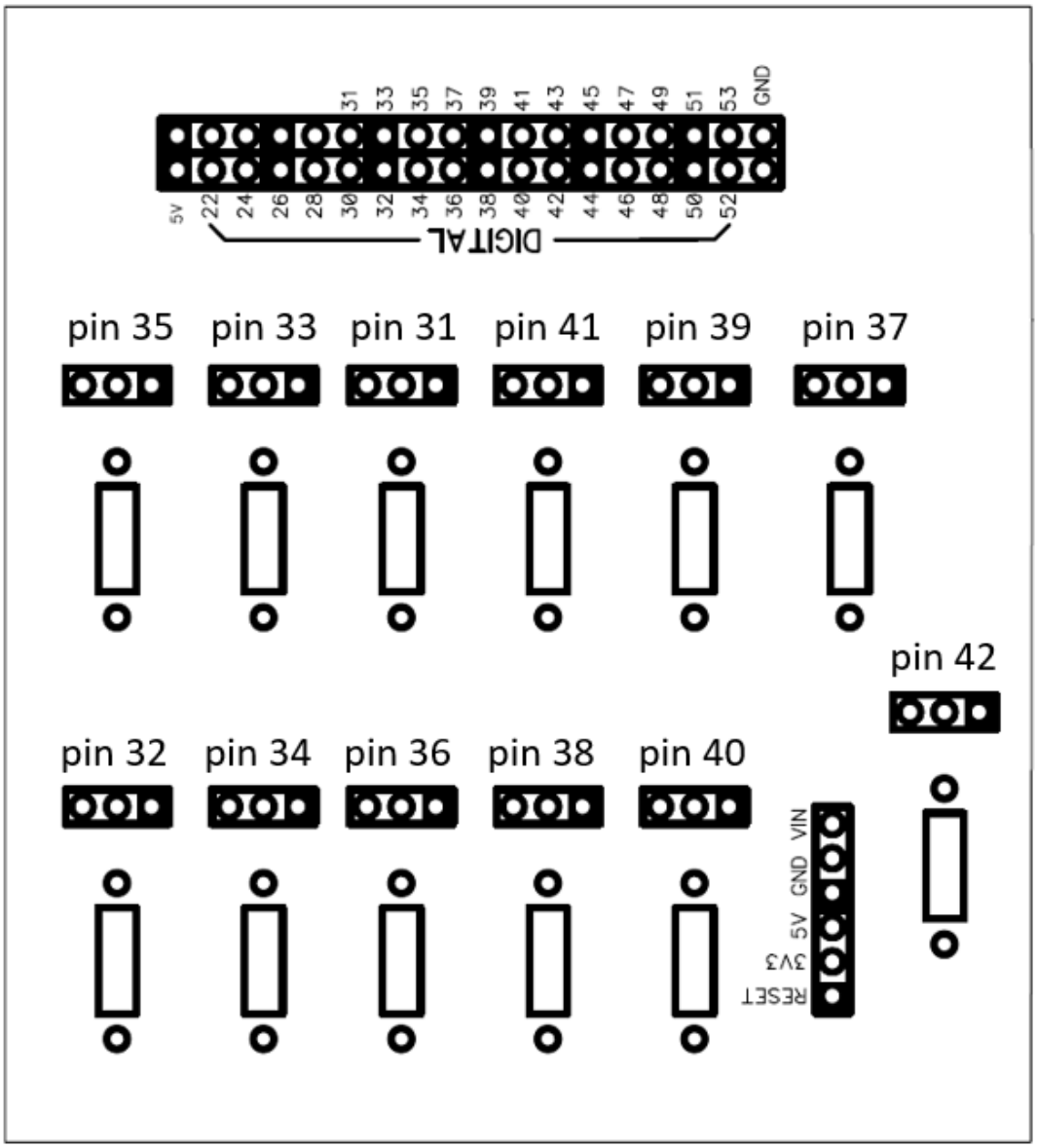


Figura 3.5- Cara de componentes con la asignación de cada zócalo sensor al pin de Arduino Mega correspondiente.

3.2.3. Resultados

Las siguientes imágenes describen el proceso de fabricación seguido, así como el resultado tras el montaje final.

a) b)

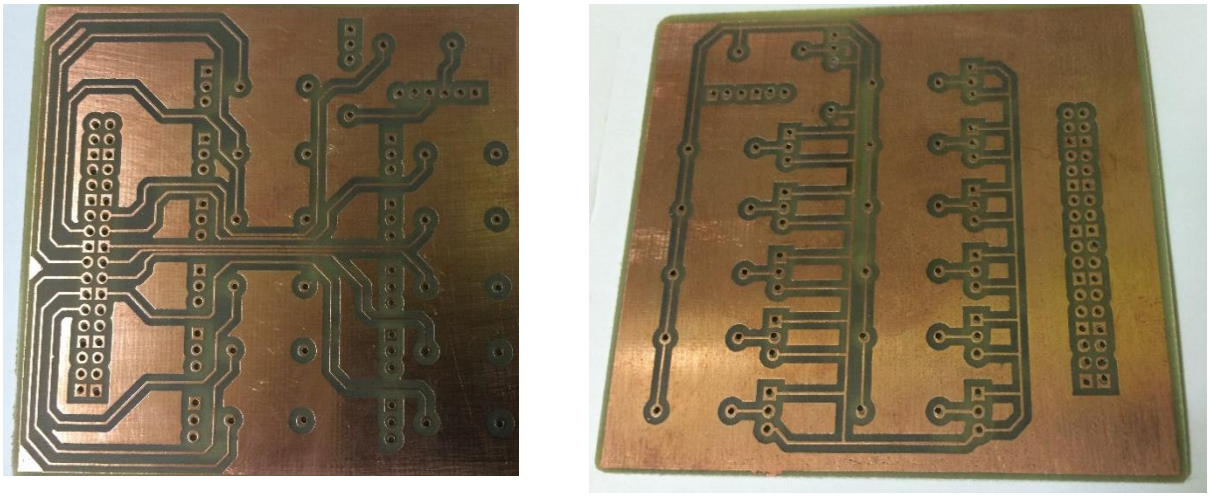


Figura 3.6- Fabricación del PCB: a) Capa superior o Top. b) Capa inferior o Bottom

a) b)

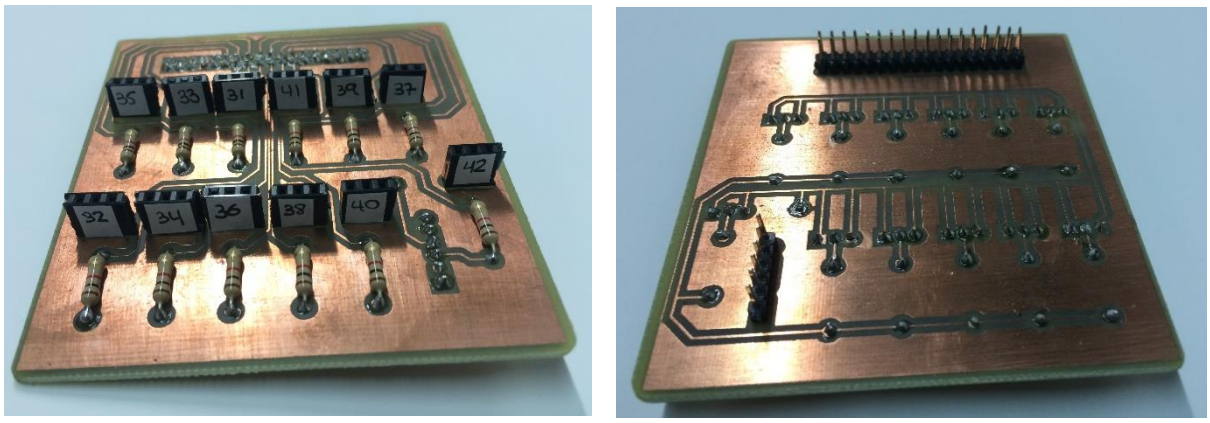


Figura 3.7- Versión final del PCB con los componentes soldados:

a) Capa superior o Top. b) Capa inferior o Bottom

CAPÍTULO 4. Elección de un software para la adquisición, procesado y presentación de la información recogida por los sensores de temperatura.

4.1. Introducción

Mediante la ayuda de un entorno de programación gráfica, conseguimos monitorear en tiempo real las temperaturas registradas por cada sensor. Dentro de los diferentes tipos de entornos de programación gráfica que existen, elegiremos los entornos esclavos (monitorización), ya que éstos nos permiten el intercambio de información con la tarjeta Arduino, ofreciendo un Instrumento Gráfico que facilita la interacción.

Para los sistemas DAQ (Data Acquisition System), se necesita de un software de instrumentación, que sea flexible para futuros cambios, y preferiblemente que sea de fácil manejo, siendo lo más poderoso e ilustrativo posible.

Programas y lenguajes de programación que cumplan con lo dicho existen en gran número en el mercado actual, como por ejemplo: Visual Basic, C++, Visual C++, Pascal, LabWindows CVI, LabVIEW, y muchos otros confeccionados específicamente para las aplicaciones que los necesiten.

Para realizar el monitoreo de toma de datos, se consideró que el lenguaje más apto era LabVIEW (Laboratory Virtual Engineering Workbench). Se trata de una herramienta diseñada especialmente para monitorizar, controlar, automatizar y realizar cálculos complejos de señales analógicas y digitales capturadas a través de tarjetas de adquisición de datos, puertos serie y GPIBs (Buses de Intercambio de Propósito General). Las razones de la elección son varias:

- Es muy simple de manejar, debido a que está basado en un nuevo sistema de programación gráfica, llamado lenguaje G.
- Es un programa enfocado hacia la instrumentación virtual, por lo que cuenta con numerosas herramientas de presentación, en gráficas, botones, indicadores y controles, los cuales son muy esquemáticos y de gran elegancia. Estos serían complicados de realizar en bases como C++, donde el tiempo para lograr el mismo efecto sería mayor.
- Es un programa de gran versatilidad donde se cuentan con librerías especializadas para manejos de DAQ, Redes, Comunicaciones, etc.
- Las horas de desarrollo de una aplicación para un ingeniero, se reducen a un nivel mínimo.
- Como se programa creando subrutinas en módulos de bloques, se pueden usar otros bloques creados anteriormente como aplicaciones por otras personas.

4.2. VIs (Instrumentos Virtuales) de LabVIEW

Los programas desarrollados mediante LabVIEW se denominan Instrumentos Virtuales (VIs), porque su apariencia y funcionamiento imitan los de un instrumento real. Sin embargo son análogos a las funciones creadas con los lenguajes de programación convencionales.

LabVIEW tiene la característica de descomposición modular ya que cualquier VI que se ha diseñado puede convertirse fácilmente en un módulo que puede ser usado como una subunidad dentro de otro VI. Esta peculiaridad podría compararse a la característica de procedimiento en los lenguajes de programación estructurada, lo que permite crear tareas muy complicadas a partir de módulos o submódulos mucho más sencillos.

Es un sistema abierto, cualquier fabricante de tarjetas de adquisición de datos o instrumentos en general puede proporcionar el driver de su producto en forma de VI dentro del entorno de LabVIEW. También es posible programar módulos para LabVIEW en lenguajes como C y C++, estos módulos son conocidos como Sub- VIs y no se difieren a los VI creados con LabVIEW salvo por la interfaz del lenguaje en el que han sido programados.

4.3. Entorno de trabajo de LabVIEW

Se podría decir que en cualquier VI de LabVIEW existen dos caras bien diferenciadas: el Front Panel (Panel Frontal) y el Block Diagram (Diagrama de Bloque). Las paletas de LabVIEW son las que proporcionan las herramientas que se requieren para crear y modificar tanto el panel frontal, como el diagrama de bloques.

4.3.1. Panel Frontal

Es la cara que el usuario del sistema está viendo cuando se está monitorizando o controlando el sistema, es decir, la interfaz del usuario. Puede ser totalmente parecido al instrumento del cual se están recogiendo los datos, de esta manera el usuario sabe de manera precisa cual es el estado actual de dicho instrumento y los valores de las señales que se están midiendo (Ver Figura 4.1).

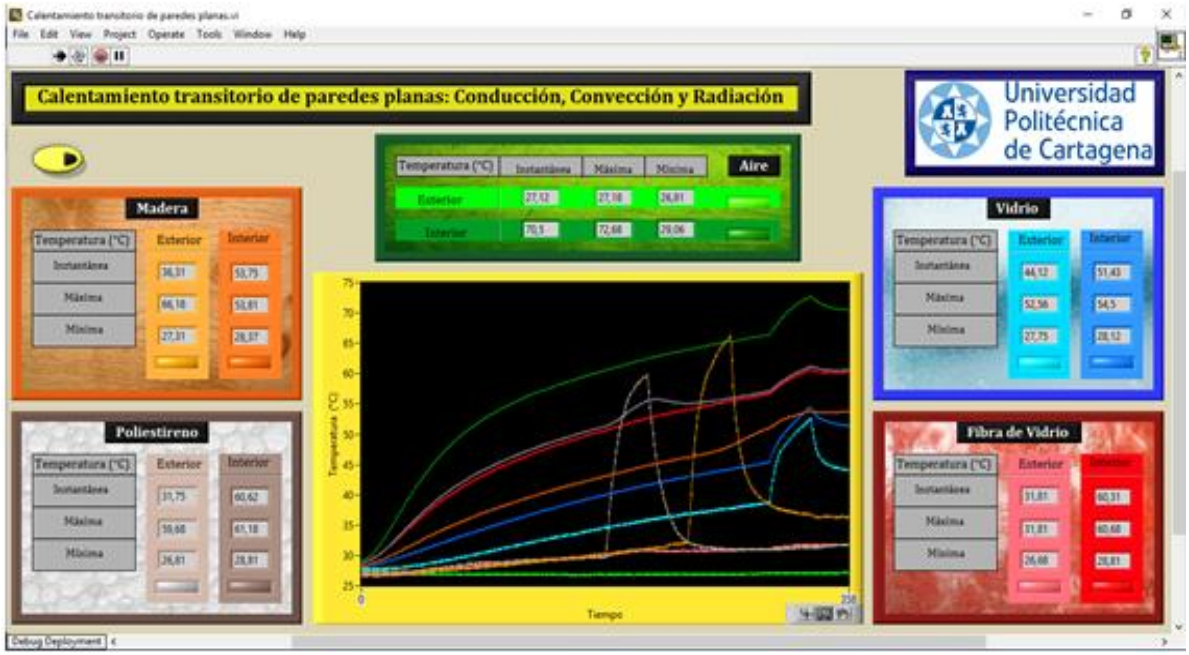


Figura 4.1- Panel frontal de la aplicación

4.3.2. Diagrama de Bloques

Es el programa propiamente dicho; el código fuente gráfico que define el funcionamiento del VI. En el diagrama de bloques es donde se realiza la implementación del programa del VI para controlar o realizar cualquier procesado de las entradas y salidas que se crearon en el panel frontal (Ver Figura 4.2).

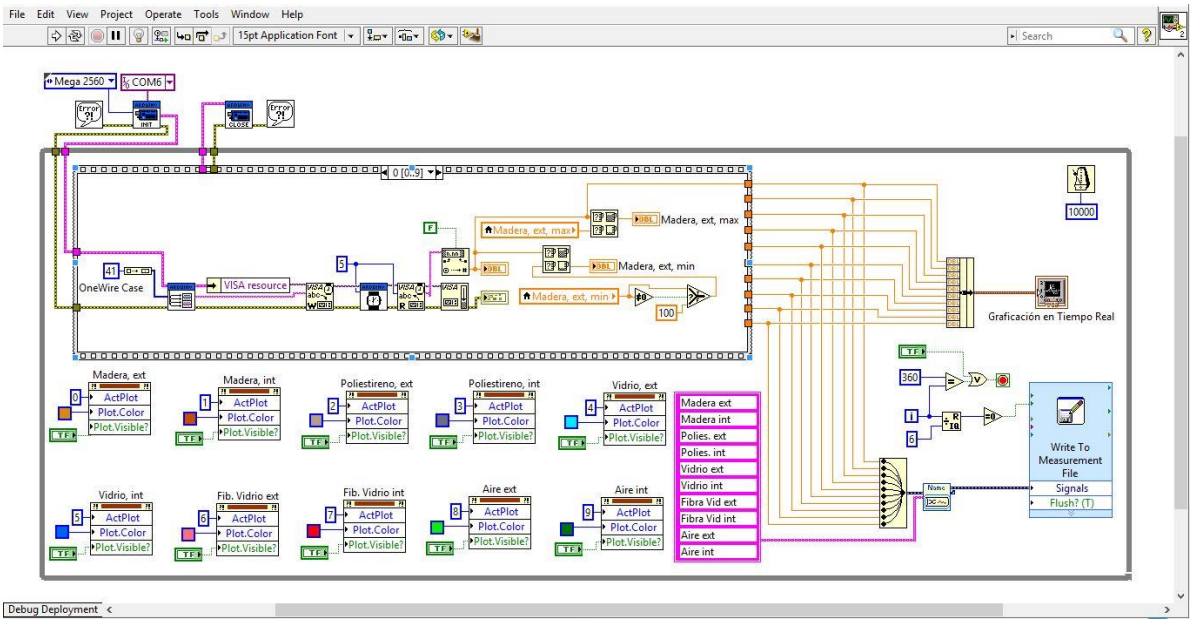


Figura 4.2- Diagrama de bloques de la aplicación

4.3.3. Paletas

Proporcionan las herramientas que se requieren para crear y modificar tanto el panel frontal como el diagrama de bloques.

- La *paleta de controles* para el panel frontal, nos permite, pinchando y arrastrando con el ratón del ordenador, insertar los controles necesarios para nuestra aplicación.

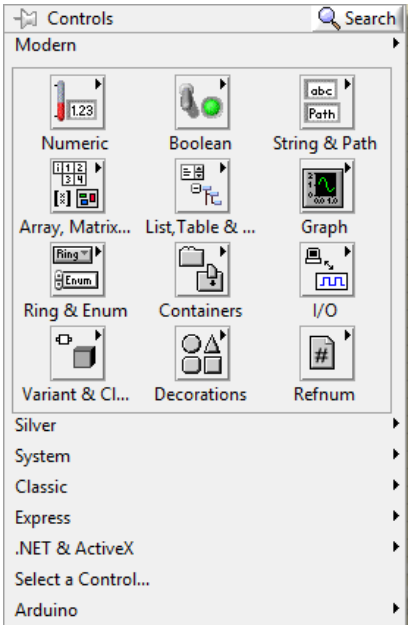


Figura 4.3- Paleta de controles para el panel frontal

- La *paleta de herramientas*, sirve tanto para el panel frontal como para el diagrama de bloques, y nos permite diseñar/editar el panel frontal, mover VIs, la inserción de textos, la edición de colores, etc.

Figura 4.4- Paleta de herramientas

- La *paleta de funciones* para la creación del diagrama de bloques nos permite elegir el tipo de función que queremos instalar en nuestro diagrama de bloques: entradas de DAQ, operaciones matemáticas, estructuras, etc. A partir de esta paleta se crea el diagrama de bloques.

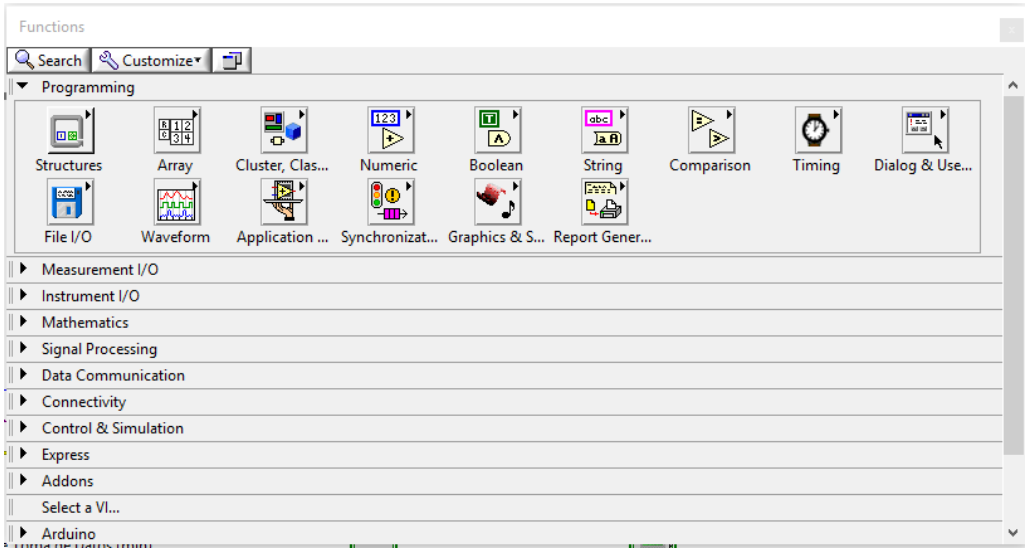


Figura 4.5- Paleta de funciones para la creación del diagrama de bloques

CAPÍTULO 5. Arduino como DAQ en LabVIEW.

5.1. Introducción

Gracias a NI es posible convertir una placa de Arduino en una tarjeta de adquisición de datos y poder manejarla por USB desde LabVIEW. Esto le confiere una gran potencialidad, dado que se trata de una herramienta de gran capacidad y muy extendida tanto en el mundo académico, como en el industrial y de laboratorio.

La conexión que actualmente ofrece LabVIEW es en modo esclavo; solo es posible visualizar y realizar control de las E/S de Arduino en dicho modo.

Esta forma de trabajo, por tratarse de LabVIEW, es interesante para usarla en el prototipado de aplicaciones de instrumentación, en las que la tarjeta Arduino juega el papel de un sencillo y versátil equipo de adquisición de datos a un costo muy bajo.

5.2. Instalación de la interfaz de LabVIEW con Arduino

La configuración de la interfaz de LabVIEW para Arduino es un proceso de seis pasos que *sólo tendrá que realizarse una sola vez*. A continuación se detallarán los pasos a seguir para realizar la conexión entre el software LabVIEW y el hardware Arduino. La instalación en su conjunto se puede subdividir en dos partes:

5.2.1. Parte de instalación de LabVIEW

1. Instalamos LabVIEW 2015.

   La UPCT cuenta con una licencia de LabVIEW que podemos utilizar en el laboratorio para su uso en las prácticas.

2. Instalamos los controladores VISA.NI.

   Éstos pueden ser descargados a partir de la web de National Instrument.

3. Instalamos el gestor de paquetes JKI VI Package Manager (VIPM).

   Este programa será el que descargue e instale los VI de Arduino en LabVIEW. Podemos descargar la versión de la comunidad (gratuitamente) de la siguiente página: http://jki.net/vipm/download

4. Ejecutamos el gestor VIPM y buscamos Arduino.

   Solo tenemos que seleccionar el paquete de la lista e instalarlo.

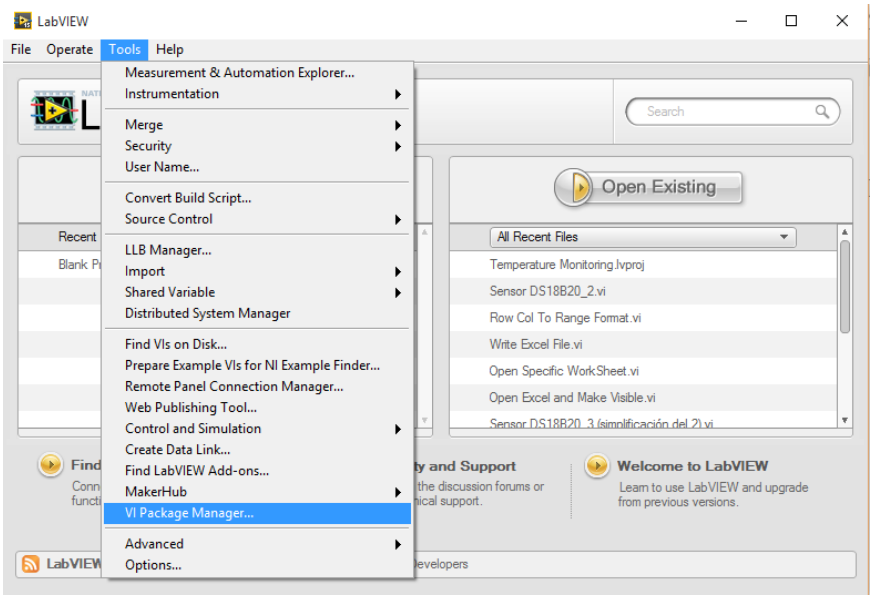


Figura 5.1- Ejecución del gestor VI Package Manager (VIPM)

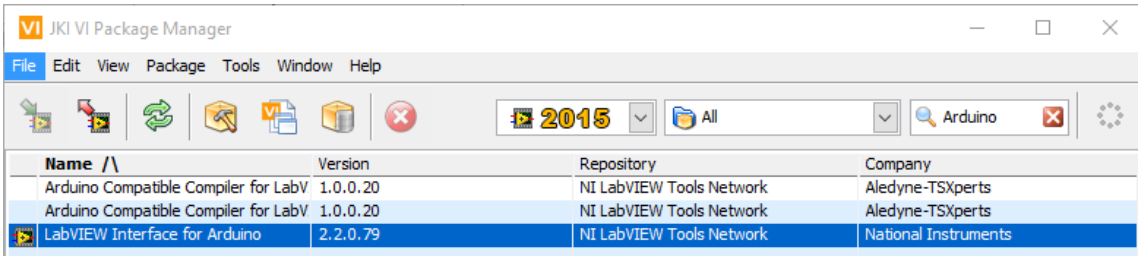


Figura 5.2- Búsqueda de Arduino en VIPM

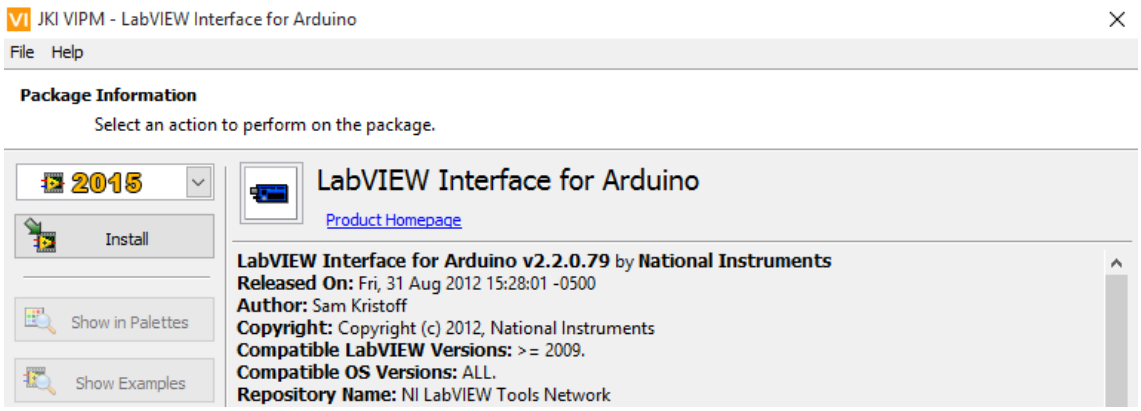


Figura 5.3- Instalación del paquete "LabVIEW Interface for Arduino"

Cuando finalice, habremos completado la parte de instalación de LabVIEW, nos queda la parte de Arduino.

5.2.2. Parte de instalación de Arduino

5. Instalamos el software de Arduino

El IDE de Arduino lo descargamos directamente de la página de Arduino, instalando así la última versión del mismo. Hemos utilizado la versión 1.6.7; éste se puede descargar aquí: https://www.arduino.cc/en/Main/Software

6. Conexión de la placa Arduino

El primer paso será conectar la placa Arduino al ordenador por medio del puerto USB. Una vez realiza dicha conexión, debemos instalar los drivers. A continuación se muestran los pasos a seguir:

- Abrir el administrador de dispositivos de Windows. Para ello hacemos: Equipo/Propiedades/Administrador de dispositivos:

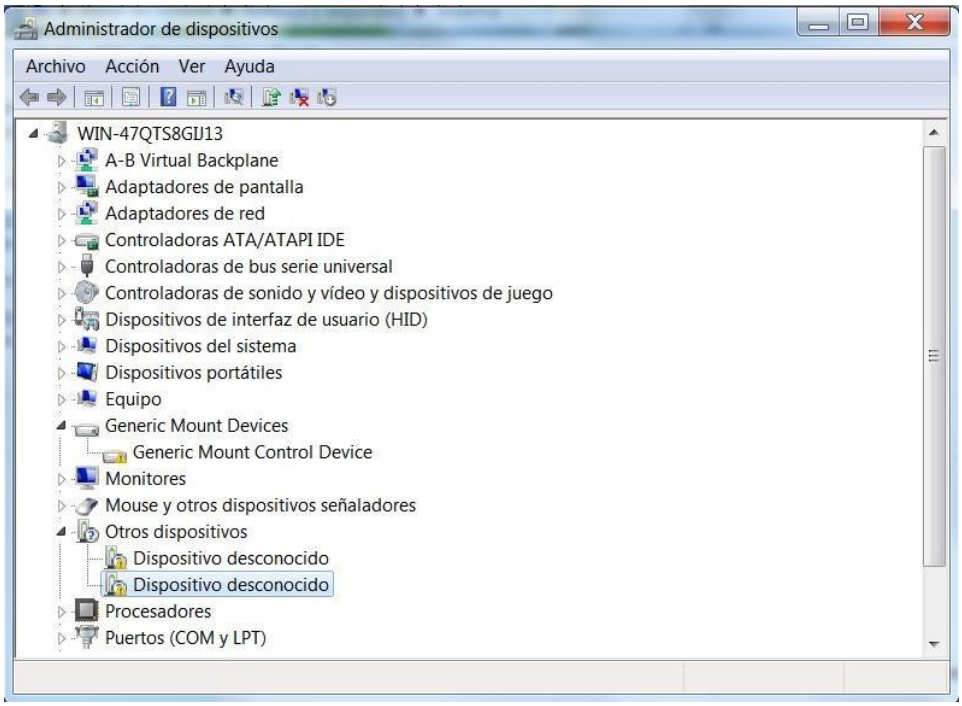


Figura 5.4- Administrador de dispositivos

- Como se puede observar en la Figura 5.4, hay dispositivos desconocidos. Abrimos el último de los dispositivos desconocidos.

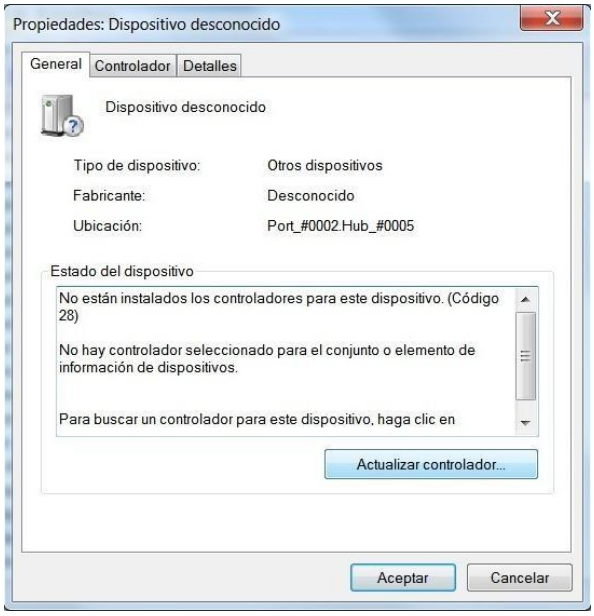


Figura 5.5- Propiedades Dispositivo desconocido

- Seleccionamos el botón actualizar controlador y a continuación seleccionamos la opción buscar software de controlador de equipo.

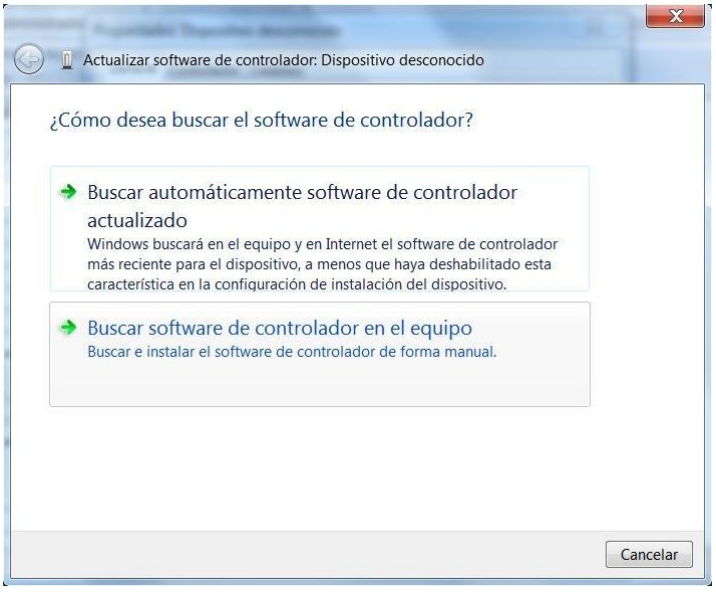


Figura 5.6- Actualizar software de controlador

- Indicamos en que carpeta están los drivers de Arduino. Tendremos que seleccionar la carpeta "drivers" que hay dentro del directorio de Arduino, y marcamos la opción “incluir subcarpetas”.

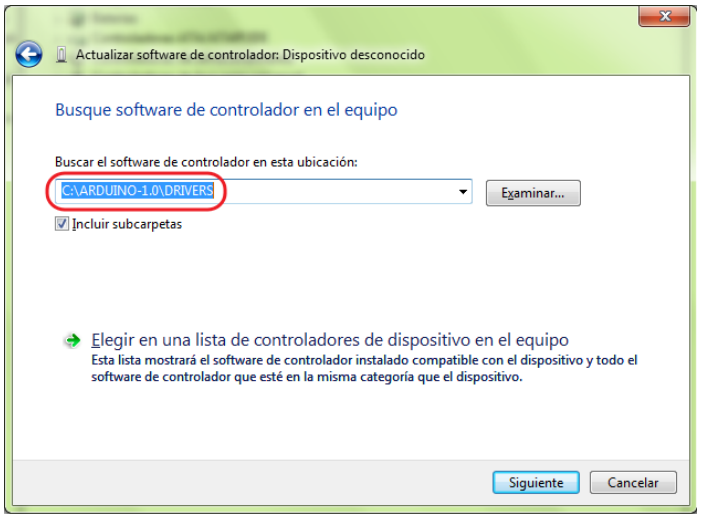


Figura 5.7- Ubicación Drivers Arduino

- Una vez instalados los drivers podemos observar que ya no aparece como dispositivo desconocido, sino que se le ha asignado un puerto COM.

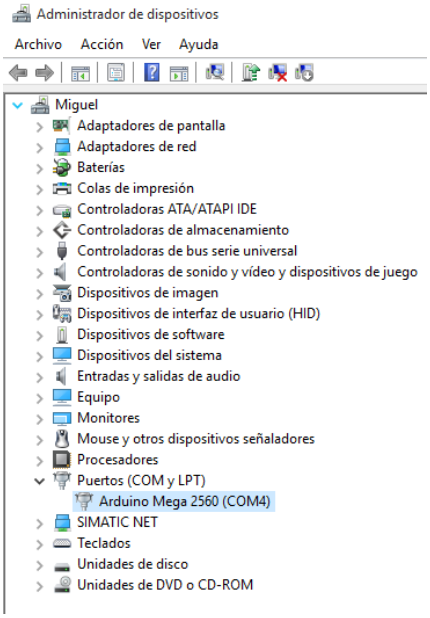


Figura 5.8- Administrador de dispositivos Arduino Mega 2560 (COM 4)

- Seleccionamos el tipo de placa que estamos utilizando, en nuestro caso es un Arduino Mega 2560. Se selecciona desde el menú *herramientas*, en la opción *placa*.

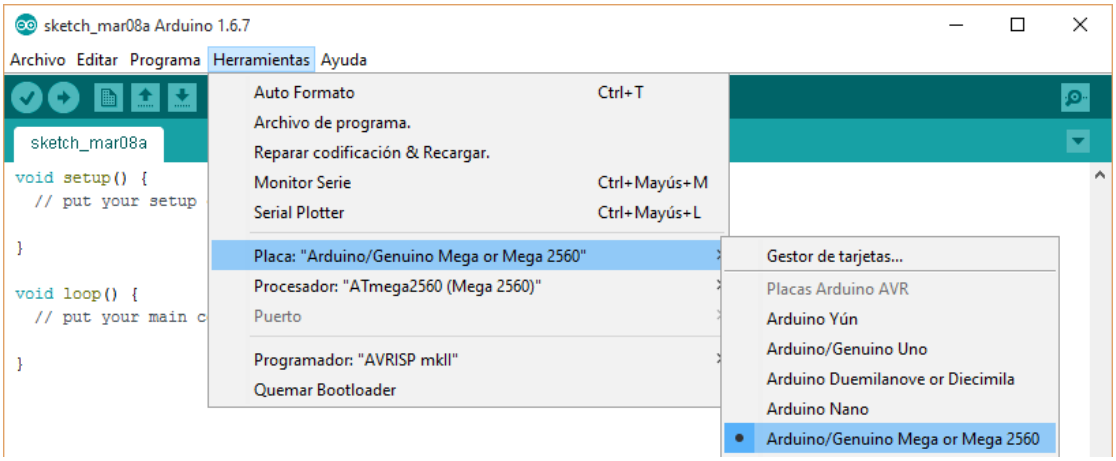


Figura 5.9- Selección del tipo de placa utilizada

- Elegimos el puerto serie (COM) que utiliza el sistema para comunicarse con la placa de Arduino. En mi caso es el COM4, pero el sistema nos puede haber asignado cualquier otro. Lo puedes averiguar en el Administrador de dispositivos. El puerto se selecciona en la opción "Serial Port" del menú *herramientas*.

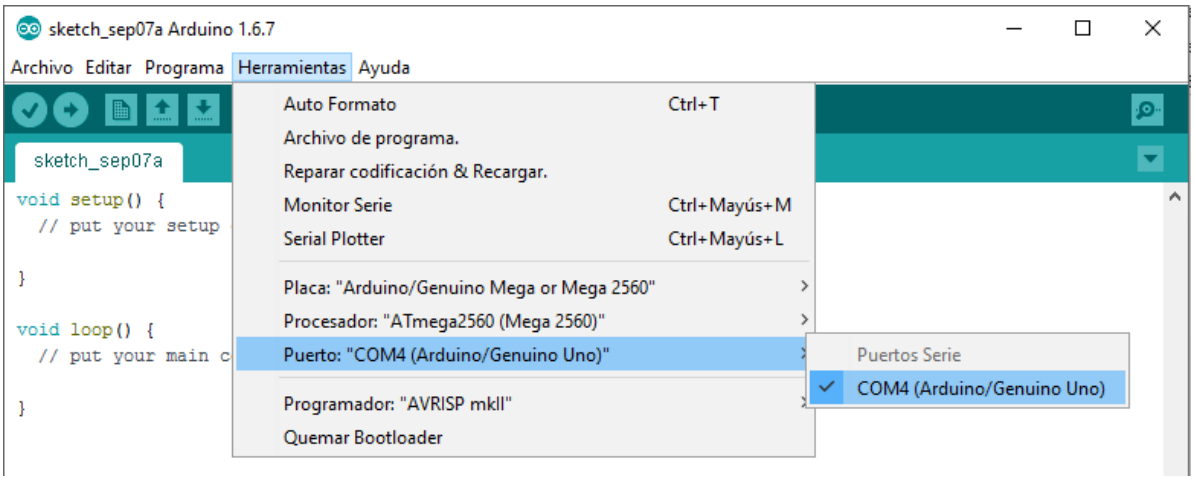


Figura 5.10- Selección del puerto serie (COM) al que está conectado nuestro Arduino

Con esto ya está todo instalado y la placa de Arduino ha sido reconocida por el sistema. Solo tenemos que programar Arduino con el sketch que comunica la placa con LabVIEW.

CAPÍTULO 6. Desarrollo del Software de Arduino

El programa que se utiliza para programar en la placa Arduino, es Arduino 1.6.7; un entorno de programación de código abierto, que hace sencilla la escritura y carga del código en la placa. Funciona en Windows, Mac OS X y Linux. El entorno está escrito en Java y basado en Processing, avr-gcc y otros programas también de código abierto.

El lenguaje utilizado es Arduino, que está basado en C/C++ y soporta todas las construcciones de C estándar y algunas funcionalidades de C++. Vincula la librería AVR Libc y permite el uso de todas sus funciones.

Programación del sistema

En este punto se expone la programación del microcontrolador Arduino. Ésta cuenta con diferentes pasos:

1. Se preparara el código para su comunicación con LabVIEW. En LabVIEW vamos a utilizar el toolkit LIFA (LabVIEW Interface for Arduino), el cual incluye el código a utilizar para dicha comunicación.

2. Diseñamos el código para nuestro sensor.

6.1 Código LIFA para LabVIEW

Ya tenemos Arduino conectado al USB y reconocido por el sistema, ahora tenemos que programar el microcontrolador con el sketch que comunica la placa con LabVIEW. El sketch de Arduino se instala con LIFA, por lo que tenemos que ir a la carpeta donde se ha instalado; la ruta es un poco larga (Ver Figura 6.1). En el directorio LIFA_Base se encuentra el archivo LIFA_Base.pde, que es el sketch que debemos abrir. Se nos abrirán además todas las librerías.

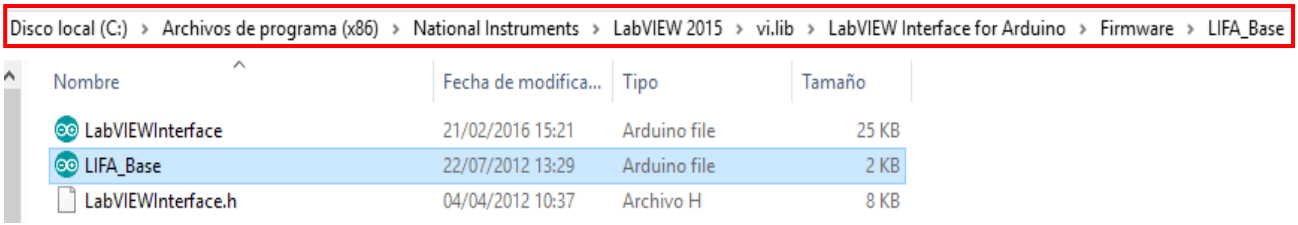


Figura 6.1- Ruta a seguir para acceder al archivo LIFA_Base

En este punto se expone el código LIFA_Base para Arduino. Este código es de libre distribución y modificación, y fue creado originalmente por Sam Kristoff en noviembre de 2010.

El código consta de 3 pestañas.

- LIFA_Base : Esta pestaña proporciona un Sketch básico para interactuar con LabVIEW.
- LabVIEWInterface.h : Esta pestaña es una librería con las funciones necesarias para interactuar con LabVIEW.
- LabVIEWInterface : Esta pestaña desarrolla las funciones para interactuar con LabVIEW.

A continuación detallamos la función que tiene cada una de ellas:

1. La pestaña **LIFA_Base** (Ver Código 6.1), es una pestaña preparada para introducir el programa que interactúe con LabVIEW.

```
/*************************************************************************
 ** Includes.
*************************************************************************/

// Standard includes.  These should always be included.
#include <Wire.h>
#include <SPI.h>
#include <Servo.h>
#include "LabVIEWInterface.h"

/*************************************************************************
 **  setup()
 **
 **  Initialize the Arduino and setup serial communication.
 **
 **  Input:  None
 **  Output: None
 *************************************************************************/
void setup()
{
  // Initialize Serial Port With The Default Baud Rate
  syncLV();

  // Place your custom setup code here

}
/*************************************************************************
 **  loop()
 **
 **  The main loop.  This loop runs continuously on the Arduino.  It
 **  receives and processes serial commands from LabVIEW.
 **
 **  Input:  None
 **  Output: None
 *************************************************************************/
```

```
void loop()
{
  // Check for commands from LabVIEW and process them.

  checkForCommand();
  // Place your custom loop code here (this may slow down communication
with LabVIEW)


  if(acqMode==1)
  {
    sampleContinously();
  }

}
```

Código 6.1- Pestaña LIFA_Base Arduino

Se divide en 3 bloques. Un primer bloque donde se definen los *includes*, las variables globales y en caso de hacer falta, algún programa corto para realizar alguna operación.

Un segundo bloque en el que encontramos el *setup()*. Este bloque se utiliza para inicializar el programa, y definir los pines digitales.

Y por último el *void loop()*, es propiamente donde se colocara el programa, debido a que es un bucle, y el programa se repetirá constantemente.

2. La pestaña **LabVIEWInterface.h**, es una librería donde se definen las variables globales y a utilizar en esta pestaña (Ver Código 6.2), y las funciones utilizadas en la pestaña LabVIEWInterface (Ver Código 6.3).

```
/*****************************************************************
**  Define Constants

**  Define directives providing meaningful names for constant values.
*****************************************************************/

#define FIRMWARE_MAJOR 02
#define FIRMWARE_MINOR 00
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define DEFAULTBAUDRATE 9600    // Defines The Default Serial Baud Rate
(This must match the baud rate specifid in LabVIEW)
#else
#define DEFAULTBAUDRATE 115200
#endif
#define MODE_DEFAULT 0            // Defines Arduino Modes (Currently Not
Used)
#define COMMANDLENGTH 15          // Defines The Number Of Bytes In A
Single LabVIEW Command (This must match the packet size specifid in
LabVIEW)
#define STEPPER_SUPPORT 1         // Defines Whether The Stepper Library
Is Included - Comment This Line To Exclude Stepper Support
```

```
// Declare Variables
unsigned char currentCommand[COMMANDLENGTH];     // The Current Command For
The Arduino To Process
//Globals for continuous aquisition
unsigned char acqMode;
unsigned char contAcqPin;
float contAcqSpeed;
float acquisitionPeriod;
float iterationsFlt;
int iterations;
float delayTime;
```

Código 6.2- LabVIEWInterface.h: Definición de Variables

```
/*****************************************************************
**  syncLV
**
**  Synchronizes with LabVIEW and sends info about the board and firmware
(Unimplemented)
**
**  Input:  None
**  Output: None
*****************************************************************/
void syncLV();

/*****************************************************************
**  setMode
**
**  Sets the mode of the Arduino (Reserved For Future Use)
**
**  Input:  Int - Mode
**  Output: None
*****************************************************************/
void setMode(int mode);

/*****************************************************************
**  checkForCommand
**
**  Checks for new commands from LabVIEW and processes them if any exists.
**
**  Input:  None
**  Output: 1 - Command received and processed
**          0 - No new command
*****************************************************************/
int checkForCommand(void);
```

Código 6.3- LabVIEWInterface.h: Ejemplo definición de funciones

Esta pestaña no se va modificar en este proyecto. Se modificaría para añadir funciones a utilizar en la tercera pestaña (LabVIEWInterface).

3. La pestaña **LabVIEWInterface**, donde se van a desarrollar las funciones definidas en la librería LabVIEWInterface.h. La pestaña se divide principalmente en dos partes:

   3.1. Una primera en la cual se definen las librerías y variables a utilizar en la pestaña.

```
// Librerias

#include <Wire.h>
#include <SPI.h>
#include <LiquidCrystal.h>
#include "OneWire.h"

// Variables

unsigned int retVal;
int sevenSegmentPins[8];
int currentMode;
unsigned int freq;
unsigned long duration;
int i2cReadTimeouts = 0;
char spiBytesToSend = 0;
char spiBytesSent = 0;
char spiCSPin = 0;
char spiWordSize = 0;
Servo *servos;
byte customChar[8];
LiquidCrystal lcd(0,0,0,0,0,0,0);
unsigned long IRdata;
```

Código 6.4- LabVIEWInterface: Definición de librerías y variables

   3.2. Y una segunda, donde se programan las funciones definidas en la librería LabVIEWInterface.h

```
/*****************************************************************
**  Functions
*****************************************************************/

// Writes Values To Digital Port (DIO 0-13).  Pins Must Be Configured As
Outputs Before Being Written To
void writeDigitalPort(unsigned char command[])
{
  digitalWrite(13, (( command[2] >> 5) & 0x01) );
  digitalWrite(12, (( command[2] >> 4) & 0x01) );
  digitalWrite(11, (( command[2] >> 3) & 0x01) );
  digitalWrite(10, (( command[2] >> 2) & 0x01) );
  digitalWrite(9, (( command[2] >> 1) & 0x01) );
  digitalWrite(8, (command[2] & 0x01) );
  digitalWrite(7, (( command[3] >> 7) & 0x01) );
  digitalWrite(6, (( command[3] >> 6) & 0x01) );
  digitalWrite(5, (( command[3] >> 5) & 0x01) );
  digitalWrite(4, (( command[3] >> 4) & 0x01) );
  digitalWrite(3, (( command[3] >> 3) & 0x01) );
```

```
  digitalWrite(2, (( command[3] >> 2) & 0x01) );
  digitalWrite(1, (( command[3] >> 1) & 0x01) );
  digitalWrite(0, (command[3] & 0x01) );
```

Código 6.5- LabVIEWInterface: Ejemplo programación de funciones

Para modificar esta pestaña hay dos opciones. Definir una nueva función en la librería LabVIEWInterface.h y programarla en la pestaña LabVIEWInterface, o añadir un *case* nuevo dentro de la función *processCommand* (Ver Código 6.6).

```
// Processes a given command
void processCommand(unsigned char command[])
{
  // Determine Command
  if(command[0] == 0xFF && checksum_Test(command) == 0)
  {
    switch(command[1])
    {
    /*****************************************************************
    LIFA Maintenance Commands
    *****************************************************************/
    case 0x00:      // Sync Packet
      Serial.print("sync");
      Serial.flush();
      break;
    case 0x01:     // Flush Serial Buffer
      Serial.flush();
      break;
```

Código 6.6- LabVIEWInterface: Función proccesCommand ejemplo case.

6.2 Programación de los sensores de temperatura DS18B20

En este punto se muestra el código a implementar para todos los sensores de temperatura.

Para ello, añadimos dentro de la función *processCommand,* tantos *case* nuevos como sensores de temperatura se vayan a emplear. Como en nuestro proyecto son necesarios 10 sensores, añadimos 10 *case*.

```
//SENSOR 1 (madera exterior)
       case 0x41:  // OneWire Read1
        OneWire_Read1();
         break;

//SENSOR 2 (madera interior)
        case 0x42:  // OneWire Read2
        OneWire_Read2();
         break;

//SENSOR 3 (poliestireno exterior)
       case 0x43:  // OneWire Read3
        OneWire_Read3();
         break;

//SENSOR 4 (poliestireno interior)
        case 0x44:  // OneWire Read4
        OneWire_Read4();
         break;

//SENSOR 5 (vidrio exterior)
       case 0x45:  // OneWire Read5
        OneWire_Read5();
         break;

//SENSOR 6 (vidrio interior)
        case 0x46:  // OneWire Read6
        OneWire_Read6();
         break;

//SENSOR 7 (fibra de vidrio exterior)
       case 0x47:  // OneWire Read7
        OneWire_Read7();
         break;

//SENSOR 8 (fibra de vidrio interior)
        case 0x48:  // OneWire Read8
        OneWire_Read8();
         break;

//SENSOR 9 (aire exterior)
       case 0x49:  // OneWire Read9
        OneWire_Read9();
         break;

//SENSOR 10 (aire interior)
        case 0x4A:  // OneWire Read10
        OneWire_Read10();
         break;
```

Código 6.7- LabVIEWInterface: Función proccesCommand para nuestros 10 sensores.

Seguidamente programamos las funciones definidas con anterioridad. Se mostrará el código empleado para un sensor, ya que éste será idéntico al de los demás sensores de temperatura, con la salvedad de modificar el pin de entrada (número subrayado en amarillo en el siguiente Código).

```
/********************************************************************
**  Functions
********************************************************************/

//SENSOR 1

void OneWire_Read1()
{
  OneWire ds(31);            // Create a OneWire Object "ds" on pin 31.
  byte OneWireData[9];
   int Fract, Whole, Tc_100, SignBit, TReading;

    // Start the Conversion
    ds.reset();       // Reset the OneWire bus in preparation for communication
    ds.skip();        // Skip addressing, since there is only one sensor
    ds.write(0x44);   // Send 44, the conversion command


  // Wait for the Conversion
    delay(850);      // Wait for the conversion to complete


  // Read back the data
    ds.reset();         // Reset the OneWire bus in preparation for communication
    ds.skip();        // Skip addressing, since there is only one sensor
    ds.write(0xBE);   // Send the "Read Scratchpad" command


    for ( byte i = 0; i < 9; i++) {
      OneWireData[i] = ds.read();     // Read the 9 bytes into data[]
    }

  // Scale the data
    TReading = (OneWireData[1] << 8) + OneWireData[0];
    SignBit = TReading & 0x8000;    // Mask out all but the MSB
    if (SignBit)                      // If the MSB is negative, take the Two's
Compliment to make the reading negative
    {
      TReading = (TReading ^ 0xffff) + 1;       // 2's comp
    }
    Tc_100 = (6 * TReading) + TReading / 4;     // Scale by the sensitivity
(0.0625°C per bit) and 100

    Whole = Tc_100 / 100; // Split out the whole number portion of the reading
    Fract = Tc_100 % 100;  // Split out the fractional portion of the reading


  // Return the data serially
    if (SignBit) {        // If the reading is negative, print a negative sign
      Serial.print("-");
    }
    Serial.print(Whole);  // Print the whole number portion and a decimal
    Serial.print(".");
    if (Fract < 10) {     // if the fraction portion is less than .1, append a 0
decimal
      Serial.print("0");
    }
    Serial.print(Fract);  // Otherwise print the fractional portion
}
```

Código 6.8- LabVIEWInterface: Programación de un sensor de temperatura DS18B20.

Visto esto, se da por terminada la programación de Arduino. En caso de ser necesario incluir nuevos sensores ya sea de temperatura, humedad, luminosidad, etc, únicamente tendremos que adaptar el código individual del componente al código global, de la manera que se ha explicado a lo largo de este apartado.

CAPÍTULO 7. Desarrollo del Software de LabView

7.1. Panel frontal

La interfaz está organizada de modo que la comunicación con el usuario sea muy simple e intuitiva.

Consta de una gráfica, alrededor de la cual, se muestran las temperaturas de los distintos materiales que conforman las paredes laterales del cubo, así como la temperatura del aire.

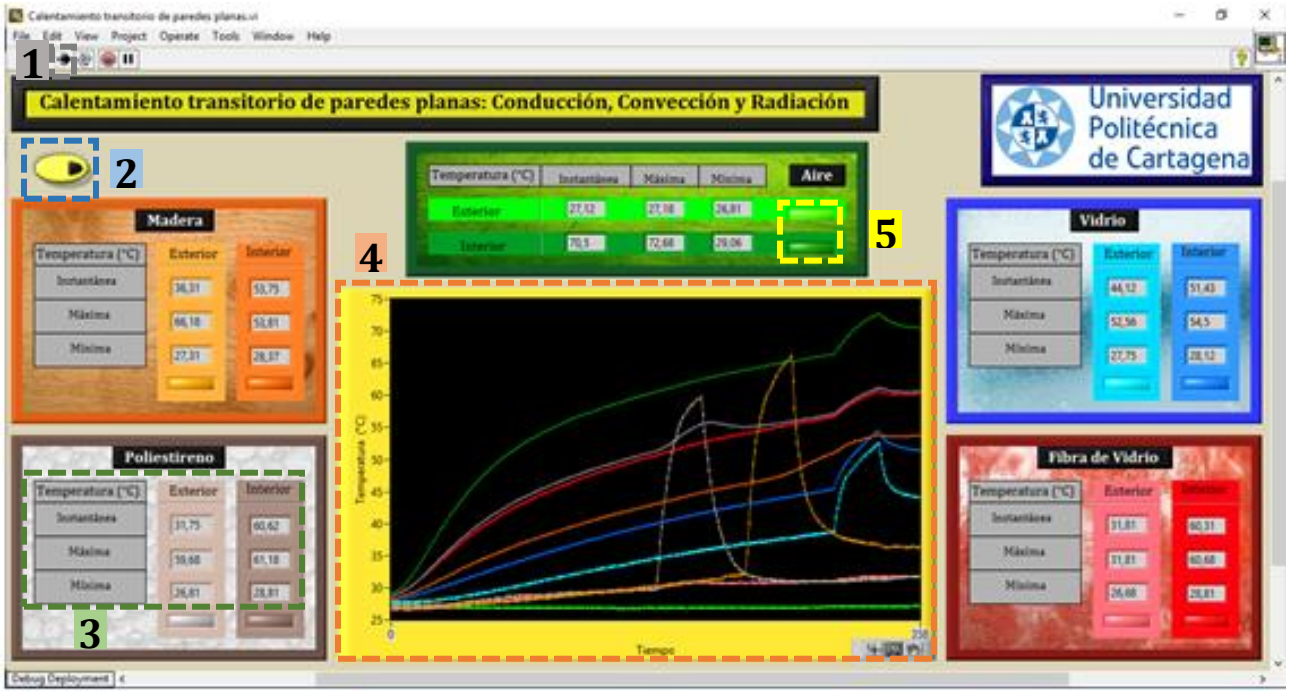


Figura 7.1- Interfaz de la aplicación

A continuación se detallan los distintos elementos que aparecen en la interfaz mostrada con anterioridad, así como las principales características del funcionamiento de nuestra aplicación:

1. **Run:** Arranca el programa
2. **Stop:** Detiene el programa; si volvemos a presionar el botón "Run" éste comenzará desde el inicio.
3. **Tabulador de temperaturas:** Muestra la temperatura de cada material, tanto por la parte interna como externa al cubo. En ambos casos se informa de su magnitud instantánea (valor que se va actualizando cada diez segundos), y de la temperatura máxima y mínima registrada a lo largo de la hora que perdura dicha monitorización.

4. **Gráfica:** Las temperaturas instantáneas son graficadas, y se van actualizando cada 10 segundos. Esta graficación perdura una hora y nos permite obtener, de forma fácil e intuitiva, diferentes conclusiones acerca del comportamiento térmico de cada material.
   También incorpora una serie de herramientas que permiten hacer zoom y movernos a través del gráfico dibujado en tiempo real. 

5. **Activación del gráfico:** Nos permite activar o desactivar la visualización gráfica de la temperatura instantánea (exterior e interior).

Transcurrida una hora de monitorización, el programa se aborta automáticamente y éste nos genera un archivo Excel donde, durante cada minuto, se ha ido tabulando la temperatura instantánea (exterior e interior) para el aire y los materiales que conforman las paredes laterales del cubo. Este archivo contiene la información necesaria para que el alumno sea capaz de contestar a las diferentes preguntas que se formulan en el guion de dicha práctica.

7.2. Diagrama de bloques

A continuación mostramos el programa propiamente dicho, es decir, el código fuente gráfico que define el funcionamiento del VI.

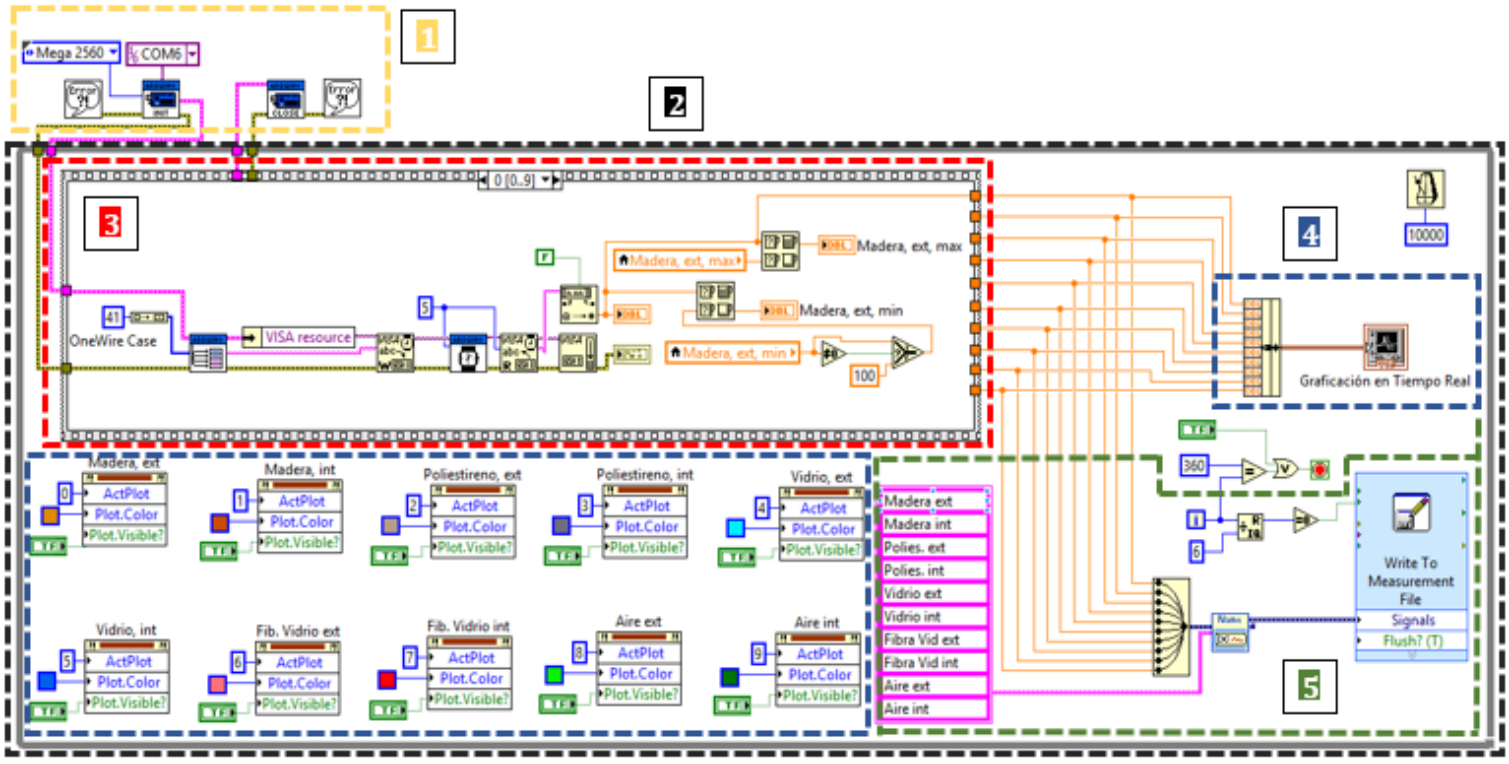


Figura 7.2- Diagrama de bloques de la aplicación

Para un mejor entendimiento del código gráfico mostrado en la anterior imagen, éste se ha subdividido en cinco partes. Procedemos a explicar con detalle cada una:

7.2.1 Inicio y finalización de la comunicación con Arduino

La librería utilizada en LabVIEW para la comunicación con la placa Arduino va a ser el toolkit LIFA (LabVIEW Interfaz For Arduino), como se comentó en apartados anteriores. Este toolkit nos proporciona una serie de subVI especialmente diseñados para dicha placa, de forma que solo tengamos que indicarle una serie de variables para su conexión.

Para iniciar la comunicación con la placa Arduino, se ha de colocar el **subVI Init.vi**, el cual cuenta con 6 variables de entrada y 2 salidas, con las que definimos:

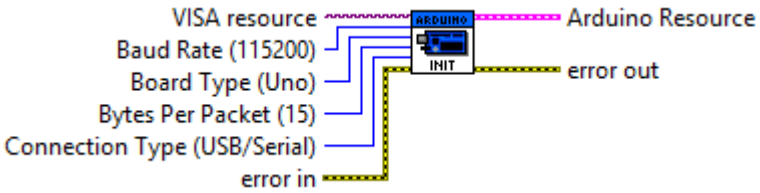


Figura 7.3- Entradas y salidas SubVI Init.vi

- ***VISA resource*:** Indicamos el puerto COM al cual tenemos conectada la placa; en nuestro caso es el COM6. Este campo es obligatorio de introducir al no contar con ningún valor predefinido.
- ***Baud Rate** (115200):* La tasa de baudios también conocida como baudaje, es el número de unidades de señal por segundo. Viene establecida de forma predeterminada en 115200, de forma que en caso de no introducir ningún otro valor utilizaría el predeterminado. En nuestro proyecto se usa 115200.
- ***Board Tipe** (Uno):* Indicamos el tipo de placa que estamos conectando. Viene predeterminado para la placa Arduino Uno; debemos de modificar dicha entrada, pues la placa utilizada en el proyecto es el Arduino Mega 2560.
- ***Connection Type*** (USB/Serial): Indicamos el tipo de conexión utilizado en la computadora para la conexión Arduino. Viene predeterminado para conexión USB/Serial, coincide con la utilizada en el proyecto.
- ***Error in:*** Es la entrada de errores.
- ***Arduino Resource:*** Es la salida que nos muestra la información obtenida en el Arduino.
- ***Error out:*** La salida de errores.

A su vez, también es necesario terminar la comunicación con Arduino, para liberar el puerto y poder darle otra función, y esto se realiza por medio del **subVI Close.vi**, el cual cuenta con dos entradas y una salida:

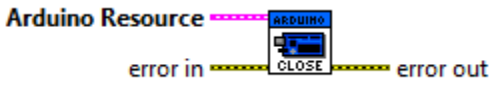


Figura 7.4- Entradas y salidas SubVI Close.vi

- ***Arduino Resource:*** Salida de información obtenida de Init.vi.
- ***Error in:*** Es la entrada de errores.
- ***Error out:*** La salida de errores.

7.2.2 Ciclo While Loop

Es el equivalente al bucle While empleado en los lenguajes convencionales de programación. Sirve para hacer que una secuencia de instrucciones se repita una cantidad de veces, siempre y cuando una afirmación sea verdadera. En LabVIEW se ejecutarán las funciones que se encuentren dentro del cuadro de ciclo.

Al seleccionar la estructura While Loop y situarla en el diagrama de bloques, un botón de stop aparece en el mismo. La condición que se le puede asignar para que se repita el ciclo puede ser:

- Stop si es verdad…
- Continúa si es verdad…

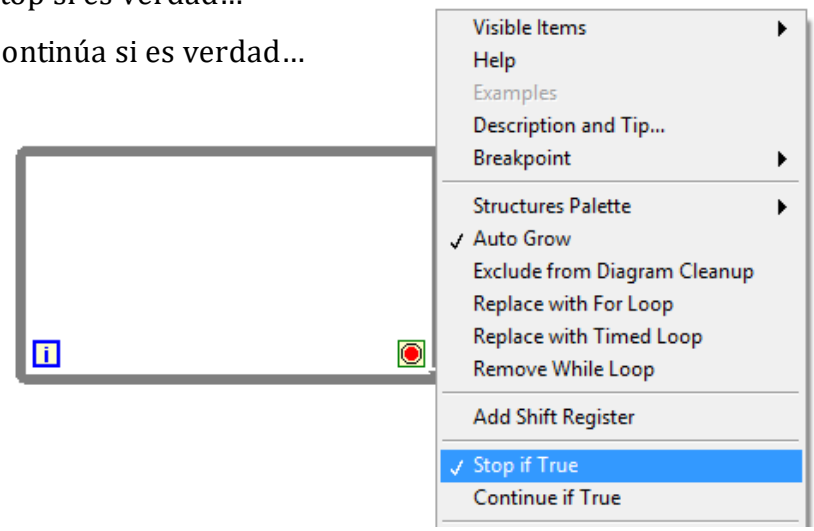
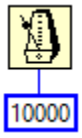


Figura 7.5- Configuración de la estructura While Loop con condicionante Stop if True

En la aplicación desarrollada se ha utilizado la estructura While Loop con la condición de Stop if true, y ésta se repetirá cada 10 segundos. Este tiempo es suficiente como para que los 10 sensores de temperatura DS18B20 registren correctamente dicha magnitud.

Recordamos que el tiempo de espera para que finalice la conversión de temperatura de nuestro sensor en Arduino, es de 850 ms (Ver Código 8), de este modo, al cabo de 8,5 segundos los 10 sensores habrán registrado su temperatura en la interfaz de LabVIEW. Se puede concluir que 10 segundos es suficiente para repetir el ciclo, es decir, realizar un nuevo registro de temperaturas.



Podemos controlar la parada del programa de dos modos:

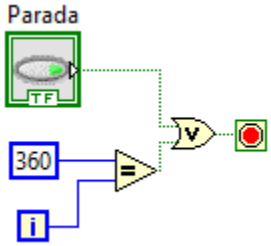
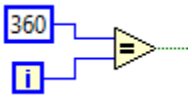


Figura 7.6- Ambas formas de controlar la parada del programa

- Clicando en el botón de parada presente en la interfaz.
- Automáticamente, transcurrida una hora desde el momento en el que se inicializó el programa. Para realizar esta parada se han tenido en cuenta las veces que se repite la estructura While Loop, **i**. Sabemos que se produce una iteración cada 10 segundos, lo que traduce en que al cabo de una hora se realizarán 360 repeticiones.
  La función de comparación *"igual"* se activará cuando el número de iteraciones, i, sea igual a 360, lo que producirá la parada del programa.



7.2.3 Obtención de temperaturas y registro de valores máximos y mínimos

A la hora de recoger los datos de los diferentes sensores de temperatura nos encontramos que éstos no tomaban las temperaturas de forma ordenada, e incluso nos limitaba el número de sensores a solamente cinco.

Esto se solucionó instalando una estructura **Stacked Sequence**, que permite ejecutar varios subdiagramas, denominados *"frames"*, en un estricto orden y donde sólo es visible uno a la vez; permitiendo así la lectura primero de un sensor y después de otro. En los lenguajes de programación convencionales basados en código de líneas no se requiere, y por lo tanto no existe una estructura análoga.

Así, esta estructura ha sido utilizada para ejecutar secuencialmente los VI correspondientes a cada sensor.

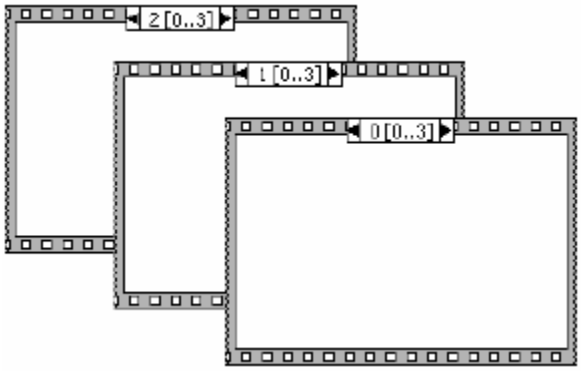


Figura 7.7- Apariencia de estructuras *Sequence*

En la parte superior del marco de cada estructura se encuentra el identificador de diagrama, que es utilizado para navegar entre *frames*.

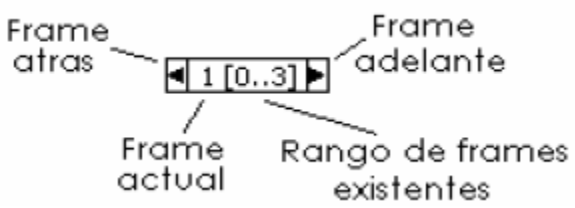


Figura 7.8- Identificador de diagrama

Por defecto la estructura *Sequence* posee un solo *frame* y no tiene identificador de diagrama. Para adicionar un *frame* después del actual debemos de seleccionar la opción *Add Frame After*

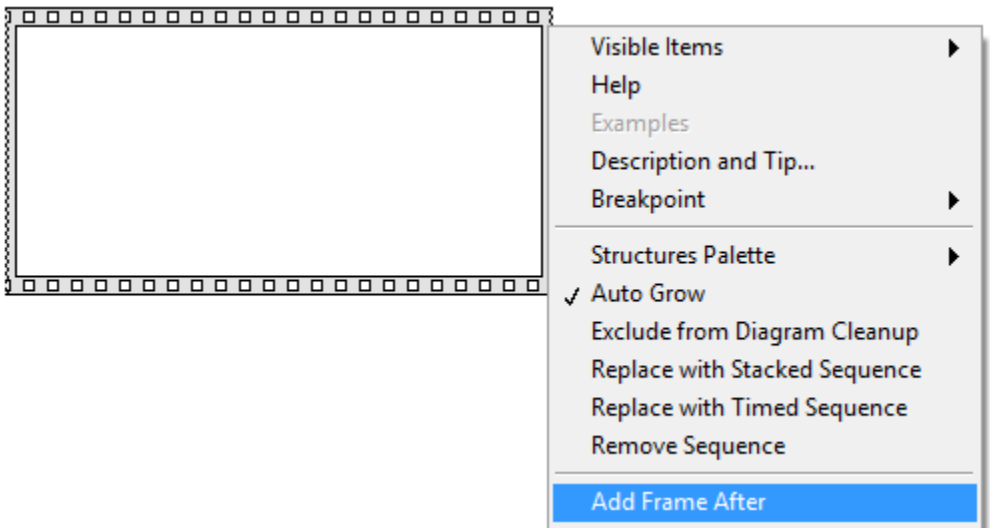


Figura 7.9- Menú de la estructura *sequence*

En nuestro proyecto, como se van a utilizar 10 sensores de temperatura, necesitamos emplear 10 *frames*, es decir, el rango de frames existentes será [0..9]. A continuación se detalla el VI correspondiente al primer sensor, el que registra la temperatura de la madera por la cara externa del cubo.

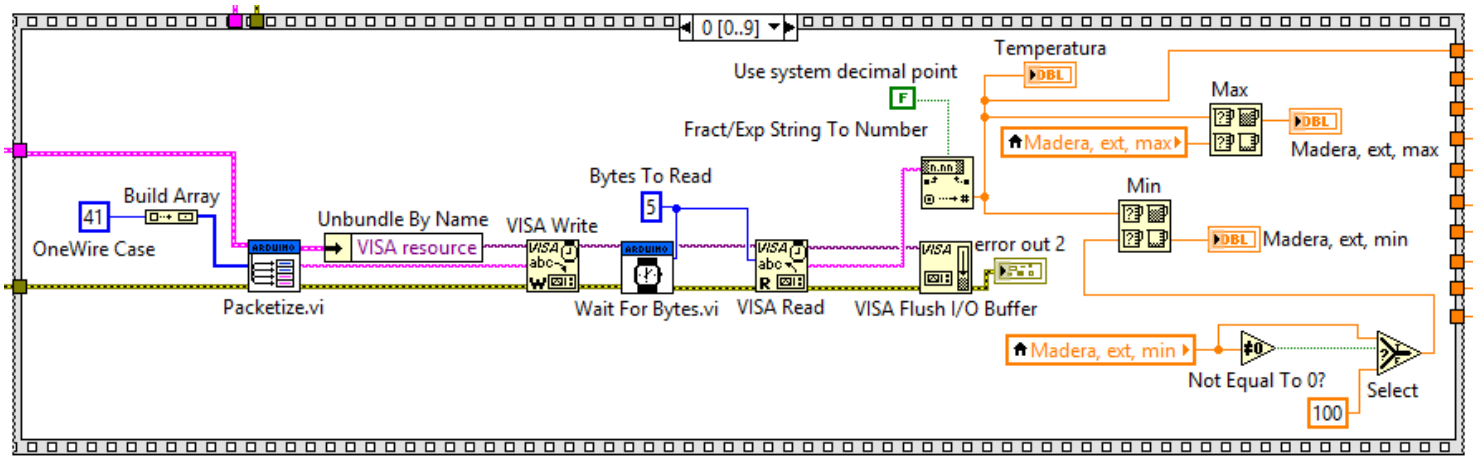


Figura 7.10- VI del primer sensor de temperatura DS18B20

A continuación se describen los pasos seguidos hasta llegar a obtener el valor de temperatura:

1. Llamamos al código cargado en Arduino, concretamente al *case 41* situado dentro de la función *processCommand* (Ver Código 6.7). Este *case* corresponde al código del sensor 1 (madera exterior), y contiene la función con la que se obtiene el dato de temperatura.

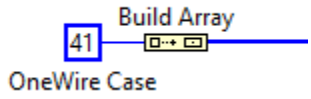


Figura 7.11- Obtención del valor de temperatura mediante el *case 41*

2. Este valor numérico en binario se paquetiza para ser enviado a través del puerto serial; posteriormente éste será monitorizado mediante LabVIEW.

Figura 7.12- subVI Packetize.vi

3. Para comunicar nuestro microcontrolador Arduino desde el puerto serie del ordenador podemos utilizar los drivers de NI VISA para LabVIEW. El envío del paquete creado en el anterior paso, hasta el ordenador, se realiza escribiendo en el puerto serial; utilizamos "VISA Write".

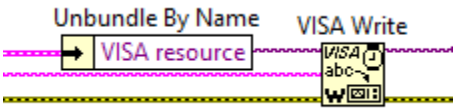


Figura 7.13- Escritura en el puerto serial mediante *VISA Write*

4. La lectura del puerto serial se realiza mediante "VISA Read". Es importante sincronizar las tareas de escritura y lectura, de tal manera que el microcontrolador o la computadora estén listos para enviar o recibir un dato en el puerto serial. Para ello se ha programado una espera, "Wait For Bytes", permitiéndonos que el programa no avance hasta recibir cierto texto.

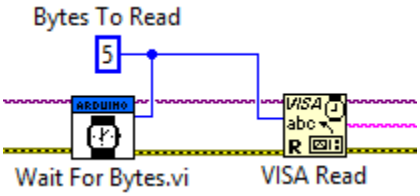


Figura 7.14- Lectura del puerto serial mediante *VISA Read*

5. Una vez leído el puerto serial, la cadena de caracteres obtenida debe ser convertida en un número decimal; para ello empleamos la función *"Fract/Exp String To Number"*. Pero si queremos que además nos muestre un valor numérico con decimales, tenemos que activar la función *"Use system decimal point".*

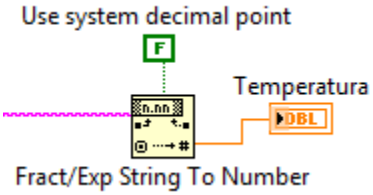


Figura 7.15- Conversión de carácter a número decimal

Una vez que se han completado estos cinco pasos, estamos en disposición del valor numérico registrado por el sensor de temperatura 1.

Para los demás sensores el código será idéntico, es decir, los pasos realizados para este sensor serán análogos para el resto, con la salvedad de que en el **paso 1** se tiene que identificar a cada sensor con el *OneWire Case* que le corresponda (Ver Código 6.7).

Ahora podemos analizar si dicho valor de temperatura se trata de un máximo o de un mínimo.

Para ello utilizamos la función *"Max & Min"*, que compara ambos valores numéricos *x e y*, devolviendo el máximo o mínimo de los dos (en función de la salida que se seleccione).

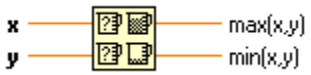


Figura 7.16- Función *Max & Min*

- **¿Es un máximo?**

Para ello debemos de comparar en la entrada de la anterior función, el valor de temperatura instantánea (actualizada cada 10 segundos) con el valor que se esté mostrando en la interfaz hasta ese momento. Como queremos obtener el valor máximo resultante de dicha comparación, tenemos que seleccionar a la salida la opción max (x,y).

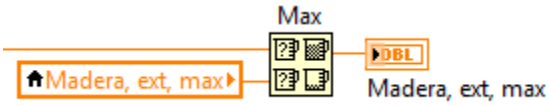


Figura 7.17- Análisis de la temperatura máxima

- **¿Es un mínimo?**

De modo análogo a lo ya comentado para el análisis de la temperatura máxima, tenemos que comparar a la entrada el valor de temperatura instantánea con el valor que se esté mostrando en la interfaz hasta ese momento. Pero aquí nos encontramos con el siguiente problema:

> Antes de arrancar el programa, todos los valores de temperatura que muestra la interfaz son nulos (Ver Figura 7.18), por lo tanto, como en el laboratorio se trabaja con temperaturas positivas, si utilizamos el mismo código empleado para el análisis de la temperatura máxima, con la salvedad de seleccionar a la salida la opción min(x,y), ésta siempre nos mostrará que el valor mínimo son 0°C.

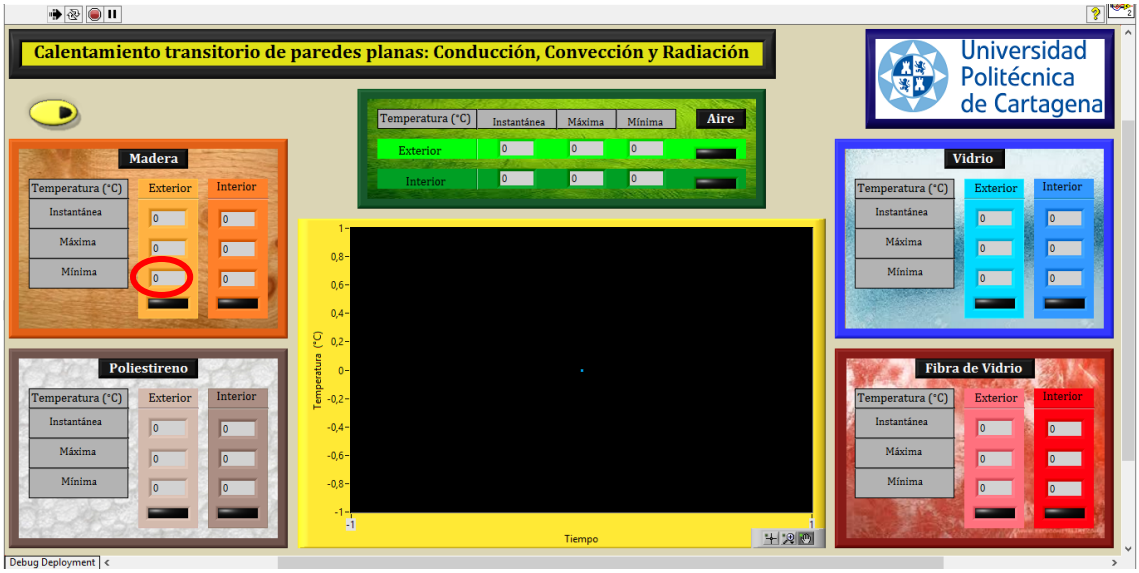


Figura 7.18- Aspecto de la interfaz antes de arrancar el programa

Para solventarlo, debemos conocer las siguientes funciones:

- **Not Equal to 0?:** Devuelve un TRUE si x no es igual a 0, o de otro modo, si x=0 devuelve un FALSE.

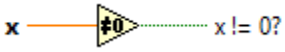


Figura 7.19- Función *Not Equal to 0?*

- **Select:** Si ***s*** es TRUE devuelve a la salida el valor numérico de la entrada ***t***, y en el caso de que ***s*** sea FALSE devuelve a la salida el valor contenido en la entrada ***f***.

Figura 7.20- Función *Select*

Y concluimos que el código que permite solventar este problema es el que se muestra en la siguiente imagen:

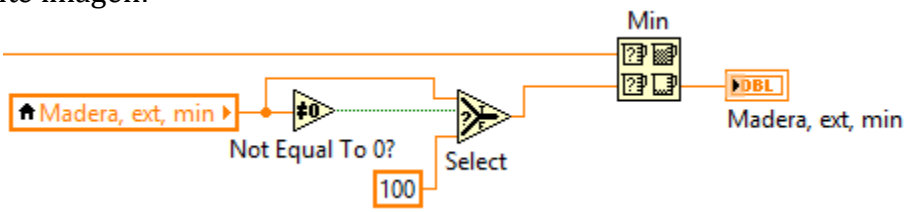


Figura 7.21- Análisis de la temperatura mínima

De este modo, existirán dos comportamientos en función de si el valor almacenado en la interfaz es o no es nulo.

- **Valor nulo** *(caso de inicialización):* la función *Not Equal To 0?* devolverá un FALSE lo que conllevará que la función *Select* muestre a la salida el valor 100. Al comparar dicho valor con la temperatura instantánea (ésta será siempre inferior a 100°C) se registrará ésta última como valor mínimo.
- **Valor no nulo:** la función *Not Equal To 0?* devolverá un TRUE lo que conllevará que la función *Select* muestre a la salida el valor almacenado en la interfaz como mínimo. Al comparar dicho valor con la temperatura instantánea se registrará la temperatura más pequeña.

7.2.4 Graficación de temperaturas en tiempo real y activación de las mismas

La *paleta de controles* del panel frontal, nos permite, pinchando y arrastrando con el ratón del ordenador, insertar los controles necesarios para nuestra aplicación. En la siguiente imagen mostramos los pasos seguidos para insertar una gráfica.

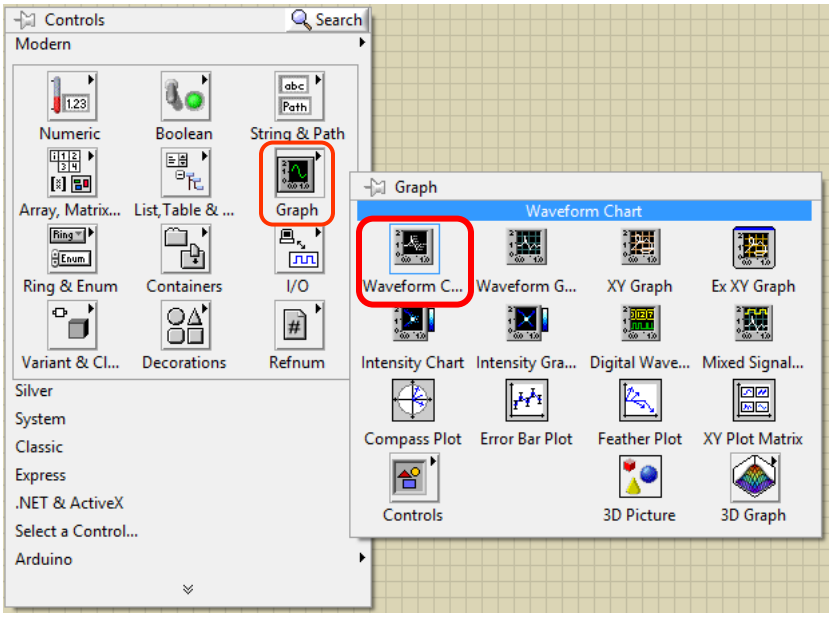


Figura 7.22- Inserción de una gráfica

Para graficar todas las temperaturas a la misma vez y en una sola gráfica, empleamos la función "*Bundle*". Ésta converge las señales de todos los sensores de temperatura en una sola salida, y las envía a graficar.

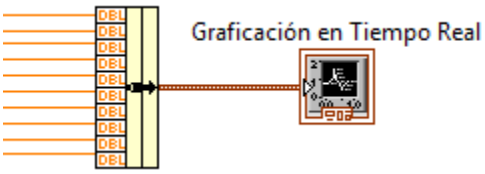


Figura 7.23- Función *Bundle* para conseguir la graficación en una sola gráfica

Tal y como ya se comentó en el apartado 7.1 de este capítulo, la interfaz nos permite activar y desactivar la visualización gráfica de la señal de temperatura de un determinado sensor.

A continuación mostramos los pasos que se han llevado a cabo para conseguir tal efecto.

1. Haciendo clic derecho encima del gráfico mostrado en la figura 7.23, y pinchando en la casilla *Activate Plot* (Ver Figura 7.24), obtenemos lo siguiente (Ver Figura 7.25):

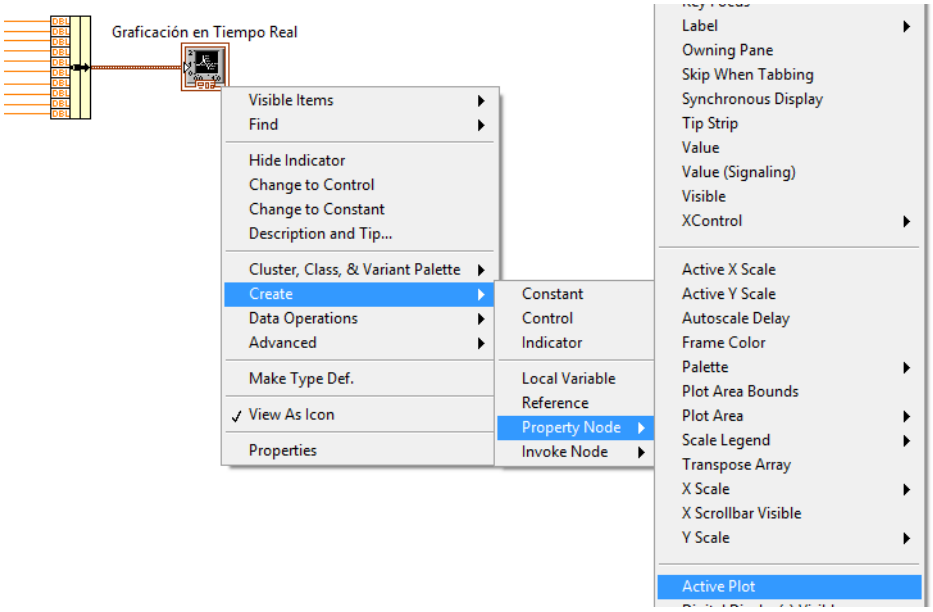


Figura 7.24- Ruta a seguir para clicar obtener la opción *Active Plot*

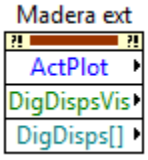


Figura 7.25- Resultado tras clicar en la anterior pestaña

2. Hacemos clic derecho encima de la anterior imagen, para cambiar al modo de escritura, *Chande To Write*.

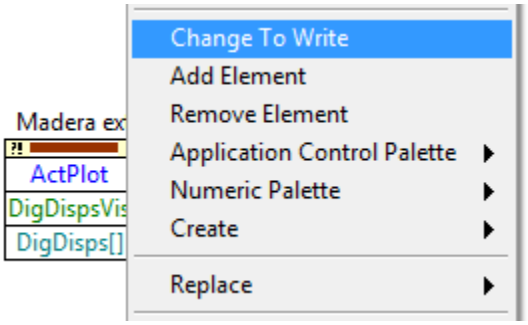


Figura 7.26- Clicamos sobre la pestaña *Change To Write*

Hasta ahora solo hemos seleccionado el sensor sobre el que estamos actuando, ***ActPlot***. Necesitamos poder elegir también el tipo de color para cada sensor de temperatura, ***Plot. Color***, así como activar y desactivar la visualización gráfica mediante un botón, ***Plot. Visible?***.

3. Haciendo clic derecho sobre la anterior figura, y siguiendo la ruta mostrada en la siguiente imagen, añadimos estas dos opciones.

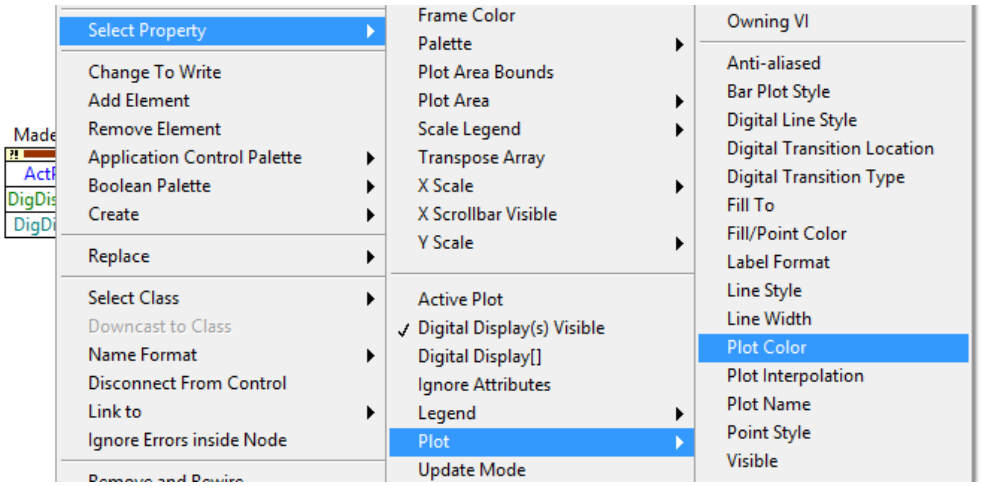


Figura 7.27- Ruta a seguir para obtener las opciones *Plot.Color* y *Plot. Visible?*

De este modo, conseguimos poder activar y desactivar la visualización gráfica de la señal de temperatura del primer sensor (madera exterior), así como facilitar su identificación mediante un color determinado.

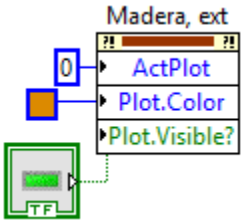


Figura 7.28- Código gráfico resultante

Extrapolando estos pasos, para el resto de sensores de temperatura, obtenemos como resultado el siguiente código gráfico:

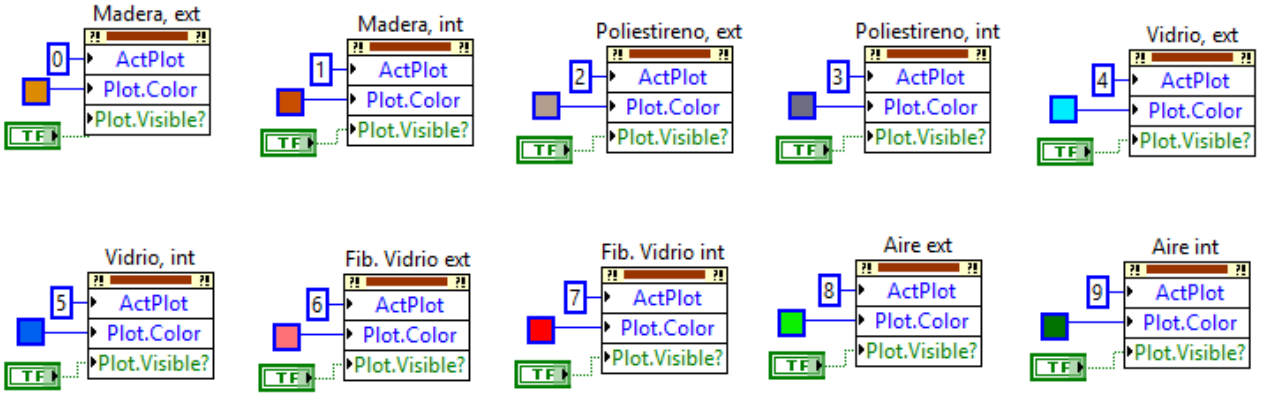


Figura 7.29- Código gráfico para controlar la visualización gráfica de todos los sensores

7.2.5 Exportación de datos a Excel

Tenemos que generar un archivo Excel donde, durante cada minuto, se vaya tabulando la temperatura instantánea (exterior e interior) para el aire y los materiales que conforman las paredes laterales del cubo. La tabla generada debe de informarnos de la fecha y hora en la que las temperaturas han sido registradas, así como, disponer cada columna de cabeceras que faciliten la identificación de las medidas efectuadas.

A partir de la paleta de funciones para la creación del diagrama de bloques, seleccionamos la función *"Write To Measurement File"*.

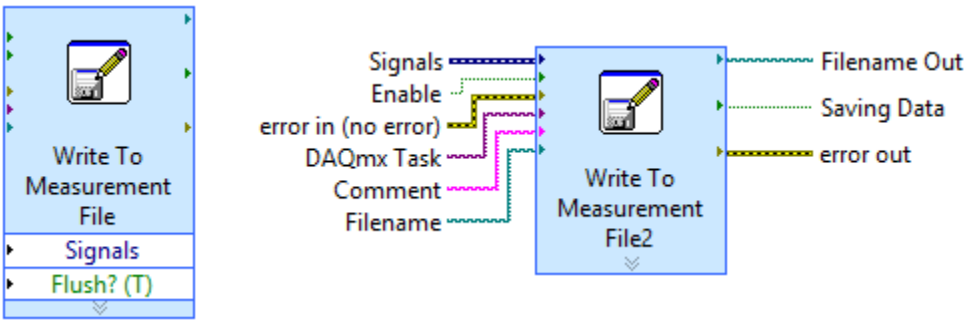


Figura 7.30- Función *Write To Measurement File*

Y establecemos la configuración mostrada en la siguiente imagen:

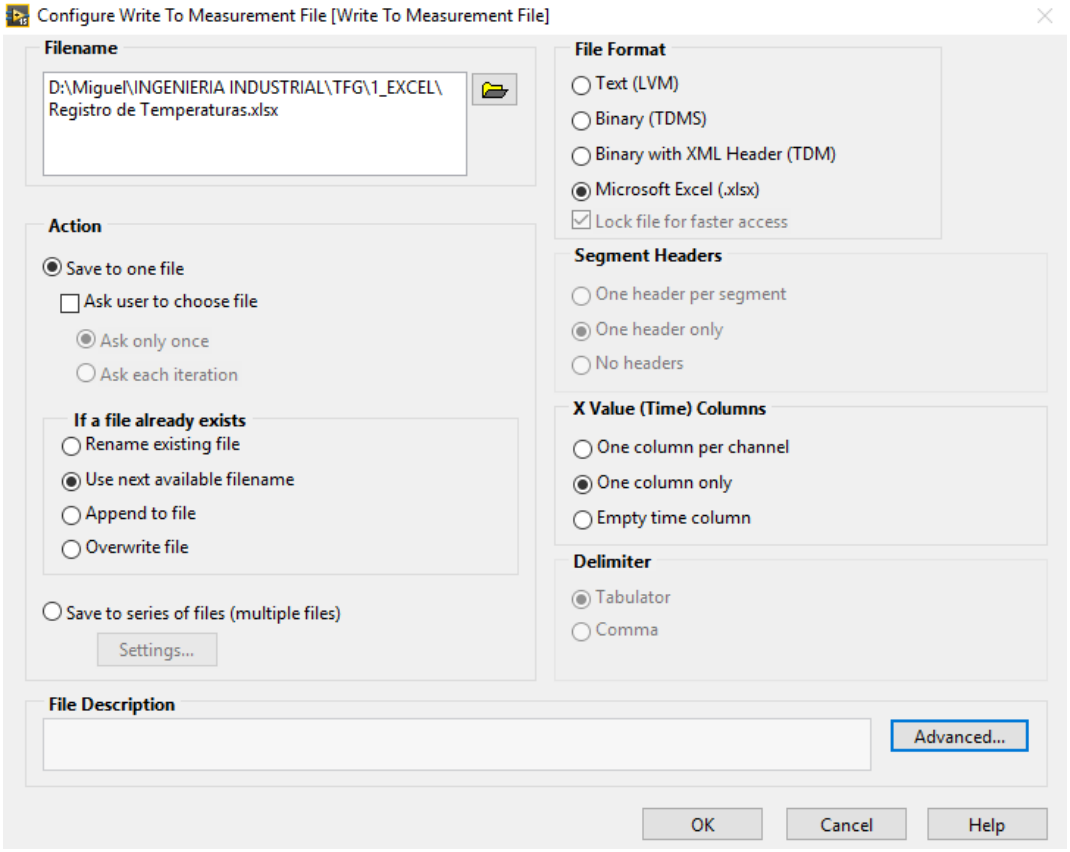


Figura 7.31- Configuración a establecer para la función *Write To Measurement File*

Para que las señales de todos los sensores de temperatura converjan en una sola salida, y sean enviadas a un archivo Excel, hemos empleado la función *"Merge Signals"*.

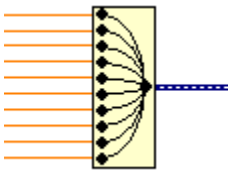


Figura 7.32- Función *Merge Signals*

Las columnas del archivo Excel deben de disponer de cabeceras, para facilitar la identificación de las señales de todos los sensores. Esto es posible mediante el **subVI Nombra señales.vi.**

Figura 7.33- subVI Nombra señales.vi.

El código gráfico que permite registrar las temperaturas con sus respectivas cabeceras tiene el siguiente aspecto:

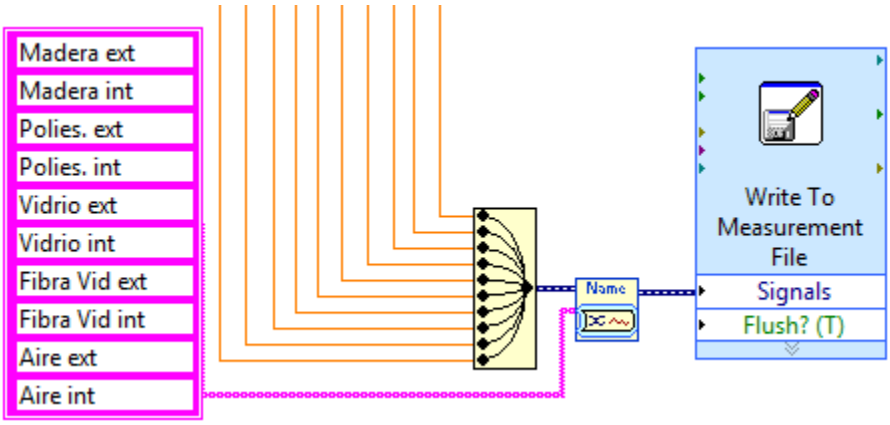


Figura 7.34- Código gráfico para exportar datos a Excel con cabeceras

Nos queda efectuar el registro de temperaturas en el archivo Excel, de forma periódica y cada minuto. Para ello empleamos la entrada *"Enable"* de la función *Write To Measurement File* (Ver Figura 7.30).

Recordando lo expuesto en el apartado 7.2.2, se repetirá todo el ciclo cada 10 segundos. Como necesitamos que la entrada *"Enable"* se active cada minuto para poder escribir en el archivo Excel, esto tendrá lugar siempre que el número de iteraciones realizadas sea múltiplo de seis, es decir, cuando el resto resultante de dividir el número de iteraciones, **i**, entre 6, sea igual a cero.

En la siguiente imagen se muestra el código que refleja lo comentado con anterioridad.

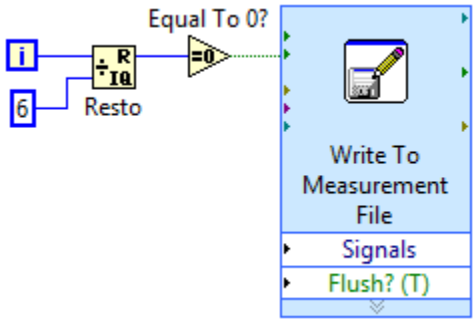


Figura 7.35- Código gráfico que habilita la escritura en Excel cada minuto

A modo de resumen, mostramos el aspecto final del código gráfico que nos permite exportar correctamente datos a Excel.

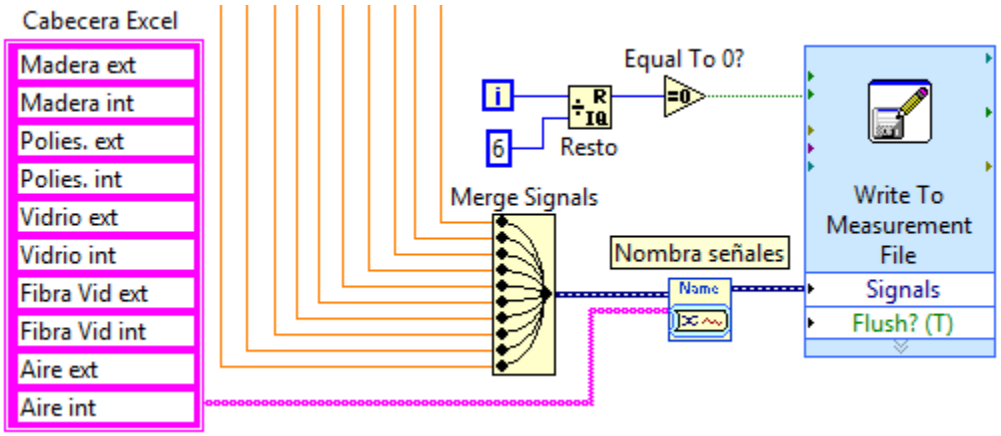


Figura 7.36- Código gráfico completo para exportar datos a Excel

Se puede comprobar la validez del código expuesto con anterioridad.

Time	Madera ext	Madera int	Polies. ext	Polies. int	Vidrio ext	Vidrio int	Fibra Vid ext	Fibra Vid int	Aire ext	Aire int
05/09/2016 09:50:04,132	27,37	28,37	26,87	28,81	27,75	28,12	26,68	28,81	26,81	29,06
05/09/2016 09:51:04,136	27,37	28,62	26,87	29,18	27,75	28,31	26,75	29,25	26,87	29,81
05/09/2016 09:52:04,113	27,31	29	26,87	29,93	27,81	28,56	26,68	30	26,87	31,31
05/09/2016 09:53:04,127	27,31	29,56	26,93	31,06	27,81	28,93	26,75	31,06	26,93	33,25
05/09/2016 09:54:04,125	27,31	30,18	27	32,37	27,87	29,31	26,75	32,25	26,93	35,43
05/09/2016 09:55:04,106	27,37	30,87	27,12	33,81	28,06	29,81	26,87	33,5	26,93	37,75
05/09/2016 09:56:04,108	27,37	31,62	27,31	35,18	28,18	30,25	27	34,75	26,93	39,87
05/09/2016 09:57:04,118	27,37	32,25	27,5	36,5	28,37	30,75	27,12	36	26,87	41,93
05/09/2016 09:58:04,117	27,43	33	27,68	37,75	28,62	31,18	27,25	37,12	26,93	43,81

Figura 7.37- Muestra del archivo Excel generado por LabVIEW

CAPÍTULO 8. Ejecución física del proyecto. Presupuesto

8.1. Ejecución física del proyecto

Tras elaborar el sistema de medición y monitorización de temperaturas, realizamos una prueba que nos permita dar validez a los resultados obtenidos con el mismo. Ésta consistirá en realizar la práctica de calentamiento transitorio de paredes planas, fin de nuestro TFE.

Los valores de las temperaturas de la casa térmica son publicados en un Excel, que se elaborará al transcurrir la hora que perdura la práctica. En la siguiente tabla se muestran las temperaturas registradas en cada una de las paredes durante el proceso de calentamiento, en intervalos de tiempo de un minuto.

PROCESO DE CALENTAMIENTO

Time	Madera ext	Madera int	Polies. ext	Polies. int	Vidrio ext	Vidrio int	Fibra Vid ext	Fibra Vid int	Aire ext	Aire int
05/09/2016 09:50:04,132	27,37	28,37	26,87	28,81	27,75	28,12	26,68	28,81	26,81	29,06
05/09/2016 09:51:04,136	27,37	28,62	26,87	29,18	27,75	28,31	26,75	29,25	26,87	29,81
05/09/2016 09:52:04,113	27,31	29	26,87	29,93	27,81	28,56	26,68	30	26,87	31,31
05/09/2016 09:53:04,127	27,31	29,56	26,93	31,06	27,81	28,93	26,75	31,06	26,93	33,25
05/09/2016 09:54:04,125	27,31	30,18	27	32,37	27,87	29,31	26,75	32,25	26,93	35,43
05/09/2016 09:55:04,106	27,37	30,87	27,12	33,81	28,06	29,81	26,87	33,5	26,93	37,75
05/09/2016 09:56:04,108	27,37	31,62	27,31	35,18	28,18	30,25	27	34,75	26,93	39,87
05/09/2016 09:57:04,118	27,37	32,25	27,5	36,5	28,37	30,75	27,12	36	26,87	41,93
05/09/2016 09:58:04,117	27,43	33	27,68	37,75	28,62	31,18	27,25	37,12	26,93	43,81
05/09/2016 09:59:04,108	27,5	33,62	27,87	38,87	28,87	31,68	27,43	38,18	26,93	45,43
05/09/2016 10:00:04,117	27,5	34,25	28	39,93	29,06	32,12	27,56	39,18	26,87	47
05/09/2016 10:01:04,115	27,62	34,87	28,18	40,93	29,31	32,62	27,81	40,12	26,93	48,37
05/09/2016 10:02:04,121	27,75	35,43	28,37	41,81	29,62	33,06	27,93	40,93	26,93	49,62
05/09/2016 10:03:04,127	27,81	36	28,62	42,62	29,87	33,5	28,06	41,75	26,93	50,68
05/09/2016 10:04:04,121	28	36,56	28,68	43,43	30,18	33,93	28,25	42,56	26,93	51,68
05/09/2016 10:05:04,121	28,12	37	28,81	44,12	30,43	34,37	28,37	43,25	26,93	52,56
05/09/2016 10:06:04,113	28,18	37,56	28,81	44,81	30,75	34,81	28,5	43,93	26,87	53,37
05/09/2016 10:07:04,119	28,31	38,06	28,87	45,43	31,06	35,25	28,62	44,56	26,87	54,12
05/09/2016 10:08:04,122	28,5	38,43	29	46	31,37	35,68	28,68	45,12	26,93	54,81
05/09/2016 10:09:04,128	28,62	38,93	29,06	46,56	31,62	36,06	28,87	45,75	26,87	55,43
05/09/2016 10:10:04,119	28,75	39,37	29,18	47,12	31,93	36,5	29	46,25	26,87	56,06
05/09/2016 10:11:04,121	28,93	39,81	29,25	47,62	32,25	36,87	29,12	46,81	26,87	56,62
05/09/2016 10:12:04,124	29,06	40,18	29,31	48,12	32,5	37,25	29,18	47,31	26,87	57,12
05/09/2016 10:13:04,125	29,18	40,62	29,37	48,56	32,81	37,62	29,25	47,81	26,87	57,62
05/09/2016 10:14:04,119	29,31	41	29,43	49,06	33,06	38	29,37	48,25	26,81	58,12
05/09/2016 10:15:04,117	29,5	41,37	29,56	49,43	33,37	38,37	29,43	48,68	26,87	58,56
05/09/2016 10:16:04,115	29,62	41,75	29,56	49,87	33,5	38,75	29,43	49,12	26,81	58,93
05/09/2016 10:17:04,122	29,75	42,12	29,62	50,25	33,81	39,06	29,56	49,5	26,81	59,43
05/09/2016 10:18:04,120	30	42,43	29,68	50,68	34,12	39,37	29,62	49,93	26,81	59,81
05/09/2016 10:19:04,126	30,12	42,81	29,75	51	34,37	39,75	29,75	50,31	26,87	60,18
05/09/2016 10:20:04,132	30,18	43,12	31,25	51,37	34,62	40,06	29,87	50,68	26,87	60,62

Figura 8.1- Valores de temperatura registrados durante el proceso de calentamiento

Tras estos 30 minutos en los que las paredes solo han recibido el flujo de calor de la bombilla de la caja, procedemos a radiar durante 5 minutos cada una de las paredes, dejando un intervalo de 5 minutos entre cada fase de iluminación. El orden en el que se radió las diferentes paredes fue: poliestireno, madera, y vidrio.

En la siguiente tabla se muestran las temperaturas registradas en cada una de las paredes, en intervalos de tiempo de un minuto.

PROCESO DE CALENTAMIENTO Y RADIACIÓN

Time	Madera ext	Madera int	Polies. ext	Polies. int	Vidrio ext	Vidrio int	Fibra Vid ext	Fibra Vid int	Aire ext	Aire int
05/09/2016 10:21:04,114	30,43	43,43	45,5	52,06	34,87	40,37	30,25	51,06	26,87	61
05/09/2016 10:22:04,121	30,62	43,81	52,93	53	35,06	40,68	30,43	51,5	26,87	61,37
05/09/2016 10:23:04,119	30,81	44,18	56,87	54	35,31	41	30,68	51,93	26,87	61,75
05/09/2016 10:24:04,124	30,93	44,5	58,62	54,87	35,56	41,31	30,81	52,31	26,87	62,12
05/09/2016 10:25:04,123	31,06	44,93	59,62	55,56	35,81	41,62	30,87	52,68	26,87	62,5
05/09/2016 10:26:04,129	31,25	45,12	46,62	55,81	36,06	41,93	30,75	53	26,93	62,81
05/09/2016 10:27:04,118	31,43	45,37	39,43	55,62	36,25	42,18	30,75	53,25	26,93	63,12
05/09/2016 10:28:04,125	31,81	45,68	35,5	55,37	36,43	42,43	30,68	53,56	26,93	63,43
05/09/2016 10:29:04,114	31,93	45,93	33,31	55,18	36,62	42,75	30,75	53,81	26,93	63,68
05/09/2016 10:30:04,129	32,12	46,18	32,25	55,12	36,87	43	30,75	54,12	26,93	64
05/09/2016 10:31:04,111	51,25	46,5	31,68	55,18	37,06	43,25	30,75	54,37	26,87	64,25
05/09/2016 10:32:04,121	58,43	46,75	31,37	55,31	37,18	43,5	30,68	54,62	26,87	64,5
05/09/2016 10:33:04,123	61,75	47,06	31,18	55,5	37,37	43,75	30,81	54,87	26,87	64,81
05/09/2016 10:34:04,117	64,25	47,43	31,12	55,68	37,62	44	30,93	55,12	26,93	65
05/09/2016 10:35:04,119	65,93	48	31	55,93	37,75	44,18	30,93	55,37	26,87	65,31
05/09/2016 10:36:04,130	50,62	48,56	30,93	56,12	37,93	44,43	31	55,62	26,93	65,5
05/09/2016 10:37:04,119	43,62	49,31	31	56,37	38,18	44,68	31,06	55,87	26,93	65,75
05/09/2016 10:38:04,121	40,81	49,93	31	56,62	38,31	44,87	31,06	56,18	26,93	66
05/09/2016 10:39:04,124	39,31	50,56	31	56,87	38,5	45,12	31	56,43	26,93	66,18
05/09/2016 10:40:04,138	38,5	51,06	31	57,12	38,81	45,56	31	56,75	27	66,56
05/09/2016 10:41:04,112	37,75	52	31,06	58,31	46,25	48,81	31,37	57,87	27,06	68,43
05/09/2016 10:42:04,118	37,31	52,62	31,12	59,18	48,5	50,56	31,43	58,81	27	70
05/09/2016 10:43:04,123	37,12	53,06	31,18	60	50,06	52,06	31,62	59,5	27,06	71,12
05/09/2016 10:44:04,118	36,87	53,5	31,25	60,62	51,31	53,37	31,75	60,12	27,12	72,06
05/09/2016 10:45:04,122	36,68	53,81	31,31	61,18	52,56	54,5	31,81	60,68	27,06	72,68
05/09/2016 10:46:04,115	36,25	53,62	31,37	60,81	46,43	52,68	31,75	60,25	27,12	71,68
05/09/2016 10:47:04,121	36,37	53,56	31,56	60,56	45,06	52,12	31,75	60,12	27,12	71,06
05/09/2016 10:48:04,124	36,37	53,56	31,68	60,5	44,43	51,75	31,75	60,18	27,12	70,68
05/09/2016 10:49:04,117	36,37	53,68	31,75	60,56	44,25	51,56	31,75	60,25	27,18	70,56
05/09/2016 10:50:04,131	36,37	53,75	31,81	60,62	44,06	51,37	31,87	60,37	27,12	70,5

Figura 8.2- Valores de temperatura registrados durante el proceso de calentamiento y radiación

Todos los valores tabulados en las Figuras 8.1 y 8.2 se han ido graficando en tiempo real (cada 10 segundos se graficaban valores de temperatura) en la interfaz del programa desarrollada para la realización de esta práctica. Concluido el tiempo que perdura la práctica, la gráfica resultante de esta monitorización tiene el aspecto de la Figura 8.3.

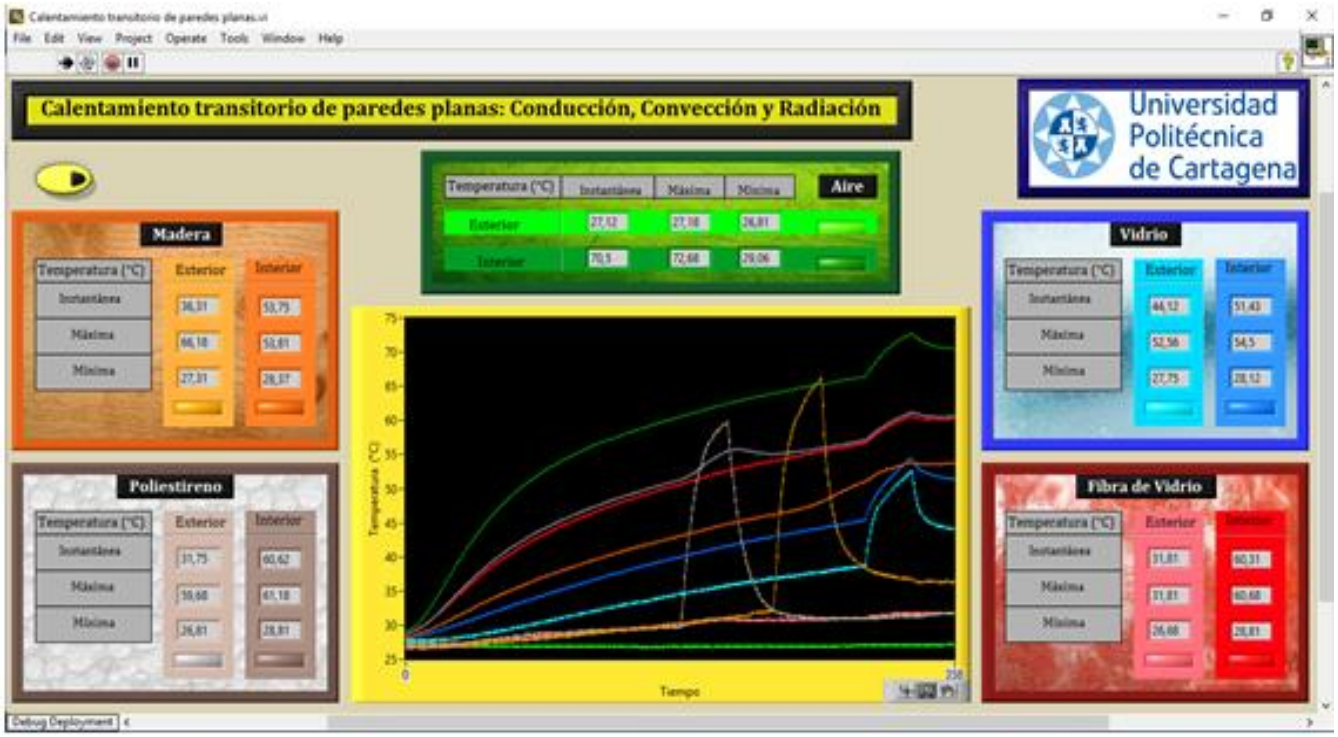


Figura 8.3- Graficación de temperaturas resultante de ambos procesos

Durante el proceso de calentamiento, generado únicamente por la bombilla de 100W, las paredes comienzan a calentarse. Se puede ver en la gráfica la diferencia de temperaturas entre las superficies interna y externa de cada material, pudiendo evaluar a simple vista la capacidad de conducción de cada pared. Las paredes con un alto coeficiente de conducción de calor presentaran menos diferencia de temperatura entre una y otra cara.

Pasada media hora, procedemos a enchufar un foco que radia, en primer lugar la pared de poliestireno durante 5 minutos. La temperatura de la superficie interior del poliestireno aumenta rápidamente ayudada por el aporte de calor de la radiación.

Dejamos una pausa de 5 minutos, y repetimos el proceso con la pared de madera. Los resultados son parecidos a los del poliestireno llegando a ser más grande la diferencia de temperaturas entre las caras exterior e interior.

Por último, radiamos la pared de cristal. Los resultados son totalmente distintos. Las variaciones de temperaturas son despreciables entre las caras, y esto se debe a que el vidrio no absorbe apenas radiación, sin embargo sí que transmite el calor por conducción.

Vemos que la madera es, de los tres materiales analizados, el que tiene un poder absorbente α mayor, de ahí el motivo de que sea un material de construcción de viviendas. Se observa que, una vez que cesa la radiación, la temperatura de la madera se estabiliza de forma más lenta como se puede ver en la gráfica.

El poliestireno, al igual que la madera, presenta un poder absorbente bastante alto y un poder transmisivo que puede considerarse despreciable.

Por el contrario, el vidrio apenas absorbe radiación pero está dotado de un poder transmisivo mucho más alto siendo este poder predominante frente a la absorción de radiación y reflexión.

El comportamiento térmico experimentado por los diferentes materiales que conforman la maqueta, responde perfectamente a las características de los mismos. De este modo, se puede concluir que los resultados proporcionados por el sistema de medición fruto de nuestro proyecto, son válidos.

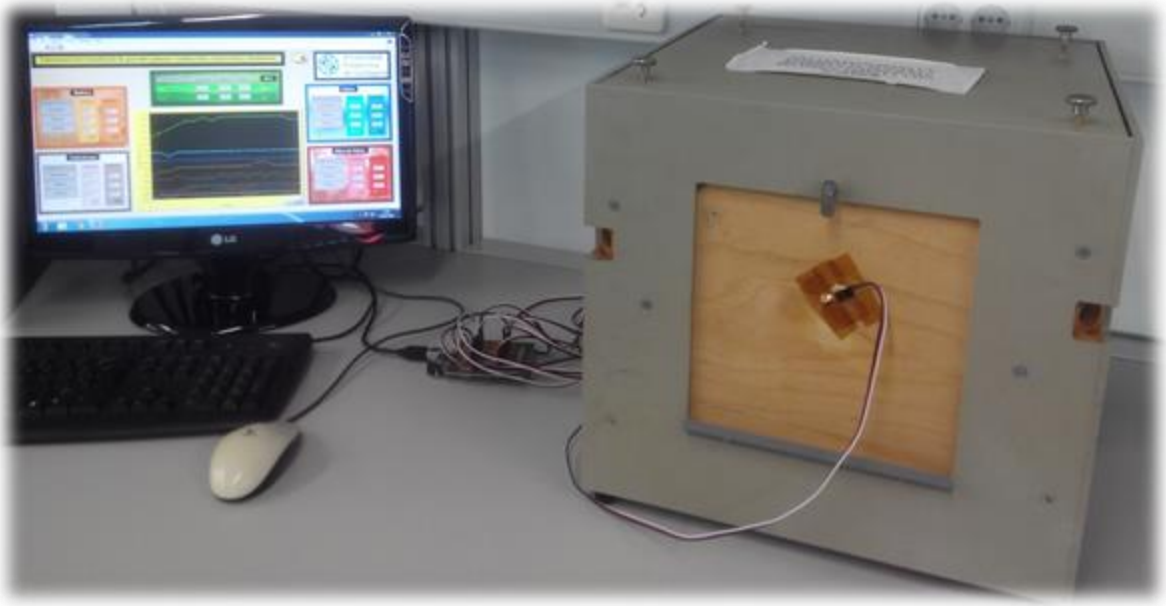


Figura 8.4- Aspecto final de la instalación

8.2. Presupuesto

En este apartado se realiza un listado tanto de los materiales necesarios para la elaboración de nuestro proyecto, como de aquellos necesarios para un sobredimensionamiento, previsiones para el futuro, y reemplazamiento de componentes.

MATERIALES NECESARIOS PARA ELABORAR ESTE PROYECTO			
Materiales	**Cantidad**	**Precio unitario (€/und.)**	**Importe (€)**
Arduino Mega 2560	1	35,00	35,00
Cable USB AM-BM de 2m	1	4,00	4,00
Protoboard (1260 puntos)	1	12,50	12,50
Sensor de temperatura digital DS18B20	10	2,50	25,00
Resistencia de 1kΩ	10	0,25	2,50
Cable macho-hembra de 3 hilos y 0,4 m de longitud	25	1,00	25,00
Cable macho-macho de 10 cm de longitud	45	-	4,00
Cable macho-macho de 20cm de longitud	15	-	1,75
		Precio Total	109,75 €

En cuanto a los materiales necesarios para un sobredimensionamiento, previsiones para el futuro, y reemplazamiento de componentes que se vayan deteriorando, se ha estimado que el presupuesto tendrá un valor cercano al precio total calculado en la anterior tabla.

De este modo, podemos concluir que el presupuesto total necesario para elaborar dicho proyecto asciende a **220 € (doscientos veinte euros).**

CAPÍTULO 9. Bibliografía

Al tratarse de un proyecto eminentemente práctico, internet y la propia experiencia del resto de usuarios del microcontrolador Arduino y LabVIEW resultarán de gran ayuda para la realización del proyecto.

[1] Sensores de Temperatura [Internet]. [consulta 4 de Enero 2016]. Disponible en: http://snsoresdetemperatura.blogspot.com.es/

[2] Sensores de temperatura para Arduino [Internet]. [consulta 5 de Enero 2016]. Disponible en: https://giltesa.com/2012/08/31/sensores-de-temperatura-para-arduino

[3] Arduino - Diseño y Manufactura- [Internet]. [consulta 13 de Febrero 2016]. Disponible en: https://sites.google.com/site/temasdedisenoymanufactura/arduino

[4] Foro: Arduino and Dallas DS1820 (one-wire) [Internet]. [consulta 15 de Marzo 2016]. Disponible en: https://decibel.ni.com/content/thread/9701?start=0&tstart=0

A pesar de la gran cantidad de información disponible en Internet, también se ha recurrido al uso de algunos ejemplares de la biblioteca:

[5] LAJARA VIZCAÍNO, José Rafael. LabVIEW entorno gráfico de programación. Barcelona: Marcombo, 2010. 2ª edición. 477p, ISBN: 9789788426714

[6] GOILAV, Nicolas. Arduino: Aprender a desarrollar para crear objetos inteligentes. En castellano. 332p, ISBN: 9782409000447