UNIVERSIDAD AUTONOMA DEL ESTADO DE MÉXICO

FACULTAD DE INGENIERÍA

"Monitoreo de temperatura a través de una aplicación Web"

REPORTE DE APLICACIÓN DE CONOCIMIENTOS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

Presenta:

Jorge Vázquez Sánchez

Asesor de reporte:

M. En Ing. Germán García Benítez

Toluca, México Junio 2013

AGRADECIMIENTOS

A mi esposa Magda,
por su amor, compresión y apoyo en mis objetivos, ideas y aventuras.

A mis mamás: Silvia y Luchita (abuelita),
por su amor, enseñanzas y apoyo incondicional.

A mis papás: Jorge, Benjamín (abuelito) y Benjamín (tío),
porque de los tres he aprendido el significado de la honradez, del amor paternal y de la tenacidad.

A mi asesor M. en I. Germán García,
por todo el apoyo que me brindó durante la realización de este trabajo.

Infinitas gracias a todos: Jorge VS

Índice

Introducción v

Descripción del problema viii

Justificación ix

Objetivo general ix

Alcances y limitaciones ix

Metodología x

Capítulo 1. Fundamentos conceptuales 1

1.1 Hardware 1

1.1.1 Microcontroladores 1

1.1.2 Diferencias entre microprocesador y microcontrolador 3

1.1.3 Los microcontroladores en la actualidad 4

1.1.4 Aplicaciones de los microcontroladores 4

1.1.5 Clasificación de microcontroladores de acuerdo a la longitud de su bus de datos 5

1.1.6 Clasificación de microcontroladores de acuerdo a su set de instrucciones 6

1.1.7 Clasificación de Microcontroladores de acuerdo a su arquitectura interna 6

1.1.8 Los Microcontroladores PIC de Microchip Technology Inc. 8

1.1.9 Selección de un microcontrolador 10

1.1.10 Microcontroladores PIC de 8-Bits 12

1.1.11 Microcontrolador PIC18F4550 18

1.1.12 Programador de PICs (MiniPIC v2.0) 23

1.1.13 USB (Universal Serial Bus) 25

1.1.14 Proceso de enumeración de un dispositivo USB 28

1.1.15 Especificaciones del bus USB 29

1.1.16 Velocidad de transmisión del puerto USB 31

1.1.17 Interfaz RS-232 31

1.1.18 Emulación de la Interfaz RS-232 sobre USB 32

1.1.19 USB CDC (Communication Device Class) 33

1.1.20 Sensores 35

1.1.21 Sensor de temperatura (LM35) 35

1.1.22 Módulo ADC 37

1.2 Software 39

1.2.1 Java 39

1.2.2 Spring framework (ver 2.5.6) 40

1.2.3 Spring MVC (ver 2.5.6) 41

1.2.4 Apache Tiles 43

1.2.5 Ajax 44

1.2.6 jQuery 45

1.2.7 Maven 45

1.2.8 Tomcat 47

1.2.9 MPLAB X IDE 47

1.2.10 Compilador CCS 48

Capítulo 2. Diseño del sistema 50

2.1 Consideraciones generales del diseño 50

2.2 Diagrama general del sistema 51

2.3 Diseño del hardware 52

2.3.1 Funcionamiento del programa en el microcontrolador 55

2.3.2 Formato de la información que envía y recibe el microcontrolador 56

2.4 Diseño del Software 57

2.4.1 Arquitectura de la aplicación 57

2.4.2 Procesamiento de las peticiones del cliente 58

2.4.3 Obtención del valor de la temperatura enviada por el microcontrolador 59

2.4.4 Presentación del valor de la temperatura al usuario 60

2.4.5 Diseño de la interfaz de usuario 61

2.4.6 Diagrama de paquetes 62

2.4.7 Diagrama de clases 64

Capítulo 3. Implementación del sistema 65

3.1 Implementación del hardware 65

3.1.1 Prueba de lectura del puerto analógico y envío de datos a través del USB 65

3.1.2 Lectura del sensor de temperatura y envío de datos a través del USB 69

3.2 Implementación del software 72

3.2.1 Construcción y administración de dependencias de la aplicación 73

3.2.2 Obtención del valor de la temperatura enviado por el microcontrolador 74

3.2.3 Presentación del valor de la temperatura al usuario 77

3.2.4 Implementación de la interfaz de usuario 80

Capítulo 4. Integración de las etapas de hardware y software 89

4.1 Integración de la etapa de potencia 97

Conclusiones 100

Recomendaciones 101

Costos del proyecto 103

Propuestas de mejoras para desarrollos futuros 105

Bibliografía 107

Apéndice A. Instalar el driver USB CDC en Windows 111

Apéndice B. Creación de un proyecto Web dinámico usando Maven en Eclipse 117

Apéndice C. Código Fuente (Parte hardware) 125

Apéndice D. Código Fuente (Parte software) 132

Glosario 161

Introducción

En la actualidad, los microcontroladores han conquistado el mundo entero, pues se encuentran presentes en todas partes, aunque no nos demos cuenta de ello. Nos acompañan a donde quiera que vayamos, ya sea a nuestro trabajo, a nuestra casa, a un centro comercial, a un hospital, en el auto, e incluso en nuestras comunicaciones telefónicas o vía Internet. Es común hallarlos controlando el funcionamiento de impresoras, teléfonos, calculadoras, hornos de microondas, monitores, sistemas de riego, lavadoras, licuadoras, televisiones, juguetes, naves espaciales, maquinaria industrial, etc.

Sin embargo, una de las aplicaciones más importantes que éstos tienen, es cuando hacen uso de las redes de computadoras, especialmente con Internet, pues abre paso a nuevas formas de interactuar con nuestro medio a distancia (es decir, sin tener que estar presente físicamente en un lugar en especial) y de esta manera atacar necesidades tales como la de controlar dispositivos electrónicos y/o electromecánicos que nos permitan realizar tareas de manera remota, o bien, como la de monitorear variables de dicho ambiente, tales como su temperatura, presión, humedad, etc.

Dado lo anterior, el presente proyecto tiene como objetivo aplicar los conocimientos adquiridos durante la formación profesional, para desarrollar una aplicación que permita monitorear la temperatura de un lugar, dispositivo o algún otro elemento de manera remota a través de una red de área local o Internet; y de esta manera poder también presentar la interacción de un microcontrolador con tecnologías Web. Para ello el presente trabajo se divide en cuatro capítulos, descritos brevemente a continuación.

En el primer capítulo, se interna al lector en el mundo de los microcontroladores, dándole una explicación detallada de la manera en la que éstos se clasifican, sus aplicaciones e importancia en la actualidad, así como de sus principales diferencias con un microprocesador. También se mencionan algunas de las compañías que los fabrican, enfocándose principalmente en aquella a la que pertenece el modelo que le da vida al presente proyecto.

Así mismo, se presenta una descripción de los componentes y tecnologías que intervienen en el desarrollo del proyecto, separándolo en dos etapas: una de software y otra de hardware. Es importante mencionar que tanto en este capítulo como en los subsecuentes, se ha optado por mantener esta división, con el propósito de mantener una separación clara y ordenada de aquellos componentes o tecnologías que integran a cada una de ellas.

En el segundo capítulo, se lleva a cabo la parte correspondiente al diseño del sistema, partiendo desde consideraciones generales del proyecto y adaptándolas a cada una de sus etapas. También se puede apreciar el diagrama general del sistema y los componentes que lo conforman, la definición del formato estándar para la comunicación entre las dos etapas y partes fundamentales en el desarrollo de aplicaciones orientadas a objetos tales como son el diagrama de clases y el diagrama de paquetes. Además, en este capítulo también se establece la arquitectura que ha de seguir la etapa de software.

De manera subsecuente, en el tercer capítulo se realiza la implementación del sistema. Es decir, se presentan temas correspondientes a la codificación, cableado y comunicación entre las etapas del proyecto. Específicamente, en la etapa de hardware se explica el cableado del microcontrolador y el desarrollo del programa que en éste se almacena para poder obtener la lectura del sensor de temperatura, procesarla y enviarla al puerto USB del servidor.
Y en la parte de software se exponen temas relacionados al desarrollo de la aplicación Web, tales como la administración de sus dependencias, la implementación de la interfaz de usuario, la integración de frameworks y su codificación, así como de la obtención de datos a partir del puerto USB.

Concretamente, en el cuarto capítulo se presenta la integración de las dos etapas del proyecto. Aquí se verifica que el diseño conceptualizado en el segundo capítulo es el adecuado, que la comunicación a través del puerto USB trabaja adecuadamente y también se corrobora el correcto funcionamiento del sistema conceptualizándolo a partir de este momento como un todo.

Posteriormente, en la sección de conclusiones y recomendaciones se realiza un recuento de los logros alcanzados y experiencias obtenidas a lo largo del desarrollo del proyecto, permitiendo así mencionar otras posibles aplicaciones del sistema, así como modificaciones o adaptaciones, e incluso la integración de otros nuevos dispositivos.

Finalmente, en las secciones de apéndices se muestran guías o pasos a seguir indispensables para el desarrollo del sistema, como la instalación del driver USB CDC en el sistema operativo Windows, la creación de un proyecto Web dinámico en Eclipse. También se encuentra disponible el código fuente perteneciente tanto a la etapa de hardware como a la de software del proyecto.

Descripción del problema

En la actualidad existe la necesidad de monitorear la temperatura de lugares, dispositivos y de muchos otros elementos constantemente; tales como invernaderos, equipos industriales, sites de cómputo, cultivos de bacterias en investigación, contenedores de alimentos, etc., que para un óptimo funcionamiento requieren estar dentro de un rango de temperatura adecuado, y que en caso de que esto no suceda se pueda realizar alguna acción correctiva. Esto involucra la presencia permanente de personal que realice la función de observar el comportamiento de la temperatura y de tomar medidas en caso de que un evento extraordinario suceda.

Lo anterior se puede resolver con la implementación de sistemas de enfriamiento existentes; sin embargo, éstos no ofrecen la posibilidad de ser monitoreados o controlados a distancia, mucho menos desde nuestra laptop, celular, tablet o algún otro dispositivo conectado a internet o a una red de área local.

Dado lo anterior, el presente trabajo propone una solución a este tipo de problemas y con el beneficio de las posibilidades antes mencionadas, a través de la creación de un sistema compuesto por una etapa de software y otra de hardware.

La etapa de software interactuará con el usuario final y aprovechará las conexiones existentes a través del protocolo HTTP, pues estará basada en una aplicación Web, haciendo uso de tecnologías y frameworks actuales tales como Spring MVC, Maven, Tiles, Ajax, HTML y Tomcat.

La parte de hardware estará compuesta principalmente por un microcontrolador que se encargará de obtener la lectura de un sensor de temperatura, y de activar o desactivar un ventilador de enfriamiento para mantener una temperatura constante, si es que el usuario así lo desea. El microcontrolador se comunicará con el servidor en donde se encuentra desplegada la aplicación Web, a través de su puerto USB.

Justificación

El monitoreo y/o control de la temperatura de un lugar, dispositivo o algún otro elemento de manera remota, solamente plantea una de las infinitas posibilidades que se pueden explotar para interactuar con ellos a distancia y con el medio que les rodea. Esto nos otorga una gran ventaja, pues ya no será necesario estar presente en el lugar de interés, suprimiendo así tiempos y costos de traslado.

Cabe mencionar que no se considera que la solución planteada sea más barata o mejor que una ya existente comercialmente, pero que si puede adecuarse a los requerimientos o modificaciones necesarias, pues se basa en tecnologías que son mucho más accesibles, flexibles y no propietarias.

Además, se espera que este trabajo ayude a promover este tipo de proyectos en la comunidad universitaria y que sirva como guía o base para el desarrollo de otros similares.

Objetivo general

Aplicar los conocimientos adquiridos durante la formación profesional para desarrollar una aplicación que permita monitorear la temperatura de un lugar, dispositivo o algún otro elemento de manera remota a través de una red de área local o Internet.

Alcances y limitaciones

El presente trabajo se limitará al desarrollo de una aplicación Web que interaccionará con un microcontrolador a través del puerto USB del servidor en donde ésta se encuentre desplegada, a fin de monitorear la temperatura de un lugar, dispositivo o algún otro elemento, incluso con la posibilidad de habilitar un ventilador y así mantener una temperatura constante.

No se considerarán temas tales como seguridad, pruebas de stress y/o volumen, sesiones múltiples, etc., que si bien son importantes, también quedan fuera del alcance del objetivo que se pretende alcanzar.

Metodología

El presente trabajo será dividido en dos etapas, una de software y otra de hardware.

Para llevar a cabo la etapa de software, debido a que será desarrollado por una sola persona y no por un equipo de trabajo y a que se trata de un proyecto pequeño y no se cuenta con un cliente como tal, me basaré en los fundamentos de la *metodología SCRUM* a fin de asegurar la obtención de un producto confiable y de calidad.

Y en cuanto a la etapa de hardware, he optado por implementar una metodología basada en mi experiencia en el desarrollo de proyectos, la cual se encuentra dividida en las siguientes fases:

- Adquirir y documentar los requerimientos.
- Recopilación de información (Definir las características de cada una de las tecnologías a utilizar para el desarrollo del proyecto).
- División del proyecto en módulos, a fin de tener control sobre el cronograma de desarrollo.
- Análisis y diseño de los módulos a construir.
- Programación, ensamble y pruebas de los módulos.

Al llegar a este punto, se hará la integración de las dos etapas y se continuará con las siguientes fases:

- Pruebas completas del sistema.
- Análisis de resultados obtenidos e identificación de futuras mejoras.
- Conclusiones.

Capítulo 1. Fundamentos conceptuales

El objetivo de este proyecto consiste en la creación de una aplicación que permita monitorear la temperatura de un lugar de manera remota, por lo que se ha optado en dividirlo en dos etapas: una de Software y otra de Hardware. Esto no quiere decir que la etapa de Software este compuesta únicamente por programas, ni tampoco que la de Hardware no incluya algún componente que requiera de codificación, si no que se ha dividido de acuerdo a la funcionalidad que realizada cada una ellas.

La etapa de Hardware está compuesta por un microcontrolador que se encargará de interactuar con el medio que le rodea, a través de sus puertos de entrada y salida. Éste recibirá la lectura analógica de temperatura realizada por un sensor, y la convertirá a una escala entendible y estandarizada para que pueda ser interpretada por el ser humano. También responderá a órdenes emitidas por el usuario (provenientes de la etapa de Software), traduciéndolas en este caso en un encendido/apagado de un ventilador.

La etapa de Software agrupa a todos aquellos elementos que forman parte de la aplicación Web, y es la que interactuará con el usuario final, presentando a través de una pantalla la temperatura obtenida por el microcontrolador. También permitirá al usuario desde sus pantallas, habilitar o deshabilitar a aquellos componentes que estén conectados a los puertos del microcontrolador.

Dado lo anterior, es importante tener una comprensión de las etapas anteriormente mencionadas, por lo que a lo largo de este capítulo se presentará una descripción de los componentes y conceptos más importantes que pertenecen a cada una de ellas.

1.1 Hardware

1.1.1 Microcontroladores

Palacios, Remiro & López (2003, p.1) indican que “Un microcontrolador es un circuito integrado programable que contiene todos los componente necesarios para controlar el funcionamiento de una tarea determinada”.

Un microcontrolador es un circuito integrado programable capaz de llevar a cabo procesos lógicos y que cuenta con características muy similares a aquellos que se encuentran en una computadora personal estándar, pero con la diferencia de que los microcontroladores son unidades autosuficientes y mucho más económicas (figura 1.1). Se emplean para controlar el funcionamiento de una tarea determinada y se encuentran constituidos principalmente por un microprocesador o unidad de procesamiento central (CPU), una memoria para almacenar el programa (Program Memory), una memoria para almacenar datos (RAM), puertos de entrada y salida (I/O Ports), timers, e incluso algunos cuentan con periféricos integrados, tales como conversores analógico-digital (A/D Converter) y conversores digital-analógico (D/A Converter), entre otros.

A diferencia de un sistema microprocesador convencional (tal como una PC), que tiene chips separados en una tarjeta de circuito impreso, el microcontrolador contiene todos éstos elementos en un solo chip (P. Bates, 2008).

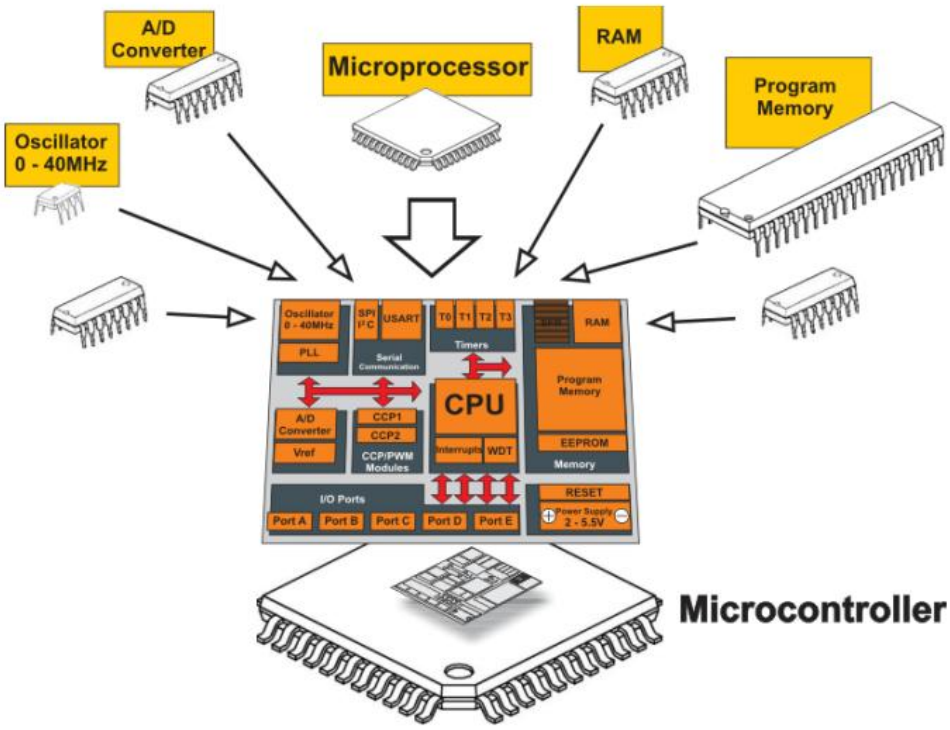


Figura 1.1 Componentes de los microcontroladores.
Fuente:[Verle, 2008]

Su funcionamiento está determinado por un único programa que se encuentra almacenado en su memoria (escrito comúnmente en lenguaje Ensamblador, aunque en la actualidad puede ser escrito en distintos leguajes de programación) y que puede ser reprogramado en repetidas ocasiones. Frecuentemente se emplea la notación µC o las siglas MCU (microcontroller unit) para referirse a los microcontroladores.

Actualmente existen varios fabricantes dedicados a la producción de familias enteras de microcontroladores, como: Atmel, NXP Semiconductor (antes Phillips), Microchip, Holtek, Intel, Freescale (antes Motorola), Renesas (antes Hitachi, Mitsubishi y NEC), STMicroelectronics, Texas Instuments, Zilog, Parallax, entre otros.

Sin embargo, de entre toda la gama existente de microcontroladores, los de Microchip son considerados como unos de los más aceptados entre aficionados y profesionales, debido a su diseño simple, facilidad de programación y a que su software de desarrollo puede ser descargado de manera gratuita, o bien a través de versiones de prueba, sin mencionar que existen toneladas de documentación acerca de ellos a lo largo y ancho de Internet.

1.1.2 Diferencias entre microprocesador y microcontrolador

Un **microcontrolador** es un sistema cerrado que contiene un computador completo y de prestaciones limitadas que no se pueden modificar (Ángulo y Ángulo, 2003).

Un **microprocesador** es un sistema abierto con el que puede construirse un computador con las características que se desee, acoplándole los módulos necesarios (Ángulo et al., 2003).

A partir de la definiciones anteriores, podemos inferir que a diferencia del microprocesador, un microcontrolador contiene todos los elementos electrónicos necesarios para hacer funcionar un sistema basado en microprocesador pero en un solo circuito integrado; es decir contiene el CPU, memoria RAM, memoria ROM, puertos de entrada y salida, así como otros periféricos, con la consiguiente reducción de espacio. Además, debido a su reducido tamaño es posible montar el controlador en el propio dispositivo que gobierna. De esta forma el controlador recibe el nombre de controlador empotrado o embebido (embedded controller).

1.1.3 Los microcontroladores en la actualidad

Actualmente se dice que los microcontroladores han conquistado el mundo, dado a que se encuentran presentes en todas partes; prácticamente en todos y cada uno de los dispositivos electrónicos que nos rodean. Simplemente basta con revisar a fondo nuestro automóvil, teléfono celular, pantalla de TV, cámara de video, cámara fotográfica, control remoto, horno de microondas, motocicleta, etc., para darnos cuenta de la importancia que éstos tienen en nuestra vida cotidiana.

Su uso es tan común, que algunos fabricantes de microcontroladores superan por mucho el millón de unidades producidas en una sola semana de un solo modelo de microcontrolador. Este dato nos da una idea de la masiva influencia de estos componentes.

En cuanto a las técnicas de fabricación, cabe mencionar que prácticamente la totalidad de los microcontroladores actuales se fabrican con tecnología CMOS (Complementary Metal Oxide Semiconductor), pues esta tecnología supera a las técnicas anteriores por su bajo consumo y alta inmunidad al ruido.

1.1.4 Aplicaciones de los microcontroladores

Cada vez existen más productos que incorporan un microcontrolador (especialmente de 8 y 16 bits) con el fin de aumentar sustancialmente sus prestaciones, reducir su tamaño y costo, mejorar su fiabilidad y disminuir el consumo de energía. Entre sus aplicaciones más habituales podemos mencionar las siguientes:

- **Sistemas de comunicación**: en grandes automatismos como centrales y en teléfonos fijos, móviles, fax, etc.
- **Electrodomésticos**: lavadoras, hornos, frigoríficos, lavavajillas, batidoras, televisores, vídeos, reproductores DVD, equipos de música, mandos a distancia, consolas, etc.
- **Industria informática**: se encuentran en casi todos los periféricos como ratones, teclados, impresoras, escáneres, etc.
- **Automoción**: climatización, seguridad, ABS, etc.
- **Industria**: Autómatas, control de procesos, etc.

- **Sistemas de supervisión, vigilancia y alarma**: ascensores, calefacción, aire acondicionado, alarmas de incendio, robo, etc.
- **Otros**: Instrumentación, electro-medicina, tarjetas (smartcard), sistemas de navegación, etc.

También es importante indicar, que en la actualidad los modernos microcontroladores de 32-bits van afianzando posiciones en el mercado, especialmente en tareas más complejas tales como el procesamiento de imágenes, comunicaciones, aplicaciones militares, procesos industriales y control de dispositivos de almacenamiento masivo de datos (Toboso, 2013).

1.1.5 Clasificación de microcontroladores de acuerdo a la longitud de su bus de datos

Actualmente la longitud del bus de datos de los microcontroladores varía entre 8, 16 y 32 bits; entre mayor sea su longitud, mayor será la eficiencia del microcontrolador al realizar operaciones con datos más grandes. Aunque también aumentará su complejidad y costo.
Por ejemplo, para los dispositivos multimedia que procesan datos de vídeo y audio, un bus de datos de 8-bits sería insuficiente.
Aunque las prestaciones de los microcontroladores de 16 y 32-bits son superiores a los de 8-bits, la realidad es que los microcontroladores de 8-bits dominan el mercado actual. La razón de esta tendencia es que los microcontroladores de 8-bits son apropiados para la gran mayoría de las aplicaciones, lo que hace absurdo emplear micros más potentes y consecuentemente más caros. Uno de los principales y más exigentes consumidores del mercado del microcontrolador es el sector automovilístico, dado a que sus componentes electrónicos deben operar bajo condiciones extremas de vibraciones, choques, ruido, etc. y seguir siendo fiables, pues el fallo en cualquiera de ellos podría dar origen a un accidente. De hecho, algunas de las familias de microcontroladores actuales se desarrollaron pensando en este sector, siendo modificadas posteriormente para adaptarse a sistemas más genéricos. (Toboso, 2013).

1.1.6 Clasificación de microcontroladores de acuerdo a su set de instrucciones

Set de instrucciones CISC (Complex Instruction Set Computer)

El set de instrucciones CISC es inherente a los primeros microcontroladores que aparecieron en el mundo, los cuales estaban inspirados en los procesadores de los grandes computadores de la época. Es complejo porque consta de muchas instrucciones, complejas y difíciles de recordar a la hora de programar en lenguaje ensamblador. Además, al crecer el número de instrucciones también crecerán los códigos de las instrucciones, lo cual deriva en una reducción de la eficiencia del microcontrolador.

Set de instrucciones RISC (Reduced Instruction Set Computer)

Estos microcontroladores cuentan con instrucciones más sencillas y en un número mucho menor. Ello permite que la programación en ensamblador sea una labor cómoda y al alcance de todos. Sin embargo, cuando se desarrollan proyectos mucho más complejos, el uso del lenguaje ensamblador se torna cada vez más engorroso. Entonces es preferible optar por compiladores de alto nivel, para los cuales un set RISC no es obstáculo. Además, la rapidez y sencillez de sus instrucciones permiten optimizar el hardware y el software del procesador.

Set de instrucciones SISC (Specific Instruction Set Computer)

El set de instrucciones SISC es usado por microcontroladores destinados a aplicaciones muy concretas. Su juego de instrucciones, además de ser reducido, es "específico", o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista.

1.1.7 Clasificación de Microcontroladores de acuerdo a su arquitectura interna

Microcontroladores con Arquitectura de Von Neumann

Los microcontroladores que se encuentran bajo está arquitectura (Figura 1.2) tienen una memoria única, que constituye tanto el segmento de memoria de programa como el de datos. Con un solo bus de comunicación entre dicha memoria y el procesador no es posible realizar diversos accesos a la vez.

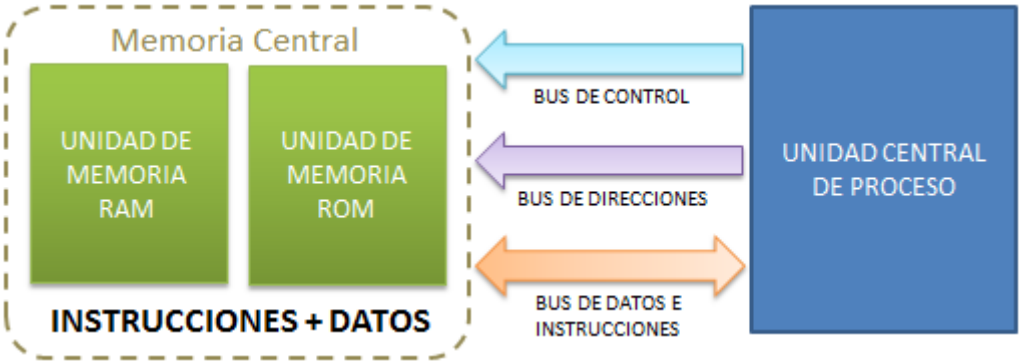


Figura 1.2 Arquitectura Von Neumann
Fuente: [Toboso, 2013]

Microcontroladores con Arquitectura Harvard

En este tipo de arquitectura (Figura 1.3), los microcontroladores disponen de dos memorias, una que contiene el programa y otra para almacenar los datos. De este modo el CPU puede tener acceso simultáneo a ambas memorias utilizando buses diferentes. Más específicamente, el CPU puede leer la siguiente instrucción de programa mientras está procesando los datos de la instrucción actual. Actualmente todos los microcontroladores se inclinan por esta arquitectura.

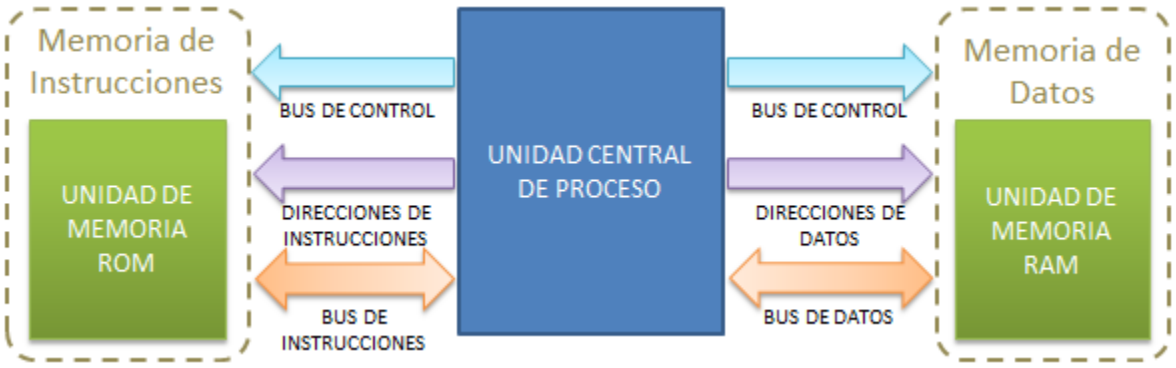


Figura 1.3 Arquitectura Harvard
Fuente: [Toboso, 2013]

1.1.8 Los Microcontroladores PIC de Microchip Technology Inc.

Los PIC fabricados por Microchip Technology Inc. son microcontroladores tipo RISC que han derivado del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instrument. Su nombre actual no es un acrónimo; en realidad el nombre completo es PICmicro, aunque generalmente se utiliza como Peripheral Interface Controller (Controlador de Interfaz Periférico).

En 1985 la división de microelectrónica de General Instrument se separó como compañía independiente y quedó incorporada como filial, hasta que el 14 de diciembre de 1987 cambió su nombre a Microchip Technology y en 1989 fue adquirida por un grupo de inversionistas. Para entonces, el PIC se mejoró con memoria EPROM a fin de obtener un controlador de canal programable (Wilmshurst, 2010).

Actualmente, Microchip ofrece un extenso portafolio de productos en el que se incluye a los microcontroladores PIC de **8, 16** y **32-bits**, agrupados de acuerdo al ancho de su **memoria de datos**.

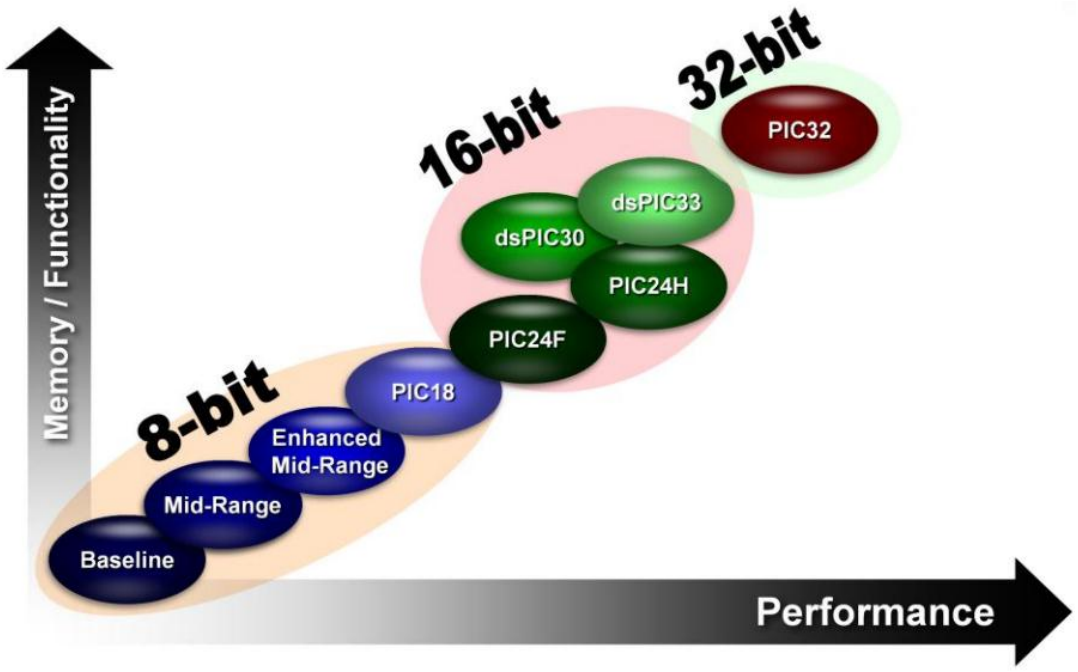


Figura 1.4 Guía de las familias de MCU PIC y dsPIC DSC
Fuente:[Castro, 2004]

Dentro de éstas agrupaciones, destacan varias arquitecturas que a su vez están compuestas por miembros de diferentes familias de PIC MCUs y dsPIC DSCs. Estas arquitecturas se clasifican de acuerdo al ancho de su **memoria de programa** (Figura 1.5) y otras características.

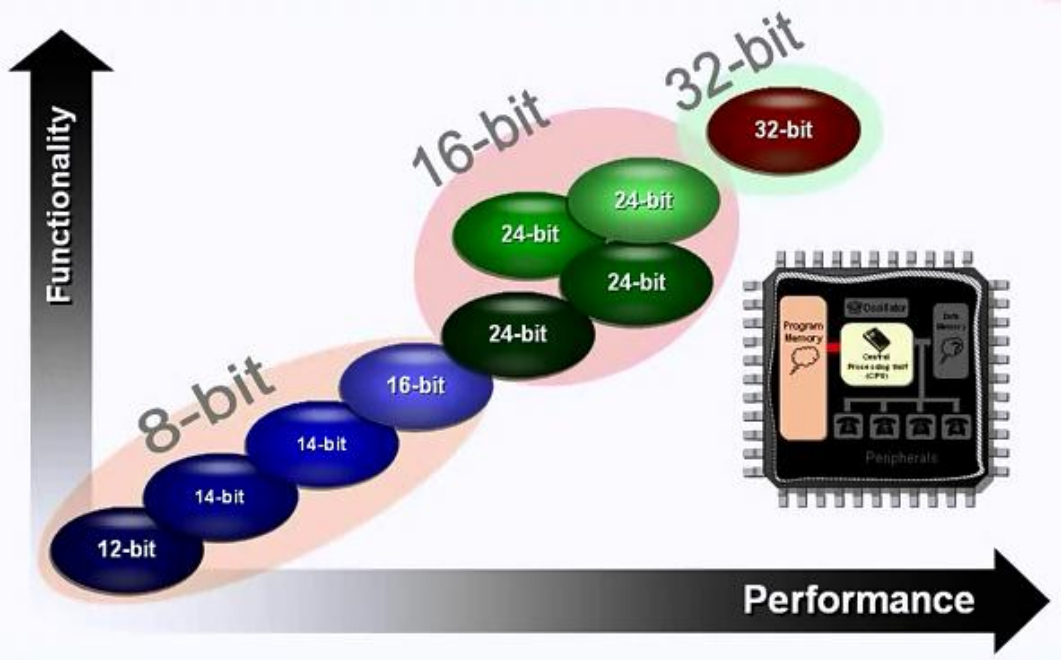


Figura 1.5 Guía de las familias de MCU PIC y dsPIC DSC
Fuente:[Microchip - PIC MCUs, 2004]

A partir de la Figura 1.4 y la Figura 1.5, podemos determinar que la arquitectura Baseline cuenta con una memoria de programa con un ancho de 12-bits, la Mid-Range y la Enhanced Mid-Range con una de 14-bits, la PIC18 con una de 16-bits y así sucesivamente (Castro, 2004).

También podemos observar que entre más nos dirijamos hacia arriba y a la derecha de las gráficas, los productos tendrán más memoria, un mejor desempeño y funcionalidades más avanzadas.

En el desarrollo de este proyecto se hará uso del microcontrolador PIC18F4550 de Microchip, pues pertenece a la serie de alto desempeño y bajo costo de los PIC18. Su

firmware reside en una memoria flash de 32 Kb y cuenta con 2 Kb de memoria RAM y 256 bytes de memoria EEPROM. También, el chip cuenta con 34 pines de Entrada/Salida que incluyen un convertidor análogo/digital de 10-bits, un puerto USART, un puerto serial síncrono que puede configurarse para utilizar I2C o SPI, capacidades PWM mejoradas y dos comparadores analógicos. Además, el modulo USB y el CPU pueden hacer uso de fuentes de reloj separadas, permitiendo que el CPU haga uso de un reloj más lento para ahorro de energía. También, es importante señalar que con ayuda de un programa bootloader se puede actualizar su firmware a través de su puerto USB.

Dado lo anterior, se presentará en los capítulos siguientes más información acerca de los **PIC de 8-bits**, específicamente de aquellos que cuentan con una arquitectura **PIC18**.

1.1.9 Selección de un microcontrolador

En la actualidad cada fabricante de microcontroladores oferta un elevado número de diferentes modelos, desde los más sencillos hasta los más poderosos; pues si sólo se dispusiese de un solo modelo de microcontrolador, éste debería contar con recursos lo suficientemente potentes para poder adaptarse a las exigencias de diferentes aplicaciones. Este exceso de potencia en la mayoría de los casos, supondría un despilfarro. Por ello, uno de los aspectos más importantes durante el diseño de un proyecto, es la selección del microcontrolador a utilizar, pues es posible variar sus características, tales como la capacidad de su memoria, el número de puertos de E/S, la cantidad y potencia de los elementos auxiliares, la velocidad de funcionamiento, etc (Ibrahim, 2008).

Para llevar a cabo esta tarea, Microchip pone a nuestra disposición un par de páginas (Figuras 1.6 y 1.7) donde podemos elegir el microcontrolador más adecuado a nuestras necesidades.

Product Selector Tool

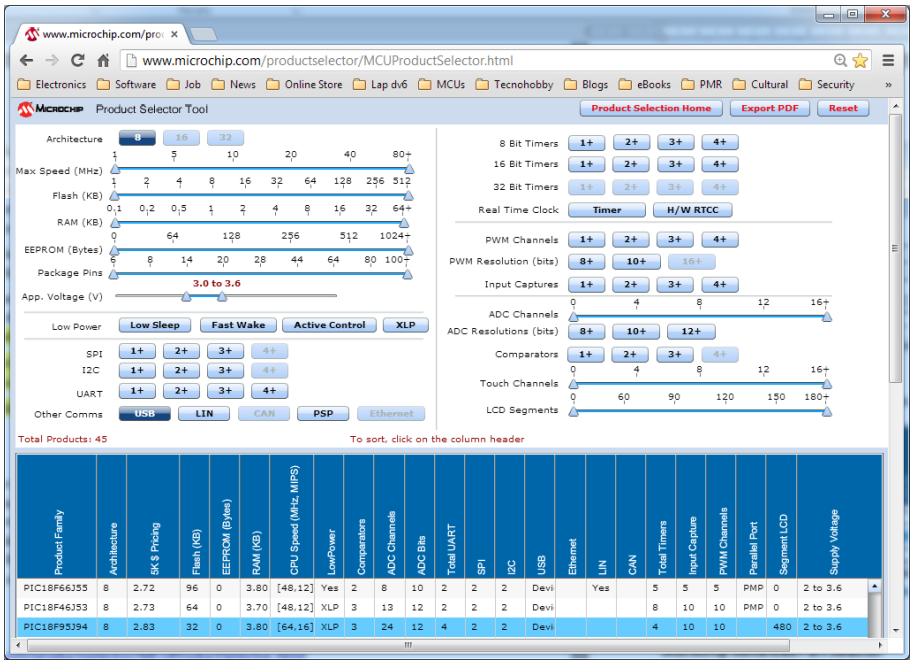


Figura 1.6 Selector de producto on-line de Microchip
Fuente: [Microchip - Product Selector, 2013]

Microchip Advanced Part Selector

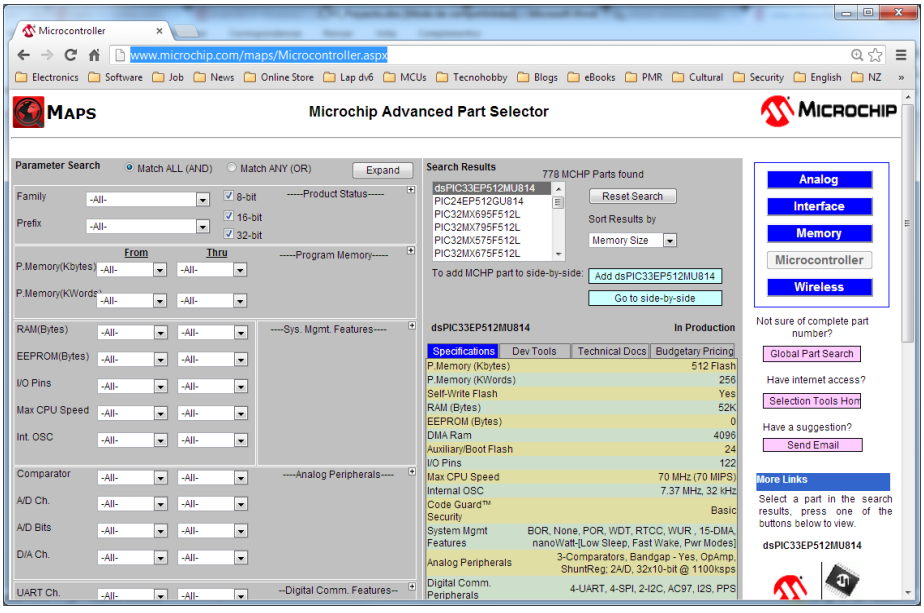


Figura 1.7 Selector avanzado de producto on-line de Microchip
Fuente: [Microchip - Part Selector, 2013]

1.1.10 Microcontroladores PIC de 8-Bits

Los microcontroladores de Microchip han fascinado al mundo entero durante los últimos años, debido principalmente a su facilidad de uso, comodidad y rapidez para el desarrollo de aplicaciones; lo que le han permitido ganar terreno en el mercado de los microcontroladores a nivel mundial, hasta convertirse en los microcontroladores de 8-bits más vendidos en la actualidad (CursoMicros, 2012).

Su grupo de microcontroladores de 8-bits está encuentra integrado por las familias PIC10, PIC12, PIC16 y PIC18, en dónde cada familia cuenta con prestaciones diferentes para adaptarse mejor a las necesidades de nuestros proyectos. Figura 1.8.

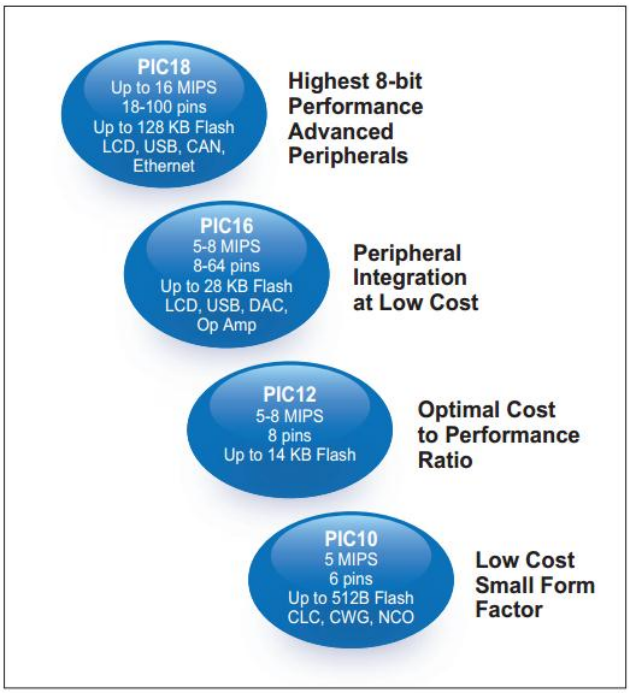


Figura 1.8 Familias de MCUs PIC de 8-bits
Fuente:[Microchip Website, 2012]

La compatibilidad tanto de código como de pines que existe entre los MCUs PIC de 8-bit nos permite migrar fácilmente de un microcontrolador a otro, y también nos permite reaccionar rápidamente ante los cambios de requerimientos y de diseño, mejorando a la vez sus

prestaciones. Esto, maximiza su reutilización en futuros desarrollos y preserva la inversión realizada en hardware, software y otras herramientas (Microchip Website, 2012).

La Figura 1.9 muestra la relación existente entre las arquitecturas (Baseline, Mid-Range, Enhanced Mid-Range y PIC18) de MCUs PIC de 8-bits con las familias (PIC10, PIC12, PIC16 y PIC18) que las integran.

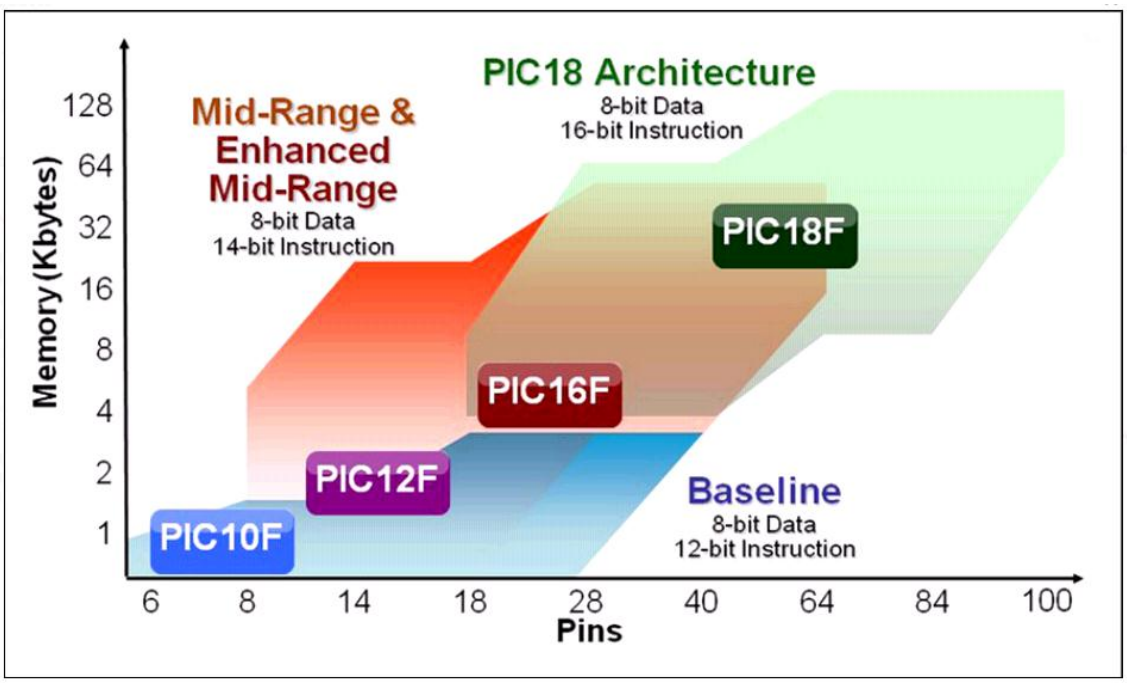


Figura 1.9 Relación entre arquitecturas de MCUs PIC de 8-bits y sus familias
Fuente:[Microchip Website, 2012]

En lo que respecta a un mercado en específico, los MCUs PIC de 8-bits son adecuados para una amplia variedad de aplicaciones, desde el control lógico hasta sistemas completamente integrados que involucren el uso de USB, Ethernet, CAN, LCD y Touch Sensing; básicamente se pueden utilizar con cualquier cosa que posea un sensor, una interfaz de usuario, control, display, etc., al que se requiera agregar un poco de “inteligencia” (Microchip – Product guide, 2012).

Es vasto el número de aplicaciones que hoy en día hacen uso de los MCUs PIC de 8-bit, como: electrodomésticos (electrodomésticos de energía inteligentes, refrigeradores y licuadoras, etc.); electrónica de consumo (cargadores de teléfonos, máquinas de afeitar eléctricas y aspiradoras, etc.); sistemas industriales (cerraduras electrónicas, seguimiento de carga, iluminación, etc.); automotor (como encendido eléctrico sin llave, asientos eléctricos y puertas de garaje, etc.), dispositivos médicos (como los dispositivos de diagnóstico y medidores médicos portátiles), ¡y la lista continúa!.

En la Tabla 1.1 se muestra la comparativa entre las Arquitecturas de los microcontroladores de Microchip de 8-bits, donde se puede apreciar que entre más lejos se encuentre de la línea base, más características y mejores prestaciones tendrá.

Tabla 1.1 Comparativa entre las Arquitecturas de 8-bits

	Arquitectura Baseline	**Arquitectura Mid-range**	**Arquitectura Enhanced Mid-range**	**Arquitectura PIC 18**
Número de Pines	6-40	8-64	8-64	18-100
Interrupciones	No	Capacidad para una sola interrupción	Capacidad para una sola interrupción con almacenaje de contexto por hardware	Capacidad para múltiples interrupciones con almacenaje de contexto por hardware
Desempeño	5 MIPS	5 MIPS	8 MIPS	Hasta 16 MIPS
Instrucciones	33, 12-bits	35, 14-bits	49, 14-bits	83, 16-bits
Memoria de Programa	Hasta 3 Kb	Hasta 14 Kb	Hasta 28 Kb	Hasta 128 Kb
Memoria de Datos	Hasta 138 Bytes	Hasta 368 Bytes	Hasta 1.5 Kb	Hasta 4 Kb
Pila por Hardware	2 niveles	8 niveles	16 niveles	32 niveles
Características	• Comparador • ADC de 8-bit • Memoria de Datos • Oscilador interno	Además de las características de la Baseline: • SPI/I2C • UART • PWMs • LCD • ADC de 10-bit • Amp Op	Además de las características de la Mid-Range: • Comunicación múltiple con periféricos • Espacio de programación lineal • PWMs con tiempo base independiente	Además de las características de la Enhanced Mid-Range: • Multiplicador hardware 8x8 • CAN • CTMU • USB • Ethernet • ADC de 12-bit
Número total de dispositivos	16	58	29	193
Familias	PIC10, PIC12, PIC16	PIC12, PIC16	PIC12FXXX, PIC16F1XX	PIC18

Arquitectura Baseline. Presentan un set de 33 instrucciones de 12-bits fácil de usar para un rápido desarrollo y operan a 5 MIPS. Tienen muy poca memoria y en cuanto a recursos periféricos, son los menos dotados. Sin embargo, las familias que poseen esta arquitectura (PIC10, PIC12 y PIC16) han sido durante mucho tiempo las favoritas por ingenieros alrededor del mundo, pues se utilizan en una amplia gama de aplicaciones. Cuentan con la cantidad exacta de características y opciones para reducir al mínimo los costos y realizar un buen trabajo. Figura 1.10

Sus principales características son las siguientes:

- Set de 33 instrucciones de **12-bit (program word)**
- 2K palabra (3 KB) de memoria de programa direccionable
- 144 bytes de memoria RAM (máximo)
- 2 niveles de pila por hardware
- 1 FSR (File Select Register) de 8 bits
- Múltiples opciones de productos y fácil migración entre ellos
- Forma más pequeña disponible factores

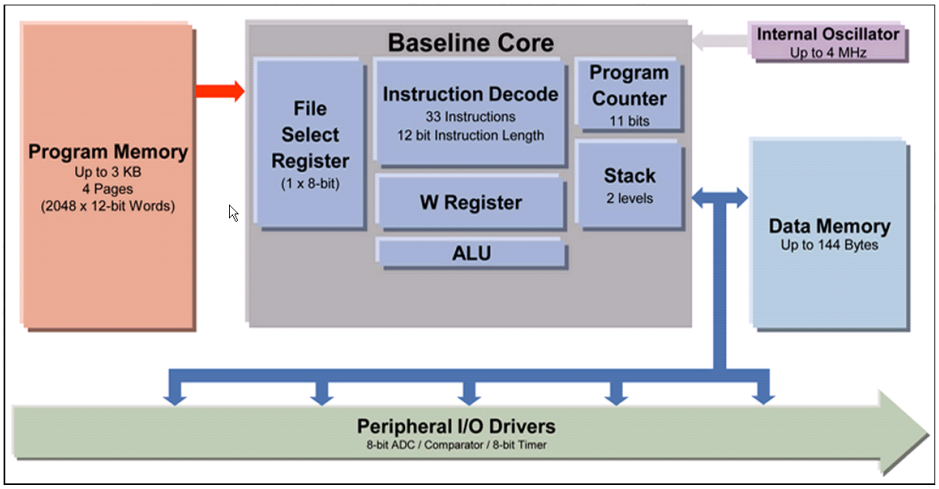


Figura 1.10 Arquitectura Baseline (8-bits)
Fuente:[Microchip Website, 2012]

Arquitectura Mid-Range. Es la arquitectura más distintiva de los PICs. Cuentan con un set de 35 instrucciones de 14-bits y no supera los 5 MIPS. En general, entre los MCUs que poseen esta arquitectura se pueden encontrar casi todos los recursos hardware que se

buscan en un microcontrolador de 8-bits, por eso se suele tomar de aquí algunos modelos como punto de partida de aprendizaje. Son ricos en dispositivos periféricos, por lo que se consideran ideales para aplicaciones multi-dimensionales que requieren de un mayor nivel de control embebido. Figura 1.11

Sus principales características son las siguientes:

- Set de 35 instrucciones de **14-bit (program word)**
- 8K palabra (14 KB) de memoria de programa direccionable
- 46 bytes de memoria RAM (máximo)
- 8 niveles de pila por hardware
- 1 FSR (File Select Register) de 9 bits
- Manejo de interrupciones por hardware
- Conjunto de características altamente integradas, tales como EEPROM, LCD, soluciones de sensores mTouch ™ y comunicación serial.

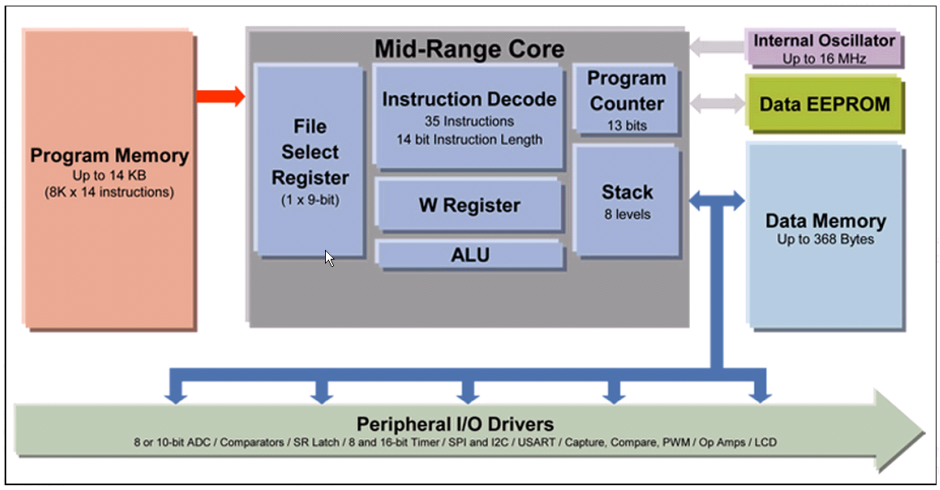


Figura 1.11 Arquitectura Mid-Range (8-bits)
Fuente:[Microchip Website, 2012]

Arquitectura Enhanced Mid-Range. Para un mejor trabajo con los compiladores de alto nivel, su repertorio básico consta de 49 instrucciones de 14-bits, y algunos modelos incluyen un pequeño conjunto de instrucciones extendidas.

Se basa en los mejores elementos core de la arquitectura Mid-range y proporciona un rendimiento adicional, mientras mantiene compatibilidad con las 35 instrucciones de la

arquitectura Mid-range. Alcanzan velocidades de operación de hasta 8 MIPS y están provistos de un modelo de memoria plana, con lo que se ahorran las tediosas operaciones de cambio de banco. Figura 1.12

Sus principales características son las siguientes:

- Set de 49 instrucciones de **14-bit (program word)** de fácil aprendizaje
- Palabra 32K (56 KB) de memoria de programa direccionable
- 4KB de memoria RAM (máximo)
- 16 niveles de pila por hardware
- 2 FSR (File Select Register) de 16 bits
- Manejo de interrupciones por hardware con guardado de contenido
- Conjunto de características avanzadas, múltiples comunicaciones seriales y capacidad para el control de motores

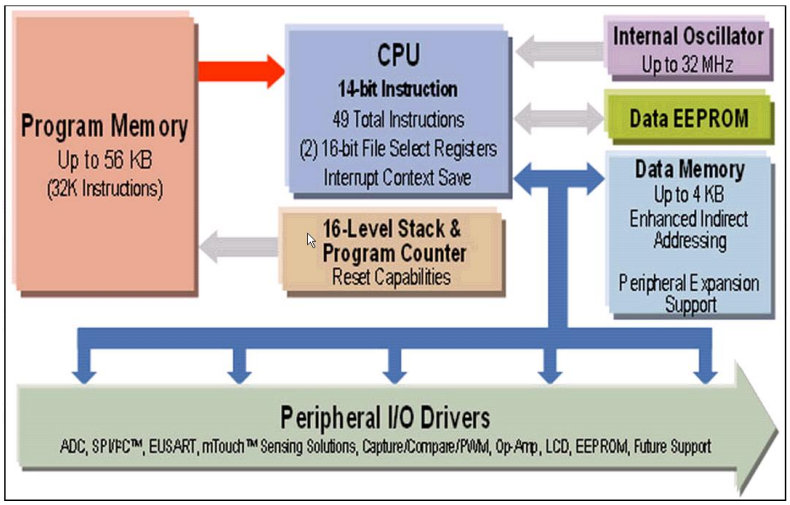


Figura 1.12 Arquitectura Enhanced Mid-Range (8-bits)
Fuente:[Microchip Website, 2012]

Arquitectura PIC 18. Combina el máximo nivel de desempeño e integración, junto con facilidad de uso bajo una arquitectura de 8-bits. Con un máximo de 16 MIPS de poder de procesamiento y memoria lineal, los microcontroladores PIC18 cuentan con periféricos avanzados, tales como CAN, USB, Ethernet, LCD y CTMU. Es la arquitectura más popular para nuevos diseños de 8-bits, especialmente si de programar en C se trata. Figura 1.13

Sus principales características son las siguientes:

- Un poderoso set de 83 instrucciones de **16-bit (program word)** optimizado para el lenguaje C
- Hasta 2 MB de memoria de programa direccionable
- USB v2.0 Full-speed de hasta 12 Mbps
- 4KB de memoria RAM (máximo)
- 32 niveles de pila por hardware
- 1 FSR (File Select Register) de 8 bits
- Multiplicador integrado de 8x8 por hardware
- Múltiples interrupciones internas y externas
- Presentan el más alto rendimiento entre las arquitecturas de 8-bit

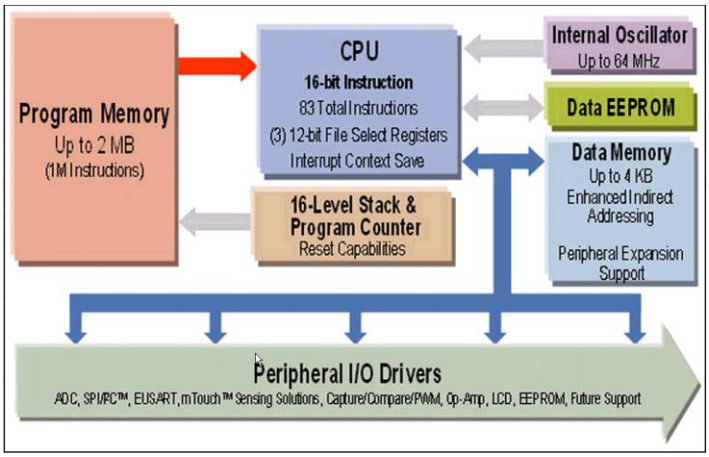


Figura 1.13 Arquitectura PIC18 (8-bits)
Fuente:[Microchip Website, 2012]

1.1.11 Microcontrolador PIC18F4550

Como se menciona en puntos anteriores, el microcontrolador a utilizar durante el desarrollo de este proyecto es el **PIC18F4550**; el cuál fue seleccionado por sus excelentes prestaciones, por el número de puertos I/O que tiene y principalmente porque cuenta con un módulo USB embebido. Además de que éste microcontrolador es uno de los que más se utiliza a nivel mundial para introducir a los estudiantes de ingeniería en el universo de los microcontroladores.

Tabla 1.2 Comparativa del PIC18F4550 y los miembros de su familia
Fuente:[PIC18F2455/2550/4455/4550 Data Sheet]

Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EUSART	Comparators	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

La familia de dispositivos PIC18F2455/2550/4455/4550 (Tabla 1.2), ofrece las ventajas de todos los microcontroladores PIC18, es decir alto desempeño computacional a un precio económico, junto con alta durabilidad y memoria de programa Flash mejorada. Además, el módulo de comunicaciones Universal Serial Bus que tienen incorporado, cumple con la especificación USB revisión 2.0 (Microchip - PIC18F2455/2550/4455/4550, 2009).

En la tabla 1.3 que a continuación se muestra, se pueden observar a mayor detalle las características principales de la familia de éstos dispositivos. En particular, las características del PIC16F4550 se pueden observar en la última columna de la derecha:

Tabla 1.3 Comparativa de las características del PIC18F4550
Fuente:[PIC18F2455/2550/4455/4550 Data Sheet]

Features	PIC18F2455	PIC18F2550	PIC18F4455	PIC18F4550
Operating Frequency	DC – 48 MHz	DC – 48 MHz	DC – 48 MHz	DC – 48 MHz
Program Memory (Bytes)	24576	32768	24576	32768
Program Memory (Instructions)	12288	16384	12288	16384
Data Memory (Bytes)	2048	2048	2048	2048
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	1	1
Enhanced Capture/ Compare/PWM Modules	0	0	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Universal Serial Bus (USB) Module	1	1	1	1
Streaming Parallel Port (SPP)	No	No	Yes	Yes
10-Bit Analog-to-Digital Module	10 Input Channels	10 Input Channels	13 Input Channels	13 Input Channels
Comparators	2	2	2	2
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT
Programmable Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled
Packages	28-Pin PDIP 28-Pin SOIC	28-Pin PDIP 28-Pin SOIC	40-Pin PDIP 44-Pin QFN 44-Pin TQFP	40-Pin PDIP 44-Pin QFN 44-Pin TQFP

Como se puede apreciar, el microcontrolador PIC15F4550 cuenta con 32Kb de memoria de programa, 2Kb de memoria de datos, 256 bytes de memoria EEPROM, 20 fuentes de interrupción, 5 puertos I/O, 4 Timers, 1 módulo CCP, 1 módulo ECCP, comunicación serial, módulo USB, 13 canales de entrada analógica, entre otras prestaciones.

En la figura 1.14, se presenta el esquema de pines del PIC18F4550.

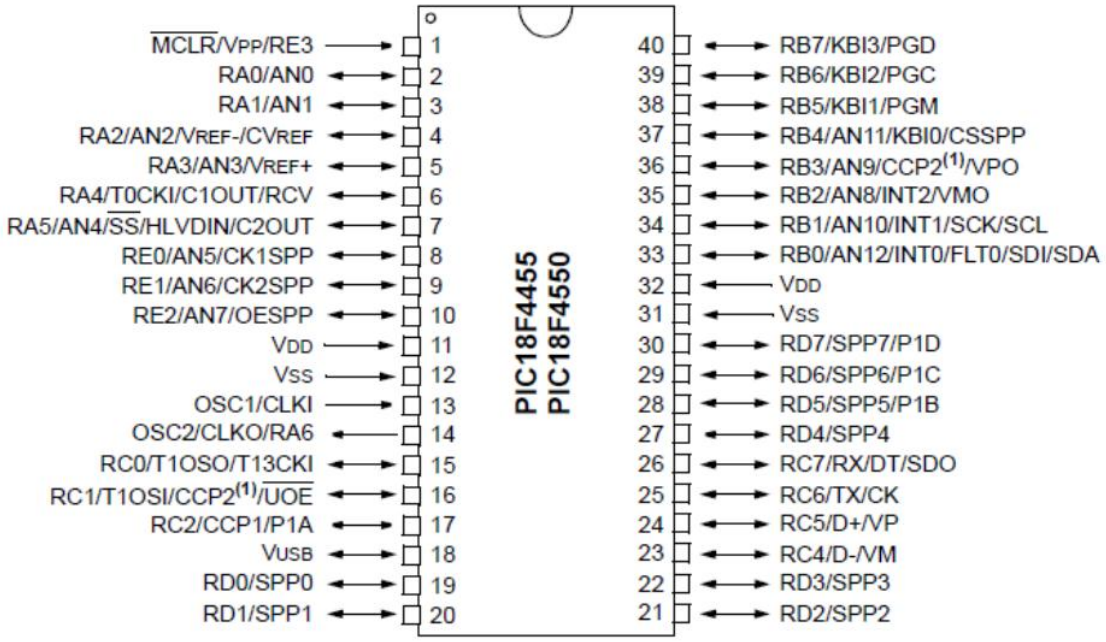


Figura1.14 Esquema de pines del PIC18F4550
Fuente:[Microchip - PIC18F2455/2550/4455/4550, 2009]

Para conocer más acerca de las características y posibles configuraciones de este microcontrolador, es indispensable leer su **hoja de datos**, disponible en el siguiente link: http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf

A continuación en la figura 1.15, se presenta el cableado para conectar el PIC18F4550 con el puerto USB de nuestra computadora (es recomendable colocar un capacitor de desacoplo de 22uF lo más cerca posible de los pines de alimentación):

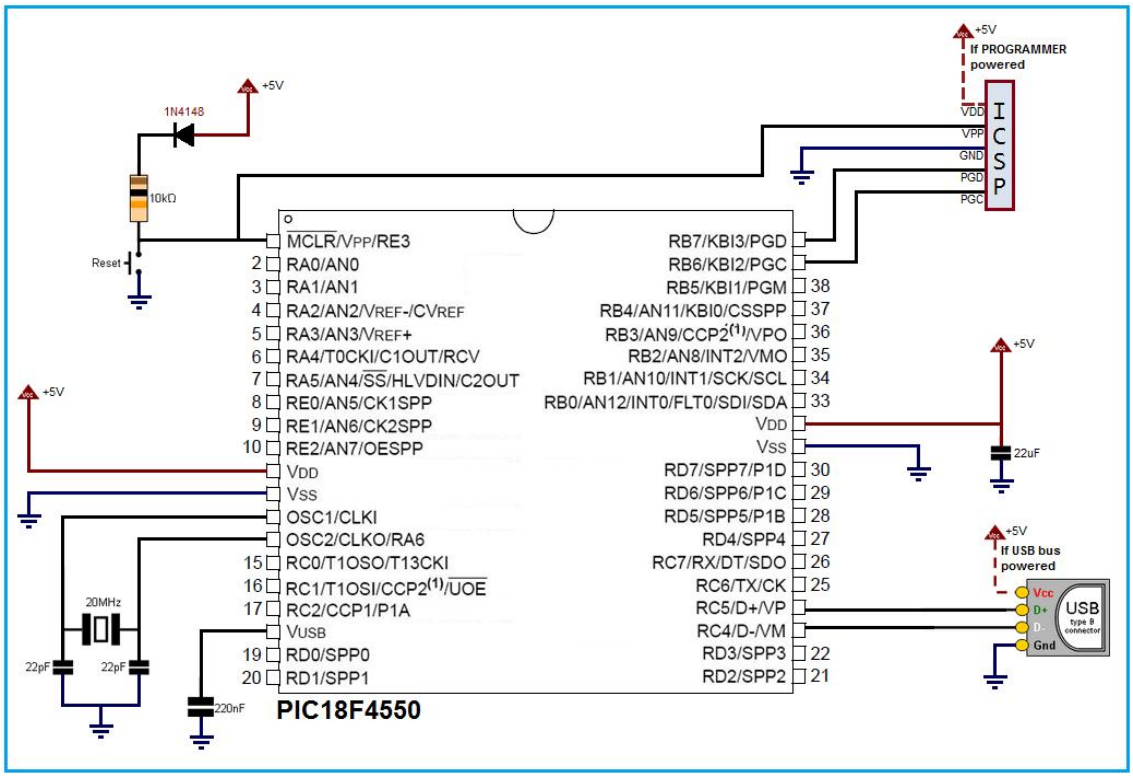


Figura1.15 Esquema de conexión del PIC18F4550
Fuente:[Elaboración propia]

Puesto a que el cableado de este circuito es común en todos los proyectos basados en este microcontrolador (y algunos de otras familias con compatibilidad en pines), se puede integrar dentro de una tarjeta de proyectos recortada (Figura 1.16). Así ya no será necesario cablearlo en cada protoboard donde se utilice; simplemente se conectará y desconectará del sistema.

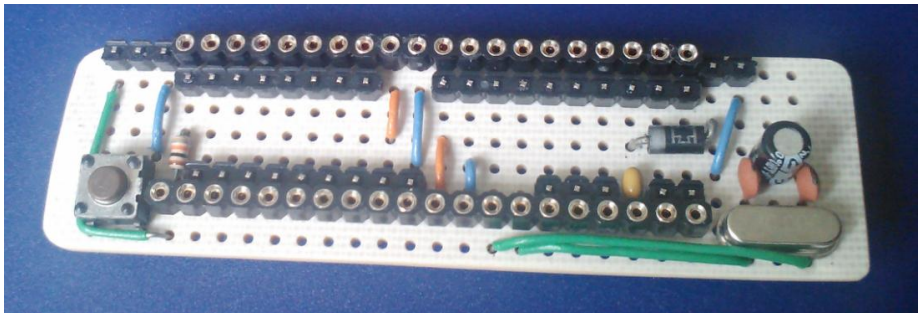


Figura1.16 Tarjeta de proyectos recortada y con componentes integrados
Fuente:[Elaboración propia]

Una vez montado el microcontrolador, luce de acuerdo a la figura 1.17.

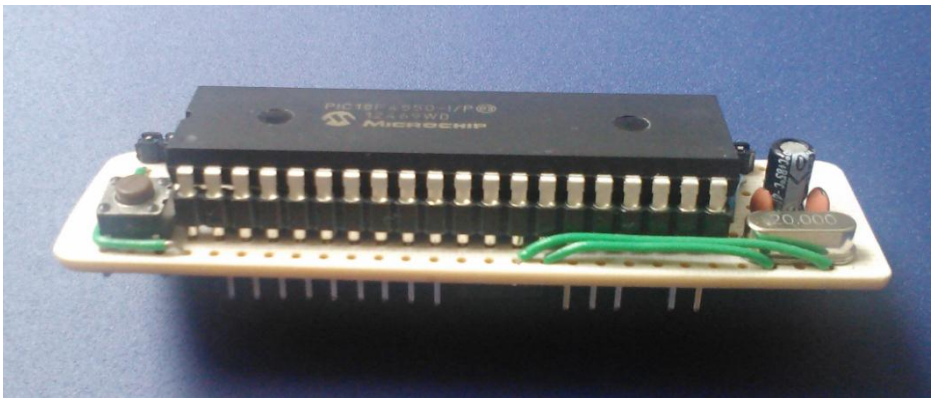


Figura1.17 Tarjeta de proyectos con PIC18F4550 montado
Fuente:[Elaboración propia]

Con la ayuda de esta tarjeta, será posible concentrarse únicamente en las conexiones complementarias del proyecto, en su funcionalidad y especialmente en su codificación.

1.1.12 Programador de PICs (MiniPIC v2.0)

El MiniPIC v2.0 (figura 1.18) es una heramienta de programación y depuración profesional para las familias de microcontroladores PIC y memorias EEPROM (al igual que el PICkit2, PICkit3 y muchos otros). Su conexión USB lo hace compatible con cualquier PC de escritorio o Laptop, y además es compatible con Windows98SE, ME, 2000, XP, Vista, 7 y Linux. El MiniPIC v2.0 puede ser utilizado como:

- Programador
- Debugger
- Fuente de alimentación (2.5V a 5.0V)
- Herramienta lógica
- Analizador lógico
- Herramienta USART

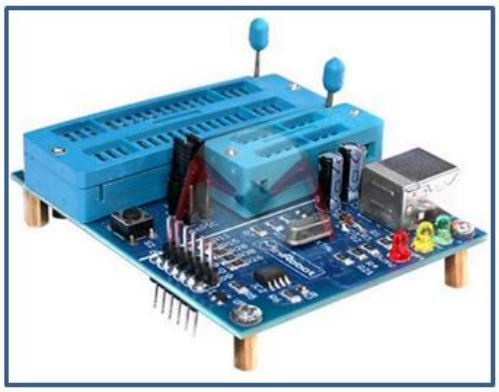


Figura1.18 Programador MiniPIC v2.0 de MiniRobot
Fuente:[http://www.minirobot.com.mx, 2012]

Sin embargo, para que pueda ser usado como programador, es necesario de alguna interfaz de software que se encargue del envío del archivo .hex al microcontrolador, como la interfaz PICKit2 o el MPLAB X IDE.

Programación en Protoboard. Con el MiniPIC v2.0 se puede programar directamente el dispositivo (figura 1.19) sin necesidad de las bases ZIF, conectando únicamente 5 señales: VPP, VDD, PGC, PGD y GND. Esto es realmente sencillo, solo se tiene que hacer coincidir los pines del programador con el pin del dispositivo con el mismo nombre, es decir, el VPP del programador con el VPP del dispositivo, el VDD del programador con el VDD del dispositivo, etc.

Esta forma de programación se conoce como ICSP (In Circuit Serial Programming) y permite programar dispositivos con diferente encapsulado al DIP.

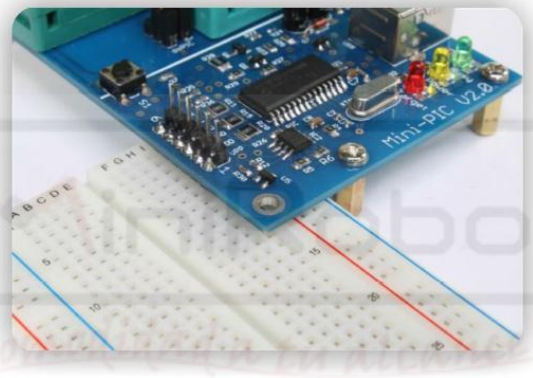


Figura1.19 Programación ICSP en Protoboard con el MiniPIC v2.0
Fuente:[http://www.minirobot.com.mx, 2012]

1.1.13 USB (Universal Serial Bus)

USB es la abreviación de Universal Serial Bus (figura 1.20), y fue creado en los años 90 por una asociación de empresas con la idea de mejorar las lentas interfaces series (RS-232) y paralelo, así como de mejorar las técnicas de plug-and-play, es decir, permitir a los dispositivos conectarse y desconectarse sin la necesidad de que fuesen reiniciados y de ser configurados automáticamente al ser conectados. Además se le dotó de transición de energía eléctrica para los dispositivos conectados.

Figura1.20 Logo Universal Serial Bus
Fuente:[http://www.usb.org, 2013]

Este bus cuenta con una estructura de árbol y se pueden ir conectando dispositivos en cadena, pudiéndose conectar hasta 127 dispositivos periféricos, como, módems, teclados, mouses, monitores, etc., y permitiendo la transferencia síncrona y asíncrona (García, 2008, p.251).

Físicamente los datos del USB se transmiten por un par trenzado (D+ y D-) además de la masa (GND) y alimentación (+5V) (Tambi, 2013).

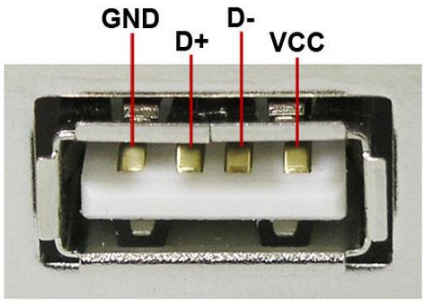
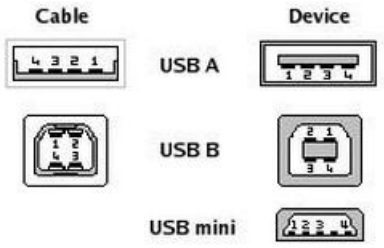


Figura1.21 Pines de un puerto USB
Fuente:[Tambi, 2013]

Los conectores están sujetos al estándar USB (Tipo A, Tipo B y su versión mini). Figura 1.22.



Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground

Figura1.22 Descripción de los pines del puerto USB
Fuente:[Tambi, 2013]

El bus USB permite el funcionamiento de hasta 127 dispositivos simultáneos, porque emplea una topología de estrella que permite la conexión de éstos dispositivos a un bus lógico sin que los dispositivos que se encuentran más abajo en la pirámide sufran retardo. La estructura de capas del bus USB se presentan en la figura 1.23, donde se puede observar que el controlador anfitrión o HOST es el responsable de controlar todo el tráfico que circula por el bus y se encuentra en la raíz o el vértice de las capas.

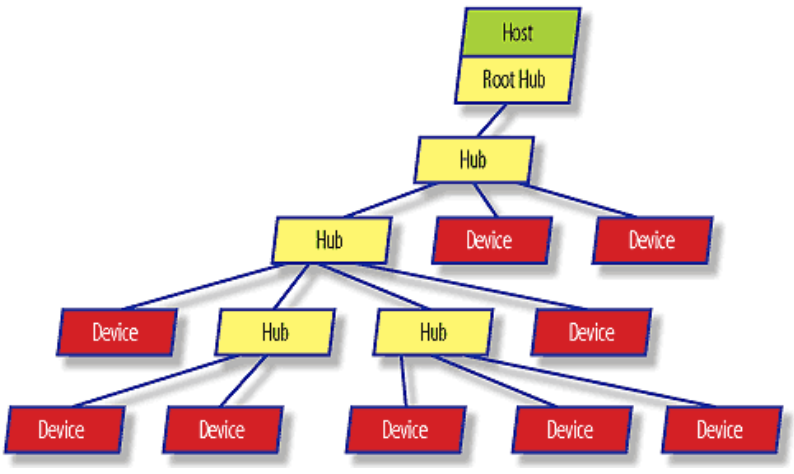


Figura1.23 Topología Universal Serial Bus
Fuente:[Weiss, 2001]

El protocolo USB, básicamente considera dos tipos de dispositivos:

- **HOST:** Dispositivo maestro que inicializa la comunicación (generalmente la PC)
- **HUB:** Es el dispositivo que contiene uno o más conectores o conexiones internas hacia otros dispositivos USB, el cuál habilita la comunicación entre el HOST con diversos dispositivos. Cada conector representa un puerto USB.

Éste protocolo de comunicación, se basa en el paso de un token (testigo), donde el HOST distribuye el token por el bus y el dispositivo cuya dirección coincida con la que porta el token responde aceptando o enviado datos al HOST.
Dado a que el bus USB es un bus punto a punto, con inicio en el HOST y destino en un dispositivo (NODO) o en un HUB; sólo puede existir un único HOST en la arquitectura USB (Peacock, 2013).

También, cabe señalar que el protocolo del bus USB soporta cuatro tipos de transferencia:

- **Control.** Modo utilizado para realizar configuraciones. Todos los dispositivos USB deben soportar este tipo de transferencia. Los datos de control sirven para configurar el periférico en el momento de conectarse al USB. Algunos drivers específicos pueden utilizar este enlace para transmitir su propia información de control. Este enlace no tiene pérdida de datos, puesto que los dispositivos de detección de recuperación de errores están activos a nivel USB.
- **Bulk.** Este modo se utiliza para la transmisión de importantes cantidades de información. Como el tipo control, este enlace no tiene pérdida de datos. Este tipo de transferencia es útil cuando la razón de transferencia no es crítica como por ejemplo, el envió de un archivo a imprimir o la recepción de datos desde un escáner. En estas aplicaciones, la transferencia es rápida, pero puede espera si fuera necesario. Solo los dispositivos de media y alta velocidad utilizan este tipo de transferencia
- **Interrupt.** Modo utilizado para transmisiones de pequeños paquetes, rápidos, orientados a percepciones humanas (mouse, punteros). Este tipo de transferencias son para dispositivos que deben recibir atención periódicamente y lo utilizan los dispositivos de baja velocidad. Este modo de transmisión garantiza la transferencia de

pequeñas cantidades de datos. El tiempo de respuesta no puede ser inferior al valor especificado por la interfaz. El mouse o cualquier otro dispositivo apuntador es una aplicación típica de este modo de transmisión.

- **Isochronous o Flujo en tiempo real.** Modo utilizado para la transmisión de audio o video comprimido. Esta forma de transmisión funciona en tiempo real. Este es el modo de mayor prioridad. La transmisión de la voz es un ejemplo de esta aplicación. Si ésta no se transmite correctamente, pueden llegar a oírse parásitos (glich) y la aplicación puede detectar ciertos errores de los llamados underruns (subdesbordamiento de búfer).

Por sus características y las ventajas que representa, el bus USB se ha convertido en la interfaz estándar para la comunicación de periféricos en computadoras personales (García, 2008).

1.1.14 Proceso de enumeración de un dispositivo USB

Cuando se conecta un dispositivo USB a la PC se produce el “proceso de enumeración”, y es cuando el HOST le pregunta al dispositivo que se presente e indique cuáles son sus parámetros, tales como:

• Consumo de energía
• Número y tipos de puntos terminales
• Clase del producto
• Tipo de transferencia (Control, Bulk, Interrupt o Isochronous)
• Razón de escrutinio, etc.

El proceso de enumeración es inicializado por el HOST cuando detecta que un nuevo dispositivo ha sido conectado al bus. Entonces, es el HOST le asigna una dirección al nuevo dispositivo conectado y habilita su configuración, permitiendo la transferencia de datos sobre el bus.

1.1.15 Especificaciones del bus USB

La especificación del bus USB ha ido mejorando a través del tiempo (figura 1.24), tanto en características como en prestaciones, sin embargo, la velocidad de transferencia de datos es una de las más notables.

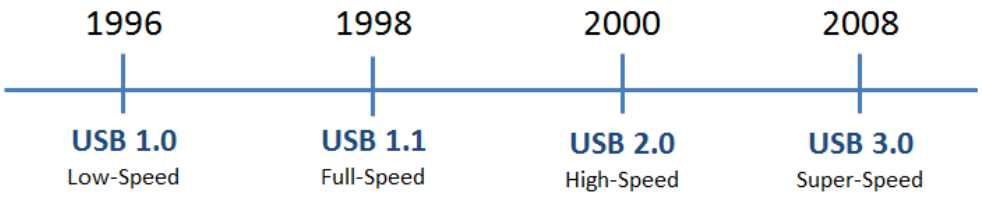


Figura1.24 Especificaciones USB y sus lanzamientos
Fuente:[Román Hernández, 2008]

- **Baja velocidad LS (Low-Speed 1.0)** Es la primera que se estableció en 1996 y sólo es utilizada para dispositivos de Interfaz Humana (HID) tales como mouses, teclado, joysticks, etc. Estos dispositivos emplean una velocidad de transmisión de datos de hasta 1.5 Mbps (187.5 KB/s).

- **Velocidad completa FS (Full-Speed 1.1**) Se lanzó en 1998 como una nueva revisión que mejora la velocidad a 12 Mbps (1.5 MB/s). Aunque fue una mejora, se volvió insuficiente para transferir mayor cantidad de información.

Figura1.25 Logo Universal Serial Bus FS
Fuente:[http://www.usb.org, 2013]

- **Alta velocidad HS (High-Speed 2.0)** Versión que salió en el año 2000 y emplea una velocidad de transferencia 40 veces más rápida que su predecesor, de hasta 480 Mbps (60 MB/s), por lo que consigue satisfacer las necesidades de transferencia y es

utilizado en discos duros externos, pen drives, conexiones a Internet, impresora, monitores, módems, scanners, equipos de audio, etc.

Figura1.26 Logo Universal Serial Bus HS
Fuente:[http://www.usb.org, 2013]

- **Super velocidad SS (Super-Speed 3.0)** Liberado en noviembre del 2008 y alcanza 10 veces la velocidad de su predecesor: 4.8 Gbps (600 MB/s).

Figura1.27 Logo Universal Serial Bus SS
Fuente:[http://www.usb.org, 2013]

De acuerdo a la tabla 1.4, cada una de las especificaciones es compatible con sus predecesoras, por lo que podemos decir por ejemplo, que la especificación USB 2.0 es compatible con dispositivos Low-Speed, Full-Speed y High-Speed, pero no con los Super-Speed (Axelson, 2009).

Tabla 1.4 Compatibilidad entre las especificaciones USB

	Low-Speed	Full-Speed	High-Speed	Super-Speed
USB 1.0	✔	✔		
USB 1.1	✔	✔		
USB 2.0	✔	✔	✔	
USB 3.0	✔	✔	✔	✔

1.1.16 Velocidad de transmisión del puerto USB

Hay 2 formas de medir la velocidad de transferencia de datos del puerto USB:

1. **En MegaBytes / segundo (MB/s)**
2. **En Megabits por segundo (Mbps)**

Es un error común el creer que lo anterior es lo mismo, debido a que los fabricantes manejan sus descripciones de producto basadas en la segunda cantidad, pero no es así. Existe una equivalencia para realizar la transformación de velocidades con una simple "regla de tres".

8 Mbps = 1 MB/s

Por lo tanto, la tabla 1.5 muestra las equivalencias correspondientes a las velocidades en las especificaciones del bus USB.

Tabla 1.5 Equivalencias de velocidad en las especificaciones USB

	Velocidad máxima en Megabits por segundo	Velocidad máxima en (Megabytes/segundo)
USB 1.0 Low-Speed	1.5 Mbps	187.5 KB/s
USB 1.0 Full-Speed	12 Mbps	1.5 MB/s
USB 1.0 High-Speed	480 Mbps	60 MB/s
USB 1.0 Super-Speed	4.8 Gbps	600 MB/s

1.1.17 Interfaz RS-232

La interfaz RS-232 es la interfaz de comunicación serial más utilizada en el mundo, y fue diseñada originalmente para conectar terminales de datos con dispositivos de comunicación, tales como módems.

Fue tal su popularidad, que se ha utilizado para conectar cualquier dispositivo imaginable, por lo que su difusión en el entorno electrodoméstico fue muy extensa. La podemos encontrar desde la conexión del mouse de la PC, el modem, agendas electrónicas, calculadoras, impresoras serie, grabadores de memorias, etc.

El estándar define voltajes que oscilan entre +3 a +15 V para el nivel alto y -3 a -15 V para el nivel bajo. Debido a la gran diferencia de voltaje que existe entre estos niveles, los valores más comunes que fija el Estándar RS-232 son: 1200, 2400, 4800, 9600, 19200, 38400, 56000, 57600, 115200, 128000, 256000 baudios. Aunque las versiones más recientes del Estándar ponen un límite de 20 kbits, es común emplear los valores altos como 115200 (siempre que sea posible), siendo la longitud del cable de unas pocas decenas de metros.

1.1.18 Emulación de la Interfaz RS-232 sobre USB

La interfaz serie RS-232 ha sido por muchos años la más utilizada para transferir datos entre una PC y dispositivos embebidos; sin embargo, con la evolución de las computadoras esta interfaz ha ido desapareciendo.

Debido a que muchas aplicaciones (legacy principalmente) hacen uso de esta interfaz, es posible emularla a través de una interfaz USB, para que así la PC identifique a esta conexión USB como una conexión COM RS-232 virtual (figura 1.28), y permita a éstas aplicaciones seguir operando normalmente, sin tener que realizar algún tipo de modificación sobre ellas.

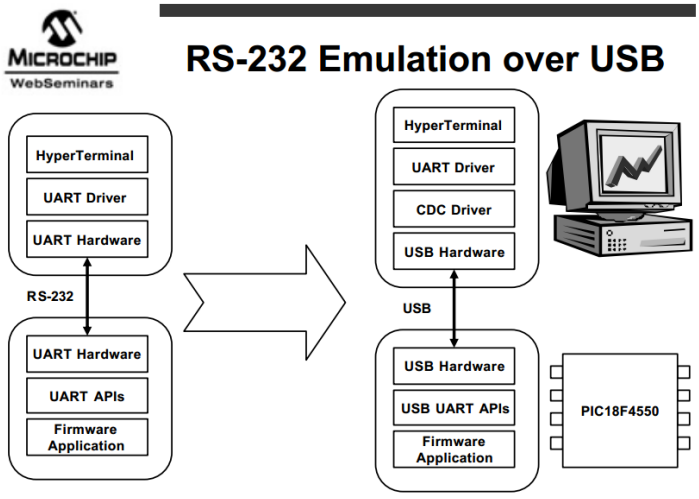


Figura 1.28 Emulación RS232 sobre USB
Fuente:[Rojvanit, 2004]

Cabe destacar que esta simulación hace uso de drivers ya suministrados por Windows, por lo que no es necesario desarrollar algún tipo de 'ad hoc'. Del lado de la PC, el driver USB CDC provee la capa de unión entre el hardware USB y el driver UART (Rojvanit, 2004).

Además, puesto que el protocolo USB maneja comunicación de bajo nivel, los conceptos 'baud rate', 'bit de paridad' y 'control de flujo' para la interfaz RS-232 ya no son de importancia (García, 2008).

1.1.19 USB CDC (Communication Device Class)

USB CDC es una especificación que define a un protocolo a nivel de dispositivos, y que envuelve a otros protocolos de comunicación permitiéndoles ser transportados sobre la interfaz USB.
Cada producto basado en esta especificación requiere tener una combinación única de VID y PID.

- **VID** corresponde al **Identificador del Fabricante**, el cual es un número de 16-bit que compra el fabricante y que es asignado por la "USB organization body" con el fin de que éste pueda comercializar y vender sus productos.
- **PID** corresponde al **Identificador del Producto**. También es un número de 16-bit, con lo que permite a cada fabricante hacer uso de hasta 65,536 PIDs.

La combinación de éstos deberá ser única para cada dispositivo existente en el mercado, ya que si hubiese dos dispositivos con el mismo VID/PID conectados a la misma PC, causarían conflictos entre ellos, impidiendo su correcto funcionamiento.
Es importante mencionar que los valores del VID y del PID son suministrados por los dispositivos (en este caso los MCUs) al sistema USB a través de los archivos descriptores (Figura 1.29), para que éste pueda clasificarlos y asignarles un Driver.

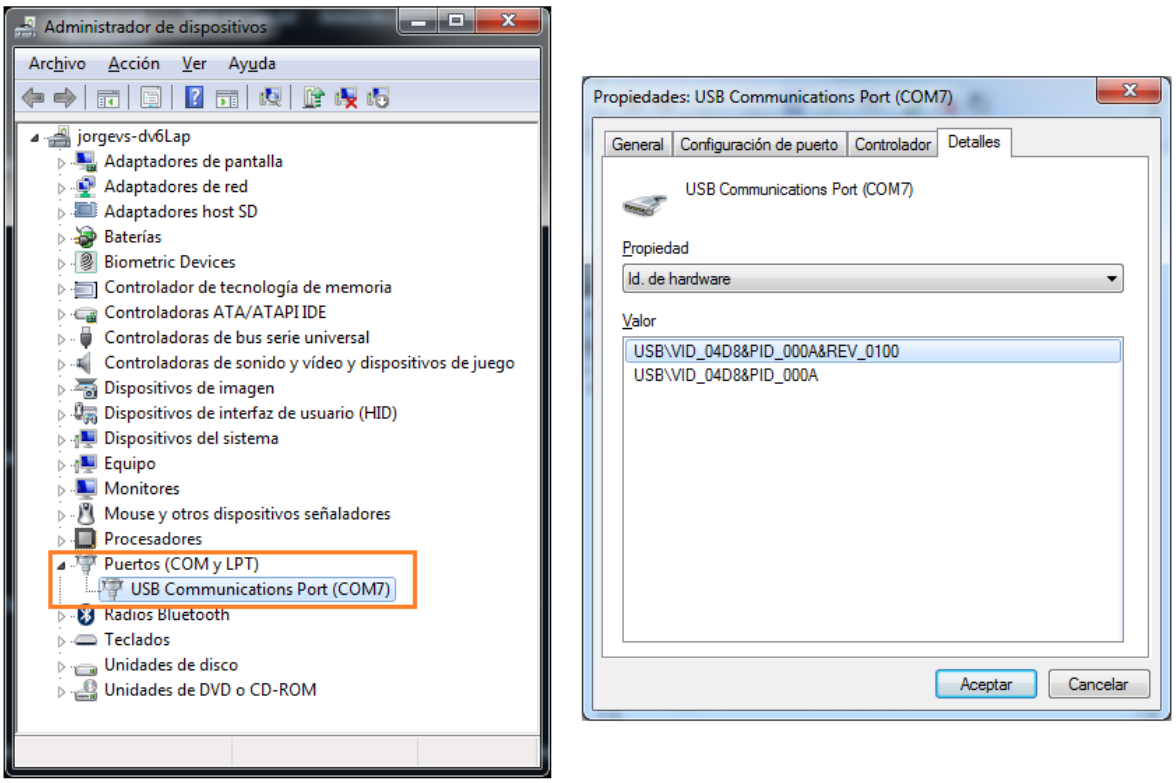


Figura 1.29 Puerto COM RS-232 virtual emulado con una interfaz USB
Fuente:[Elaboración propia]

Para la especificación CDC, Windows suministra el driver **usbser.sys**; sin embargo no en todas sus versiones cuenta con un archivo *.inf estándar para el driver, por lo que es necesario suministrarlo la primera vez que un dispositivo USB se conecta al sistema. Microchip suministra el archivo **mchpcdc.inf** (que es parte de su **USB Framework***) necesario para sus dispositivos.

***USB Framework** is parte de la **Microchip Applications Library**. Para más información visitar:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044

1.1.20 Sensores

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, y transformarlas en variables eléctricas. Algunas de estas magnitudes pueden ser: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, movimiento, pH, etc.

A continuación, se mencionan algunas de las características de un sensor:

- Rango de medida
- Precisión
- Offset o desviación de cero
- Linealidad
- Sensibilidad
- Resolución
- Rapidez de respuesta
- Repetitividad
- Etc.

Actualmente, los sensores se encuentran embebidos en múltiples aparatos electrónicos que utilizamos en nuestra vida cotidiana, así como en la industria automotriz, aeroespacial, manufactura, robótica, medicina, etc.

1.1.21 Sensor de temperatura (LM35)

El LM35 (figura 1.30) es un sensor de temperatura integrado de precisión, cuya tensión de salida es linealmente proporcional a la temperatura en ºC (cada grado centígrado equivale a 10mV).

No requiere calibración externa o ajuste para proporcionar una precisión típica de ± 1.4 ºC a temperatura ambiente y ± 3.4 ºC a lo largo de su rango de temperatura (de -55 a 150 ºC). El dispositivo se ajusta y calibra durante el proceso de producción. La baja impedancia de salida, la salida lineal y la precisa calibración inherente, permiten la creación de circuitos de lectura o control especialmente sencillos.

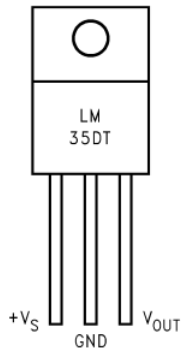


Figura 1.30 LM35 TO-220 Paquete plástico
Fuente:[Texas Instruments - LM35, 2000]

Requiere sólo 60 µA para alimentarse, y cuenta con un bajo factor de auto-calentamiento (menos de 0,1 ºC en aire estático).

El LM35 está preparado para trabajar en una gama de temperaturas que abarca desde los-55 ºC bajo cero a 150 ºC, mientras que el LM35C está preparado para trabajar entre -40 ºC y 110 ºC (con mayor precisión) (Texas Instruments - LM35, 2000).

Características

- Calibrado directamente en grados Celsius (Centígrados)
- Factor de escala lineal de +10 mV / ºC
- 0,5ºC de precisión a +25 ºC
- Rango de trabajo: -55 ºC a +150 ºC
- Apropiado para aplicaciones remotas
- Bajo costo
- Funciona con alimentaciones entre 4V y 30V
- Menos de 60 µA de consumo
- Bajo auto-calentamiento (0,08 ºC en aire estático)
- Baja impedancia de salida, 0,1W para cargas de 1mA

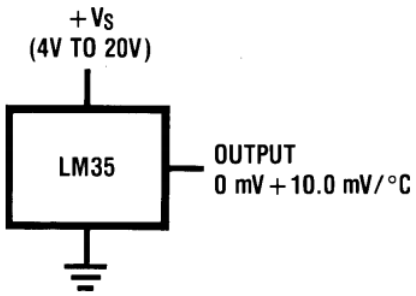


Figura 1.31 Esquema típico del LM35
Fuente:[Texas Instruments - LM35, 2000]

1.1.22 Módulo ADC

Un Convertidor Análogo-Digital (ADC por sus siglas en inglés) tiene como entrada un nivel de voltaje (valor analógico) y produce en su salida un número binario de n-bits, proporcional al nivel de la entrada (valor digital).

El módulo convertidor ADC del PIC18F4550 (figura 1.32), cuenta con 13 canales de entrada (AN0 – AN12). La conversión de la señal analógica aplicada (a uno de los canales) se convierte en número binario de 10-bits.

Su módulo ADC posee voltajes de referencia que pueden ser seleccionados para emplear las tensiones VDD y VSS del microcontrolador; o bien, puede emplear tensiones aplicadas a los pines AN2 (Vref-) y AN3 (Vref+). Incluso es posible establecer combinaciones de los anteriores valores.

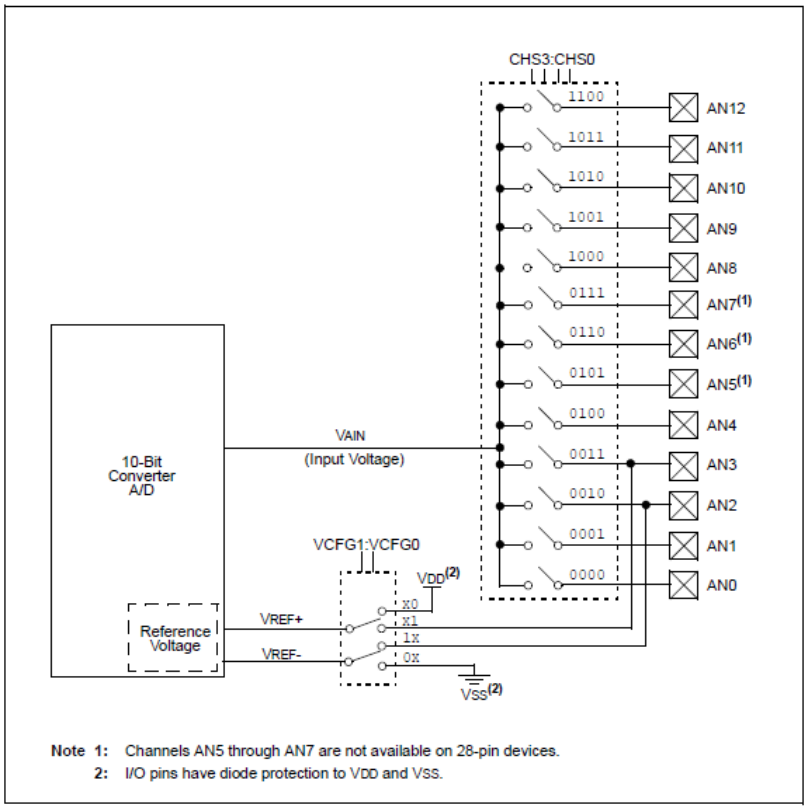


Figura 1.32 Diagrama de bloques del módulo ADC en el PIC18F4550
Fuente:[Microchip - PIC18F2455/2550/4455/4550, 2009]

La **resolución** (ecuación 1.1) es uno de los aspectos más importantes en un ADC, y corresponde a la mínima variación de voltaje en la entrada que produce un cambio del valor digital en la salida.

(1.1)

$$resolucionADC = \frac{Vref_{+} - Vref_{-}}{2^{n} - 1}$$

Por ejemplo un convertidor de 10-bits tiene un total de 2^{10} valores (1024 valores, de 0 a 1023). Por lo que, si tenemos 10V en la entrada, de acuerdo a la ecuación 1.1 la resolución seria de 9.775mV.

$$resolutionADC = \frac{10V - 0V}{1024 - 1} = 9.775mV$$

En este caso el voltaje es de 10V a 0V pero pueden variar.

Otro ejemplo es, si tenemos de 10V a 5V la resolución será:

$$resolutionADC = \frac{10V - 5V}{1024 - 1} = 4.88mV$$

Donde las tensiones de referencia son 10V y 5V.

1.2 Software

1.2.1 Java

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en el 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva mucho de C y C++, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente. Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "*write once, run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir del 2012, uno de los lenguajes de

programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados (Flanagan, 2005).

Java es la base para prácticamente todos los tipos de aplicaciones de red, además del estándar global para desarrollar y distribuir aplicaciones móviles, juegos, contenido basado en web y software de empresa.

- 1,100 millones de escritorios ejecutan Oracle Java
- 930 millones de descargas de Java Runtime Environment cada año
- mil millones de teléfonos móviles ejecutan Java
- Se entregan 31 veces más al año teléfonos Java que Apple y Android juntos
- El 100% de los reproductores de Blu-ray ejecutan Java
- Se fabrican 1400 millones de tarjetas Java cada año
- Más de 9 millones de desarrolladores en todo el mundo utilizan Java

Además, Java se incluye en decodificadores, impresoras, juegos, sistemas de navegación en vehículos, cajeros automáticos, terminales de loterías, dispositivos médicos, estaciones de pago de aparcamientos y mucho más (Java Oracle Webpage, 2013).

1.2.2 Spring framework (ver 2.5.6)

Spring es un framework de código abierto para el desarrollo de aplicaciones basadas en la plataforma Java. La primera versión fue escrita por Rod Jonhson, quien lo lanzó primero con la publicación de su libro Expert One-on-One Java EE Design and Development (Wrox Press, octubre 2002). Rod Johnson es reconocido por crear un framework que está basado en las mejores prácticas aceptadas, y ello las hizo disponibles para todo tipo de aplicaciones, no sólo aquellas basadas en web. Estas ideas también estaban plasmadas en su libro y, tras la publicación, sus lectores le solicitaron que el código que acompañaba al libro fuera liberado bajo una licencia open source.

El framework fue lanzado inicialmente bajo la Apache 2.0 License en junio de 2003. El primer gran lanzamiento hito fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005.

A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al ser considerado como una alternativa y sustituto del modelo de Enterprise JavaBeans. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Mientras que las características fundamentales de este framework pueden emplearse en cualquier aplicación hecha en Java, existen muchas extensiones y mejoras para construir aplicaciones basadas en web por encima de la plataforma empresarial de Java (Java Enterprise Plataform).

1.2.3 Spring MVC (ver 2.5.6)

Spring MVC es parte del core Spring Framework y es tecnología líder en el desarrollo de aplicaciones Web. Spring MVC provee un limpio modelo de componentes que hace que la construcción de aplicaciones Web sea escalable, testeable y con base en las mejores prácticas del desarrollo de software. A través de la adopción de “convención-sobre-configuración”, permite que su implementación en una aplicación sea tan simple como unas cuantas anotaciones y el registro del servlet Spring MVC (SpringSource - Spring Framework, 2011).

El diseño del Framework Spring MVC (model-view-controller) gira alrededor de un servlet llamado **DispatcherServlet** Figura 1.33, que es una implementación del patrón de diseño “**Front Controller**” (Sampaleanu, 2011).

Este servlet en realidad no lleva a cabo ninguna actividad relacionada con la lógica de negocio de la aplicación, sin embargo con ayuda de los **HandlerMapping** (objetos que indican hacia donde debe dirigir una petición basándose en su URL) delega las peticiones o **requests** que recibe del cliente a los objetos **Controller**, que son objetos en donde el verdadero trabajo se lleva a cabo (Minter, 2008).

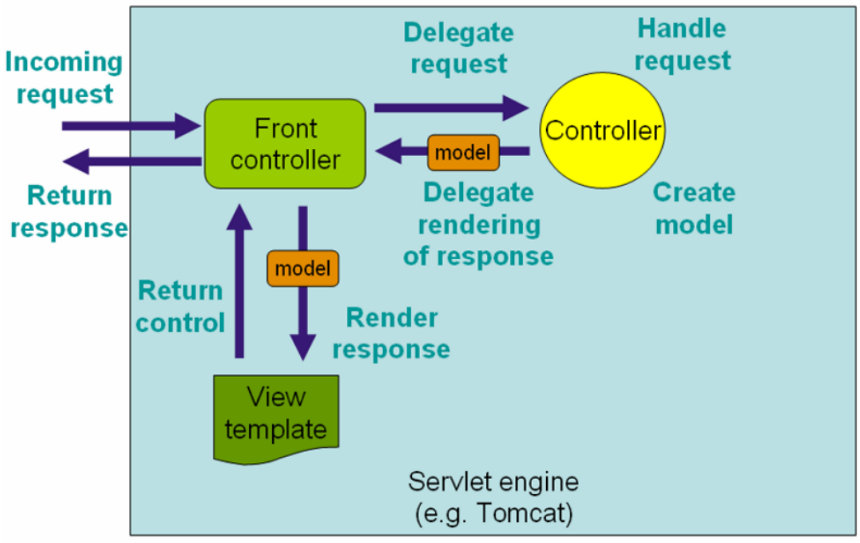


Figura 1.33 Componentes del Spring Web MVC
Fuente:[SpringSource – Spring Framework, 2008]

Los objetos **Controller** son los encargados de procesar la petición (**request**) y crear un objeto **model**, que no es otra cosa más que la información que necesita llevarse de vuelta para presentarse al usuario a través del browser o navegador. Pero enviar el modelo de regreso no es suficiente, es necesario darle un formato apropiado (típicamente HTML) a través de un objeto **View** (usualmente una página JSP) quien se encargará de renderizar la información que ha de presentarse al usuario, por lo que también el **Controller** debe identificar el nombre del objeto **View** y enviarlo de vuelta al **DispatcherServlet** junto con la petición (**request**) y el modelo (**model**).

El nombre del objeto **View** no identifica directamente a una página JSP, de hecho no necesariamente sugiere que este objeto sea una página JSP, pues solo representa el nombre lógico que será usado para identificar al objeto **View** encargado de producir el resultado. Es entonces cuando el **DispatcherServlet** consulta a un objeto **ViewResolver** para mapear y relacionar el nombre lógico con un objeto **View** en específico, el cual podría no ser una JSP.

Ahora que el **DispatcherServlet** conoce cuál es el objeto **View**, le entrega la petición (**request**) y el modelo (**model**) para que éste renderice y genere la salida a través de un

objeto de respuesta (**response**) el cuál será interpretado por el navegador o browser para finalmente presentarse al usuario (Shafqat, 2011).

1.2.4 Apache Tiles

Apache Tiles es un framework de plantillas que ayuda a simplificar el desarrollo de interfaces de usuario para aplicaciones Web (Apache Tiles Website, 2013).

Con el uso de Tiles, se pueden crear componentes reutilizables para la capa de presentación de aplicaciones Web. Primero, se define una plantilla base (base layout) con diferentes secciones (figura 1.34) y después se establece que página JSP debe rellenar cada una de las regiones correspondientes en la plantilla, a través de un archivo de configuración externo. La misma plantilla (layout) puede ser reutilizada “n” número de veces especificando diferentes páginas JSPs.

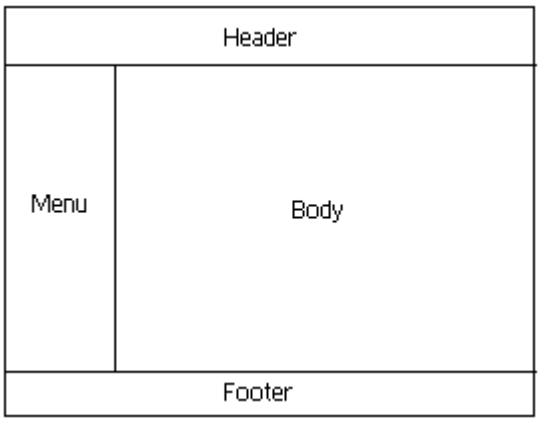


Figura 1.34 Plantilla genérica de una aplicación Web
Fuente:[Elaboración propia]

Una plantilla normalmente contiene una sección para el menú, una cabecera para publicidad, pie de página para notificaciones, un cuerpo que es donde se presenta en el contenido o la información principal, etc.

Apache Tiles se puede integrar con diferentes frameworks de la capa de presentación como Spring, Struts, Shale, etc para trabajar en conjunto y simplificar el desarrollo de aplicaciones Web.

1.2.5 Ajax

AJAX, es el acrónimo de Asynchronous JavaScript And XML. Es un grupo de técnicas utilizadas para el desarrollo de aplicaciones Web interrelacionadas con la intención de crear aplicaciones interactivas o RIA (Rich Internet Applications) (Ullman, 2007).

Estas aplicaciones se ejecutan en el lado del cliente, es decir, en el navegador de los usuarios. Con Ajax, las aplicaciones pueden enviar y recibir datos desde el servidor de manera asíncrona (en segundo plano) sin interferir en la presentación y comportamiento de la página existente. De esta forma es posible interactuar con la página que se está visualizando sin necesidad de recargarla, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

JavaScript es un lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante el objeto XMLHttpRequest, que se encuentra disponible en la mayoría de los navegadores actuales. A pesar del nombre, el uso de XML no es requerido y las peticiones no tienen que ser necesariamente asíncronas (Crane, Bibeault & Sonneveld, 2007).

Ajax es una combinación las siguientes tecnologías ya existentes:

- HTML y CSS (Cascade Style Sheet) utilizados normalmente para el etiquetado y el estilo utilizado en la presentación de la información.
- DOM (Document Object Model) que es accedido con JavaScript para presentar información dinámicamente al usuario y permitirle interactuar con ella.
- JavaScript y el objeto XMLHttpRequest que son quienes proveen la manera de intercambiar información de manera asíncrona entre el navegador y el servidor, evitando la recarga completa de las páginas Web.

1.2.6 jQuery

jQuery es una biblioteca JavaScript pequeña, rápida y rica en funcionalidades. Permite simplificar la manera de interactuar con los documentos HTML, con el árbol DOM, manejar eventos, animaciones y agregar interacción con la técnica AJAX a páginas Web de una manera mucho más sencilla y trabajando con múltiples navegadores (jQuery Website, 2013).

Fue creada inicialmente por John Resig y es software libre y de código abierto lo que permite su uso en proyectos libres y privativos.
jQuery, consiste en un único archivo JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX. La característica principal de la biblioteca es que permite cambiar el contenido de una página Web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX.
También, al igual que otras bibliotecas ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio (Swedberg & Chaffer, 2010).

1.2.7 Maven

Maven es una herramienta para la creación y gestión de cualquier proyecto basado en Java. Fue creado por Jason Van Zyl, de Sonatype, en 2002. Cuenta con un modelo de configuración de construcción simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation (Apache Maven Website, 2013).

Maven está basado en la filosofía “convención sobre configuración”. Es decir, asume un comportamiento por defecto que permite empezar a trabajar sin necesidad de configuración inicial.
Maven utiliza un Project Object Model (POM) para describir características propias del proyecto de software a construir, tales como:

- **Coordenadas del proyecto:** Es la información que permite identificar de forma única a un proyecto.
- **Propiedades administrativas del proyecto:** Licencia, miembros del proyecto.
- **Dependencias del proyecto:** Puede referirse a otros proyectos que ya se encuentran compilados, empaquetados y dentro de algún repositorio, ya sea local o remoto.
- **Repositorios remotos:** Se puede hacer referencia a otros repositorios de artefactos Maven, a partir de los cuales los proyectos pueden obtener sus dependencias.
- **Plugins de terceros:** Permiten añadir al proyecto funcionalidad específica para su desarrollo.

Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores.

Maven provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local (Casey, Massol, Porter, Sanchez & Van Zyl, 2006).

Maven cuenta con objetivos predefinidos para realizar ciertas tareas claramente definidas las cuales constituyen su ciclo de vida, y éstas son:

validate: Verifica que el proyecto este correcto y que toda la información necesaria esté disponible

compile: Compila el código fuente del proyecto

test: Realiza pruebas unitarias sobre el código fuente, haciendo uso de algún Unit Testing Framework. Estas pruebas no requieren que el código sea empaquetado ni desplegado.

package: Toma el código compilado y lo empaqueta en un format distribuible (JAR file)

integration-test: Procesa y despliega el paquete si es necesario dentro de un ambiente en donde las pruebas de integración puedan ser ejecutadas

verify: Ejecuta algunas chequeos a fin de verificar que el paquete generado es válido y cumple con ciertos criterios de calidad

install: Instala el paquete dentro del repositorio local (un directorio de la PC), para que pueda ser usado como dependencia en otros proyectos localmente

deploy: Copia el paquete final a un repositorio remoto para que pueda ser compartido con otros desarrolladores y proyectos (Apache Maven Website, 2013).

1.2.8 Tomcat

Apache Tomcat es un servidor Web con soporte para Servlets y JSPs, pues implementa las especificaciones establecidas por Sun Microsystems para dichos componentes. Incluye el compilador Jasper, el cuál se encarga de compilar los JSPs, a fin de convertirlos en Servlets. Fue desarrollado bajo el proyecto Jakarta en la Apache Software Foundation (Apache Tomcat Website, 2013).

Inicialmente se tenía la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Sin embargo, en la actualidad ya no existe esa percepción, y Tomcat es utilizado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

1.2.9 MPLAB X IDE

MPLAB X IDE (figura 1.35) es una interfaz de para desarrollar aplicaciones con los microcontroladores PIC de Microchip de 8-bit, 16-bit y 32-bit y dsPIC DSCs (Digital Signal Controllers). Incluye un editor rico en funcionalidades, un debugger a nivel código, un simulador de software y también soporta otras herramientas de hardware muy populares de Microchip, tales como el MPLAB ICD 3 in-circuit debugger, PICkit™ 3, y el programador MPLAB PM.

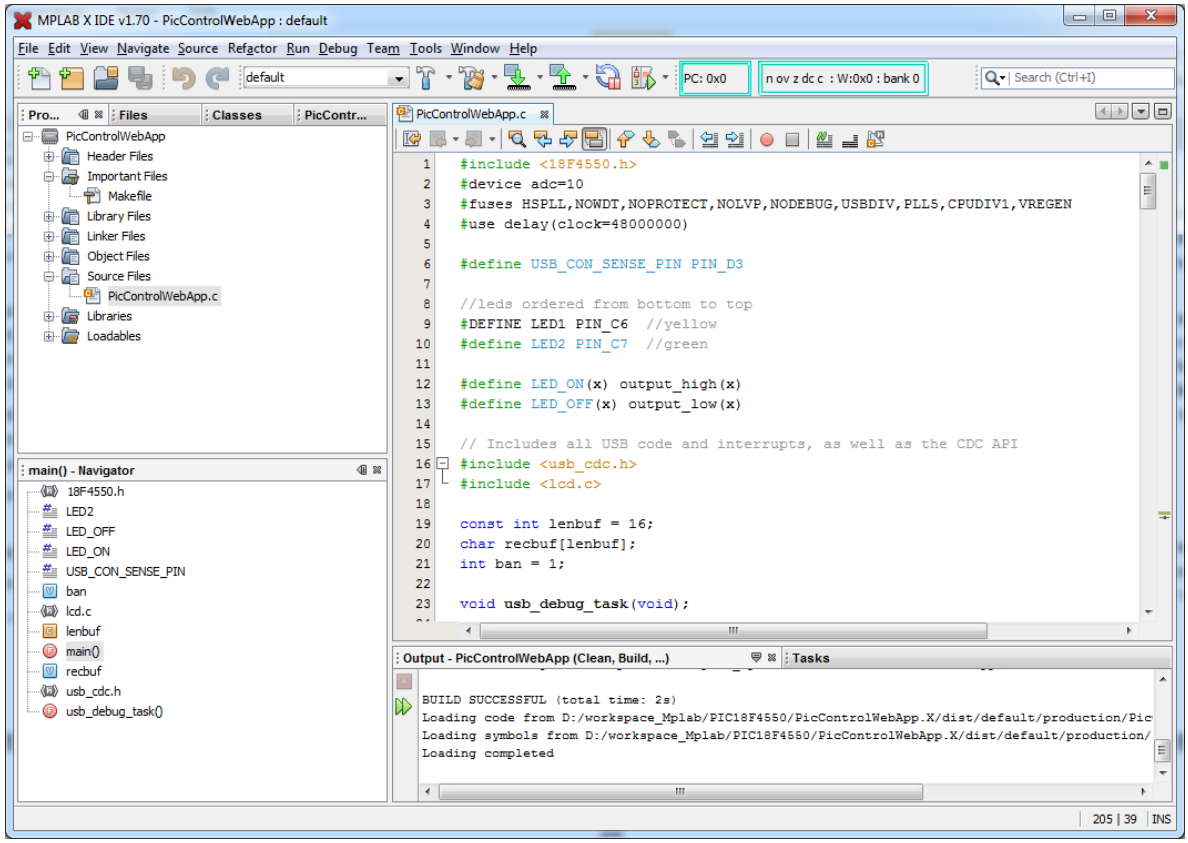


Figura 1.35 MPLAB X IDE v1.70
Fuente:[Elaboración propia]

Actualmente, esta versión “X” está basada en la plataforma open-source NetBeans y corre sobre Windows, MAC OS y Linux. También soporta herramientas de terceros y es compatible con muchos plug-ins de NetBeans (Microchip - MPLAB X, 2013).

1.2.10 Compilador CCS

Es un compilador de lenguaje C desarrollado por CCS Inc., totalmente compatible con el MPLAB IDE de Microchip.

Incorpora operadores del C Estándar y funciones específicamente diseñadas para los registros de los microcontroladores PIC, con lo que provee a los desarrolladores de herramientas muy poderosas para acceder a las funcionalidades de dichos dispositivos a partir de un lenguaje de nivel medio como lo es C. Sus pre-procesadores, operadores y

sentencias, pueden combinarse con directivas específicas de hardware, así como de sus funciones y librerías de ejemplo incorporadas, para desarrollar rápidamente aplicaciones que hagan uso de dispositivos con tecnología de punta, tales como pantallas touch capacitivas, comunicación alámbrica e inalámbrica, movimiento y control de motores y de administración de energía (CCS Inc. Webpage, 2013).

Una versión DEMO de este compilador se puede descargar de manera gratuita directamente de la página de www.ccsinfo.com La versión DEMO es totalmente funcional durante un periodo de 45 días.

Capítulo 2. Diseño del sistema

Una vez que se han presentado las tecnologías y herramientas que se utilizarán en el desarrollo del proyecto, es turno de listar las características a tomar en cuenta durante su diseño.

2.1 Consideraciones generales del diseño

El sistema a desarrollar deberá cumplir con las siguientes cualidades:

- La temperatura se mostrará al usuario en grados centígrados.
- El microcontrolador y el control que éste realiza, necesita hacer uso del voltaje suministrado por el puerto USB, evitando así el uso de alguna fuente externa.
- La aplicación Web debe ser sencilla, amigable y basada en hojas de estilo (CSS) para facilitar su mantenimiento.
- La aplicación Web deberá mostrar al usuario los puertos digitales disponibles del microcontrolador con los que podrá interactuar, de tal forma que pueda habilitar o deshabilitar cada uno de ellos.
- El sistema necesitará mostrar al usuario en tiempo real las variaciones en las lecturas realizadas por el puerto analógico del microcontrolador sobre el sensor de temperatura.
- La aplicación necesitará estar diseñada de manera tal, que pueda soportar actualizaciones relacionadas a las tecnologías de software. Esto quiere decir que si es necesario reemplazar o modificar algún componente de software, el impacto no debe afectar a toda la aplicación sino solamente a la capa en donde se realice dicha actualización.
- El acceso a datos deberá poder ser reutilizado por servicios pertenecientes a otras aplicaciones, o bien, por componentes de la misma aplicación que podrían desarrollarse en el futuro.

2.2 Diagrama general del sistema

El sistema debe contemplar la creación de módulos tanto de hardware como de software, y que en conjunto permitirán llevar a cabo el objetivo principal del proyecto. En la figura 2.1 se puede apreciar el diagrama general.

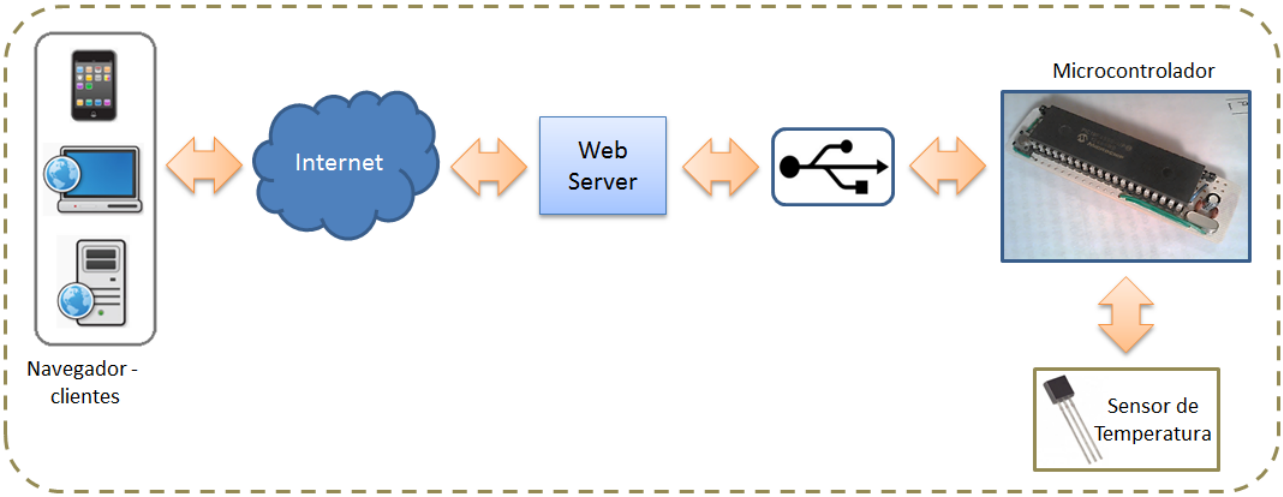


Figura 2.1 Diagrama general del sistema
Fuente:[Elaboración propia]

El sistema estará compuesto por seis partes fundamentales, que son: el cliente, la red, el servidor de aplicaciones encargado de hostear la aplicación Web, el puerto USB de comunicación perteneciente a la PC que funge como servidor, el microcontrolador y el sensor de temperatura. Cada una de estas partes se detalla a continuación.

Cliente – El cliente corresponde al browser o navegador de cualquier dispositivo que pueda conectarse al servidor, a la red de área local o Internet, dependiendo de cómo sea desplegada la aplicación. Por ejemplo: una PC de escritorio, un Smartphone, una Tablet, una laptop, etc.

Red – La red es el elemento por el cual el cliente se comunica con el servidor. Puede ser una red LAN, WAN, Internet o no existir ninguna y tener una conexión directa entre el browser del usuario y el servidor Web ejecutándose en el mismo equipo.

Servidor Web – Se encarga de hostear la aplicación Web y de exponerla a través de la red haciendo uso del protocolo HTTP, permitiendo que uno o más clientes interactúen con ella.
Puerto USB – Es el uno de los puertos USB existentes en el equipo que funge como servidor Web.
Microcontrolador – Es un circuito integrado programable capaz de llevar a cabo procesos lógicos y se encarga de controlar el funcionamiento de una tarea determinada.
Sensor de temperatura – En este proyecto hacemos uso del sensor LM35 que es un componente electrónico que envía una tensión de salida a uno de los puertos analógicos del MCU, y que es linealmente proporcional al valor de la temperatura de su entorno en grados centígrados.

Mediante la interacción de estos componentes, el usuario podrá seleccionar y ejecutar comandos desde la aplicación Web para interactuar con los puertos digitales de entrada/salida del microcontrolador, así como poder visualizar en tiempo real la lectura de la temperatura que entrega el sensor al microcontrolador.

2.3 Diseño del hardware

Como hemos visto en las secciones anteriores, el microcontrolador deberá tener comunicación con diferentes componentes electrónicos para poder llevar a cabo dos tareas principales:

1) Recibir las instrucciones enviadas por el usuario, procesarlas e identificar cuáles son los puertos que se deben encender o apagar.
2) Hacer uso de su módulo ADC para leer el voltaje presente en el sensor de temperatura, realizar su transformación a grados centígrados, imprimirlos en la pantalla LCD y enviarlos a través del puerto USB al servidor para que la lectura pueda ser presentada al usuario.

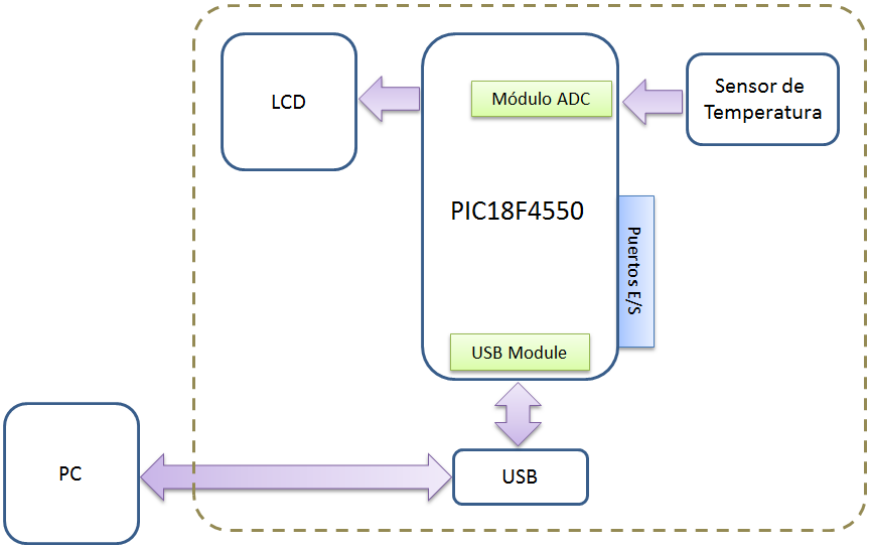


Figura 2.2 Diagrama de control del microcontrolador
Fuente:[Elaboración propia]

En esta etapa de diseño, es conveniente hacer uso de un simulador de hardware que permita conocer como se comporta el circuito, incluso antes de conectar cualquier componente en el protoboard. Actualmente existen varios simuladores de circuitos, sin embargo para el desarrollo de este trabajo se ha optado por utilizar ISIS de Labcenter Electronics (figura 2.3), porque se encuentra ampliamente difundido y existen muchos ejemplos y tutoriales en la red. Además, se pueden descargar muchos componentes en forma de librerías, incluso para muchos componenetes nuevos y que se acaban de introducir en el mercado.

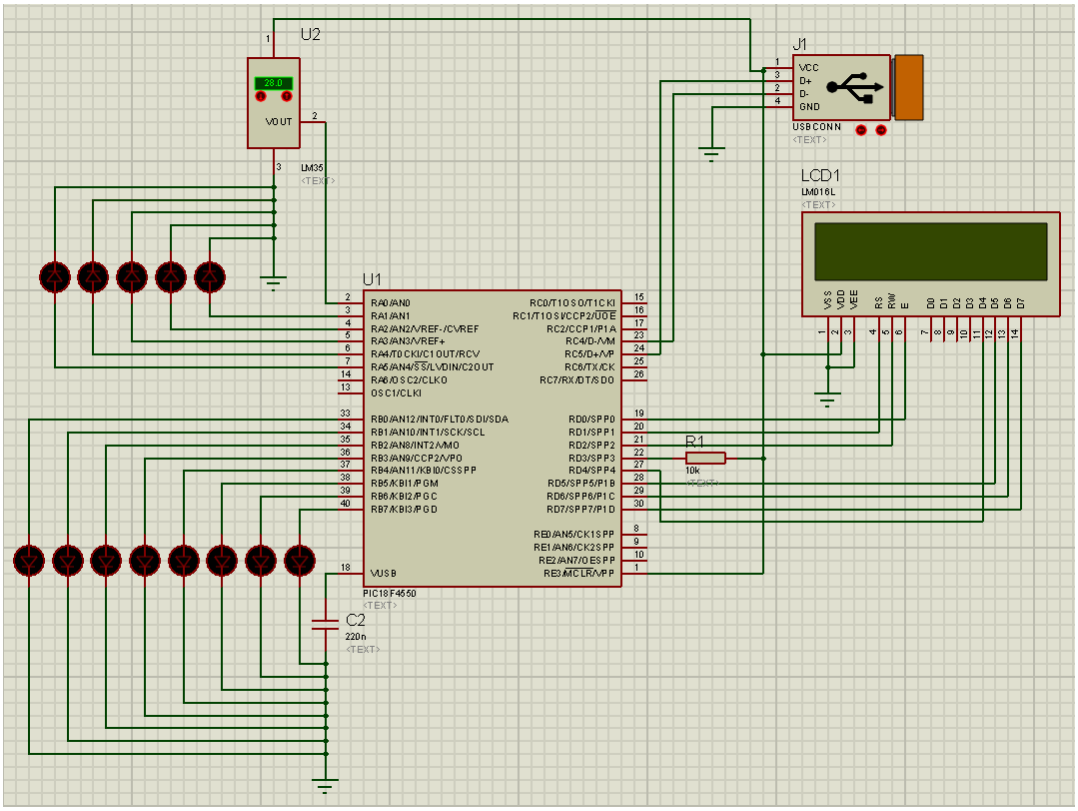


Figura 2.3 Componentes de la etapa de Hardware en ISIS
Fuente:[Elaboración propia]

Después de efectuar pruebas y realizar el diseño del circuito en el simulador, se han establecido los siguientes puntos:

- El sensor de temperatura estará conectado al puerto AN0, el cuál deberá estar configurado como entrada analógica para poder leer el voltaje en el sensor, y a través del módulo ADC realizar la conversión a grados centígrados.
- Todas las terminales del puerto A (excepto AN0) y del puerto B, se deberán configurar como salidas digitales, de tal manera que el programa pueda controlar el encendido/apagado de los led que estarán conectados a cada una de ellas. Es importante mencionar que los niveles de tensión que manejan las terminales de éstos puertos permiten trabajar con cargas de bajo consumo, tales como leds, displays de siete segmentos o LCD; sin embargo, en caso de que se requiera activar cargas de mayor consumo, será necesario hacer uso de transistores y de alguna fuente externa.

- El voltaje para la operación del circuito, se deberá obtener a partir de los 5 Volts de alimentación que entrega el puerto USB del servidor.
- El LCD conectado al puerto D no es indispensable para el funcionamiento del sistema, sin embargo es muy útil para observar la información que envía y recibe el microcontrolador y así poder verificar que su programación funciona adecuadamente.

2.3.1 Funcionamiento del programa en el microcontrolador

Para que el microcontrolador pueda desempeñar las tareas que le corresponden como parte del sistema, es necesario desarrollar un pequeño programa que ejecute ciertos pasos y operaciones de manera contínua. Estos pasos se muestran en el siguiente diagrama de flujo correspondiente a la figura 2.4.

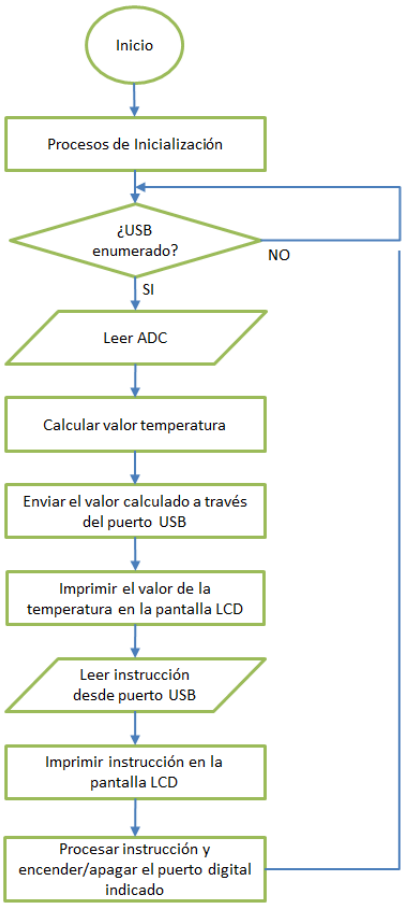


Figura 2.4 Diagrama de flujo de funcionamiento
Fuente:[Elaboración propia]

Tan pronto sea reconocido y enumerado el microcontrolador por el puerto USB del servidor, esta serie de pasos y operaciones se ejecutarán constantemente.

2.3.2 Formato de la información que envía y recibe el microcontrolador

Se necesita establecer un formato para la información que se va a tranferir entre la etapa de hardware y software, a fin de que cada etapa pueda interpretar correctamente lo que recibe; por lo que a continuación se detalla cada uno de ellos.

Los valores de temperatura que se van a transmitir desde el microcontrolador al servidor deberán ser tener el siguiente formato: **NN.N>**

Donde

- **NN.N** representa el envío de dos números enteros y un decimal.
- El carácter **">"** es utilizado como separador, con la intención de marcar el fin de cada valor que envía el microcontrolador.

Las instrucciones para encender o apagar los leds conectados a las terminales de los puertos A y B, y que se van a transmitir desde el servidor al microcontrolador deberán contar con el siguiente formato: **PN:E<**

Donde

- **P** indica el puerto (puede ser A o B)
- **N** indica el número del terminal del puerto (1-5 para el puerto A, 0-7 para el puerto B)
- **E** establece si se enciende o se apaga el led conectado al terminal. Sus posibles valores son:

  0 = apagado / 1 = encendido
- El carácter **"<"** es utilizado como separador, con la intención de marcar el fin de cada instrucción que recibe el microcontrolador.

2.4 Diseño del Software

Se ha planteado el desarrollo de una Aplicación Web a través de la cual el usuario podrá interactuar con el microcontrolador, ya sea encendiendo o apagando sus puertos digitales, o bien realizando la lectura del sensor de temperatura a través de uno de sus puertos analógicos. Y para poder llevar a cabo estas tareas, la aplicación Web debe ser una interfaz robusta, fácil de manejar e intuitible para el usuario.

2.4.1 Arquitectura de la aplicación

Para cumplir con las especificaciones de diseño, la arquitectura de esta aplicación deberá estar dividida en capas, desacopladas entre sí y dependientes en una sola dirección. Esto quiere decir, que las capas inferiores no tienen ninguna dependencia con las capas superiores, permitiéndoles ser reutilizables en otros escenarios. Además, su aislamiento permitirá realizar mejoras en el código, actualizaciones en las tecnologías utilizadas, corrección de defectos, etc., reduciendo el impacto en las otras capas y por lo tanto en todo el sistema.

A continuación, en la figura 2.5 se presentan las 3 capas principales en las que estará dividida la aplicación Web.

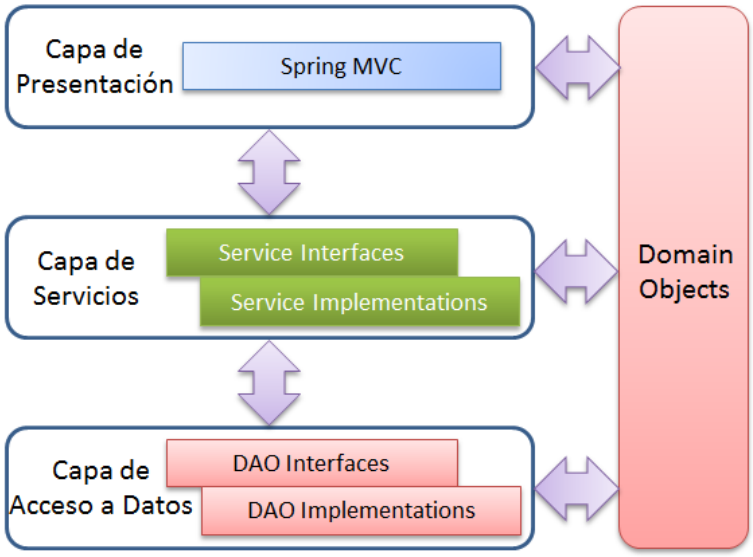


Figura 2.5 Arquitectura basada en capas
Fuente:[Elaboración propia]

Como se puede observar, el microcontrolador será accedido únicamente por la capa de acceso a datos popularmente llamada también como DAO Layer (la cual accedería también a algún otro origen de datos, en caso de que existiese en el sistema, tal como lo es una base de datos, archivo plano, repositorio, etc.). Esta capa interactuará con el microcontrolador a través del puerto USB del servidor.

La capa de acceso a datos será invocada por la capa de negocio o servicios, la cual contiene las reglas de negocio de la aplicación, y que a su vez es consumida por la capa de presentación. Ésta última, es la que hace uso del Spring Framework MVC.

2.4.2 Procesamiento de las peticiones del cliente

Dado lo anterior, se puede observar en la figura 2.6 el diagrama del flujo de procesamiento de las peticiones realizadas por el cliente.

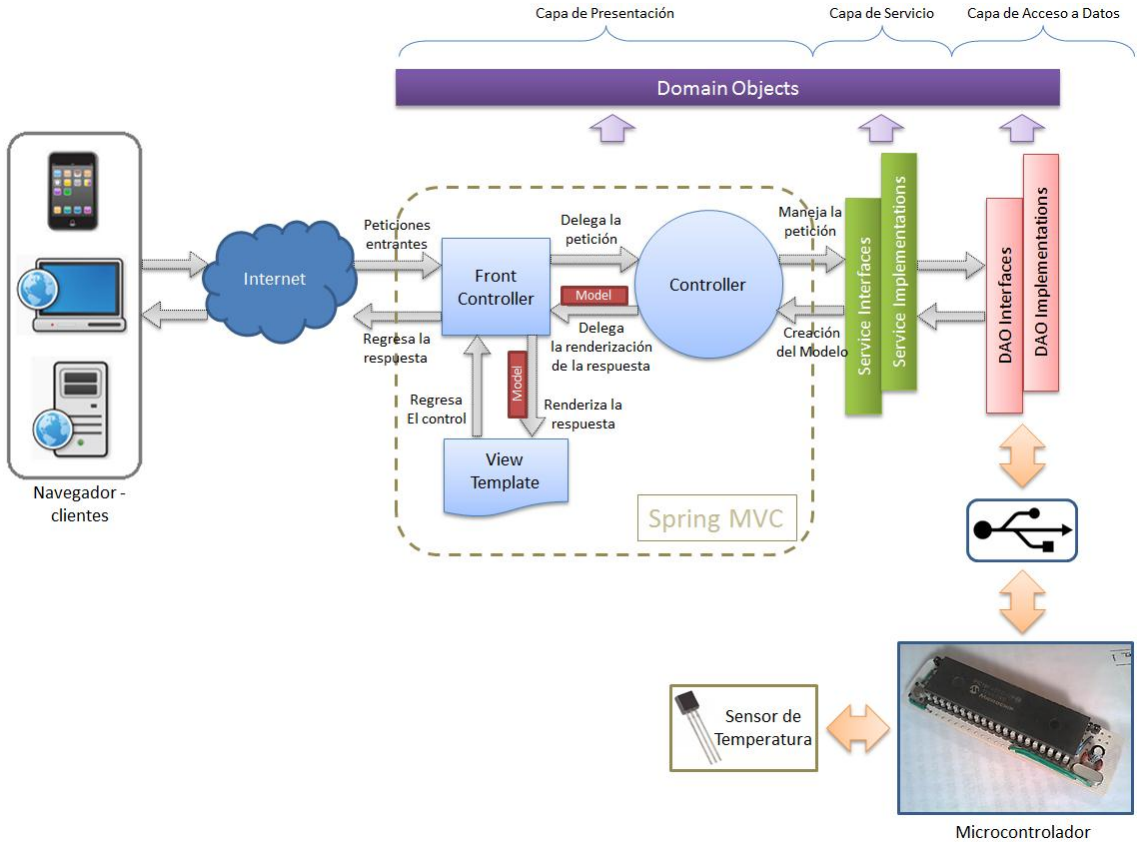


Figura 2.6 Procesamiento de peticiones realizadas por el cliente
Fuente:[Elaboración propia]

Cada petición HTTP realizada por el cliente, será recibida y procesada por los componentes de la capa de presentación (principalmente por los del framework Spring MVC). El principal de éstos es un Servlet central que se encarga de despachar todas las peticiones http que recibe, hacia los controladores. Esto no es más que una expresión del patrón de diseño "Front Controller".

Los controladores al recibir cada petición, la procesarán y consumirán los métodos expuestos por la capa de servicios, la cual a su vez consumirá a aquellos disponibles por la capa de acceso a datos. Ésta última será la que se comunique directamente con el microcontrolador a través del puerto USB del servidor, de tal forma que pueda controlar el estado de sus puertos digitales y obtener la lectura del sensor de temperatura que envía el microcontrolador.

Una vez que los controladores han terminado de procesar una petición, devolverán la información obtenida, encapsulada en un objeto llamado "modelo" el que es utilizado para renderizar la información en un formato de presentación que será mostrado al usuario.

2.4.3 Obtención del valor de la temperatura enviada por el microcontrolador

Dado a que las variaciones en las lecturas realizadas por el microcontrolador sobre el sensor de temperatura deben ser obtenidas en tiempo real, y sus variaciones deben presentarse inmediatamente al usuario, es necesario contar con un hilo (Thread) perteneciente a la capa de acceso a datos que se encargue de recibir ésta información de manera constante (figura 2.7). Cada valor que éste obtenga, será almacenado y expuesto a través de una variable de memoria, de manera tal que su valor pueda ser consultado desde la capa de servicios por algún componente de otra aplicación, o bien, por algún otro componente que pertenezca a la capa de presentación de la aplicación principal.

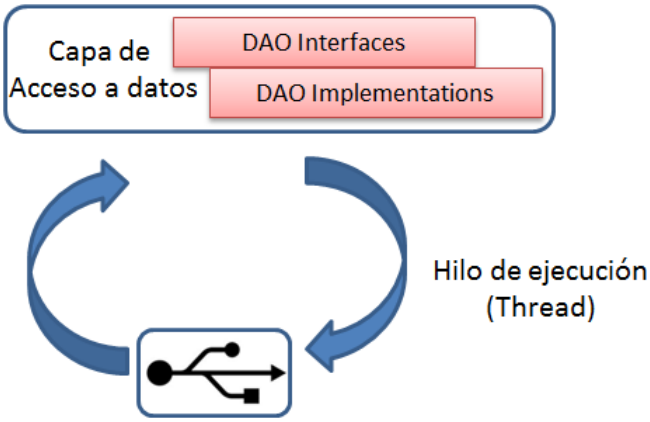


Figura 2.7 Thread encargado de leer toda la información enviada por el MCU
Fuente:[Elaboración propia]

El Thread procesa continuamente las lecturas enviadas por el microcontrolador y almacena su valor en una variable, lo que permite contar con el valor de temperatura más actual constantemente.

Sin la existencia de éste Thread, la lectura se realizaría directamente del puerto USB (en vez del valor de la variable en donde el Thread almacena la última lectura) lo que ocasionaría que la lectura no fuera la más actual, pues éstas se encontrarían encoladas en el puerto y se tomarían valores previos.

2.4.4 Presentación del valor de la temperatura al usuario

Como se mencionó anteriormente, el valor de la temperatura se almacenará en una variable en memoria, la cual será accedida por un Servlet perteneciente a la capa de presentación. Éste hará uso de la capa de servicios para llegar a la capa de datos y así obtener dicho valor; sin embargo, es importante mencionar que el Servlet será invocado desde una página JSP, también perteneciente a la capa de presentación. La página contiene un script de jQuery que hace uso de Ajax para presentar el valor de la temperatura en un componente HTML de la misma aplicación cada cierto periodo de tiempo, a fin de presentar las variaciones de la temperatura.

El diagrama de la obtención de la temperatura y de cómo es consumida para ser presentada al usuario se muestra en la figura 2.8 a continuación.

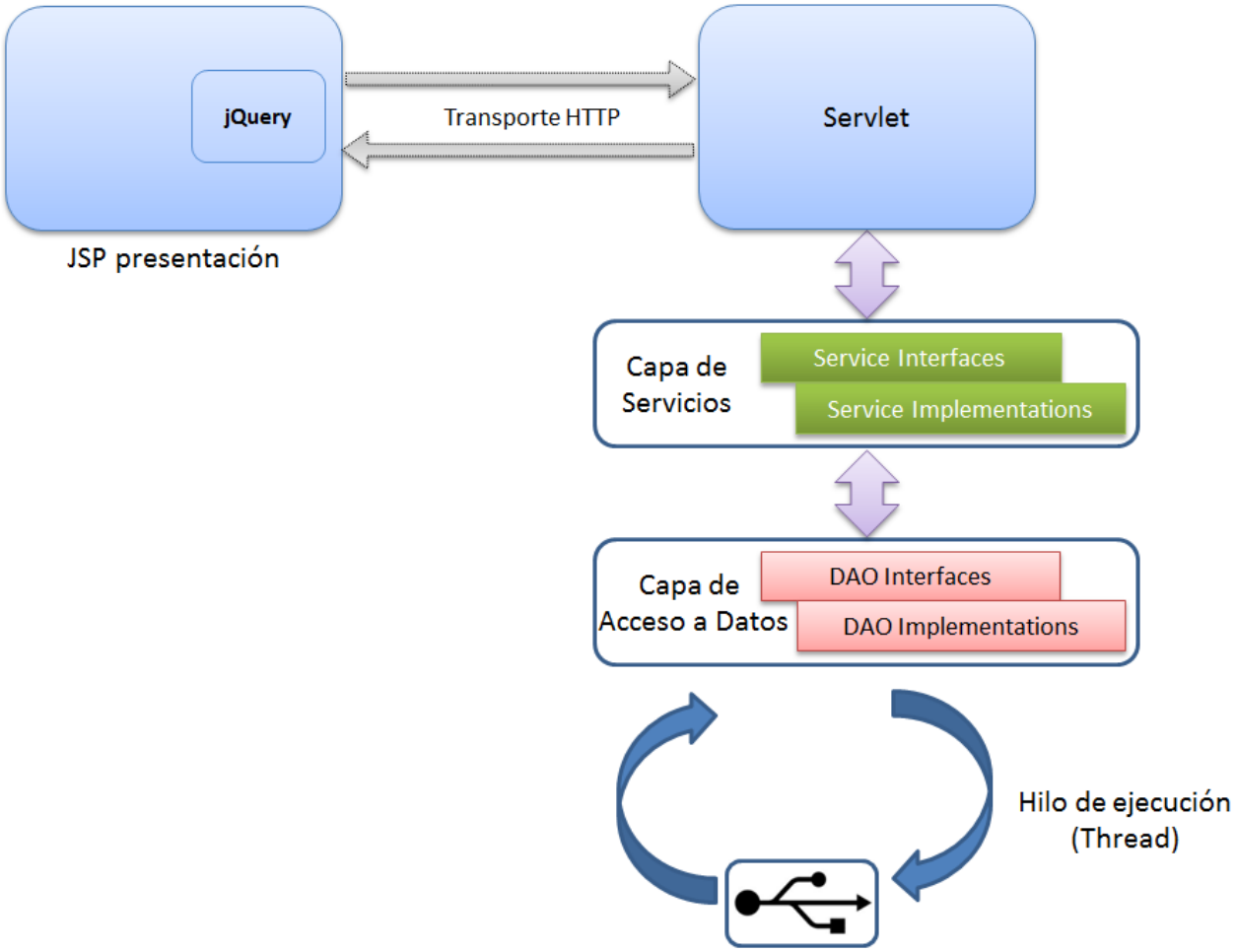


Figura 2.8 Obtención del valor de la temperatura desde la capa de presentación
Fuente:[Elaboración propia]

2.4.5 Diseño de la interfaz de usuario

Usualmente una aplicación Web, está formada por una o varias plantillas en la que su funcionalidad cambia únicamente una porción del contenido mostrado respecto a las restantes (normalmente la parte central); por lo que es de gran ayuda integrar algún framework para el manejo de plantillas y de ésta manera simplificar el desarrollo de aplicaciones Web.

Para el caso del presente proyecto, se deberá definir una plantilla que cuente con las siguientes secciones:

- **Menu** (menú) – Sección que permitirá cambiar el contenido que se presenta en la sección “Body” a través de vínculos dinámicos
- **Header** (cabecera) – en esta sección se presentará el nombre e imagen descriptiva de la aplicación.
- **Footer** (pié de página) – sección donde se mostrará la fecha actual
- **Body** (cuerpo) – Esta sección cambiará constantemente y en ella se presentará información y datos importantes para el usuario.

Su ubicación en la presentación deberá ser de acuerdo al esquema presente en la figura 2.9.

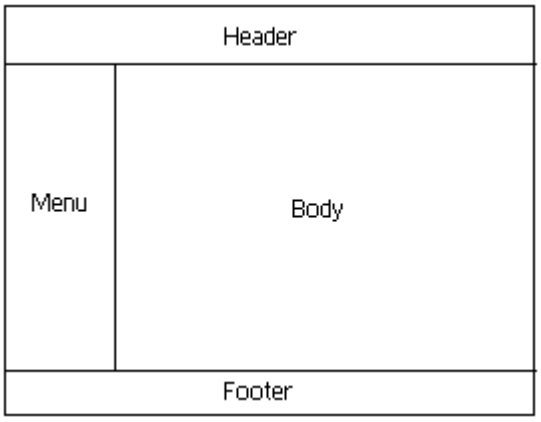


Figura 2.9 Secciones de la interfáz de usuario
Fuente:[Elaboración propia]

2.4.6 Diagrama de paquetes

Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias existentes entre dichas agrupaciones.

Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema. Usualmente

están organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre ellos.

De acuerdo con la definición anterior, a continuación se presenta el diagrama de paquetes correspondiente a la aplicación de este proyecto.

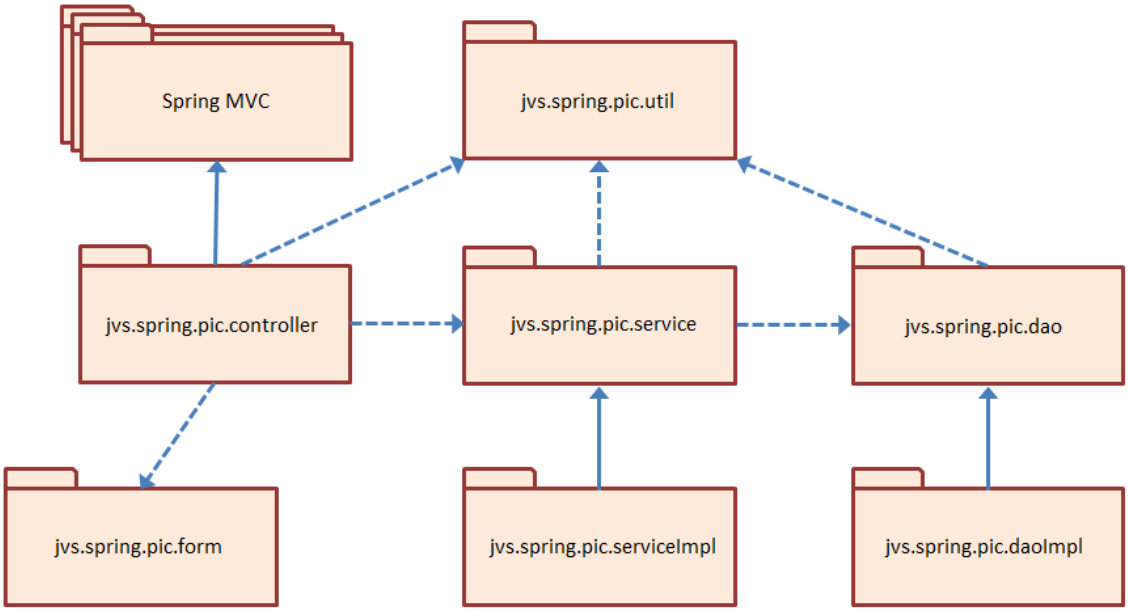


Figura 2.10 Diagrama de paquetes
Fuente:[Elaboración propia]

Descripción de los paquetes que forman la aplicación Web:

SpringMVC - Hace referencia a los paquetes que integran el Framework Spring MVC.

jvs.spring.pic.controller - Contiene las clases que integran la capa de presentación. En este caso, implementan el Framework Spring MVC.

jvs.spring.pic.service - Contiene las interfaces que definen a las clases de la capa de servicios

jvs.spring.pic.serviceImpl - Contiene las clases que implementan la capa de servicios

jvs.spring.pic.dao - Contiene las interfaces que definen la capa de acceso a datos

jvs.spring.pic.daoImpl - Contiene las clases que implementan la capa de acceso a datos

jvs.spring.pic.util - Clases de utilerías que son utilizadas principalmente por las capas de servicio y acceso a datos. Entre ellas se pueden encontrar clases de formato de texto, fechas o de números, o clases de manejo de cadenas, zip, encripción, etc.

jvs.spring.pic.form - Contiene clases de dominio, son utilizadas por todas las capas que integran la aplicación

2.4.7 Diagrama de clases

El diagrama a presete en la figura 2.11, presenta la vista estática del sistema en términos de clases y de las relaciones entre ellas, describiendo también su comportamiento.

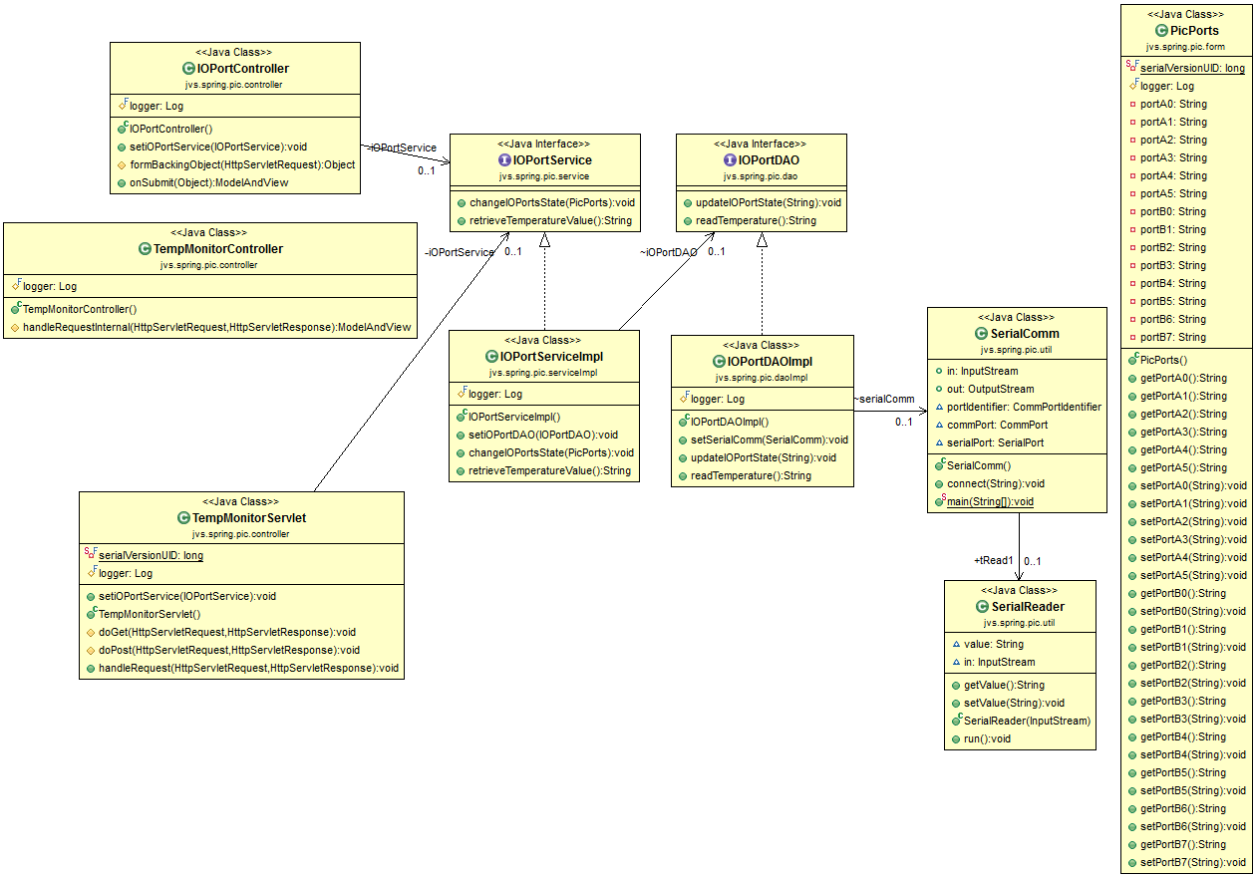


Figura 2.11 Diagrama de clases de la aplicación Web
Fuente:[Elaboración propia]

Capítulo 3. Implementación del sistema

3.1 Implementación del hardware

En los siguientes apartados se describe el código y el cableado para la comunicación entre el módulo USB del microcontrolador con el puerto USB del servidor, y para la obtención de la temperatura indicada por el sensor, que conforman la etapa de hardware del proyecto.

En la figura 3.1 se observan los módulos del diagrama general que corresponden a esta etapa.

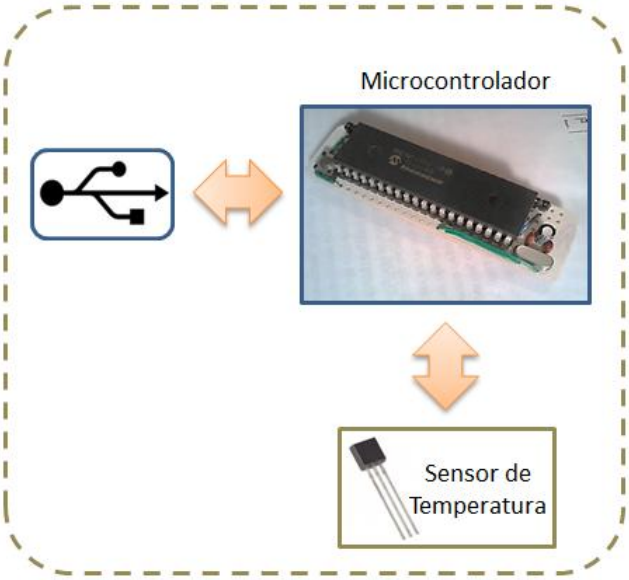


Figura 3.1 Etapa de Hardware
Fuente:[Elaboración propia]

3.1.1 Prueba de lectura del puerto analógico y envío de datos a través del USB

Para establecer la comunicación entre el microcontrolador y el puerto USB se hace uso de un potenciómetro cuyo valor pueda variar manualmente, con la intención de asegurar que la comunicación y respuesta entre éstos funciona adecuadamente. Una vez que esto opere correctamente, se reemplazará el potenciómetro por un sensor de temperatura, a fin de cumplir con el objetivo del proyecto.

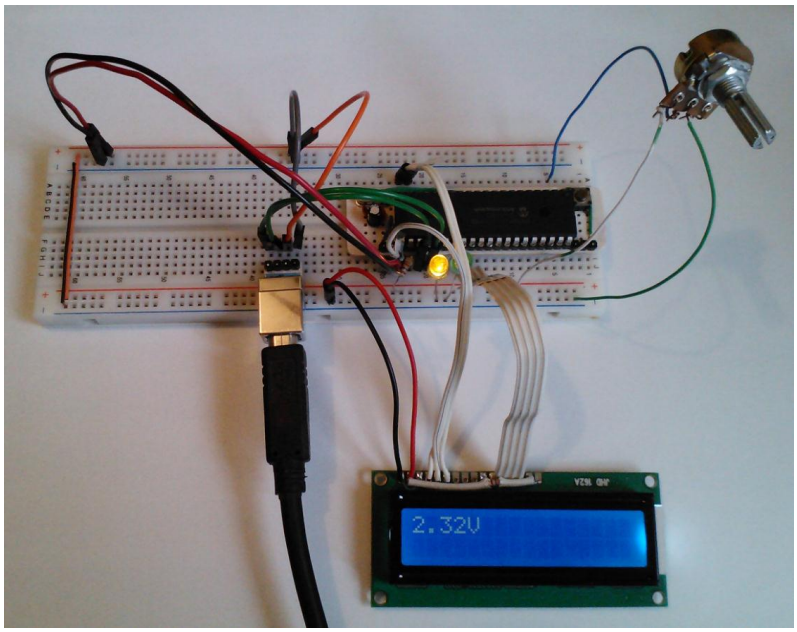


Figura 3.2 Envío de la lectura del POT a través del USB
Fuente:[Elaboración propia]

Para llevar a cabo esta tarea, se ha realizado una sencilla aplicación de prueba: **POT_ADC_USB_TestProject.c**, cuyo código se encuentra disponible en el Apéndice C.

Esta aplicación servirá como base para el desarrollo del proyecto, pues permite probar el módulo ADC del microcontrolador y su comunicación con el puerto USB.

```
while (TRUE) {
    usb_task();
    usb_debug_task();

    if (usb_enumerated()) {

        //Read the analog port///////////////
        q = read_adc();
        p = 5.0 * q / 1023.0;
        printf(lcd_putc, "\f%1.2fV", p);
        //Send the value to the port/////////  sample: '3.08V>'
        printf(usb_cdc_putc, "%1.2fV>", p);
        ////////////////////////////////////

        delay_ms(50);
    }
}
```

Listado 3.1 Programa POT_ADC_USB_TestProject.c
Fuente:[Elaboración propia]

El fragmento de código del Listado 3.1, es la parte más importante de la aplicación de prueba, y en él se puede observar la lectura que realiza el módulo ADC del voltaje presente en el potenciómetro a través de un puerto del microcontrolador configurado como entrada analógica, la conversión a nivel de voltaje que en la siguiente línea se le aplica, y la impresión del resultado tanto en la pantalla LCD como en el puerto USB.

La conversión anterior, es necesaria porque el módulo ADC convierte el valor del voltaje analógico presente en el potenciómetro en un valor digital dentro de un rango de [0 – 1023]. Por lo que, para transformar el valor digital devuelto por el módulo ADC en un valor de voltaje que pueda ser interpretado, se realiza una sencilla regla de tres: se multiplica por 5V, porque los extremos del potenciómetro están conectados al Vss y al GND del protoboard (o sea, 0V y 5Volts son los voltajes de referencia) y después se divide entre 1024, porque el ADC del microcontrolador es de 10-bits. (ADC 2^10 = 1024).

Una vez que se haya cargado el código de la aplicación de prueba en el microcontrolador y conectado al puerto USB del servidor, se podrá observar que es reconocido por el Sistema Operativo como un puerto COM RS-232, debido a la clase CDC (Communication Device Class) que se ha utilizado en su programación.

La instalación del Driver USB CDC en Windows, se encuentra disponible en el Apéndice A del presente documento.

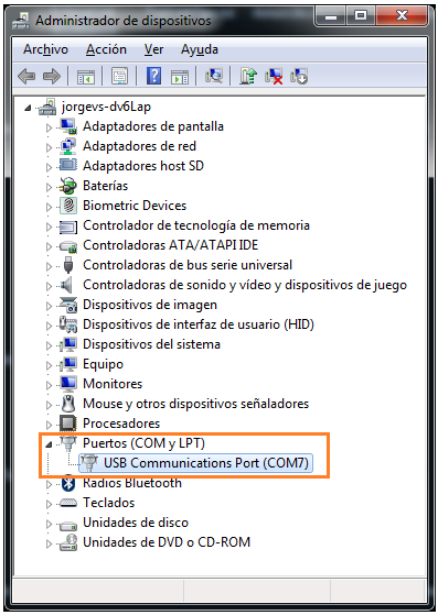


Figura 3.3 Puerto COM RS-232 emulado a través de la interfaz USB
Fuente:[Elaboración propia]

Gracias a la emulación COM sobre el puerto USB se puede hacer uso del HyperTerminal (Figura 3.4) de Windows para verificar los datos que envía el microcontrolador al servidor.

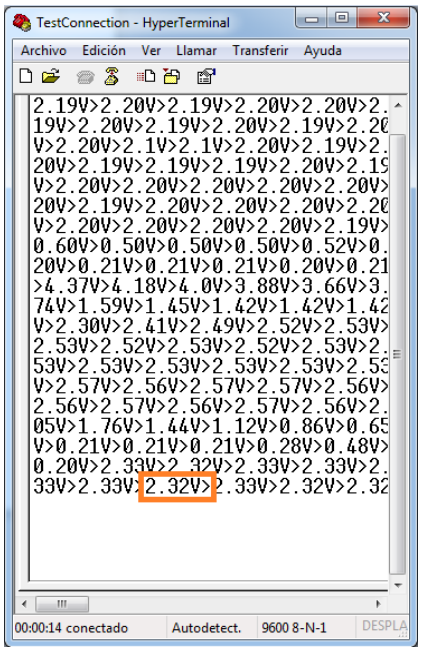


Figura 3.4 HyperTerminal recibiendo datos provenientes del MCU
Fuente:[Elaboración propia]

Como resultado de la prueba efectuada, se puede corroborar lo siguiente:

- Que el módulo USB del microcontrolador es reconocido y enumerado satisfactoriamente por el SO
- La comunicación entre el microcontrolador y el puerto USB funciona correctamente
- Los cambios de valor en el POT de manera manual se ven reflejados en los datos recibidos
- Que la información cumple con el formato esperado (en este ejercicio se esperan Volts y no grados centígrados, por lo que el formato es de la forma "**N.NNV>**")

3.1.2 Lectura del sensor de temperatura y envío de datos a través del USB

En el apartado anterior se realizó una prueba para verificar la lectura que efectúa el módulo ADC sobre el POT y que la información se envía correctamente desde el módulo USB del microcontrolador al puerto USB del servidor; por lo que ahora se realizarán ciertos ajustes para alcanzar el objetivo principal de la etapa de hardware, que consiste en realizar la lectura del sensor de temperatura y enviarla al servidor para ser procesada por la aplicación Web.

Obtención de la temperatura en grados Celsius

El sensor de temperatura LM35 es un sensor cuya tensión de salida es linealmente proporcional a la temperatura en ºC; cada grado centígrado equivale a 10mV. Eso quiere decir que si la salida es de 300mV entonces la temperatura es de 30°C.

El módulo ADC del microcontrolador PIC18F4550 es de 10-bits, por lo que entrega un valor de entre 0-1023 por un voltaje de entrada de 0 a 5V. Por lo que si la lectura es 0 entonces la entrada es de 0V, si la lectura es de 1023 entonces el voltaje de entrada es de 5V.

De manera general, si la lectura del módulo ADC es *val*, entonces el voltaje está dado por:

$$val = read\_adc();$$

$$voltaje = \frac{val \times 5}{1023}$$

La ecuación previa indica el voltaje en Volts; y para obtenerlo en mili volts (mV) se deberá multiplicar por 1000, entonces:

$$voltaje = \frac{val \times 5}{1023} \times 1000$$

Recordemos que 10mV = 1 grado; por lo que para obtener la temperatura se deberá dividirlo entre 10, entonces:

$$voltaje = \frac{val \times 5}{1023} \times 100$$

Simplificando un poco más, se tiene lo siguiente:

(3.1)

$$voltaje = \frac{val}{1023} \times 500$$

La ecuación 3.1 es la que se ha de utilizar en el desarrollo de la aplicación.

```
//Read the analog port///////////////
val = read_adc();
temp = ((val / 1023.0)*500);

if (tempAux != temp) {
    printf(lcd_putc, "\f%2.1f C", temp);
    //Send the value to the port/////////  sample: '27.9>'
    printf(usb_cdc_putc, "%2.1f>", temp);
    tempAux = temp;
}
////////////////////////////////////
```

Listado 3.2 Obtención de la temperatura en grados Celsius
Fuente:[Elaboración propia]

El valor de temperatura obtenido, se imprime en la pantalla LCD y se envía a través del puerto USB al servidor para que éste sea leído y procesado por la aplicación Web. Todo esto dentro de un bucle infinito para garantizar que la lectura del valor de la temperatura es en tiempo real.

Como se puede apreciar en el código, el formato de la información consiste en el establecido durante la etapa de diseño, de la forma: "**NN.N>**"

Con ayuda del simulador ISIS se puede verificar la funcionalidad del código una vez compilado. El codigo del programa desarrollado durante esta etapa **PicControlWebApp.c**, se encuentra disponible en el Apéndice C del presente documento.

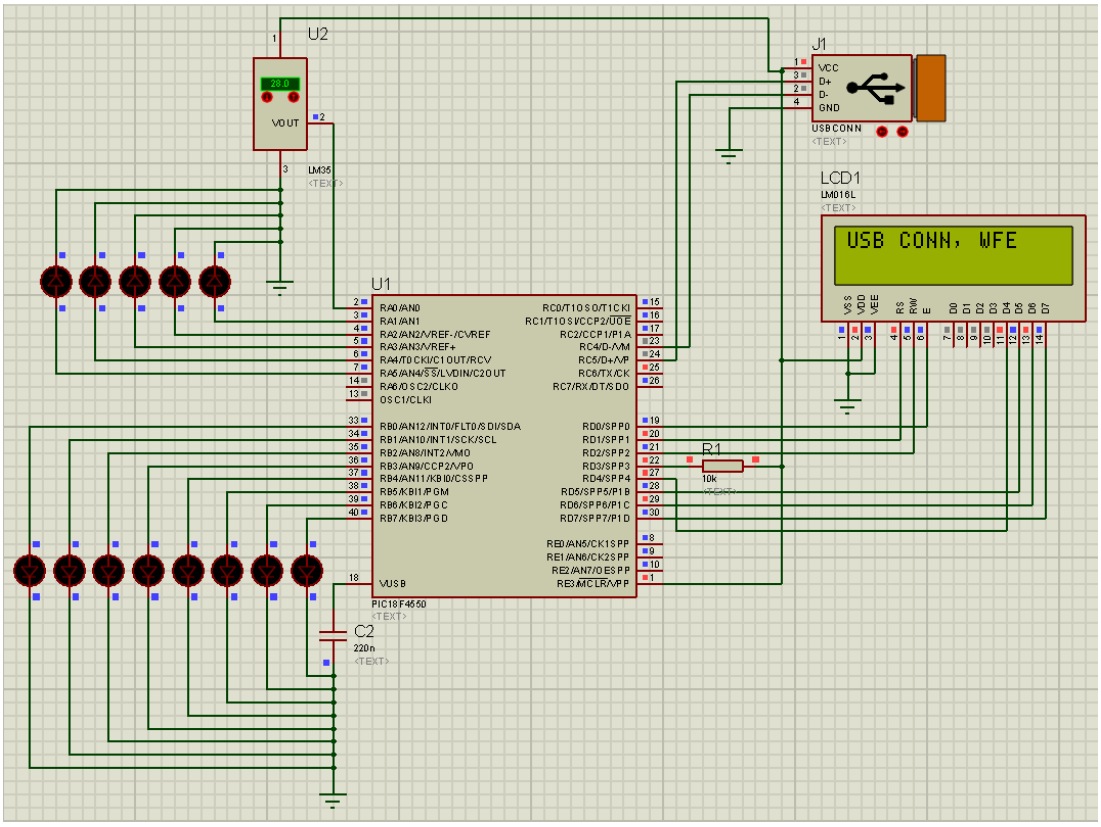


Figura 3.5 Simulación de la obtención de temperatura en ISIS
Fuente:[Elaboración propia]

Otra forma en que se puede verificar el funcionamiento del código, sería cargarlo en el microcontrolador, conectarlo al puerto USB del servidor y observar la información que éste envía haciendo uso del HyperTerminal de Windows.

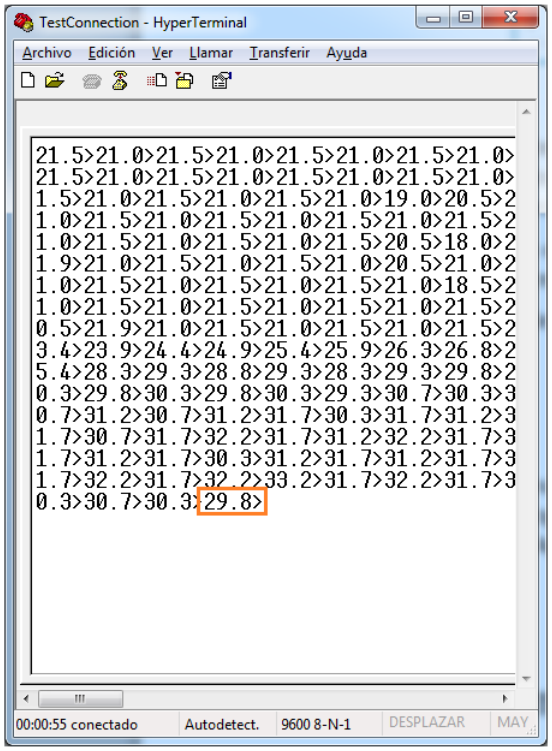


Figura 3.6 HyperTerminal recibiendo la lectura de la temperatura proveniente del MCU
Fuente:[Elaboración propia]

3.2 Implementación del software

En los siguientes apartados se profundizará en el desarrollo de la aplicación Web y en la comunicación que existe entre ella y el puerto USB del servidor, que conforman la etapa de software del proyecto.

En la figura 3.7 se observan los módulos del diagrama general que corresponden a esta etapa.

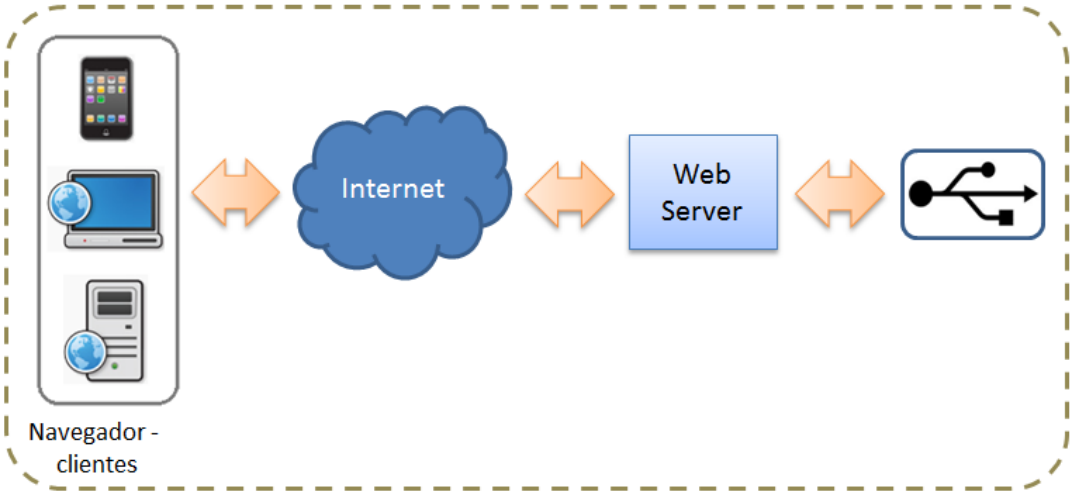


Figura 3.7 Diagrama general del sistema
Fuente:[Elaboración propia]

3.2.1 Construcción y administración de dependencias de la aplicación

Apache Maven es utilizado en este proyecto para la construcción y administración de dependencias de software. Para que el proyecto soporte Maven, es necesario crear un proyecto en Eclipse del tipo:"**Maven Dynamic Web Project**" como la arquitectura base de la aplicación. La creación de este tipo de proyectos, se puede consultar en el Apéndice B.

En el archivo **pom.xml** es donde se especifican los artefactos a utilizar tanto en la construcción, como en el desarrollo de la aplicación (dependencias y sus respectivas versiones).

PicControlWebApp/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>jvs.spring.pic</groupId>
    <artifactId>PicControlWebApp</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>PicControlWebApp Maven Webapp</name>
    <url>http://maven.apache.org</url>

    <properties>
        <spring.version>2.5.6.SEC03</spring.version>
        <java.version>1.6</java.version>
        <servlet.version>2.5</servlet.version>
        <jstl.version>1.1.2</jstl.version>
        <jUnit.version>3.8.1</jUnit.version>
        <tiles.version>2.0.7</tiles.version>
        <RXTXcomm.version>2.1.7</RXTXcomm.version>

    </properties>

    <dependencies>

        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
```

Listado 3.3 Archivo pom.xml de Maven
Fuente:[Elaboración propia]

Los artefactos de Spring, Tiles, RXTXComm, etc que son utilizados en la aplicación, son declarados en este archivo y a través de la contrucción con Maven, éstos serán descargados desde el repositorio local (o el de internet) para obtener un archivo que empaquete a todos los componentes de la aplicación, y así poder ser distribuida o desplegada en un servidor de aplicaciones como Tomcat.

3.2.2 Obtención del valor de la temperatura enviado por el microcontrolador

De acuerdo con el diseño que se realizó previamente para la obtención de la temperatura, se ha desarrollado un Thread que se conecte al puerto de comunicación y que realice una lectura constante de la información enviada por el microcontrolador.

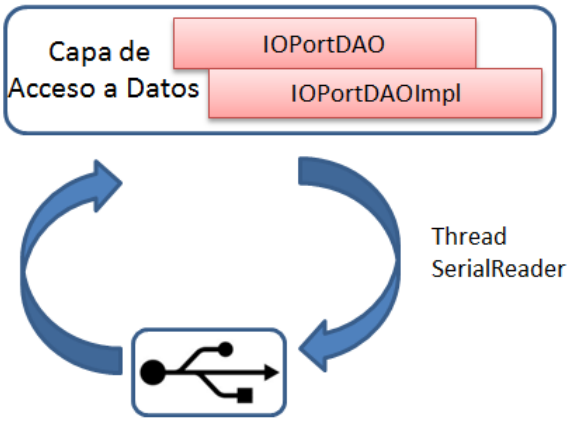


Figura 3.8 Thread SerialReader que poolea constantemente el puerto USB
Fuente:[Elaboración propia]

La clase **SerialComm** hace uso del Thread **SerialReader**, cuya funcionalidad puede comprobarse a través de la ejecución de su método **main()**.

En el Listado 3.4 a continuación, se muestrá el código perteneciente al método **main()** de la clase **SerialComm**. Para fines prácticos se ha omitido el resto del código perteneciente a esta clase, el cuál se encuentra disponible en el Apéndice C del presente trabajo.

```
public static void main(String[] args) {
    SerialComm serialComm = new SerialComm();

    try {
        serialComm.connect("COM7");

        byte[] bout = new byte[5];

        bout[0] = (byte) 'A'; // sample: 'A'
        bout[1] = (byte) '1'; // sample: '1'
        bout[2] = (byte) ':'; // sample: ':'
        bout[3] = (byte) '1'; // sample: '1'
        bout[4] = (byte) '<'; // sample: '<'
        serialComm.out.write(bout);

        bout[0] = (byte) 'A'; // sample: 'A'
        bout[1] = (byte) '3'; // sample: '3'
        bout[2] = (byte) ':'; // sample: ':'
        bout[3] = (byte) '1'; // sample: '1'
        bout[4] = (byte) '<'; // sample: '<'
        serialComm.out.write(bout);

        bout[0] = (byte) 'A'; // sample: 'A'
        bout[1] = (byte) '5'; // sample: '5'
        bout[2] = (byte) ':'; // sample: ':'
        bout[3] = (byte) '1'; // sample: '1'
        bout[4] = (byte) '<'; // sample: '<'
        serialComm.out.write(bout);

        for (int i = 0; i < 5; i++) {
            System.out.println("Reading tRead1: " + serialComm.tRead1.getValue());
            Thread.sleep(500);
        }

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        System.out.println("Closing port and ending thread");
        serialComm.tRead1 = null;

    }
}
```

Listado 3.4 Test de comunicación
Fuente:[Elaboración propia]

El objetivo de este pequeño fragmento de código (listado 3.4) consiste en enviar 3 instrucciones al microcontrolador, para encender los leds conectados a tres terminales del puerto A: A1, A3 y A5

Después de encender los tres leds realiza la lectura de 5 bloques de información enviadas por el microcontrolador, cuyos valores son impresos a través de la consola standard.

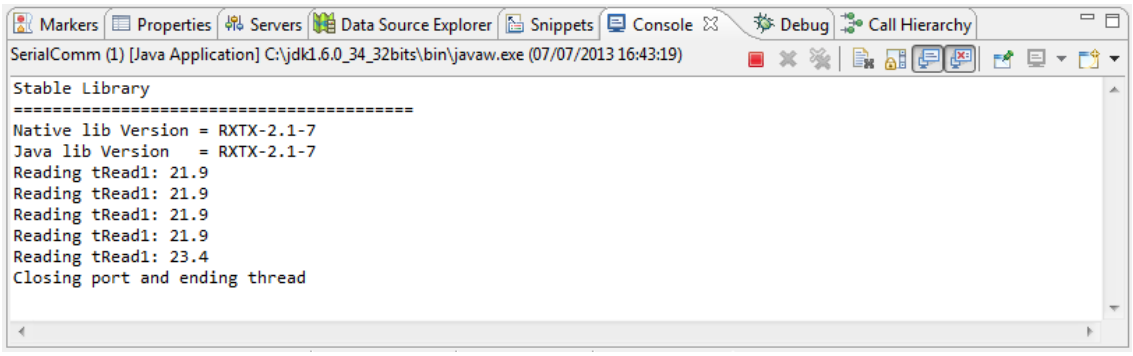


Figura 3.9 Salida estándard de la consola de Java
Fuente:[Elaboración propia]

Una vez que el valor de la temperatura ha sido obtenido, es el turno de ver los componentes que se encargan de presentarla al usuario.

3.2.3 Presentación del valor de la temperatura al usuario

Tal como se mencionó en el durante la etapa de diseño, es necesario la construcción de un Servlet que acceda a la variable de la capa de acceso a datos a través de la capa de servicios, a fin de obtener su valor. Éste a su vez, será llamado desde una página JSP que contiene un script de jQuery y que hace uso de rutinas Ajax para llamar al Servlet y mostrar los cambios en la pantalla sin la necesidad de que se refresque o actualice la página en el navegador.

El diagrama de la figura 3.10 muestra los componentes y su interacción que permiten mostrar el valor de la temperatura al usuario, así como sus variaciones.

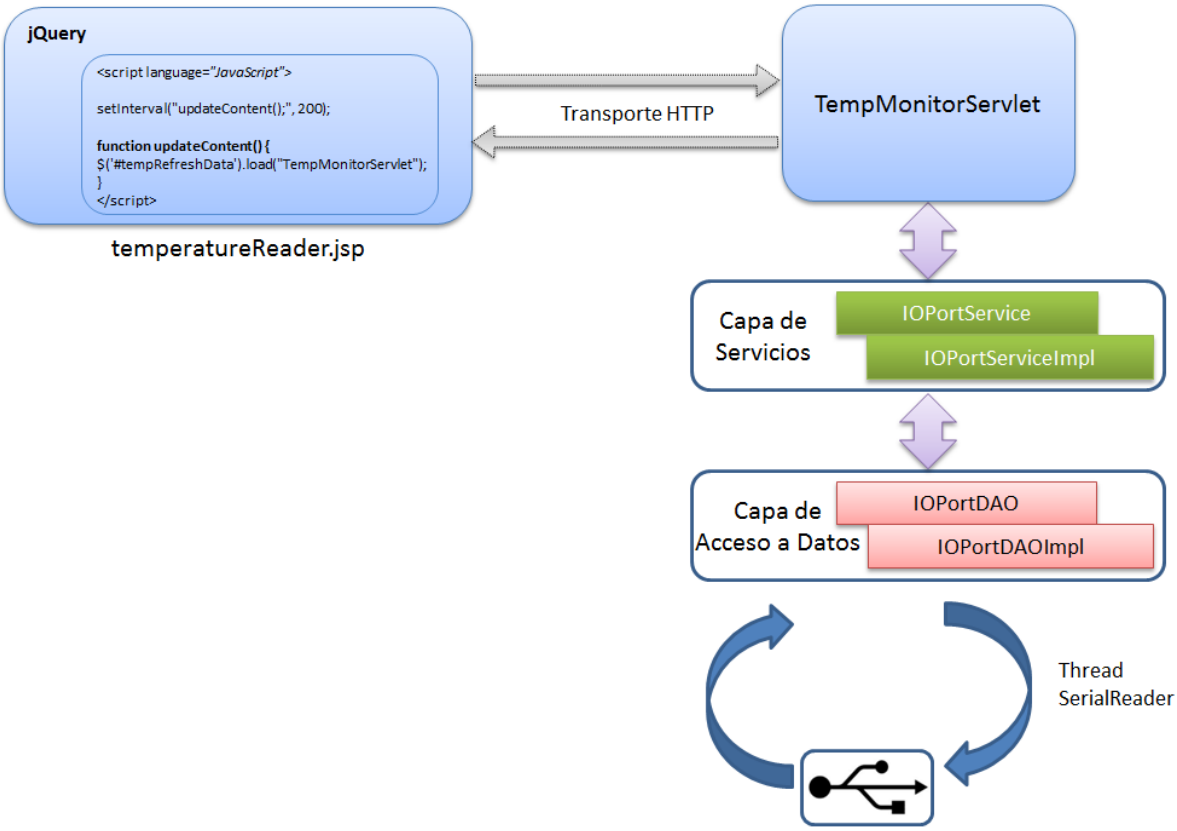


Figura 3.10 Interacción de componentes para la obtención de la temperatura
Fuente:[Elaboración propia]

La página **temperatureReader.jsp** contiene un script que realiza llamadas al Servlet **TempMonitorServlet.java** cada 250 milisegundos, permitiendo así que cada este periodo de tiempo sea obtenido la variable que almacena el valor de la temperatura.

Para que el Servlet **TempMonitorServlet.java** pueda hacer uso de las capas de servicio, cuyos componentes se encuentran declarados como Beans de Spring es necesario inyectárselos declarándolos en los archivos de configuración de dicho framework.

El proceso para inyectar un bean de Spring en un Servlet se detalla a continuación:

1. Implementar la interface **HttpRequestHandler**

El Servlet debe implementar la interface **org.springframework.web.HttpRequestHandler**, así como también realizar la implementación del método **handleRequest()** y escribir el código a ejecutar, tal como si se tratase de del método **doPost().**

2. Declarar el Servlet como un Bean de Spring

En el archivo de configuración de Spring se declara el Servlet, así como el o los objetos a inyectarse.

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="h
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">


    <bean id="iOPortService" class="jvs.spring.pic.serviceImpl.IOPortServiceImpl">
        <property name="iOPortDAO" ref="iOPortDAO" />
    </bean>

    <bean id="iOPortDAO" class="jvs.spring.pic.daoImpl.IOPortDAOImpl">
        <property name="serialComm" ref="serialComm" />
    </bean>

    <bean id="serialComm" class="jvs.spring.pic.util.SerialComm" />


    <!-- TempMonitorServlet implements HttpRequestHandler so that Spring Beans can be injected, since it is a Servlet -->
    <bean id="TempMonitorServlet" class="jvs.spring.pic.controller.TempMonitorServlet">
        <property name="iOPortService" ref="iOPortService" />
    </bean>
</beans>
```

Listado 3.5 Contenido del archivo applicationContext.xml
Fuente:[Elaboración propia]

3. Declarar el Servlet en el archivo **web.xml**

La declaración del Servlet se lleva a cabo de la misma manera que declaramos a un Servlet común y corriente, pero indicando que el tipo de Servlet es de la clase **org.springframework.web.context.support.HttpRequestHandlerServlet**

web.xml

```
</servlet>

<servlet-mapping>
  <servlet-name>springapp</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>

<session-config>
  <session-timeout>30</session-timeout>
</session-config>

<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>default.html</welcome-file>
  <welcome-file>default.htm</welcome-file>
  <welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
  <description></description>
  <display-name>TempMonitorServlet</display-name>
  <servlet-name>TempMonitorServlet</servlet-name>
  <servlet-class>org.springframework.web.context.support.HttpRequestHandlerServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>TempMonitorServlet</servlet-name>
  <url-pattern>/TempMonitorServlet</url-pattern>
</servlet-mapping>

</web-app>
```

Listado 3.6 Contenido del archivo web.xml
Fuente:[Elaboración propia]

Después de éstos pasos, el Servlet podrá hacer uso de los Beans de Spring que le han sido inyectados, y con esto, el Servlet de la aplicación tendrá acceso a los bean que forman parte de la capa de servicios, para que a través de ellos pueda llegar a la capa de acceso a datos.

3.2.4 Implementación de la interfaz de usuario

De acuerdo al diseño especificado para el desarrollo de la aplicación, es necesario elaborar un esquema que cuente con las siguientes secciones o regiones: Header, Menu, Body y Footer, conforme al siguiente bosquejo:

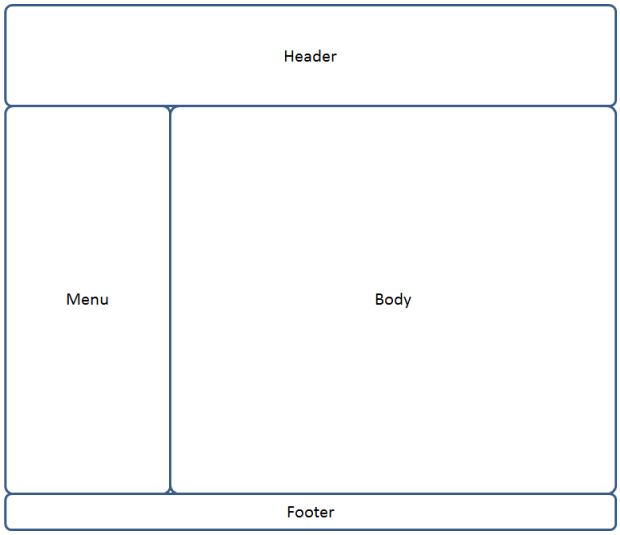


Figura 3.11 Esquema de la interfaz de usuario
Fuente:[Elaboración propia]

Por lo tanto, la página que indica la definición de las regiones que se estará utilizando en la aplicación se define en el código HTML perteneciente al listado 3.7.

baseLayout.jsp

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<link href="css/styles.css" rel="stylesheet" type="text/css" />

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><tiles:insertAttribute name="title" ignore="true" /></title>
</head>
<body>
    <table border="0" cellpadding="0" cellspacing="0" align="left" width="80%" height="100%">

        <tr height="72px">
            <td width="100%" valign="middle" align="left" colspan="2"><tiles:insertAttribute name="header" /></td>
        </tr>

        <tr height="85%">
            <td width="20%" valign="top" align="left"><tiles:insertAttribute name="menu" /></td>
            <td width="80%" valign="top" align="left"><tiles:insertAttribute name="body" /></td>
        </tr>

        <tr height="5%">
            <td width="100%" valign="middle" align="left" colspan="2"><tiles:insertAttribute name="footer" /></td>
        </tr>

    </table>
</body>
</html>
```

Listado 3.7 Contenido del archivo baseLayout.jsp
Fuente:[Elaboración propia]

Todas las páginas JSP que hagan uso de Apache Tiles, deberán de contar con la siguiente taglib

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
```

Haciendo uso de las etiquetas de Apache Tiles se define la plantilla base y a aquellas que la extenderán para establecer cuáles páginas permanecerán estáticas y cuáles cambiarán dinámicamente en las regiones definidas previamente (listado 3.8).

tiles-defs.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
"http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
    <definition name="tileBaseLayout" template="/tiles/baseLayout.jsp">
        <put-attribute name="title" value="Template" />
        <put-attribute name="header" value="/tiles/header.jsp" />
        <put-attribute name="menu" value="/tiles/menu.jsp" />
        <put-attribute name="body" value="/tiles/body.jsp" />
        <put-attribute name="footer" value="/tiles/footer.jsp" />
    </definition>

    <definition name="tileIOPortOperations" extends="tileBaseLayout">
        <put-attribute name="title" value="IOPort Operations" />
        <put-attribute name="body" value="/jsp/iOPortOperations.jsp" />
    </definition>

    <definition name="tileTemperatureReader" extends="tileBaseLayout">
        <put-attribute name="title" value="Temperature Reader" />
        <put-attribute name="body" value="/jsp/temperatureReader.jsp" />
    </definition>
</tiles-definitions>
```

Listado 3.8 Contenido del archivo tiles-defs.xml
Fuente:[Elaboración propia]

El atributo “name” de la etiqueta “put” especifica la región en la plantilla base, en la cuál su correspondiente página especificada por el atributo “value” será desplegada.

En este proyecto, la region “header” será ocupada por la página header.jsp, la región “menu” por menu.jsp, la región “body” por body.jsp y la región “footer” por footer.jsp.

Como se puede observar en las definiciones Tiles: “**tileIOPortOperations**” y “**tileTemperatureReader**” (que extienden de la plantilla base “**tileBaseLayout**”), las únicas

secciones que cambiarán cuando el usuario solicite una página diferente serán las secciones “title” y “body” .

Para integrar Apache Tiles con Spring MVC, es necesario agregar su definición en el archivo de configuración **springapp-servlet.xml** (listado 3.9).

springapp-servlet.xml

```
        <property name="commandClass" value="jvs.spring.pic.form.PicPorts" />
        <property name="sessionForm" value="true" />

        <property name="formView" value="tileIOPortOperations" />
        <property name="successView" value="tileIOPortOperations" />
    </bean>

    <bean name="/temperatureReader.htm" class="jvs.spring.pic.controller.TempMonitorController" />


    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="messages" />
    </bean>

    <bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
        <property name="definitions">
            <list>
                <value>/WEB-INF/tiles-defs.xml</value>
            </list>
        </property>
    </bean>

    <!-- Declare the Resolver -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.tiles2.TilesView" />
    </bean>

</beans>
```

Listado 3.9 Contenido del archivo springapp-servlet.xml
Fuente:[Elaboración propia]

Al ejecutar la aplicación, la siguiente pantalla será desplegada. (El borde de la tabla se ha establecido a un valor de “1” con la intención de hacer más clara la separación entre las regiones).

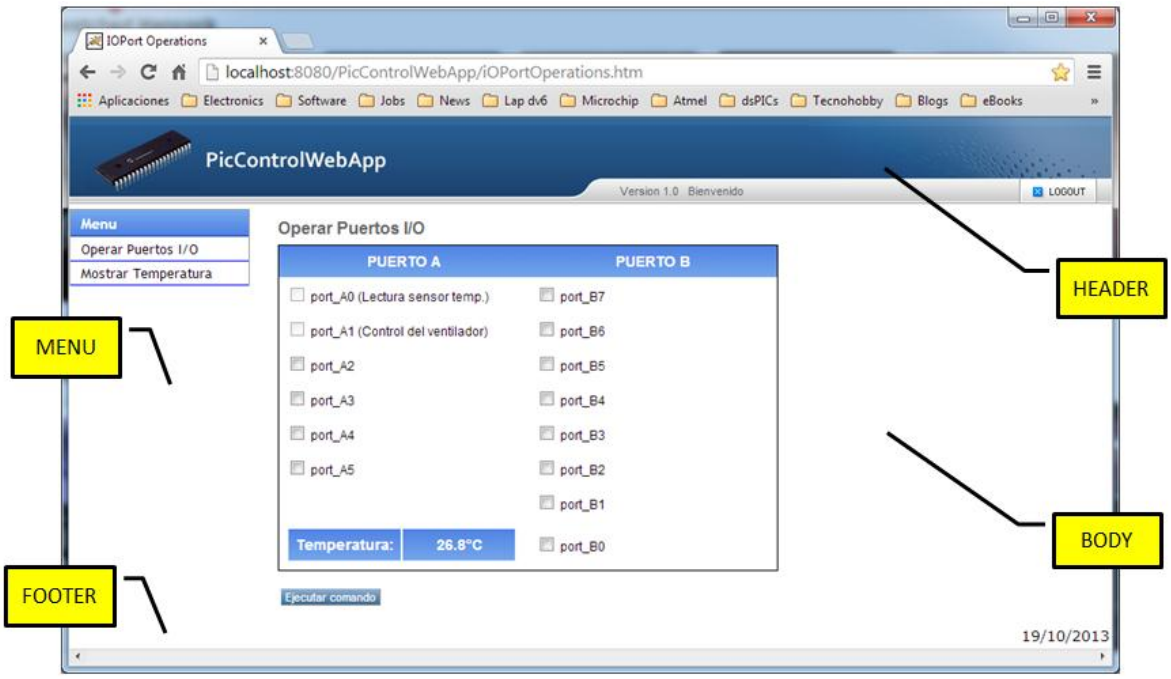


Figura 3.12 Página principal de la aplicación Web
Fuente:[Elaboración propia]

Dando clic en los links del menú izquierdo, las secciones “title” y “body” son las únicas que cambiarán de acuerdo a las definiciones Tiles establecidas.

La página menu.jsp contiene el código del listado 3.10.

menu.jsp

```
        collapseprev : true, //Collapse previous content (so only one open at any time)? true/false
        defaultexpanded : [], //index of content(s) open by default [index1, index2, etc] [] denotes no content
        onemustopen : false, //Specify whether at least one header should be open always (so never all headers closed)
        animatedefault : false, //Should contents open by default be animated into view?
        persiststate : true, //persist state of opened contents within browser session?
        toggleclass : [ "", "" ], //Two CSS classes to be applied to the header when it's collapsed and expanded, respectivel
        togglehtml : [ "suffix",
                "<img src='images/menu/plus.gif' class='statusicon' />",
                "<img src='images/menu/minus.gif' class='statusicon' />" ], //Additional HTML added to the header when it's c
        animatespeed : "fast", //speed of animation: integer in milliseconds (ie: 200), or keywords "fast", "normal", or "slo
        oninit : function(headers, expandedindices) { //custom code to run when headers have initalized
            //do nothing
        },
        onopenclose : function(header, index, state, isuseractivated) { //custom code to run whenever a header is opened or c
            //do nothing
        }
    })
</script>

</head>

<body>
    <div class="glossymenu">
        <a class="menuitem submenuheader">Menu</a>
        <div class="submenu">
            <ul>
                <li><a href="${pageContext.request.contextPath}/iOPortOperations.htm">IOPort Operations</a></li>
                <li><a href="${pageContext.request.contextPath}/temperatureReader.htm">Temperature Reader</a></li>
            </ul>
        </div>
    </div>
</body>
</html>

```

Listado 3.10 Contenido del archivo menu.jsp
Fuente:[Elaboración propia]

Dado lo anterior, las definiciones Tiles serán invocadas por el direccionamiento que realizan las clases controladoras establecidas en el archivo de configuración **springapp-servlet.xml**

springapp-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">


    <!-- the application context definition for the springapp DispatcherServlet -->
    <bean name="/iOPortOperations.htm" class="jvs.spring.pic.controller.IOPortController">
        <property name="iOPortService" ref="iOPortService" />

        <property name="commandName" value="picPorts" />
        <property name="commandClass" value="jvs.spring.pic.form.PicPorts" />
        <property name="sessionForm" value="true" />

        <property name="formView" value="tileIOPortOperations" />
        <property name="successView" value="tileIOPortOperations" />
    </bean>

    <bean name="/temperatureReader.htm" class="jvs.spring.pic.controller.TempMonitorController" />


    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="messages" />
    </bean>

    <bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
        <property name="definitions">
            <list>
                <value>/WEB-INF/tiles-defs.xml</value>
            </list>
        </property>
    </bean>

    <!-- Declare the Resolver -->
```

Listado 3.11 Contenido del archivo springapp-servlet.xml
Fuente:[Elaboración propia]

Así, cuando se de clic sobre el menú "**IOPort Operations**", se desplegará la pantalla indicada en la figura 3.13.

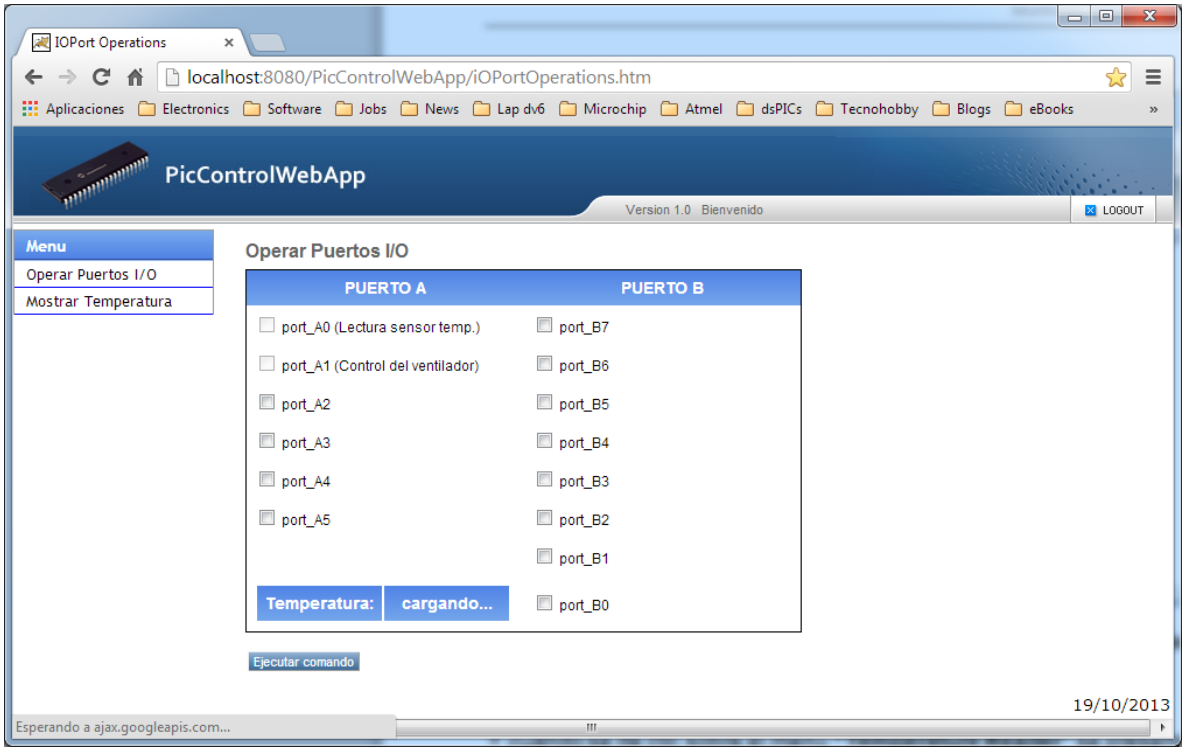


Figura 3.13 Página principal de la aplicación Web
Fuente:[Elaboración propia]

Y cuando se de clic sobre el menú **"Temperature Reader"** se presentará la de la figura 3.14.

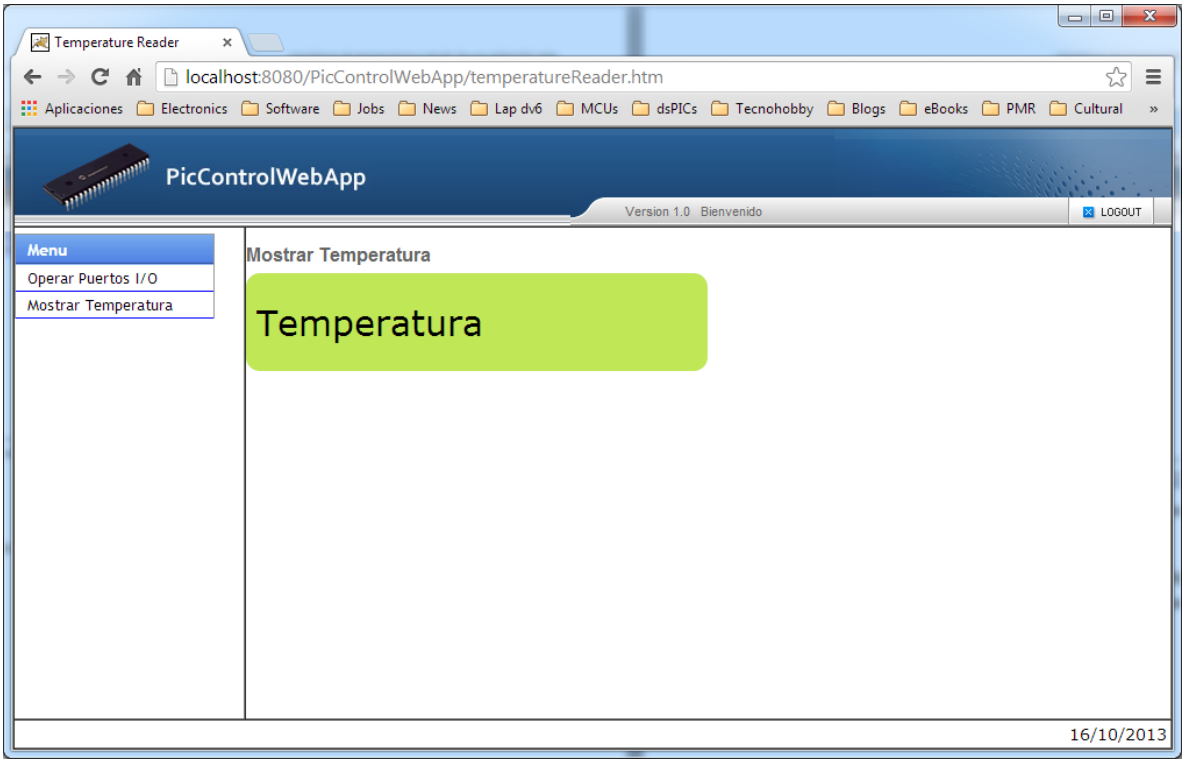


Figura 3.14 Página que muestra la lectura de temperatura
Fuente:[Elaboración propia]

Como se mencionó anteriormente, el borde de la tabla dentro de la página que establece el esquema de las secciones se ha dejado visible (con un valor de “1”) para hacer más clara la separación entre sus regiones.

Capítulo 4. Integración de las etapas de hardware y software

En este apartado se lleva a cabo la integración de las 2 etapas que se desarrollaron anteriormente, y verificar el correcto funcionamiento de todo el sistema.

En primer lugar se conecta el microcontrolador a la PC por medio del puerto USB, esperando a ser reconocido y enumerado por el sistema operativo.

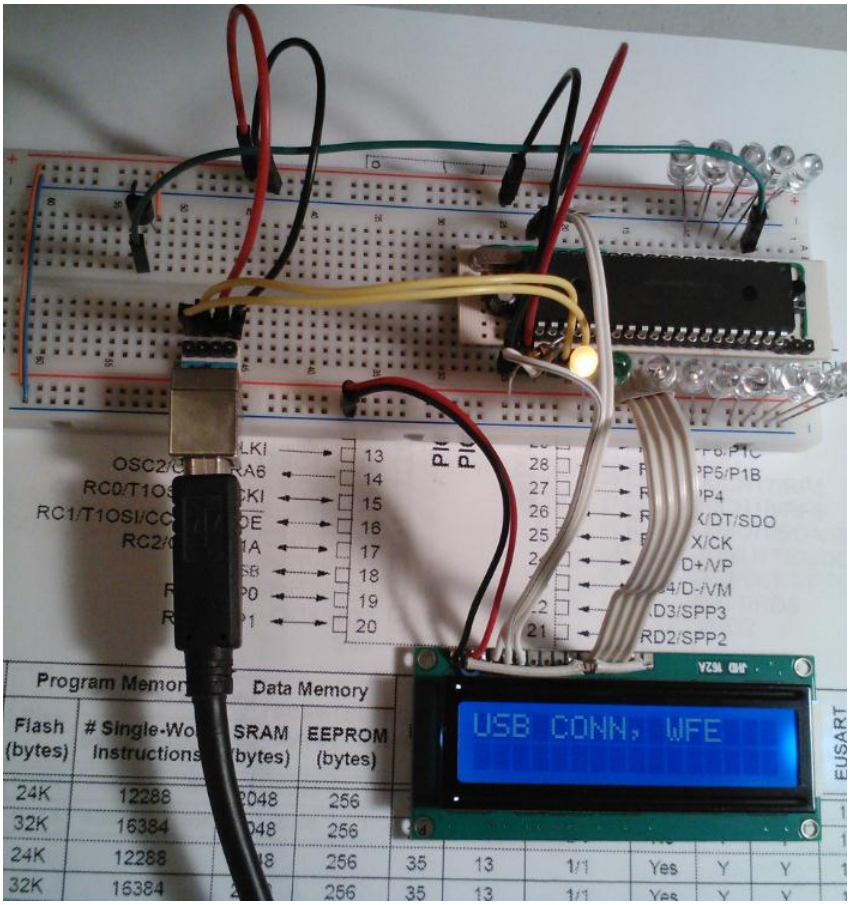


Figura 4.1 Conexión y enumeración del MCU por la PC
Fuente:[Elaboración propia]

Una vez que esto suceda, se podrá visualizar el valor de la lectura del sensor de temperatura a través del display. Esto indica que la lectura del sensor es correcta, así como el envío de esta información a través del puerto USB del servidor.

Con esto se verifica que la parte de Hardware trabaja adecuadamente.

Ahora es turno arrancar el servidor Web y de desplegar la aplicación Web para echar a andar la parte del software. Para esto, se copia la librería **RXTXcomm.jar** en el directorio **\jre\lib\ext** del **JDK** presente en el servidor.

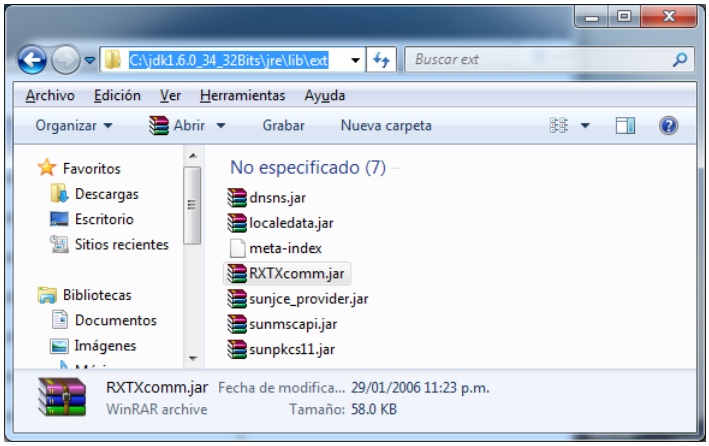


Figura 4.2 Directorio /jre/lib/ext del JDK local
Fuente:[Elaboración propia]

Para arrancar o inicializar el servidor Tomcat se ejecuta el archivo **startup.bat,** que se encuentra en el directorio **\bin** de su directorio de instalación.

Nota: Es necesario tener declarada la variable de ambiente **JAVA_HOME** apuntando al directorio donde se encuentra instalado el **JDK** de Java para que Tomcat funcione correctamente.

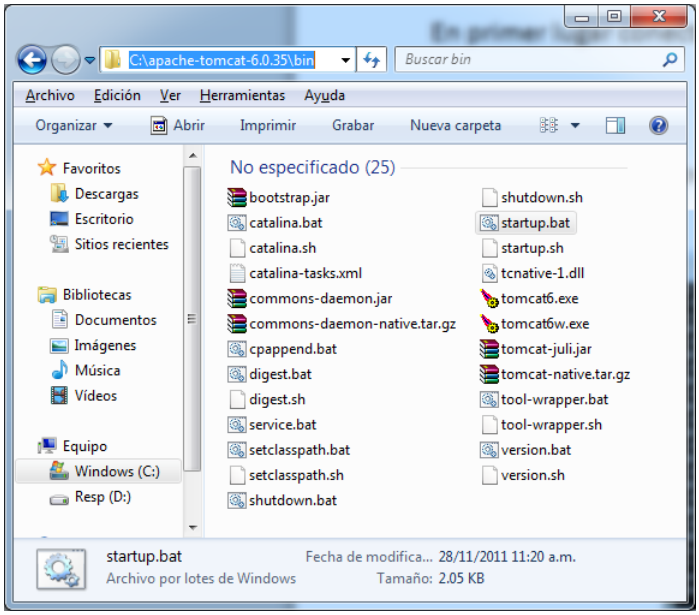


Figura 4.3 Directorio /bin de Tomcat
Fuente:[Elaboración propia]

Al ejecutar el archivo, una ventana de DOS se abrirá, donde se podrá visualizar la información del arranque del servidor. Esta información es importante, sobre todo para identificar si existió algún problema durante su inicialización y de las aplicaciones que se encuentren instaladas en él.

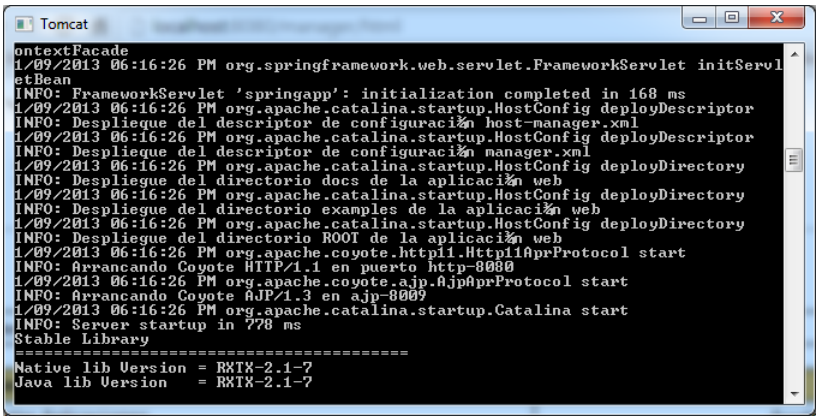


Figura 4.4 Arranque del servidor Tomcat
Fuente:[Elaboración propia]

Tan pronto como se hayan terminado de ejecutar los procesos de arranque, se abrirá el navegador y se deberá escribir la dirección junto con el puerto donde está escuchando el servidor Web, que en este caso es de manera local (localhost) y el puerto es el 8080 que es el que utiliza Tomcat por default.

Así, se visualizará en el navegador la página de inicio de Tomcat:

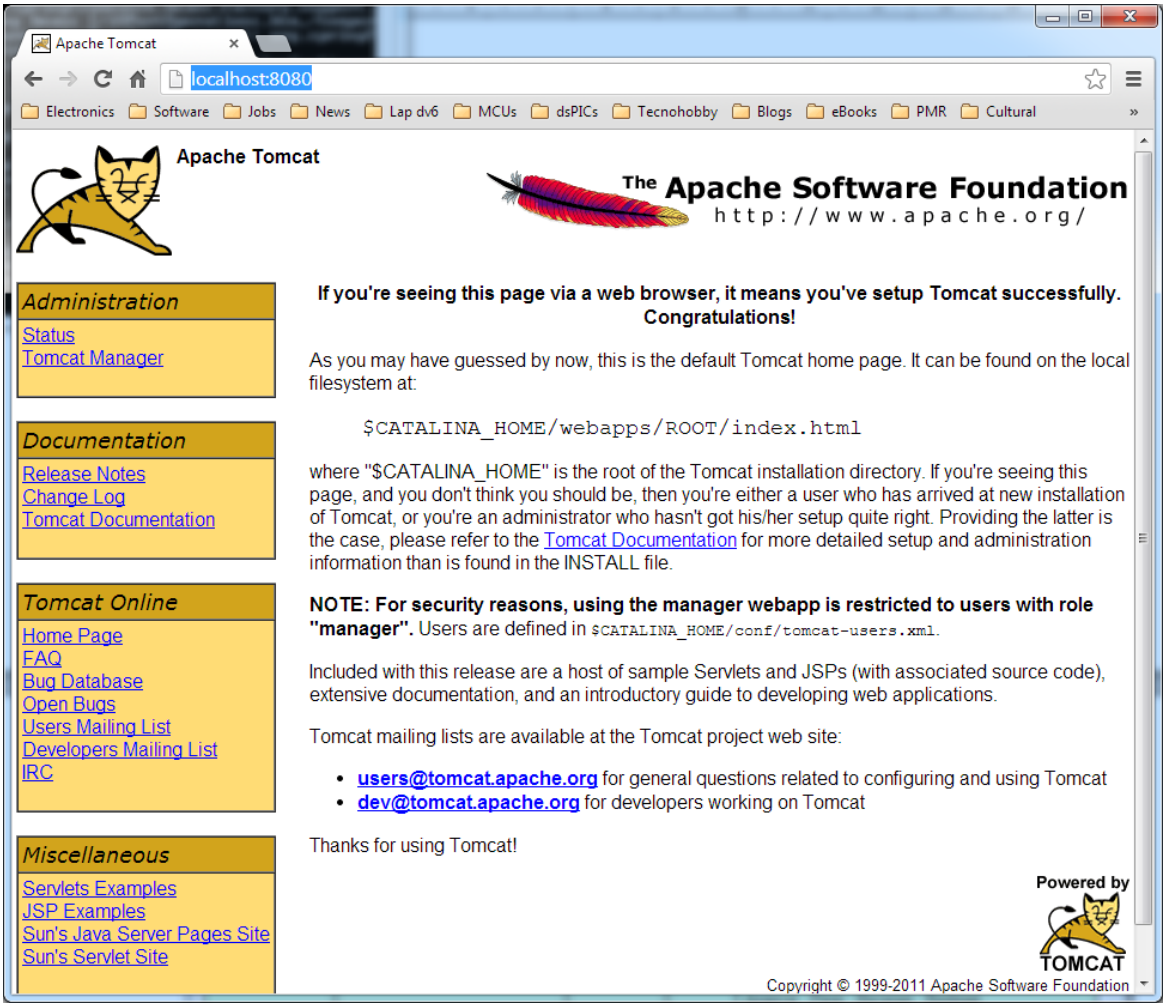


Figura 4.5 Página principal del servidor Tomcat
Fuente:[Elaboración propia]

Click en el menu **Tomcat Manager** para que Tomcat muestre la página de administración.

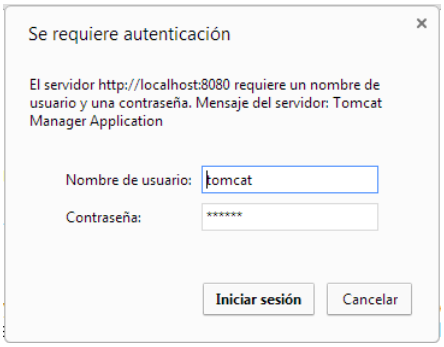


Figura 4.6 Autenticación de Tomcat
Fuente:[Elaboración propia]

La aplicación pedirá ingresar el user/pwd configurado para la administración de Tomcat.

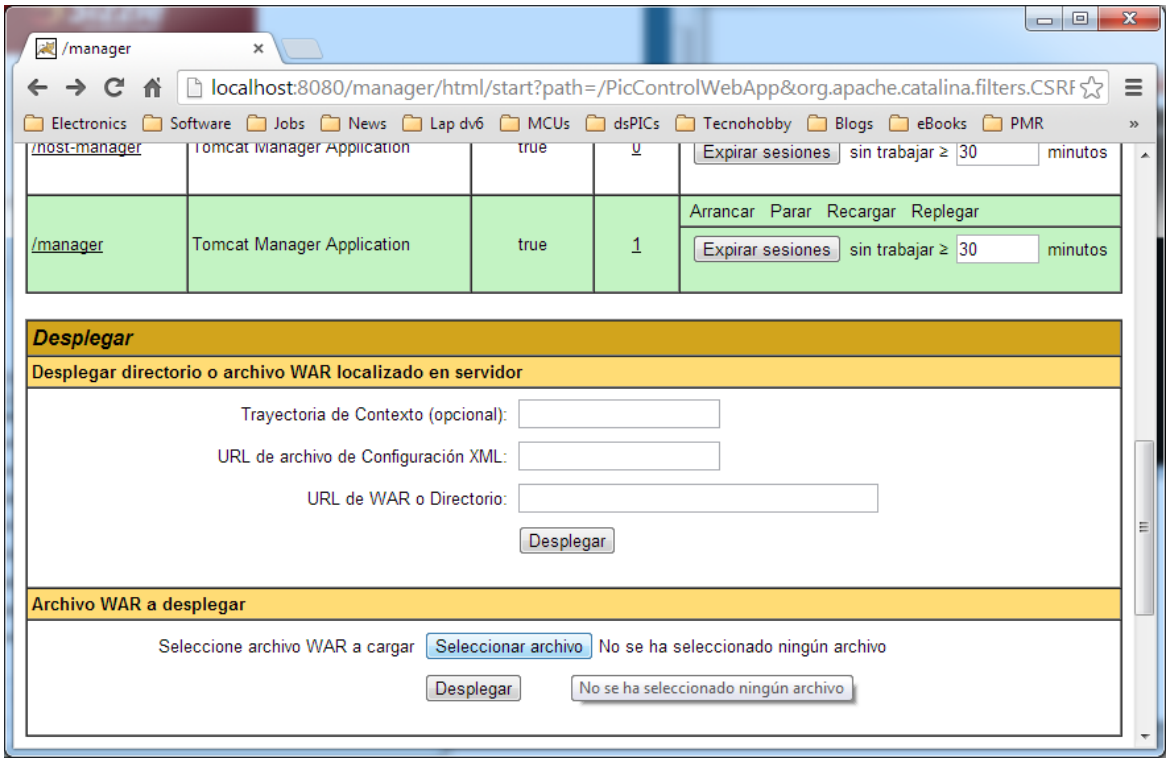


Figura 4.6 Página de administración de Tomcat
Fuente:[Elaboración propia]

Luego, dar click en el botón **Seleccionar Archivo** de la sección "**Archivo WAR a desplegar**" y buscar el archivo WAR que ha sido creado durante la compilación de la aplicación con Eclipse (PicControlwebApp.war)

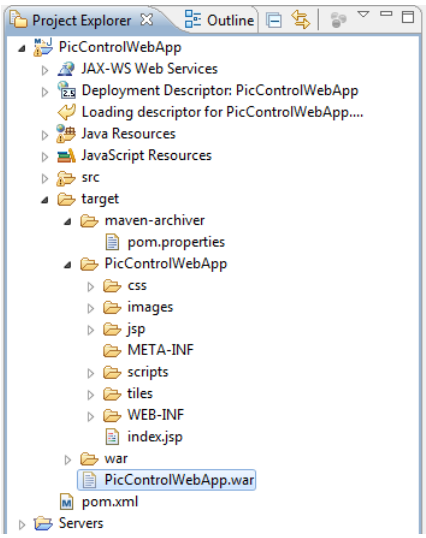


Figura 4.7 Selección del archivo war
Fuente:[Elaboración propia]

Una vez localizado, dar click en el botón **Desplegar**.

Al terminar los pasos anteriores, es necesario escribir en el navegador la ruta donde se encuentra la aplicación, que es la siguiente: http://localhost:8080/PicControlWebApp/

Con esto, se presentará la página definida como página principal dentro de la programación de la aplicación.

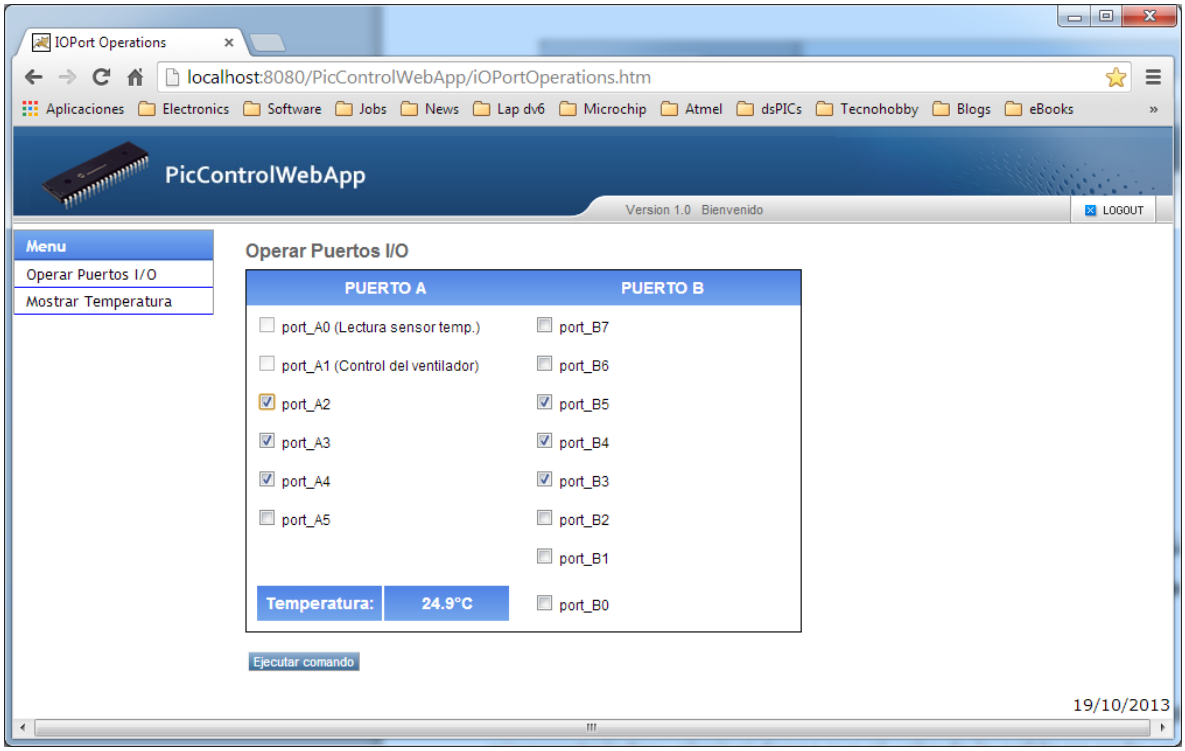


Figura 4.8 Pantalla para realizar operaciones sobre los puertos de Entrada/Salida del microcontrolador
Fuente:[Elaboración propia]

A través de esta página el usuario podrá seleccionar los terminales de los puertos A y B del microcontrolador y elegir si desea que el valor de la salida en cada uno de éstos sea un cero o un uno lógico, que será visualizado a través del encendido y apagado de los leds conectados a ellos.

También, desde esta página se podrá visualizar en tiempo real la temperatura que detecta el sensor, que en este caso es de 26.3 C

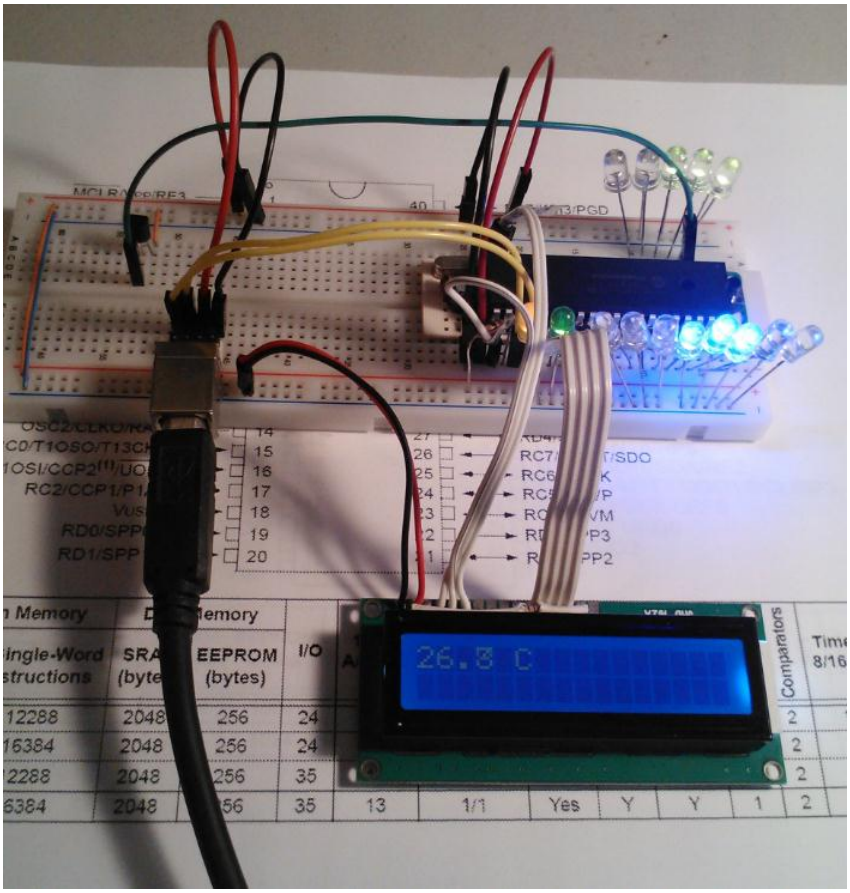


Figura 4.9 Display conectado al MCU, presentando la lectura de la temperatura
Fuente:[Elaboración propia]

Al revisar el display conectado al microcontrolador, se podrá corroborar que la temperatura que en él se muestra es la misma que la indicada en la aplicación Web y que esta varía de acuerdo al cambio de temperatura de la habitación donde éste se encuentra.

También, a través de la otra página Web que se desarrolló se podrá visualizar la información correspondiente a la temperatura en un tamaño de letra mayor.

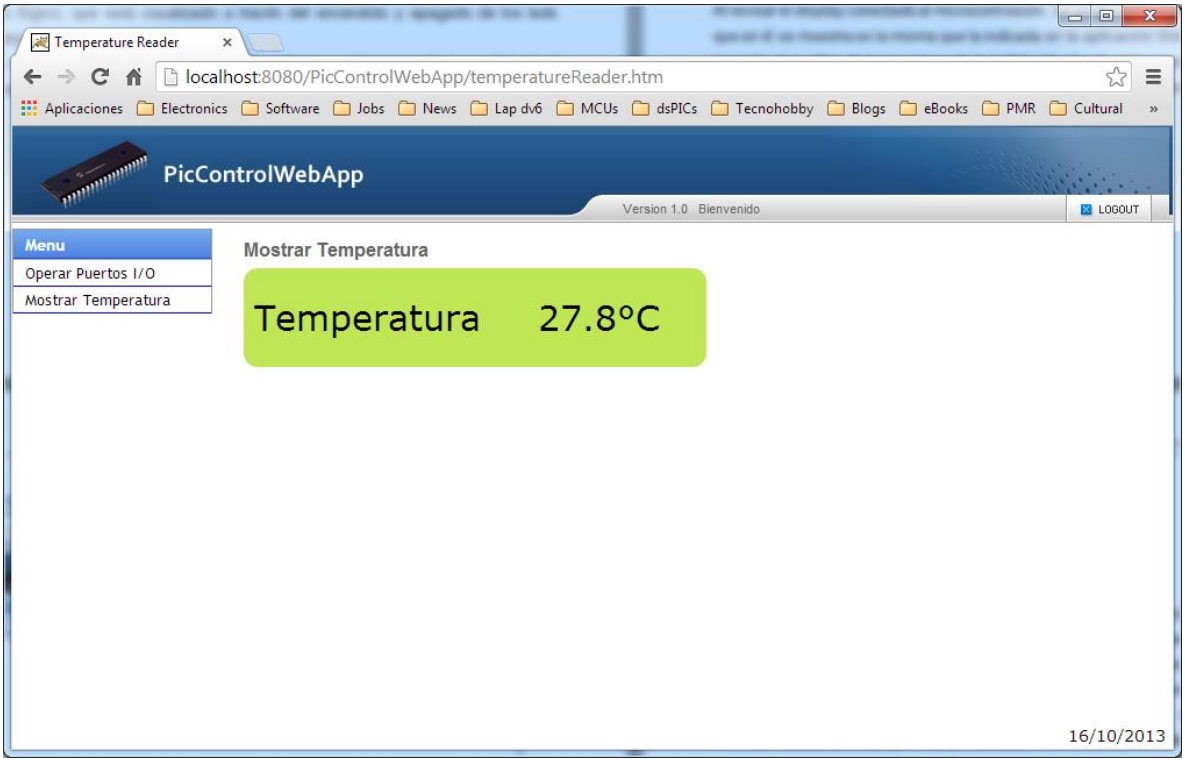


Figura 4.10 Pantalla para obtener en tiempo real la medición de la temperatura
Fuente:[Elaboración propia]

Una vez funcionando el sistema, se podrá corroborar que el control de cualquier dispositivo puede llevarse a cabo con tan solo incluir una pequeña etapa de potencia para accionar dispositivos que trabajen con voltajes y cargas más elevadas.

4.1 Integración de la etapa de potencia

Como se mencionó en el punto anterior, es muy fácil integrar una etapa de potencia para sacar provecho de la comunicación existente entre nuestro microcontrolador y la aplicación Web. Es por eso que en la Figura 4.11 se presenta el diagrama para realizar una sencilla integración de un pequeño ventilador, que será controlado por el microcontrolador y cuya puesta en marcha dependerá de si el valor de la temperatura alcanzado supera los 28 ºC, de lo contrario éste permanecerá sin actividad.

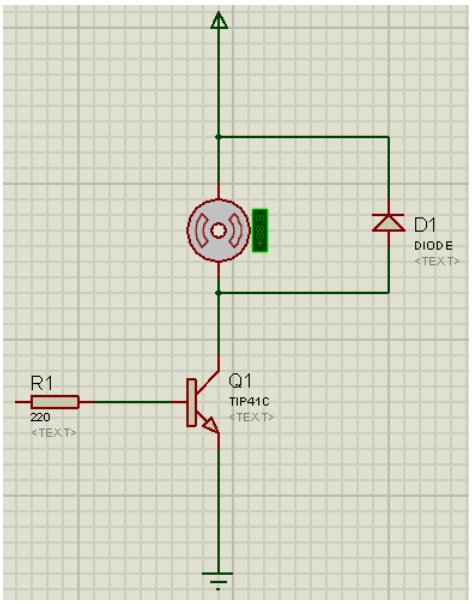


Figura 4.11 Circuito para activar un ventilador desde el puerto del MCU
Fuente:[Elaboración propia]

Se puede apreciar que, de acuerdo al código del Listado 4.1, el microcontrolador determinará si debe enviar un valor alto/bajo al puerto A1 para activar/desactivar el ventilador con la ayuda de un transistor TIP41C que es tipo NPN.

```
if(temp > 28){
    LED_ON(PIN_A1);
}else{
LED_OFF(PIN_A1);
}
```

Listado 4.1 Código para accionar el ventilador
Fuente:[Elaboración propia]

En la Figura 4.12 se visualiza el ventilador, su diodo de protección, el transistor TIP41C, así como la resistencia de 220 ohms conectada a la salida del puerto A1 a través de la cual el microcontrolador controlará la puesta en marcha del ventilador.

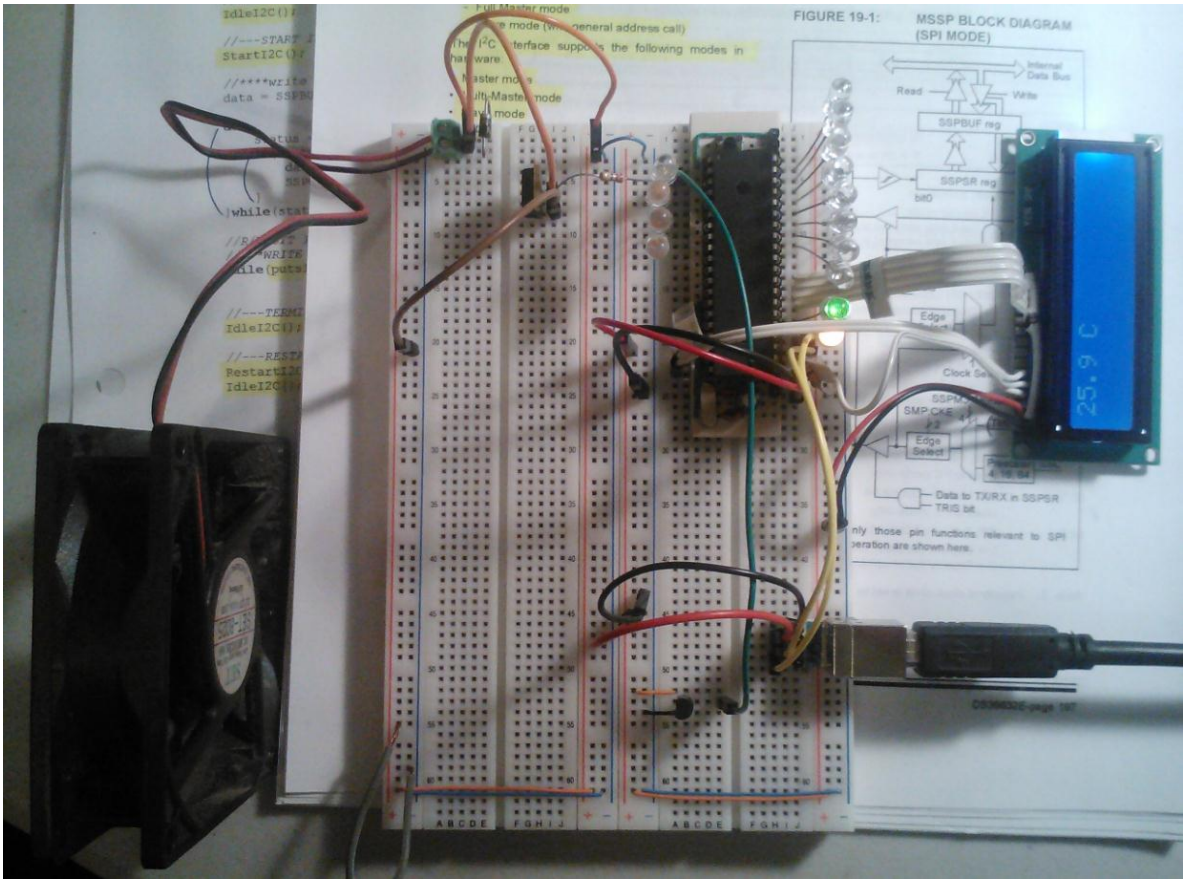


Figura 4.12 Integración de una etapa de potencia para activar un ventilador
Fuente:[Elaboración propia]

De esta manera, el usuario podrá visualizar y constatar desde la aplicación Web que el valor de la temperatura siempre se mantendrá por debajo de los 28 ºC.

Una vez concluida la presentación del ejemplo anterior, es importante indicar que diferentes tipos de integraciones se pueden llevar a cabo gracias a la flexibilidad del sistema desarrollado; siendo suficiente realizar algunos pequeños ajustes al código en caso de ser necesario.

Conclusiones

El proyecto desarrollado permite obtener información de dispositivos externos (sensores principalmente, tales como: humedad, temperatura, presión, radiación solar, movimiento, etc.) conectados a los puertos de un microcontrolador, para después procesarla internamente y enviarla a través de una red local o Internet a un navegador dentro de algún dispositivo ubicado en un sitio remoto.

Gracias a su flexibilidad, puede tener diferentes aplicaciones: en el hogar podría ser adaptado para controlar electrodomésticos, luces, persianas, cerraduras, etc., y en la industria para controlar la temperatura de hornos o congeladores, para monitorear el funcionamiento de dispositivos, o para sincronizar la ejecución de procesos electromecánicos sin la necesidad de un operador presencial.
Dado lo anterior, su gama de aplicaciones es prácticamente infinita pues solo es necesario realizar ciertos cambios o ajustes para adaptarlo y satisfacer diferentes necesidades.

Las pruebas de concepto (POC) son de gran utilidad en el desarrollo de proyectos, especialmente en aquellos en los que se hace uso de alguna tecnología o producto que no se conoce en su totalidad, o que no se ha utilizado previamente; pues nos permiten conocer su alcance, funcionamiento y sobre todo si va a cumplir con nuestras necesidades y/o expectativas.

Derivado de las pruebas de concepto realizadas durante la fase de desarrollo, se llegó a la conclusión de que el uso de la clase CDC para emular a la Interfaz RS-232 sobre USB era una mejor opción sobre las otras clases de la interfaz USB, por su facilidad de manejo e integración con el sistema operativo Windows, que fue sobre el que se desarrolló este proyecto.

El uso de metodologías en el desarrollo de cualquier tipo de proyecto es esencial, pues éstas nos orientan y nos ayudan a llevarlo a cabo de una manera sistemática, disciplinada y controlada; o sea, nos permiten asegurar la obtención de un producto fiable, con calidad y que satisfaga nuestras necesidades y/o requerimientos.

Las metodologías ágiles se utilizan cuando se requiere de un proceso que pueda responder de manera eficiente a los cambios en los productos en desarrollo, permitiendo una mayor flexibilidad que las metodologías tradicionales, pues éstas se bloquean muy pronto en los detalles del proyecto y son menos capaces de ajustarse a las cambiantes necesidades y a los desafíos imprevistos que plantea la tecnología.

El uso de la metodología SCRUM durante la elaboración de este trabajo, permitió reaccionar rápidamente ante las complicaciones que se presentaron, así como a las variaciones de los requerimientos; las cuales son sus principales ventajas, principalmente frente al desarrollo de proyectos de pequeña y mediana escala.

Como parte de la adopción de alguna metodología, es de vital importancia elaborar un análisis y diseño previo, pues durante estas etapas es en donde se pueden identificar posibles complicaciones que pudieran afectar nuestras estimaciones de tiempo y de costos principalmente.

Recomendaciones

La principal recomendación para el desarrollo de proyectos basados en microcontroladores, es hacer uso de alguno cuyo fabricante cuente con librerías, ejemplos y documentación acerca de sus productos, tales como Microchip, Atmel o Freescale.
También se puede hacer uso de Arduino o de alguna de sus muchas variantes (ChipKit UNO32, Pingüino, Netduino, Freeduino, Olimexino, etc.), que es una plataforma basada en microcontroladores AVR de Atmel, en el lenguaje de programación Processing, con una interfaz de programación sencilla y amigable, y que además cuenta con infinidad de librerías

y funcionalidades disponibles desde su página Web así como en muchos otros sitios de Internet.

Arduino es un excelente entrenador para ingresar de lleno al mundo de los microcontroladores, pues está diseñado principalmente para ser utilizado por artistas, hobbistas, estudiantes y recién iniciados. También nos permite obtener un prototipo rápido y sencillo, liberando al usuario de soldar y crear su propio PCB.

Además, con el uso de Arduino podemos reducir las complicaciones que nos podría traer el trabajar directamente con un microcontrolador, pues provee una capa de abstracción de muchos de los aspectos del chip y oculta algunos detalles de bajo nivel, logrando que nuestro trabajo sea mucho más fácil.

Sin embargo, el trabajar directamente con los microcontroladores no es tan malo, pues obtenemos varias ventajas, especialmente si se trata de los PICs de Microchip. Por ejemplo, con Arduino es prácticamente imposible realizar la depuración de un programa conectado al circuito, cosa que es una de las principales potencias de los PICs. Asimismo, con los PICs podemos revisar sus hojas de datos (Datasheet) como si se tratase de cualquier otro dispositivo electrónico, permitiéndonos acceder completamente a todas sus funcionalidades y explotar así toda su potencia. Esto nos permite ser capaces de especificar el tipo de oscilador a utilizar, si será interno o externo, la magnitud de su frecuencia, así como de hacer un manejo más apropiado de las interrupciones y periféricos, e incluso podremos activar opciones para el ahorro de energía y manipular adecuadamente sus registros de funciones especiales y de memoria, etc.

Además, los PICs se pueden programar en varios lenguajes, pero principalmente en C que es uno de los lenguajes de mayor crecimiento en materia de programación de microcontroladores, gracias a la simplificación de comandos y a que es de alto nivel. Por otro lado, existe MPLAB X que es una herramienta gratuita distribuida por Microchip y que brinda la posibilidad de trabajar tanto en C como en ASM directamente (incluso es posible configurar compiladores de terceros) para poder manejar los tiempos del microcontrolador a la perfección y con la precisión exacta que brinde el cristal que se utilice como reloj del sistema; Es decir, con MPLAB X se tiene el dominio absoluto de la acción a cada instante.

Con Arduino, uno queda atado al uso del software que viene con el sistema, y a pesar de contar con un nivel avanzado de programación podría ser muy complicado variar sus características. Se puede interactuar con otros programas; pero para grabar el firmware dentro del microcontrolador se requiere usar el específico de Arduino.

Finalmente, es conveniente mencionar que el costo de producir un proyecto en serie utilizando la plataforma Arduino sería obsceno. Por ejemplo, supongamos que deseamos realizar un proyecto que hace uso de tan solo un par de puertos digitales, un contador y el módulo PWM del microcontrolador; con la plataforma Arduino sería un total desperdicio de periféricos, memoria y funcionalidades que quedarían sin utilizar, además los costos se dispararían hasta el cielo, haciendo imposible la venta de dicho proyecto. Esto no sucedería con los PICs de Microchip, pues en este tipo de problemas es en lo que más se especializa el fabricante que cuenta con un muy amplio portafolio de productos que se ajustan al diseño y especificaciones de nuestro proyecto, de tal forma que podemos seleccionar un modelo de entre varios para reducir el tamaño de la memoria, el número de periféricos, los puertos, eliminar el uso de un reloj externo y todas aquellas otras funcionalidades que no utilicemos, reduciendo al máximo nuestros costos de producción.

Costos del proyecto

El tema de los costos para el desarrollo de este proyecto es un punto muy importante, pues en su mayoría el uso de sus componentes de software es libre o gratuito, y respecto a los componentes de hardware, solo se considerarán los costos originados por la compra del microcontrolador y su circuitería, de la pantalla LCD y del sensor de temperatura. El costo de éstos últimos dos se aproxima a los $130 pesos M.N.

Si hablamos del costo de los microcontroladores de Microchip, podemos indicar que éste es muy bajo comparado con otras opciones existentes hoy en día; incluso si lo comparamos con el popular Arduino UNO; en donde salta a la vista que esta versión de Arduino tiene un costo que ronda alrededor de los $400 pesos M.N., cuando el costo del PIC18F4550 se puede conseguir en menos de la tercera parte de ese precio, incluyendo la tabla prototipo para montarlo y la circuitería necesaria para su funcionamiento (un capacitor electrolítico, un par

de capacitores cerámicos, algunos trozos de cable, leds, un reloj de cristal de cuarzo, un diodo, un botón de presión y una resistencia y el Header USB).

Es también importante indicar que dentro de los costos del proyecto no se incluye el del programador de PICs, pues sólo se utiliza para la programación del microcontrolador y no se considera como una parte funcional del proyecto. Sin embargo, mencionaremos que actualmente existe una gran variedad de ellos en el mercado, cuyo costo aproximado es de entre $300 y $600 pesos M.N.

Lo único que podría considerarse como un gasto importante es la licencia del compilador C CCS utilizado para la programación del microcontrolador; sin embargo se puede utilizar su versión demo, o bien, utilizar uno de los muchos compiladores existentes y disponibles de manera gratuita o en versión con limitantes, que se apegan más al estándar del ANSI C y que bien podrían sustituir al C CCS. Tal es el caso del compilador MikroC, que cuenta con una versión gratuita, cuya limitante consiste en poder programar hasta un máximo de 2K en la memoria del programa. Otra opción es el Compilador SDCC (Small Device C Compiler) que es libre bajo licencia GPL y que permite la programación de los PIC16 y PIC18 de Microchip. También, el compilador XC8, que el mismo Microchip pone a disposición de sus usuarios de manera gratuita, aplicando costos de licencia únicamente sobre la versión profesional, la cual maneja un nivel superior de optimización del código para una mejor eficiencia en el uso de la memoria comparado con la versión gratuita, etc.

Una vez mencionados los puntos anteriores, podemos indicar que los costos de la elaboración de este proyecto no superan los $350 pesos M.N.

Sin embargo, es conveniente tener presente que los costos mencionados son basados en que su desarrollo será con fines didácticos, o llevado a cabo por personas que cuenten con el conocimiento técnico para comprenderlo, mejorarlo e incluso adaptarlo a sus necesidades. Pero que si la intención fuese comercializarlo, entonces deberíamos adicionar el costo derivado de su programación, que podrían superar los $37,000 pesos M.N., dependiendo del tiempo en que se requiera contar una fase productiva y de las adaptaciones o modificaciones que se vayan a realizar. También deberían considerarse los costos originados por el diseño

de la nueva funcionalidad electrónica y por la migración de prototipo a circuito impreso, que también dependerán de esto último.

Propuestas de mejoras para desarrollos futuros

A continuación se mencionan algunas propuestas que podrían utilizarse en futuras implementaciones:

- Hacer uso de microcontroladores dsPICs que son especializados en el procesamiento de señales, con la intención de poder recibir y procesar ondas de radio provenientes de sensores más robustos para desarrollar una estación meteorológica.
- Modificar la aplicación para que pueda interactuar simultáneamente con varios usuarios, sin presentar problemas de sincronización.
- Agregar módulos de radiofrecuencia para extender la funcionalidad del sistema, logrando así interactuar con dispositivos que no se encuentren conectados al puerto USB del servidor. En este caso, el microcontrolador podría recibir la información por IR de muchos otros dispositivos, y estaría encargado de procesar dicha información y de enviarla a la aplicación Web para ponerla a disposición del usuario, y recíprocamente de recibir los comandos de control que ingrese el usuario y enviarlos a los dispositivos remotos.
- Añadir un módulo encargado del control de acceso a la aplicación, de tal forma que restrinja el ingreso a usuarios no autorizados; e incluso se podría manejar un esquema basado en roles o perfiles, para que así solo algunos usuarios puedan realizar ciertas operaciones, que para otros no estarían disponibles.
- Incluir una cámara IP para poder visualizar el entorno del microcontrolador.
- Integrar bases de datos y diferentes tipos de sensores: humedad, temperatura, radiación solar, etc., de tal forma que se pueda almacenar el registro de varios de sus eventos durante periodos de tiempo prolongados, y permitir así su explotación e interpretación.

- Al integrar una base de datos, se pueden aplicar diferentes tipos de procesamiento a los datos, e incluso realizar cálculos estadísticos para obtener una mejor interpretación del medio que rodea al microcontrolador.
- Adicionar dispositivos electro-mecánicos como servo-motores, motores de pasos y relevadores para controlar el movimiento de brazos robóticos, etc.
- Implementar mejores opciones de control al sistema.

Bibliografía

1. Ángulo Usategui, J. M., Ángulo Martínez, I. (2003). ***Microcontroladores PIC. Diseño práctico de aplicaciones.*** (3a Ed.) España: McGraw-Hill
2. Axelson, J. (2009). ***Usb Complete, the Developer´s Guide.*** (4a Ed.) US: Independent Pub Group.
3. Casey, J., Massol, V., Porter, B., Sanchez, C. & Van Zyl, J. (Marzo 2006). ***Better Builds with Maven. The How-to Guide for Maven 2.0*** US:Mergere Library Press, Inc.
4. Crane, D., Bibeault, B & Sonneveld, J. (2007). ***Ajax in Practice*** (1a Ed). US: Manning Publications Co.
5. Flanagan, D. (2005). ***Java in a Nutshell.*** (5a Ed). US:O'Reilly
6. García Brejo, E. (2008). ***Compilador C CCS y simulador proteus para microcontroladores PIC*** (1a Ed.) Barcelona, España: Alfaomega Grupo Editor, S.A. de C.V.
7. Ibrahim, D. (2008). ***Advanced PIC Microcontroller Projects in C: From USB to RTOS with the PIC18F Series.*** (1a Ed.) UK:Newnes
8. Minter, D. (2008). ***Beginning Spring 2: From Novice to Professional.*** (1a Ed.) NY, USA: Apress
9. Palacios Municio, E., Remiro Dominguez, F. & López Pérez, L.J. (2004) **Micocontrolador PIC16F84. Desarrollo de proyectos**. (1ª. Ed.) México: Alfaomega Grupo Editar, México
10. P. Bates, M. (2008). ***Programming 8-bit Pic Microcontrollers in C: With Interactive Hardware Simulation.*** (1a Ed.) UK: Newnes
11. Swedberg, K. & Chaffer, J. (2010). ***jQuery 1.4 Reference Guide. A comprehensive exploration of the popular JavaScript library.*** UK:Packt Publishing
12. Ullman, C. (Marzo 2007). ***Beginning Ajax.*** (1a Ed.) US:Wrox
13. Verle, M. (2008). ***PIC Microcontrollers.*** (1a Ed.) US:mikroElektronika press
14. Wilmshurst, T. (2010). ***Designing Embedded Systems with PIC Microcontrollers.*** (2a Ed.) UK:Newnes

Referencias electrónicas

15. Apache Maven Website. (2013). ***Maven in 5 Minutes.*** Consultado el 09 de mayo 2013. Disponible en: http://maven.apache.org/
16. Apache Tiles Website. (2013). ***Apache Tiles.*** Consultado el 14 de junio 2013. Disponible en: http://tiles.apache.org/
17. Apache Tomcat Website. (2013). ***Apache Tomcat.*** Consultado el 16 de junio 2013. Disponible en: http://tomcat.apache.org/
18. Castro, P. (2004). ***Microchips 8-bit PIC Microcontrollers - WebSeminars.*** Consultado el 09 de Junio 2013. Disponible en: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2813&dDocName=en557096
19. CCS Inc. Webpage (2013). ***CCS C Compiler Details.*** Consultado el 02 de mayo 2013. Disponible en: http://www.ccsinfo.com/content.php?page=compiler-details
20. CursoMicros. (2012). ***Los microcontroladores PICMicro o PIC de Microchip***. Consultado el 15 de noviembre 2012. Disponible en: http://www.cursomicros.com/pic/microcontroladores/los-microcontroladores-pic.html
21. Java Oracle Webpage (2013). ***Conozca más sobre la tecnología Java.*** Recuperado el 09 de mayo 2013. Disponible en: http://www.java.com/es/about/
22. jQuery Website. (2013). ***jQuery.*** Consultado el 25 de agosto 2013. Disponible en: http://jquery.com/
23. Microchip – Product guide (2012). ***Microchip 8-bit Product Guide.*** [Versión electrónica]. Disponible en: http://www.microchip.com/stellent/groups/picmicro_sg/documents/devicedoc/en023527.pdf
24. Microchip - Part Selector. (2013). ***Microchip Advanced Part Selector.*** Consultado el 07 de Junio 2013. Disponible en: http://www.microchip.com/maps/Microcontroller.aspx
25. Microchip - PIC18F2455/2550/4455/4550. (2009). ***PIC18F2455/2550/4455/4550 Datasheet.*** [Versión electrónica]. Disponible en: http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf

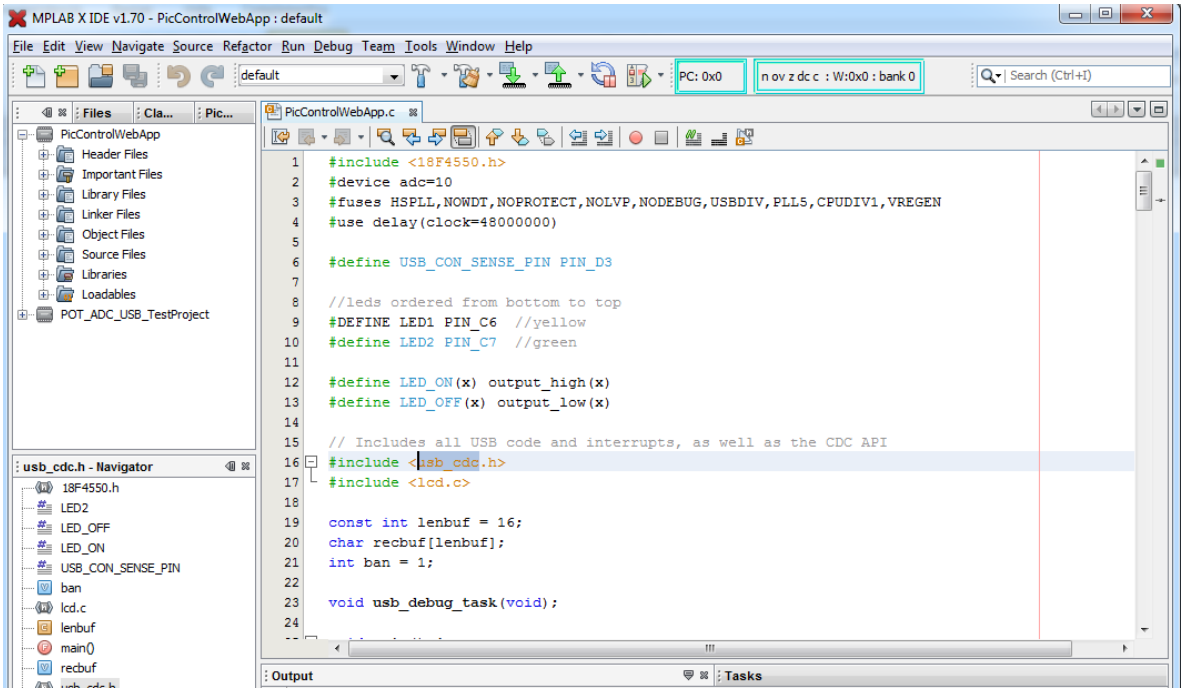
26. Microchip - Product Selector. (2013). ***Microchip Product Selector.*** Consultado el 07 de Junio 2013. Disponible en: http://www.microchip.com/productselector/MCUProductSelector.html
27. Microchip - PIC MCUs. (2004). ***PIC MCUs and dsPIC DSCs. A quick overview – WebSeminars.*** Consultado el 09 de Junio 2013. Disponible en: http://www.microchip.com/microchip.webcontent.provider/Video.aspx?id=en546906
28. Microchip - MPLAB X. (2013). ***MPLAB X Integrated Development Environment (IDE).*** Disponible en: http://www.microchip.com/pagehandler/en-us/family/mplabx/
29. Microchip Website. (2012). ***8-bit PIC Microcontrollers.*** Consultado el 15 de noviembre 2012. Disponible en: http://www.microchip.com/pagehandler/en-us/family/8bit/
30. Peacock, C. (2013). ***USB in a Nutshell. Making sense of the USB standard.*** Consultado el 28 de Julio de 2013. Disponible en: http://www.beyondlogic.org/usbnutshell/usb1.shtml
31. Rojvanit, R. (2004). ***Emulating RS-232 over USB using the PIC18F4550 - WebSeminars.*** Consultado el 09 de Junio 2013. Disponible en: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2813&dDocName=en528378
32. Román Hernández, J. (2008). ***USB1.0, USB1.1, USB2.0 y USB3.0.*** Consultado el 12 de Junio 2013. Disponible en: http://www.emezeta.com/articulos/usb10-usb11-usb20-usb30
33. Sampaleanu, C. (4 Enero 2011). ***Green Beans: Getting Started with Spring MVC.*** Disponible en: http://blog.springsource.com/2011/01/04/green-beans-getting-started-with-spring-mvc/
34. Shafqat, A. (2011). ***Web Application Development using Spring MVC.*** Consultado el 25 de Agosto de 2013. Disponible en: http://eiconsulting.blogspot.mx/2011/09/web-application-development-using.html
35. SpringSource - Spring Framework (2011). ***Spring Java Application Framework 3.0. Reference Documentation*.** [Versión electrónica]. Disponible en:

http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf

36. SpringSource - Spring Framework (2008). ***Spring Java/j2ee Application Framework 2.5.6. Reference Documentation.*** [Versión electrónica]. Disponible en: http://static.springsource.org/spring/docs/2.5.6/spring-reference.pdf
37. Tambi, Y. (2013). ***Serial Communication – Introduction.*** Consultado el 07 de Junio 2013. Disponible en: http://maxembedded.com/2013/09/03/serial-communication-introduction/
38. Texas Instruments - LM35 (2000). ***LM35 Precision Centigrade Temperature Sensors - Datasheet.*** [Versión electrónica]. Disponible en: http://www.ti.com/lit/ds/symlink/lm35.pdf
39. Toboso, E. (2013). ***Aplicaciones de los Microcontroladores.*** Consultado el 07 de Junio 2013. Disponible en: http://perso.wanadoo.es/pictob/microcr.htm#aplicaciones_de_los_microcontroladores
40. Weiss, R. (2001). ***Universal Serial Bus 2.0.*** Consultado el 28 de Julio de 2013. Disponible en: http://electronicdesign.com/embedded/universal-serial-bus-20

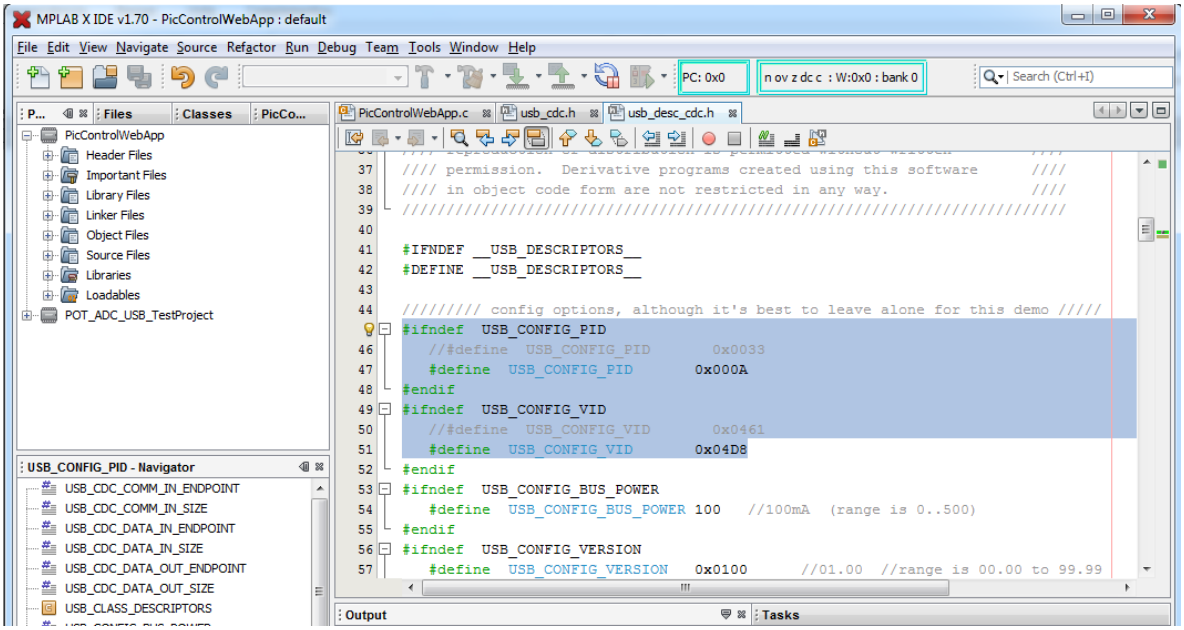
Apéndice A. Instalar el driver USB CDC en Windows

Paso 1. Grabar en el Micro el código de la aplicación USB CDC, que hace uso del driver **usb_cdc.h**, el cual es declarado en el archivo de la aplicación (ex: **PicControlwebApp.c**). Este driver permite utilizar una clase de dispositivos **USB CDC**, emulando un dispositivo **RS232** y lo muestra como un puerto **COM** en Windows.



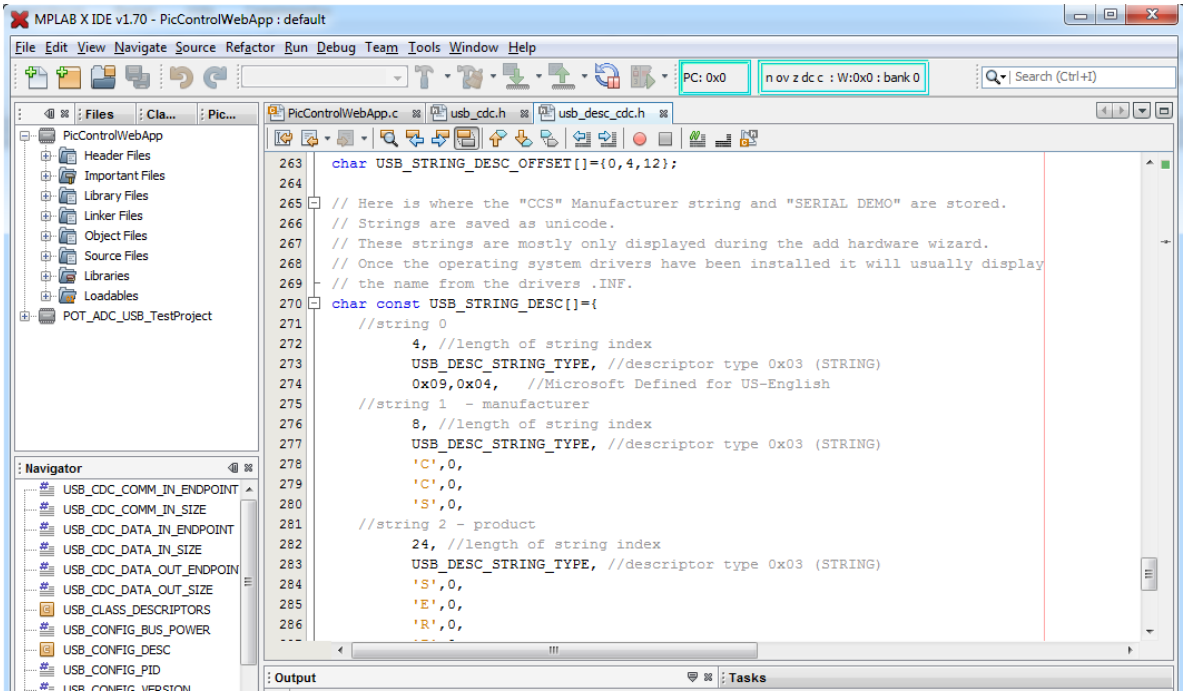
El driver **usb_cdc.h** incluye a la librería de descriptores **usb_desc_cdc.h,** en la cuál será necesario realizar una modificación para especificar el **VID** y **PID** correspondiente:

- **VID** – “Vendor Identification” Número de 16 bits (04D8H para Microchip)
- **PID** – “Product Identification” Número de 16 bits (000AH para familia de los PIC18 de Microchip)



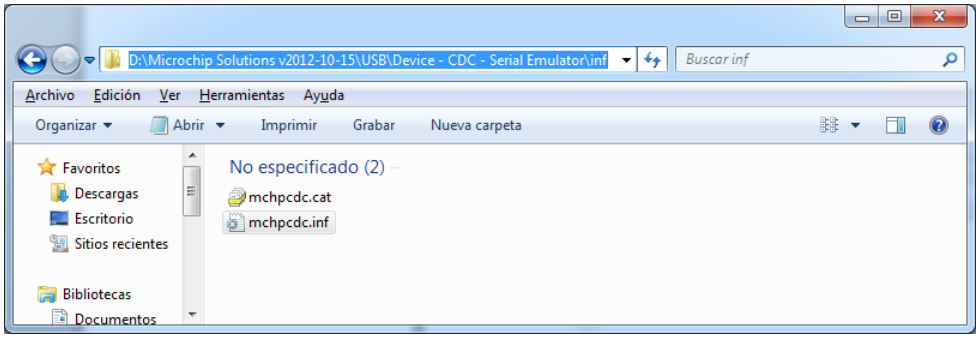
Así como el **USB_STRING_DESC_OFFSET**[] y la **USB_STRING_DESC[]** que se encuentran al final del archivo**:**

- **USB_STRING_DESC_OFFSET[] –** Contiene las posiciones iniciales de las cadenas de descriptores del dispositivo detectado por el driver de Windows, y que son las que se mostrarán en la lista del HW.
- **USB_STRING_DESC[] –** Contiene las cadenas de descriptores del dispositivo



Paso 2. Es posible actualizar la descripción dentro del archivo **mchpusb.inf** que es parte del **USB Framework*** de Microchip, y que se encuentra en la siguiente ruta:

Microchip Solutions vXXXX-XX-XX\USB\Device - CDC - Serial Emulator\inf\mchpcdc.inf



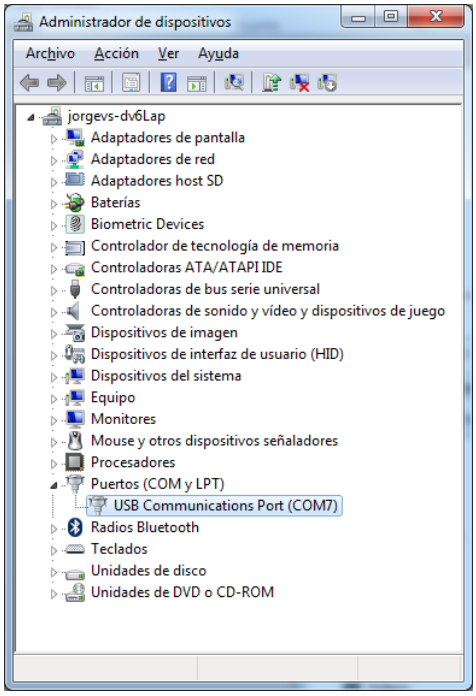
Especialmente las secciones correspondientes al **VID** y al **PID**.

```
;------------------------------------------------------------------------------
;  Vendor and Product ID Definitions
;------------------------------------------------------------------------------
; When developing your USB device, the VID and PID used in the PC side
; application program and the firmware on the microcontroller must match.
; Modify the below line to use your VID and PID.  Use the format as shown below.
; Note: One INF file can be used for multiple devices with different VID and PIDs.
; For each supported device, append ",USB\VID_xxxx&PID_yyyy" to the end of the line.
;------------------------------------------------------------------------------
[SourceDisksFiles]
[SourceDisksNames]
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_000A

[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_000A


;------------------------------------------------------------------------------
;  String Definitions
;------------------------------------------------------------------------------
;Modify these strings to customize your device
;------------------------------------------------------------------------------
[Strings]
MFGFILENAME="mchpcdc"
DRIVERFILENAME ="usbser"
MFGNAME="Microchip Technology, Inc."
INSTDISK="Microchip Technology, Inc. Installation Disc"
DESCRIPTION="USB Communications Port"
SERVICE="USB RS-232 Emulation Driver"
```

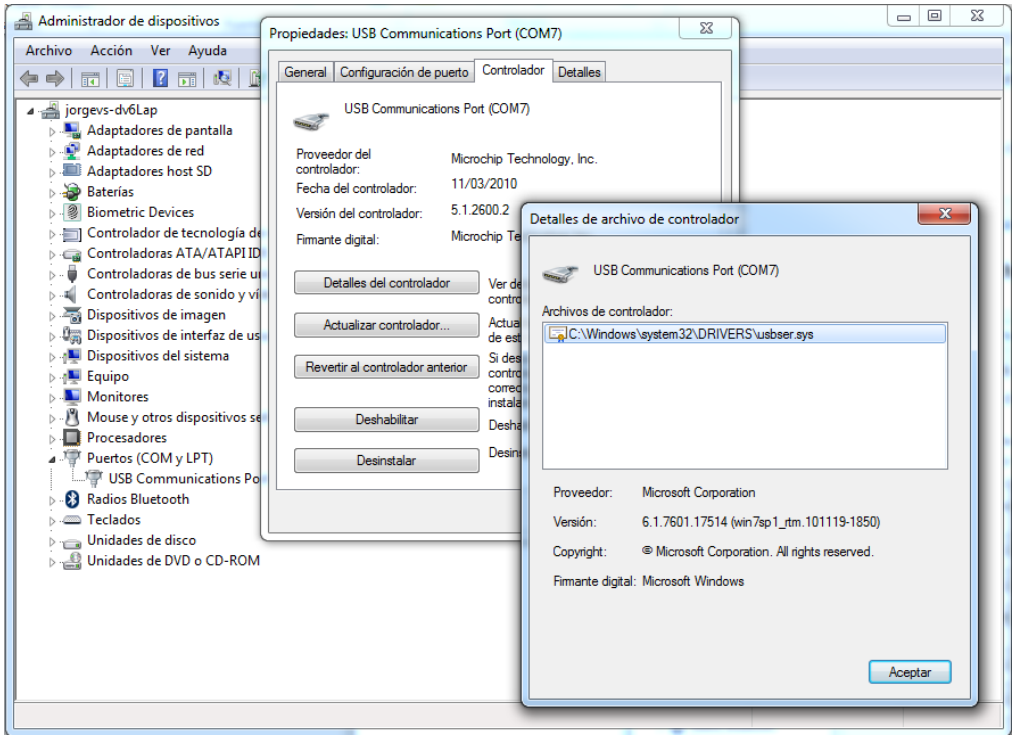
Es importante señalar que el driver que se utiliza es **usbser.sys**, y que es suministrado por Windows.

(The CDC based RS-232 emulation firmware makes use of standard drivers which distribute with Windows. The .INF file is the only PC side file that is needed with this firmware).

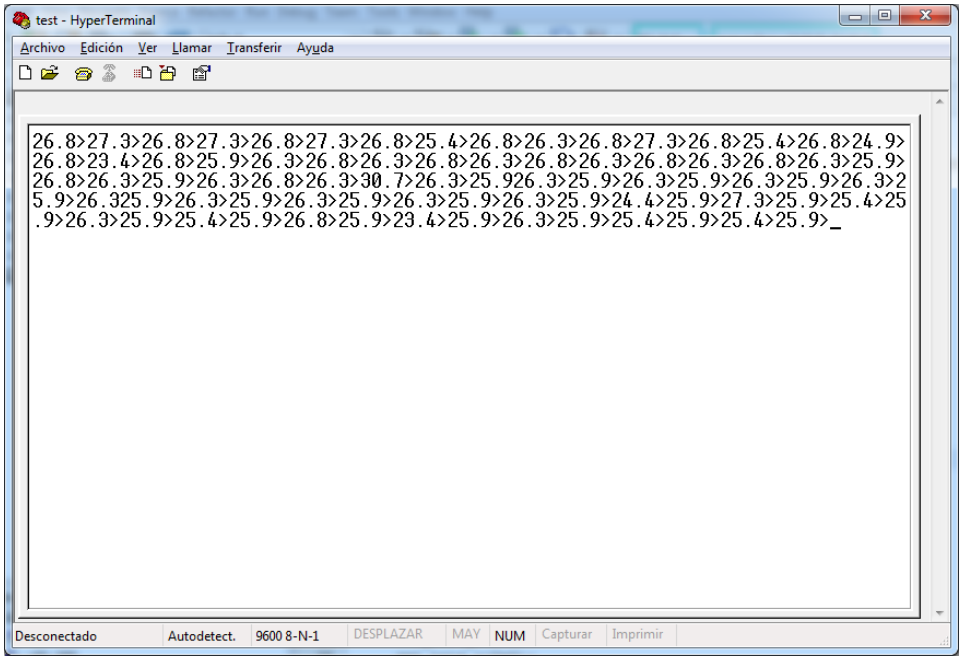
Paso 3. Conectar el micro al USB de la PC, y al ser detectado por Windows indicarle la ruta del archivo *.inf, en este caso: **D:\Microchip Solutions v2012-10-15\USB\Device - CDC - Serial Emulator\inf**



Windows tomará la información del archivo **mchpusb.inf** y utilizará el driver al sistema: **usbser.sys**



Paso 4. Una vez lista la configuración, es posible comunicarse con el MCU como si se tratase de un puerto COM. Por ejemplo, podemos hacer uso del HyperTerminal de Windows para interactuar con él.



*El **USB Framework** is part of the **Microchip Applications Library**. Para más información visita:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044

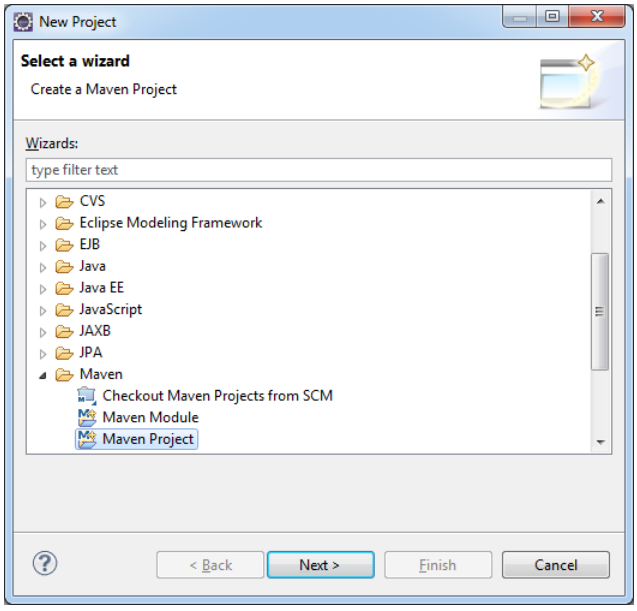
Apéndice B. Creación de un proyecto Web dinámico usando Maven en Eclipse

Es necesario contar con los siguientes componentes instalados

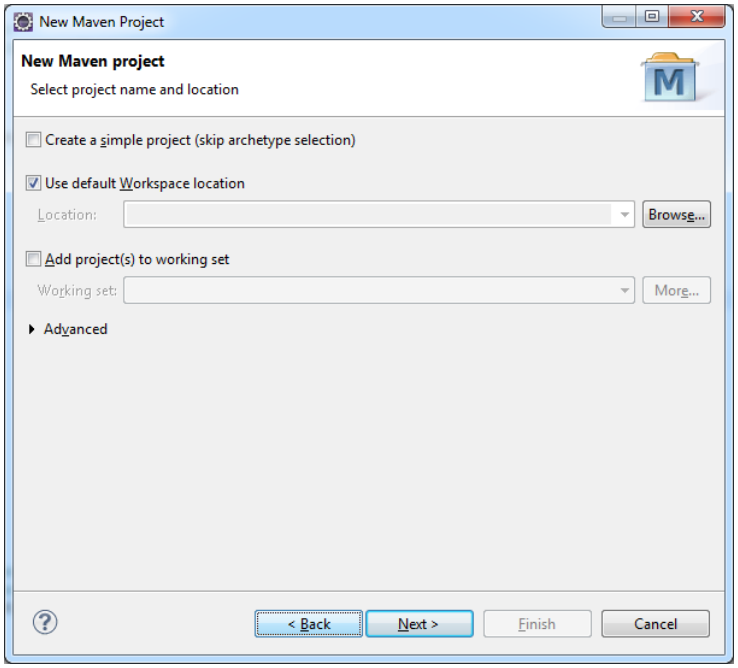
1. JDK 1.6 o superior
2. Eclipse Indigo SR2 o superior (Podría funcionar con alguna versión anterior)
3. Maven 2.0 o superior (Es necesario declarar la ruta del directorio \bin de Maven en la variable PATH)
4. M2Eclipse Plugin

Paso 1. Crear un proyecto en Eclipse.

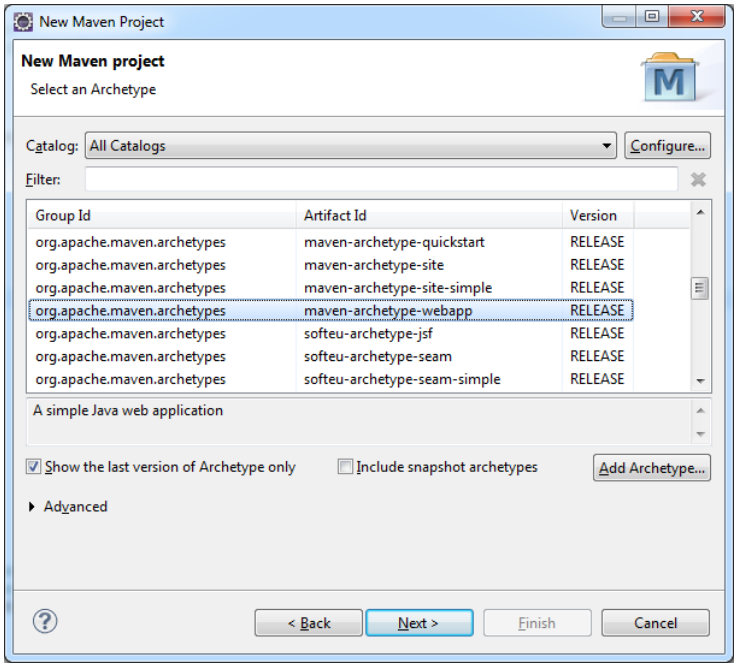
Clic en **File** > **New** > **Project** y seleccionar **Maven Project** de la lista.



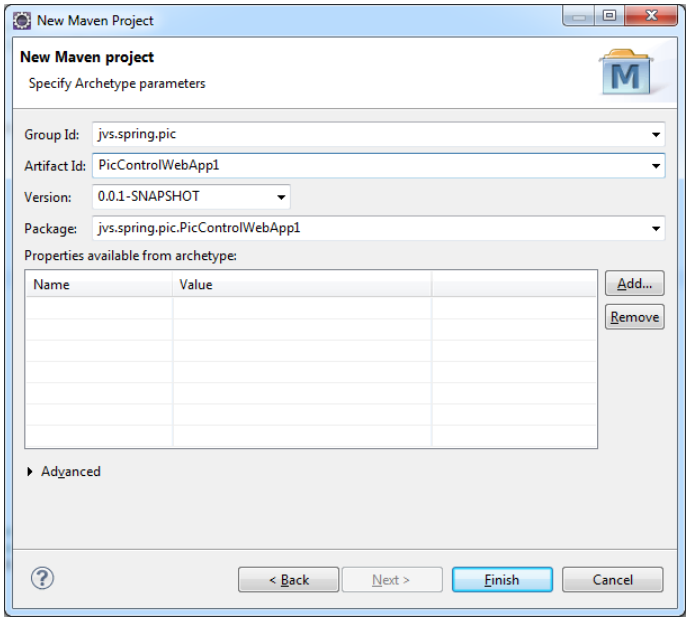
Dejar las opciones que aparecen por default y dar clic en **Next >**



Seleccionar el arquetipo de Maven para WebApps, dar clic en **Next >**



Establecer el nombre del grupo, del artefacto, su versión y del paquete. Después dar clic en **Finish**

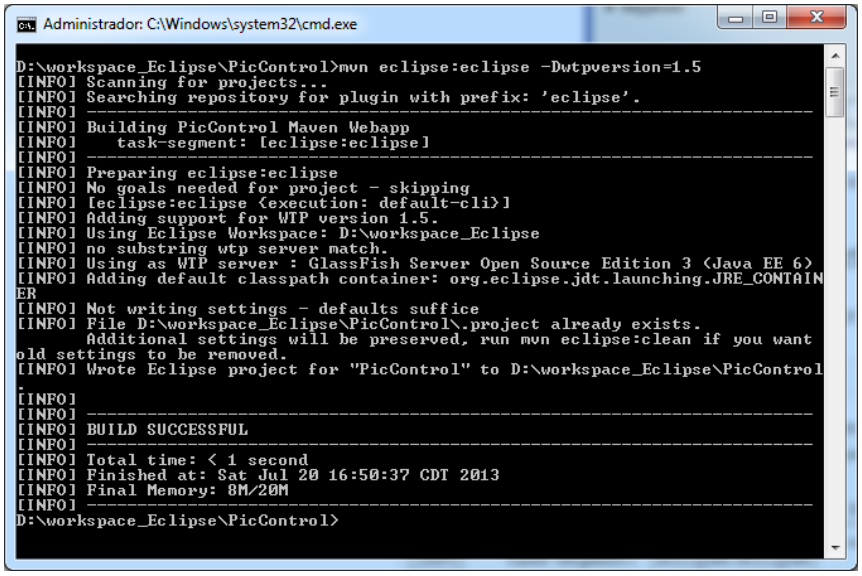


Paso 2. Convertir el **Maven Project** en un **Maven Dynamic Web Project** con **WTP**

La conversión de proyecto, se hace con la siguiente instrucción Maven

```
mvn eclipse:eclipse -Dwtpversion=1.5
```

Por lo que ejecutamos éste comando en la carpeta donde se aloja el proyecto a convertir:

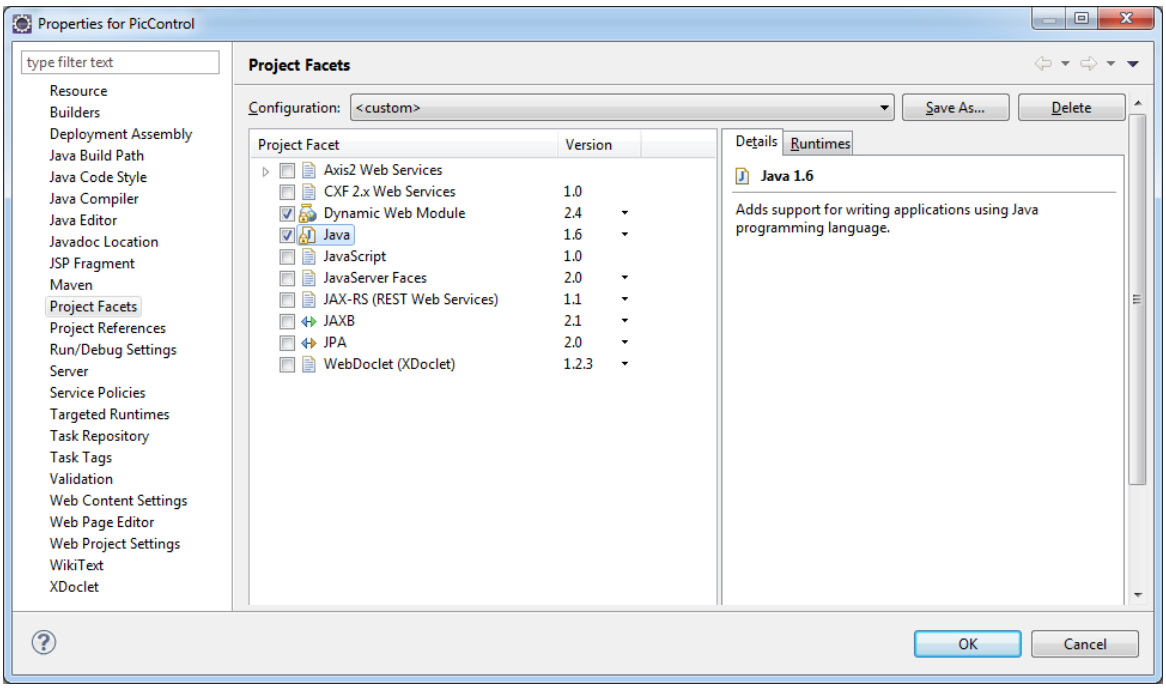


Una vez completado el comando, es necesario refrescar el proyecto en Eclipse presionando **F5**.

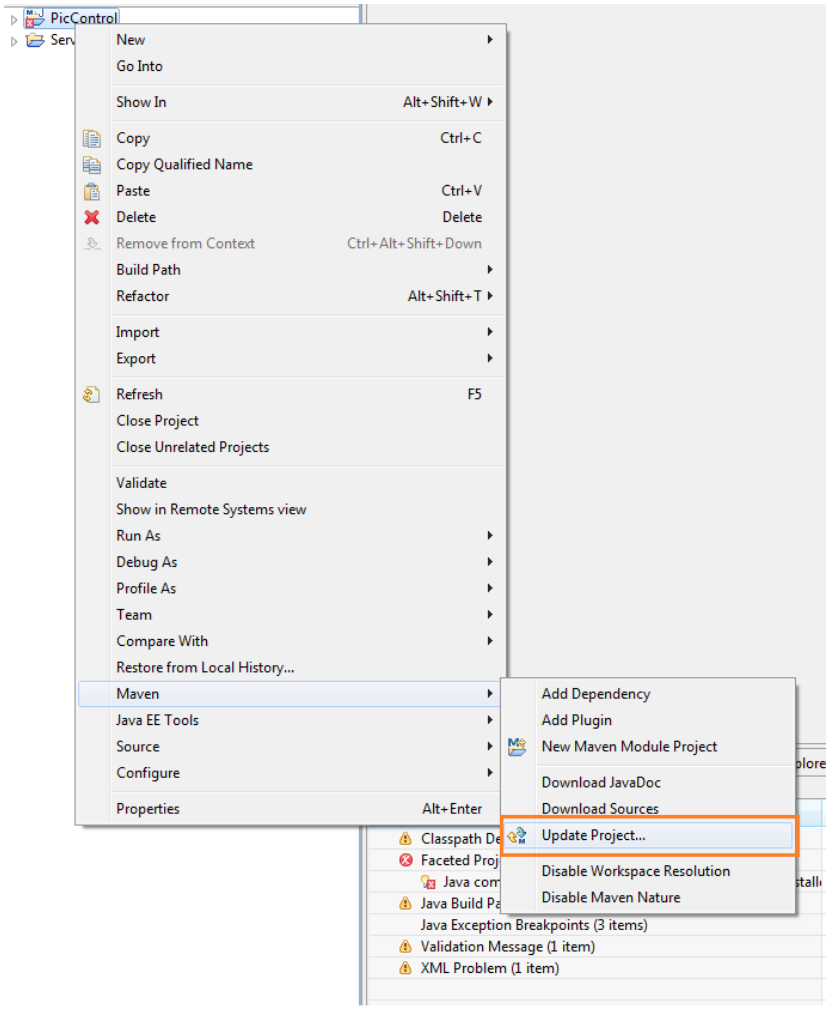
Paso 3. Verificar la versión del **Dynamic Web Module** y **Java** del proyecto

Es necesario que la versión establecida en los Facets del proyecto corresponda con la versión del JDK que se esté utilizando (en este caso es JDK 1.6), por lo que si no es así, no será necesario cambiarlo.

La versión del Dynamic Web Module debe ser la 2.4

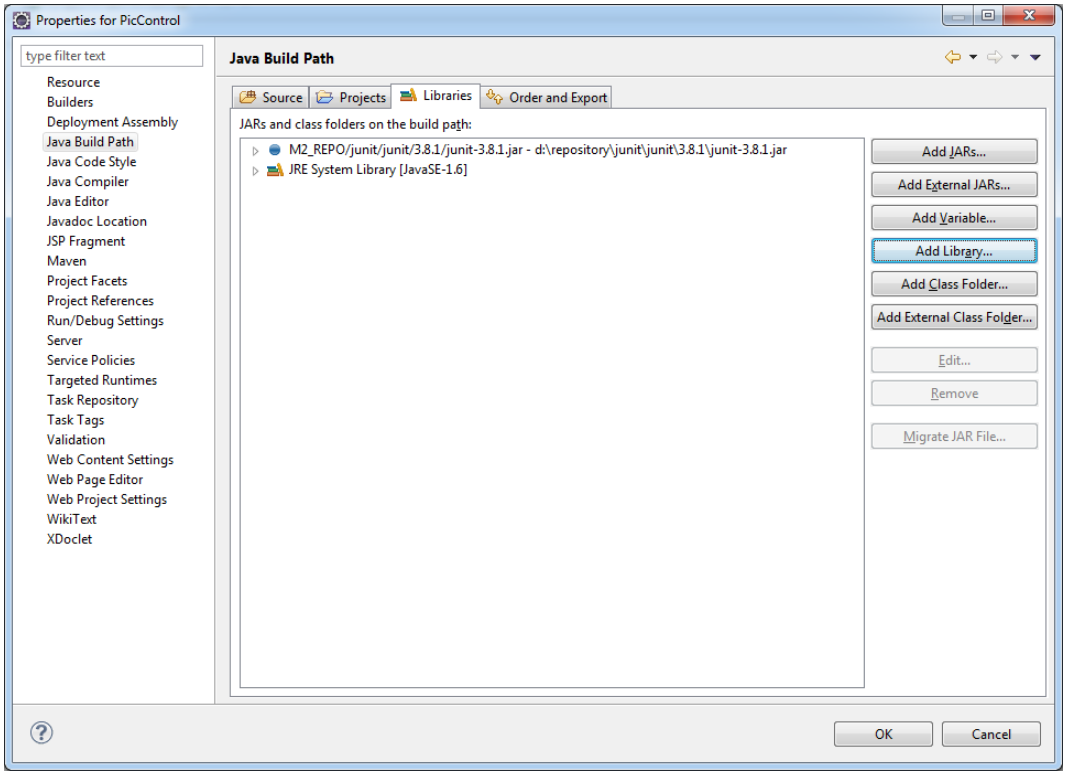


Después se actualiza el proyecto con Maven, dando clic derecho sobre el proyecto y luego en **Maven** > **Update Maven**

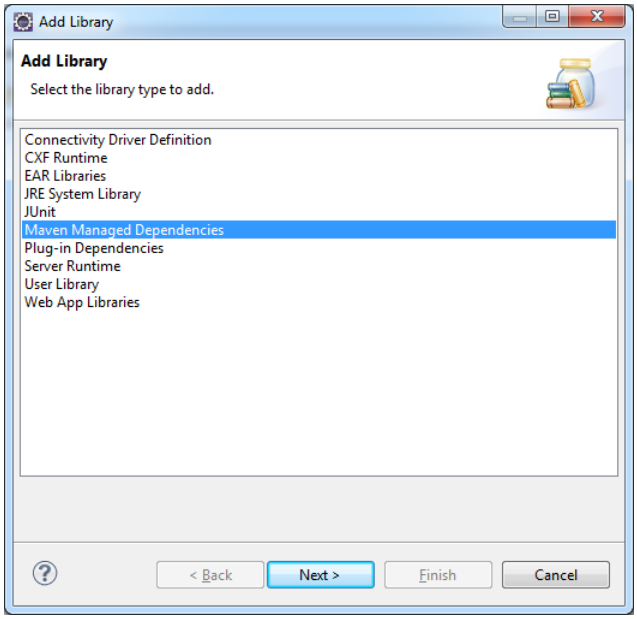


Paso 4. Establecer la ruta de construcción

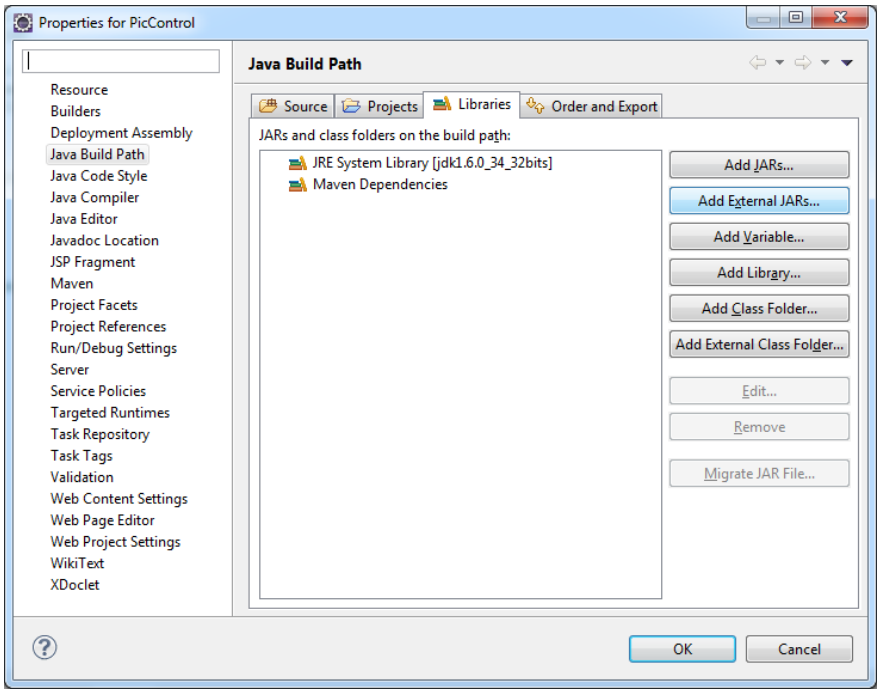
Es necesario establecer la ruta de construcción del proyecto para que consideren las dependencias a través de Maven



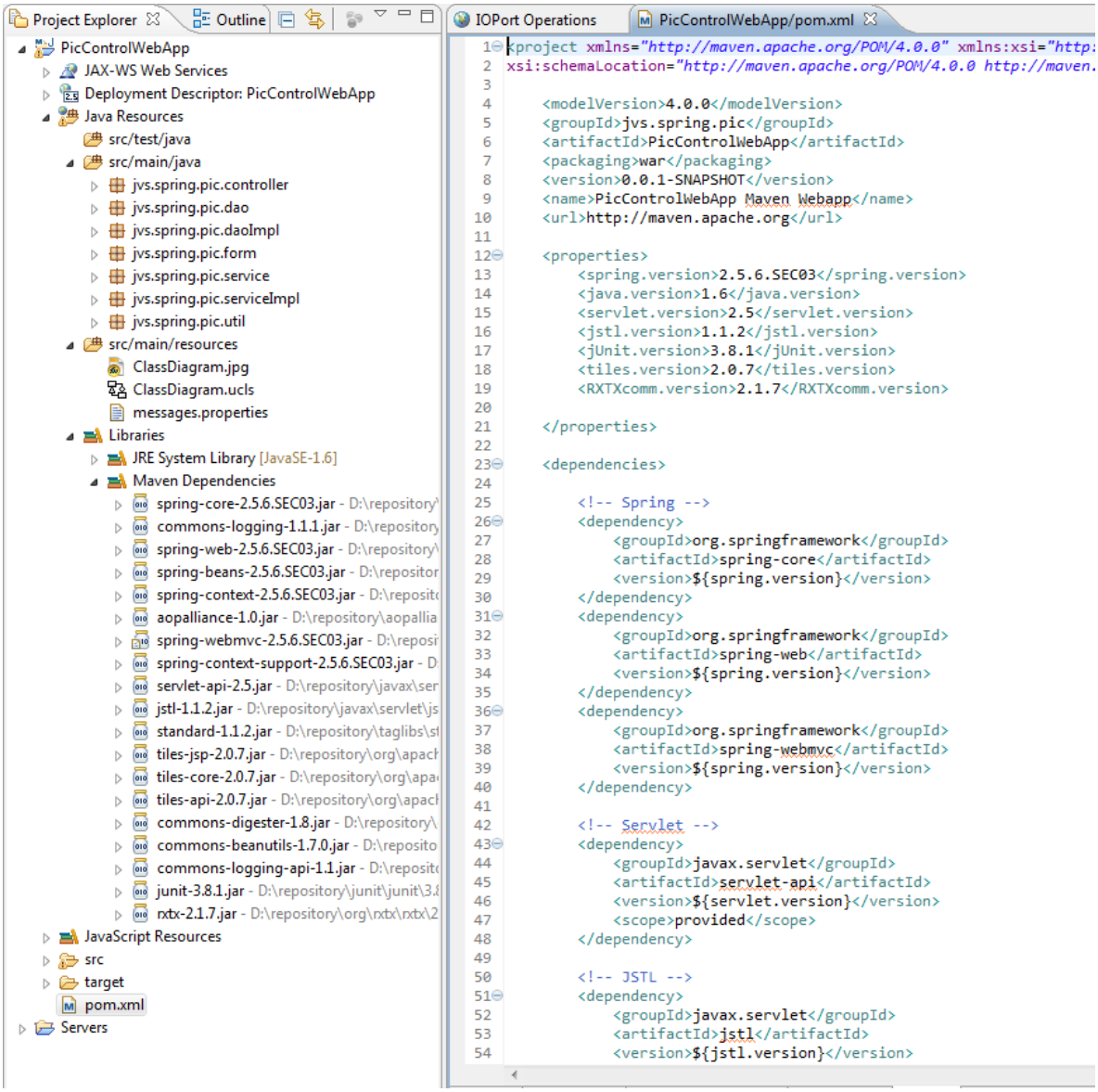
Se da clic en **Add Library**, y se selecciona **Maven Managed Dependencies**, después clic en **Next** >



Con estos cambios, desaparece la variable **M2_REPO** de las librerías del **Java Build Path** y queda el proyecto de la siguiente manera



De esta forma, todas las dependencias que se integren al proyecto, serán administradas desde el **pom.xml** de **Maven**.


Project Explorer
Outline
PicControlWebApp
JAX-WS Web Services
Deployment Descriptor: PicControlWebApp
Java Resources
src/test/java
src/main/java
jvs.spring.pic.controller
jvs.spring.pic.dao
jvs.spring.pic.daoImpl
jvs.spring.pic.form
jvs.spring.pic.service
jvs.spring.pic.serviceImpl
jvs.spring.pic.util
src/main/resources
ClassDiagram.jpg
ClassDiagram.ucls
messages.properties
Libraries
JRE System Library [JavaSE-1.6]
Maven Dependencies
spring-core-2.5.6.SEC03.jar - D:\repository
commons-logging-1.1.1.jar - D:\repository
spring-web-2.5.6.SEC03.jar - D:\repository
spring-beans-2.5.6.SEC03.jar - D:\repositor
spring-context-2.5.6.SEC03.jar - D:\reposit
aopalliance-1.0.jar - D:\repository\aopallia
spring-webmvc-2.5.6.SEC03.jar - D:\reposi
spring-context-support-2.5.6.SEC03.jar - D
servlet-api-2.5.jar - D:\repository\javax\ser
jstl-1.1.2.jar - D:\repository\javax\servlet\js
standard-1.1.2.jar - D:\repository\taglibs\st
tiles-jsp-2.0.7.jar - D:\repository\org\apach
tiles-core-2.0.7.jar - D:\repository\org\apa
tiles-api-2.0.7.jar - D:\repository\org\apach
commons-digester-1.8.jar - D:\repository\
commons-beanutils-1.7.0.jar - D:\reposito
commons-logging-api-1.1.jar - D:\reposit
junit-3.8.1.jar - D:\repository\junit\junit\3.
rxtx-2.1.7.jar - D:\repository\org\rxtx\rxtx\2
JavaScript Resources
src
target
pom.xml
Servers
IOPort Operations
PicControlWebApp/pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http:
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
<modelVersion>4.0.0</modelVersion>
<groupId>jvs.spring.pic</groupId>
<artifactId>PicControlWebApp</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>PicControlWebApp Maven Webapp</name>
<url>http://maven.apache.org</url>
<properties>
<spring.version>2.5.6.SEC03</spring.version>
<java.version>1.6</java.version>
<servlet.version>2.5</servlet.version>
<jstl.version>1.1.2</jstl.version>
<jUnit.version>3.8.1</jUnit.version>
<tiles.version>2.0.7</tiles.version>
<RXTXcomm.version>2.1.7</RXTXcomm.version>
</properties>
<dependencies>
<!-- Spring -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-web</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>${spring.version}</version>
</dependency>
<!-- Servlet -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>servlet-api</artifactId>
<version>${servlet.version}</version>
<scope>provided</scope>
</dependency>
<!-- JSTL -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version>${jstl.version}</version>

Apéndice C. Código Fuente (Parte hardware)

C.1 POT_ADC_USB_TestProject.c

```
#include <18F4550.h>
#device adc=10
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000)

#define USB_CON_SENSE_PIN PIN_D3

//leds ordered from bottom to top
#DEFINE LED1 PIN_C6  //yellow
#define LED2 PIN_C7  //green

#define LED_ON(x) output_high(x)
#define LED_OFF(x) output_low(x)

// Includes all USB code and interrupts, as well as the CDC API
#include <usb_cdc.h>
#include <lcd.c>


void usb_debug_task(void);

void main() {
    //set_tris_a(0x0); // configure ports as output
    set_tris_b(0x0);
    set_tris_c(0x0);
    set_tris_d(0x0);
    set_tris_e(0x0);
    //-----------------------------------
    disable_interrupts(global);
    disable_interrupts(int_timer1);
    disable_interrupts(int_rda);
    disable_interrupts(int_ext);
    disable_interrupts(int_ext1);
    disable_interrupts(int_ext2);
    //setup_adc_ports(NO_ANALOGS);
    //setup_adc(ADC_OFF);
    setup_spi(FALSE);
    setup_psp(PSP_DISABLED);
    setup_comparator(NC_NC_NC_NC);
    setup_vref(FALSE);
    port_b_pullups(FALSE);
    //-----------------------------------------

    //output_a(0); // saca un nivel bajo de salida en los puertos
    output_b(0);
    output_c(0);
    output_d(0);
    output_e(0);

    setup_adc_ports(AN0);
    setup_adc(ADC_CLOCK_INTERNAL);
    set_adc_channel(0);
```

```
    //Tiempo para que se configure el puerto como ADC
    delay_us(20);

    LED_OFF(LED1);
    LED_OFF(LED2);

    lcd_init();
    usb_init_cs();

    lcd_putc("\fInitializing...");

    int16 q;
    float p;

    while (TRUE) {
        usb_task();
        usb_debug_task();

        if (usb_enumerated()) {

            //Read the analog port///////////////
            q = read_adc();
            p = 5.0 * q / 1024.0;
            printf(lcd_putc, "\f%1.2fV", p);
            //Send the value to the port/////////  sample: '3.08V>'
            printf(usb_cdc_putc, "%1.2fV>", p);
            ////////////////////////////////////

            delay_ms(50);
        }
    }
}

void usb_debug_task(void) {
    static int8 last_connected;
    static int8 last_enumerated;
    int8 new_connected;
    int8 new_enumerated;

    new_connected = usb_attached();
    new_enumerated = usb_enumerated();

    if (new_connected)
        LED_ON(LED1);
    else
        LED_OFF(LED1);

    if (new_enumerated)
        LED_ON(LED2);
    else
        LED_OFF(LED2);

    if (new_connected && !last_connected)
        lcd_putc("\fUSB CONN, WFE"); //"USB connected, waiting for enumaration...
    if (!new_connected && last_connected)
        lcd_putc("\fUSB DIS, WFC"); //USB disconnected, waiting for connection...
    if (new_enumerated && !last_enumerated)
        lcd_putc("\fUSB ENUM"); //"USB enumerated by PC/HOST
    if (!new_enumerated && last_enumerated)
```

```
        lcd_putc("\fUSB ENUM, WFE"); //"USB unenumerated by PC/HOST, waiting for
enumeration...

    last_connected = new_connected;
    last_enumerated = new_enumerated;
}
```

C.2 PicControlWebApp.c

```
#include <18F4550.h>
#device adc=10
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000)

#define USB_CON_SENSE_PIN PIN_D3

//leds ordered from bottom to top
#DEFINE LED1 PIN_C6  //yellow
#define LED2 PIN_C7  //green

#define LED_ON(x) output_high(x)
#define LED_OFF(x) output_low(x)

// Includes all USB code and interrupts, as well as the CDC API
#include <usb_cdc.h>
#include <lcd.c>

const int lenbuf = 16;
char recbuf[lenbuf];
int ban = 1;

void usb_debug_task(void);

void main() {
    //set_tris_a(0x0); // configure ports as output
    set_tris_b(0x0);
    set_tris_c(0x0);
    set_tris_d(0x0);
    set_tris_e(0x0);
    //-----------------------------------
    disable_interrupts(global);
    disable_interrupts(int_timer1);
    disable_interrupts(int_rda);
    disable_interrupts(int_ext);
    disable_interrupts(int_ext1);
    disable_interrupts(int_ext2);
    //setup_adc_ports(NO_ANALOGS);
    //setup_adc(ADC_OFF);
    setup_spi(FALSE);
    setup_psp(PSP_DISABLED);
    setup_comparator(NC_NC_NC_NC);
    setup_vref(FALSE);
    port_b_pullups(FALSE);
    //-------------------------------------------

    //output_a(0); // saca un nivel bajo de salida en los puertos
    output_b(0);
    output_c(0);
    output_d(0);
```

```
output_e(0);

setup_adc_ports(AN0);
setup_adc(ADC_CLOCK_INTERNAL);
set_adc_channel(0);
delay_us(20);

LED_OFF(LED1);
LED_OFF(LED2);

lcd_init();
usb_init_cs();

lcd_putc("\fInitializing...");

int16 val;
float temp = 0;
float tempAux = 0;

while (TRUE) {
    usb_task();
    usb_debug_task();

    if (usb_enumerated()) {

        //Cleans the buffer//////////////////
        int x = 0;
        for (x = 0; x < sizeof (recbuf); x++) {
            recbuf[x] = ' ';
        }
        ////////////////////////////////////


        //Read the analog port///////////////
        val = read_adc();
        temp = ((val / 1023.0)*500);

        if (tempAux != temp) {
            printf(lcd_putc, "\f%2.1f C", temp);
            //Send the value to the port/////////  sample: '27.9>'
            printf(usb_cdc_putc, "%2.1f>", temp);
            tempAux = temp;
        }
        ////////////////////////////////////


        //Receives values from port/////////  sample: 'A2:1<'
        char c;
        int a = 0;
        while (usb_cdc_kbhit()) {
            c = usb_cdc_getc();
            recbuf[a] = c;
            a++;
            if (c == '<') {
                break;
            }
        }
        printf(lcd_putc, "\n%s", recbuf);
        ////////////////////////////////////
```

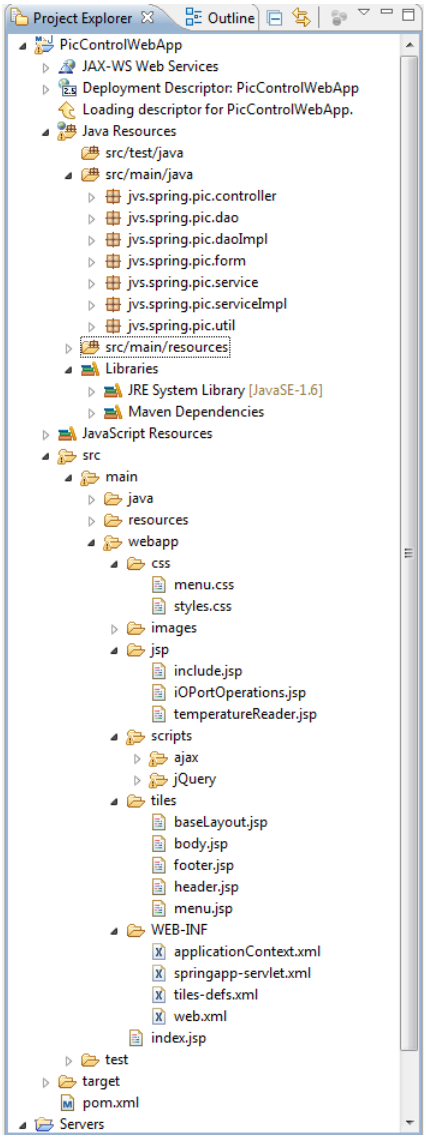
```
//Filters and operate the PIC IO/////
if (recbuf[0] == 'A') {
    if (recbuf[1] == '0') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_A0);
        } else {
            LED_OFF(PIN_A0);
        }
    }
    if (recbuf[1] == '1') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_A1);
        } else {
            LED_OFF(PIN_A1);
        }
    }
    if (recbuf[1] == '2') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_A2);
        } else {
            LED_OFF(PIN_A2);
        }
    }
    if (recbuf[1] == '3') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_A3);
        } else {
            LED_OFF(PIN_A3);
        }
    }
    if (recbuf[1] == '4') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_A4);
        } else {
            LED_OFF(PIN_A4);
        }
    }
    if (recbuf[1] == '5') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_A5);
        } else {
            LED_OFF(PIN_A5);
        }
    }
} else if (recbuf[0] == 'B') {
    if (recbuf[1] == '0') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_B0);
        } else {
            LED_OFF(PIN_B0);
        }
    }
    if (recbuf[1] == '1') {
        if (recbuf[3] == '1') {
            LED_ON(PIN_B1);
        } else {
            LED_OFF(PIN_B1);
        }
    }
```

```
                if (recbuf[1] == '2') {
                    if (recbuf[3] == '1') {
                        LED_ON(PIN_B2);
                    } else {
                        LED_OFF(PIN_B2);
                    }
                }
                if (recbuf[1] == '3') {
                    if (recbuf[3] == '1') {
                        LED_ON(PIN_B3);
                    } else {
                        LED_OFF(PIN_B3);
                    }
                }
                if (recbuf[1] == '4') {
                    if (recbuf[3] == '1') {
                        LED_ON(PIN_B4);
                    } else {
                        LED_OFF(PIN_B4);
                    }
                }
                if (recbuf[1] == '5') {
                    if (recbuf[3] == '1') {
                        LED_ON(PIN_B5);
                    } else {
                        LED_OFF(PIN_B5);
                    }
                }
                if (recbuf[1] == '6') {
                    if (recbuf[3] == '1') {
                        LED_ON(PIN_B6);
                    } else {
                        LED_OFF(PIN_B6);
                    }
                }
                if (recbuf[1] == '7') {
                    if (recbuf[3] == '1') {
                        LED_ON(PIN_B7);
                    } else {
                        LED_OFF(PIN_B7);
                    }
                }
            }
            ////////////////////////////////////
            delay_ms(100);
        }
    }
}

void usb_debug_task(void) {
    static int8 last_connected;
    static int8 last_enumerated;
    int8 new_connected;
    int8 new_enumerated;

    new_connected = usb_attached();
    new_enumerated = usb_enumerated();

    if (new_connected)
        LED_ON(LED1);
```

```
    else
        LED_OFF(LED1);

    if (new_enumerated)
        LED_ON(LED2);
    else
        LED_OFF(LED2);

    if (new_connected && !last_connected)
        lcd_putc("\fUSB CONN, WFE"); //"USB connected, waiting for enumaration...
    if (!new_connected && last_connected)
        lcd_putc("\fUSB DIS, WFC"); //USB disconnected, waiting for connection...
    if (new_enumerated && !last_enumerated)
        lcd_putc("\fUSB ENUM"); //"USB enumerated by PC/HOST
    if (!new_enumerated && last_enumerated)
        lcd_putc("\fUSB ENUM, WFE"); //"USB unenumerated by PC/HOST, waiting for
enumeration...

    last_connected = new_connected;
    last_enumerated = new_enumerated;
```

Apéndice D. Código Fuente (Parte software)

Los componentes que conforman la aplicación Web se presentan a continuación de acuerdo a la siguiente imagen de estructura de paquetes y directorios:



IOPortController

```
-----------------------------------------------------------------------------------------------
package jvs.spring.pic.controller;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;

import jvs.spring.pic.form.PicPorts;
```

```
import jvs.spring.pic.service.IOPortService;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;

public class IOPortController extends SimpleFormController {

	/** Logger for this class and subclasses */
	protected final Log logger = LogFactory.getLog(getClass());

	private IOPortService iOPortService = null;

	public void setiOPortService(IOPortService iOPortService) {
		this.iOPortService = iOPortService;
	}

	protected Object formBackingObject(HttpServletRequest request) throws
ServletException {
		PicPorts picPorts = new PicPorts();
		return picPorts;
	}

	public ModelAndView onSubmit(Object command) throws ServletException {
		PicPorts picPorts = (PicPorts) command;

		iOPortService.changeIOPortsState(picPorts);

		String now = (new java.util.Date()).toString();
		ModelAndView mv = new ModelAndView(getSuccessView());
		mv.addObject("picPorts", picPorts);
		mv.addObject("actualTime", now);

		logger.info("Returning from IOPortController, view to " + getSuccessView());
		return mv;
	}

}
```

TempMonitorController

```
package jvs.spring.pic.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;

public class TempMonitorController extends AbstractController {

	/** Logger for this class and subclasses */
	protected final Log logger = LogFactory.getLog(getClass());

	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest request,
HttpServletResponse response) throws Exception {
```

```
		ModelAndView mv = new ModelAndView("tileTemperatureReader");
		return mv;
	}

}
```

TempMonitorServlet

```
package jvs.spring.pic.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jvs.spring.pic.service.IOPortService;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.web.HttpRequestHandler;

/**
 * Servlet implementation class TempMonitorServlet
 */
public class TempMonitorServlet extends HttpServlet implements HttpRequestHandler {
	private static final long serialVersionUID = 1L;

	/** Logger for this class and subclasses */
	protected final Log logger = LogFactory.getLog(getClass());

	private IOPortService iOPortService = null;

	public void setiOPortService(IOPortService iOPortService) {
		this.iOPortService = iOPortService;
	}

	public TempMonitorServlet() {
		super();
	}

	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html;charset=UTF-8");
		PrintWriter out = response.getWriter();

		try {

			String temp = iOPortService.retrieveTemperatureValue();
			if (temp == null) {
				temp = "loading...";
			}
			out.println(temp + "°C");
			out.close();

		} catch (Exception e) {
			// e.printStackTrace();
		}
```

```
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		doGet(request, response);
	}

	@Override
	public void handleRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		doGet(request, response);
	}

}
```

IOPortDAO

```
package jvs.spring.pic.dao;

public interface IOPortDAO {

	public void updateIOPortState(String instruction);

	public String readTemperature();

}
```

IOPortDAOImpl

```
package jvs.spring.pic.daoImpl;

import java.io.IOException;

import jvs.spring.pic.dao.IOPortDAO;
import jvs.spring.pic.util.SerialComm;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class IOPortDAOImpl implements IOPortDAO {

	/** Logger for this class and subclasses */
	protected final Log logger = LogFactory.getLog(getClass());

	SerialComm serialComm;

	public IOPortDAOImpl() {
	}

	public void setSerialComm(SerialComm serialComm) {
		this.serialComm = serialComm;
	}

	public void updateIOPortState(String instruction) {

		try {
			serialComm.connect("COM7");
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
```

```
			}

			if (instruction != null) {

				byte[] bout = new byte[5];
				bout[0] = (byte) instruction.charAt(0); // sample: 'A'
				bout[1] = (byte) instruction.charAt(1); // sample: '0'
				bout[2] = (byte) instruction.charAt(2); // sample: ':'
				bout[3] = (byte) instruction.charAt(3); // sample: '1'
				bout[4] = (byte) instruction.charAt(4); // sample: '<'
                                                                                    // '0'
= OFF / '1' = ON
				String value = new String(bout);
				logger.info("Writting: " + value);

				try {
					serialComm.out.write(bout);
				} catch (IOException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}

			}
		}

		public String readTemperature() {
			try {
				serialComm.connect("COM7");
			} catch (Exception e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

			// Reads the value from the readingThread
			return this.serialComm.tRead1.getValue();
		}

}
```

PicPorts

```
-----------------------------------------------------------------------------------------------
package jvs.spring.pic.form;

import java.io.Serializable;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class PicPorts implements Serializable {

	private static final long serialVersionUID = -589586375522595949 6L;

	/** Logger for this class and subclasses */
	protected final Log logger = LogFactory.getLog(getClass());

	private String portA0;
	private String portA1;
	private String portA2;
	private String portA3;
	private String portA4;
```

```
private String portA5;

private String portB0;
private String portB1;
private String portB2;
private String portB3;
private String portB4;
private String portB5;
private String portB6;
private String portB7;

public String getPortA0() {
	return portA0;
}

public String getPortA1() {
	return portA1;
}

public String getPortA2() {
	return portA2;
}

public String getPortA3() {
	return portA3;
}

public String getPortA4() {
	return portA4;
}

public String getPortA5() {
	return portA5;
}

public void setPortA0(String portA0) {
	this.portA0 = portA0;
	// logger.info("this.portA0 = " + portA0);
}

public void setPortA1(String portA1) {
	this.portA1 = portA1;
	// logger.info("this.portA1 = " + portA1);
}

public void setPortA2(String portA2) {
	this.portA2 = portA2;
	// logger.info("this.portA2 = " + portA2);
}

public void setPortA3(String portA3) {
	this.portA3 = portA3;
	// logger.info("this.portA3 = " + portA3);
}

public void setPortA4(String portA4) {
	this.portA4 = portA4;
	// logger.info("this.portA4 = " + portA4);
}
```

```
public void setPortA5(String portA5) {
	this.portA5 = portA5;
	// logger.info("this.portA5 = " + portA5);
}

public String getPortB0() {
	return portB0;
}

public void setPortB0(String portB0) {
	this.portB0 = portB0;
	// logger.info("this.portB0 = " + portB0);
}

public String getPortB1() {
	return portB1;
}

public void setPortB1(String portB1) {
	this.portB1 = portB1;
	// logger.info("this.portB1 = " + portB1);
}

public String getPortB2() {
	return portB2;
}

public void setPortB2(String portB2) {
	this.portB2 = portB2;
	// logger.info("this.portB2 = " + portB2);
}

public String getPortB3() {
	return portB3;
}

public void setPortB3(String portB3) {
	this.portB3 = portB3;
	// logger.info("this.portB3 = " + portB3);
}

public String getPortB4() {
	return portB4;
}

public void setPortB4(String portB4) {
	this.portB4 = portB4;
	// logger.info("this.portB4 = " + portB4);
}

public String getPortB5() {
	return portB5;
}

public void setPortB5(String portB5) {
	this.portB5 = portB5;
	// logger.info("this.portB5 = " + portB5);
}

public String getPortB6() {
```

```
		return portB6;
	}

	public void setPortB6(String portB6) {
		this.portB6 = portB6;
		// logger.info("this.portB6 = " + portB6);
	}

	public String getPortB7() {
		return portB7;
	}

	public void setPortB7(String portB7) {
		this.portB7 = portB7;
		// logger.info("this.portB7 = " + portB7);
	}

}
```

IOPortService

```
package jvs.spring.pic.service;

import jvs.spring.pic.form.PicPorts;

public interface IOPortService {

	public void changeIOPortsState(PicPorts picPorts);

	public String retrieveTemperatureValue();

}
```

IOPortServiceImpl

```
package jvs.spring.pic.serviceImpl;

import jvs.spring.pic.dao.IOPortDAO;
import jvs.spring.pic.form.PicPorts;
import jvs.spring.pic.service.IOPortService;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class IOPortServiceImpl implements IOPortService {

	/** Logger for this class and subclasses */
	protected final Log logger = LogFactory.getLog(getClass());

	IOPortDAO iOPortDAO;

	public IOPortServiceImpl() {
	}

	public void setiOPortDAO(IOPortDAO iOPortDAO) {
		this.iOPortDAO = iOPortDAO;
	}

	public void changeIOPortsState(PicPorts picPorts) {
```

```
			if (picPorts != null) {

				// usbSendCommand.sendCommand(picCommand.getPortA0() == null ?
				// "A0:0" :
				// "A0:1");
				iOPortDAO.updateIOPortState(picPorts.getPortA1() == null ? "A1:0<" :
"A1:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortA2() == null ? "A2:0<" :
"A2:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortA3() == null ? "A3:0<" :
"A3:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortA4() == null ? "A4:0<" :
"A4:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortA5() == null ? "A5:0<" :
"A5:1<");

				iOPortDAO.updateIOPortState(picPorts.getPortB0() == null ? "B0:0<" :
"B0:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB1() == null ? "B1:0<" :
"B1:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB2() == null ? "B2:0<" :
"B2:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB3() == null ? "B3:0<" :
"B3:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB4() == null ? "B4:0<" :
"B4:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB5() == null ? "B5:0<" :
"B5:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB6() == null ? "B6:0<" :
"B6:1<");
				iOPortDAO.updateIOPortState(picPorts.getPortB7() == null ? "B7:0<" :
"B7:1<");
			}
		}

		public String retrieveTemperatureValue() {
			return iOPortDAO.readTemperature();
		}

}
```

SerialComm

```
package jvs.spring.pic.util;

import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

import java.io.InputStream;
import java.io.OutputStream;

public class SerialComm {

		public InputStream in;
		public OutputStream out;

		public SerialReader tRead1;

		CommPortIdentifier portIdentifier;
```

```
	CommPort commPort;
	SerialPort serialPort;

	// ------------------------------------------------------------
	public SerialComm() {
		super();
	}

	// ------------------------------------------------------------
	public void connect(String portName) throws Exception {
		portIdentifier = CommPortIdentifier.getPortIdentifier(portName);

		if (!portIdentifier.isCurrentlyOwned()) {
			commPort = portIdentifier.open(this.getClass().getName(), 2000);
			if (commPort instanceof SerialPort) {
				serialPort = (SerialPort) commPort;
				serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8,
SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);

				in = serialPort.getInputStream();
				out = serialPort.getOutputStream();

				tRead1 = new SerialReader(in);
				tRead1.start();

			} else {
				System.out.println("The selected port, is not a SerialPort");
			}
		} else {
			// System.out.println("Port is currently in use");
		}

	}

	// ------------------------------------------------------------
	public static void main(String[] args) {
		SerialComm serialComm = new SerialComm();

		try {
			serialComm.connect("COM7");

			byte[] bout = new byte[5];

			bout[0] = (byte) 'A'; // sample: 'A'
			bout[1] = (byte) '1'; // sample: '1'
			bout[2] = (byte) ':'; // sample: ':'
			bout[3] = (byte) '1'; // sample: '1'
			bout[4] = (byte) '<'; // sample: '<'
			serialComm.out.write(bout);

			bout[0] = (byte) 'A'; // sample: 'A'
			bout[1] = (byte) '3'; // sample: '3'
			bout[2] = (byte) ':'; // sample: ':'
			bout[3] = (byte) '1'; // sample: '1'
			bout[4] = (byte) '<'; // sample: '<'
			serialComm.out.write(bout);

			bout[0] = (byte) 'A'; // sample: 'A'
			bout[1] = (byte) '5'; // sample: '5'
			bout[2] = (byte) ':'; // sample: ':'
```

```
                    bout[3] = (byte) '1'; // sample: '1'
                    bout[4] = (byte) '<'; // sample: '<'
                    serialComm.out.write(bout);

                    for (int i = 0; i < 5; i++) {
                          System.out.println("Reading tRead1: " +
serialComm.tRead1.getValue());
                          Thread.sleep(500);
                    }

             } catch (Exception e) {
                    e.printStackTrace();
             } finally {
                    System.out.println("Closing port and ending thread");
                    serialComm.tRead1 = null;

             }
       }
}
```

SerialReader

```
package jvs.spring.pic.util;

import java.io.IOException;
import java.io.InputStream;

public class SerialReader extends Thread {
       String value = "";

       public String getValue() {
             return value;
       }

       public void setValue(String value) {
             this.value = value;
       }

       InputStream in;

       // ------------------------------------------------------------
       public SerialReader(InputStream in) {
             this.in = in;
       }

       // ------------------------------------------------------------
       public void run() {
             int len = -1;
             StringBuilder cadena = new StringBuilder();
             char c;
             try {
                    while (true) {
                          while ((len = this.in.read()) > -1) {
                                c = (char) len;
                                if (c == '>') {
                                      value = cadena.toString();
                                      cadena = new StringBuilder();
                                      // System.out.println(value);
                                } else {
                                      cadena.append(c);
```

```
                                        }
                                  }
                            }
                      } catch (IOException e) {
                            e.printStackTrace();
                      }
               }
}
```

messages.properties

```
version.number=1.0
home.welcome=Welcome

iOPortOperations.heading=IOPort Operations
TemperatureReader.heading=Temperature Reader
```

menu.css

```
.glossymenu {
       margin: 5px 0;
       padding: 0;
       width: 170px; /*width of menu*/
       border: 1px solid #9A9A9A;
       border-bottom-width: 0;
}

.glossymenu a.menuitem {
       background: black url(../images/menu/glossyback.gif) repeat-x bottom left;
       font: bold 14px "Lucida Grande", "Trebuchet MS", Verdana, Helvetica,
             sans-serif;
       color: white;
       display: block;
       position: relative;
       /*To help in the anchoring of the ".statusicon" icon image*/
       width: auto;
       padding: 4px 0;
       padding-left: 10px;
       text-decoration: none;
}

.glossymenu a.menuitem:visited,.glossymenu .menuitem:active {
       color: white;
}

.glossymenu a.menuitem .statusicon {
       /*CSS for icon image that gets dynamically added to headers*/
       position: absolute;
       top: 5px;
       right: 5px;
       border: none;
}

.glossymenu a.menuitem:hover {
       background-image: url(../images/menu/glossyback2.gif);
}

.glossymenu div.submenu { /*DIV that contains each sub menu*/
       background: white;
}
```

```
.glossymenu div.submenu ul { /*UL of each sub menu*/
	list-style-type: none;
	margin: 0;
	padding: 0;
}

.glossymenu div.submenu ul li {
	border-bottom: 1px solid blue;
}

.glossymenu div.submenu ul li a {
	display: block;
	font: normal 13px "Lucida Grande", "Trebuchet MS", Verdana, Helvetica,
		sans-serif;
	color: black;
	text-decoration: none;
	padding: 2px 0;
	padding-left: 10px;
}

.glossymenu div.submenu ul li a:hover {
	background: #DFDCCB;
	colorz: white;
}
```

styles.css

```
BODY {
	PADDING-RIGHT: 0px;
	PADDING-LEFT: 0px;
	FONT-SIZE: 11px;
	BACKGROUND-IMAGE: url(../images/imgBackGral.jpg);
	PADDING-BOTTOM: 0px;
	MARGIN: 0px;
	WIDTH: 100%;
	PADDING-TOP: 0px;
	BACKGROUND-REPEAT: repeat-x;
	FONT-FAMILY: Arial, Helvetica, sans-serif;
	HEIGHT: 100%;
	BACKGROUND-COLOR: #ffffff;
	min-width: 990px;
}

BODY#foriframe {
	PADDING-RIGHT: 0px;
	PADDING-LEFT: 0px;
	FONT-SIZE: 11px;
	BACKGROUND-IMAGE: none;
	PADDING-BOTTOM: 0px;
	MARGIN: 0px;
	PADDING-TOP: 0px;
	FONT-FAMILY: Arial, Helvetica, sans-serif;
	BACKGROUND-COLOR: #fff
}

HTML {
	HEIGHT: 100%;
	min-width: 990px
}
```

```
TABLE#maincontent {
	HEIGHT: 83%
}

#header {
	BACKGROUND-POSITION: right 50%;
	BACKGROUND-IMAGE: url(../images/imgBackHeader.jpg);
	BACKGROUND-REPEAT: no-repeat;
	HEIGHT: 83px;
	min-width: 995px
}

.hedaderbar {
	BACKGROUND-POSITION: 50% bottom;
	BACKGROUND-IMAGE: url(../images/imgBackHeader_bar.jpg);
	VERTICAL-ALIGN: bottom;
	BACKGROUND-REPEAT: repeat-x
}

.boxheaderbar {
	PADDING-RIGHT: 15px;
	DISPLAY: block;
	PADDING-LEFT: 15px;
	PADDING-BOTTOM: 0px;
	WIDTH: 455px;
	PADDING-TOP: 0px;
	HEIGHT: 24px
}

.boxheaderbar .txtgris {
	MARGIN-TOP: 5px;
	FLOAT: left;
	COLOR: #666666
}

.boxheaderbar .usr {
	FONT-WEIGHT: bold;
	FLOAT: left;
	MARGIN: 5px 0px 0px 5px;
	COLOR: #33689b
}

H1 {
	FONT-WEIGHT: bold;
	FONT-SIZE: 16px;
	MARGIN: 0px 0px 5px;
	COLOR: #666666
}

H2 {
	FONT-WEIGHT: bold;
	FONT-SIZE: 14px;
	MARGIN: 0px 0px 5px;
	COLOR: #26598e
}

H3 {
	FONT-WEIGHT: bold;
	FONT-SIZE: 12px;
```

```
	MARGIN: 0px 0px 7px;
	COLOR: #132e4f
}

P {
	FONT-WEIGHT: normal;
	MARGIN: 0px 0px 5px;
	COLOR: #333
}

P.bold {
	FONT-WEIGHT: bold;
	MARGIN: 5px 0px;
	COLOR: #000
}

SELECT {
	BORDER-RIGHT: #a4a8ab 1px solid;
	BORDER-TOP: #a4a8ab 1px solid;
	FONT-SIZE: 11px;
	BORDER-LEFT: #a4a8ab 1px solid;
	COLOR: #000;
	BORDER-BOTTOM: #a4a8ab 1px solid;
	FONT-FAMILY: Arial, Helvetica, sans-serif
}

TEXTAREA {
	BORDER-RIGHT: #a4a8ab 1px solid;
	BORDER-TOP: #a4a8ab 1px solid;
	FONT-SIZE: 11px;
	BORDER-LEFT: #a4a8ab 1px solid;
	COLOR: #000;
	BORDER-BOTTOM: #a4a8ab 1px solid;
	FONT-FAMILY: Arial, Helvetica, sans-serif
}

INPUT.txt {
	BORDER-RIGHT: #a4a8ab 1px solid;
	BORDER-TOP: #a4a8ab 1px solid;
	FONT-SIZE: 11px;
	BORDER-LEFT: #a4a8ab 1px solid;
	COLOR: #000;
	BORDER-BOTTOM: #a4a8ab 1px solid;
	FONT-FAMILY: Arial, Helvetica, sans-serif
}

INPUT.txt_disabled {
	BACKGROUND-COLOR: #efebde;
	BORDER-RIGHT: #a4a8ab 1px solid;
	BORDER-TOP: #a4a8ab 1px solid;
	FONT-SIZE: 11px;
	BORDER-LEFT: #a4a8ab 1px solid;
	COLOR: #000;
	BORDER-BOTTOM: #a4a8ab 1px solid;
	FONT-FAMILY: Arial, Helvetica, sans-serif
}

.text {
	BORDER-RIGHT: #a4a8ab 1px solid;
	BORDER-TOP: #a4a8ab 1px solid;
```

```
	FONT-SIZE: 11px;
	BORDER-LEFT: #a4a8ab 1px solid;
	COLOR: #000;
	BORDER-BOTTOM: #a4a8ab 1px solid;
	FONT-FAMILY: Arial, Helvetica, sans-serif
}

INPUT.txt_white {
	BORDER-TOP-WIDTH: 0px;
	BORDER-LEFT-WIDTH: 0px;
	FONT-SIZE: 11px;
	BORDER-BOTTOM-WIDTH: 0px;
	COLOR: #000;
	FONT-FAMILY: Arial, Helvetica, sans-serif;
	BORDER-RIGHT-WIDTH: 0px
}

UNKNOWN {
	BORDER-TOP-WIDTH: 0px;
	BORDER-LEFT-WIDTH: 0px;
	BORDER-BOTTOM-WIDTH: 0px;
	BORDER-RIGHT-WIDTH: 0px
}

INPUT.file {
	BORDER-RIGHT: #a4a8ab 1px solid;
	BORDER-TOP: #a4a8ab 1px solid;
	FONT-SIZE: 11px;
	BORDER-LEFT: #a4a8ab 1px solid;
	COLOR: #000;
	BORDER-BOTTOM: #a4a8ab 1px solid;
	FONT-FAMILY: Arial, Helvetica, sans-serif;
}

INPUT.blue_button {
	BORDER-TOP-WIDTH: 1px;
	PADDING-RIGHT: 4px;
	PADDING-LEFT: 4px;
	BORDER-LEFT-WIDTH: 1px;
	FONT-SIZE: 11px;
	BACKGROUND-IMAGE: url(../images/imgBgButtomBlue.gif);
	BORDER-BOTTOM-WIDTH: 1px;
	PADDING-BOTTOM: 0px;
	COLOR: #ffffff;
	PADDING-TOP: 0px;
	BACKGROUND-COLOR: #2e5f8d;
	BORDER-RIGHT-WIDTH: 1px;
	width: auto;
	min-width: 50px;
	height: auto;
	min-height: 17px;
}

INPUT.gray_button {
	BORDER-TOP-WIDTH: 1px;
	PADDING-RIGHT: 4px;
	PADDING-LEFT: 4px;
	BORDER-LEFT-WIDTH: 1px;
	FONT-SIZE: 11px;
	BACKGROUND-IMAGE: url(../images/imgBgButtomGray.jpg);
```

```
	BORDER-BOTTOM-WIDTH: 1px;
	PADDING-BOTTOM: 0px;
	COLOR: #000;
	PADDING-TOP: 0px;
	BACKGROUND-COLOR: #2e5f8d;
	BORDER-RIGHT-WIDTH: 1px;
	height: auto;
	min-height: 17px;
}

table.CSSTablePorts {
	width: 60%;
	height: 100%;
	margin: 0px;
	padding: 0px;
	border: 1px solid #000000;
}

table.CSSTablePorts tr:hover td {
	background-color: #d6e7ff;
}

table.CSSTablePorts td {
	vertical-align: middle;
	background-color: #ffffff;
	border: 0px solid #000000;
	border-width: 0px 0px 0px 0px;
	text-align: left;
	padding: 7px;
	font-size: 12px;
	font-family: Arial;
	font-weight: normal;
	color: #000000;
}

table.CSSTablePorts tr:first-child td {
	background: -o-linear-gradient(bottom, #5285e6 5%, #74a6ec 100%);
	background: -webkit-gradient(linear, left top, left bottom, color-stop(0.05,
#5285e6), color-stop(1, #74a6ec) );
	background: -moz-linear-gradient(center top, #5285e6 5%, #74a6ec 100%);
	filter:
progid:DXImageTransform.Microsoft.gradient(startColorstr="#5285e6",endColorstr="#74a6ec" );
	background: -o-linear-gradient(top, #5285e6, 74a6ec);
	background-color: #5285e6;
	border: 0px solid #000000;
	text-align: center;
	border-width: 0px 0px 0px 0px;
	font-size: 14px;
	font-family: Arial;
	font-weight: bold;
	color: #ffffff;
}


table.foggycountry {
	width: 50%;
	height: 20%;
	margin: 0px;
	padding: 0px;
```

```
	border-collapse: collapse;
	border: 0px solid #000000;
	font: normal 30px verdana, arial, helvetica, sans-serif;
	color: #000000;
	background: #BFE756;


	-moz-border-radius-bottomleft:12px;
	-webkit-border-bottom-left-radius:12px;
	border-bottom-left-radius:12px;

	-moz-border-radius-bottomright:12px;
	-webkit-border-bottom-right-radius:12px;
	border-bottom-right-radius:12px;

	-moz-border-radius-topright:12px;
	-webkit-border-top-right-radius:12px;
	border-top-right-radius:12px;

	-moz-border-radius-topleft:12px;
	-webkit-border-top-left-radius:12px;
	border-top-left-radius:12px;
}

table.foggycountry td {
	color: #000000;
	border: 0px;
	padding: .3em;
	width: 50%;
	height: 100%;
	margin: 0px;
	text-align: center;
}
```

include.jsp

```
<%@ page session="false"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<script src="<%=request.getContextPath()%>/scripts/ajax/prototype.js"
type="text/javascript"></script>
<script src="<%=request.getContextPath()%>/scripts/jQuery/jquery-1.9.1.js"
type="text/javascript"></script>

<%@ include file="include.jsp"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@page contentType="text/html;charset=UTF-8"%>
```

iOPortOperations.jsp

```
<html>
<head>
	<title><fmt:message key="title" /></title>
</head>

<body>

	<br>
```

```
	<h1>
		<fmt:message key="iOPortOperations.heading" />
	</h1>

	<form:form method="post" commandName="picPorts">
		<table cellspacing="0" cellpadding="0" class="CSSTablePorts">
			<tr>
				<td width="50%">PORT A</td>
				<td width="50%">PORT B</td>
			</tr>
			<tr>
				<td><form:checkbox path="portA0" value="true" disabled="true" />
port_A0 (Config as analog)</td>
				<td><form:checkbox path="portB7" value="true" /> port_B7</td>
			</tr>
			<tr>
				<td><form:checkbox path="portA1" value="true" /> port_A1</td>
				<td><form:checkbox path="portB6" value="true" /> port_B6</td>
			</tr>
			<tr>
				<td><form:checkbox path="portA2" value="true" /> port_A2</td>
				<td><form:checkbox path="portB5" value="true" /> port_B5</td>
			</tr>
			<tr>
				<td><form:checkbox path="portA3" value="true" /> port_A3</td>
				<td><form:checkbox path="portB4" value="true" /> port_B4</td>
			</tr>
			<tr>
				<td><form:checkbox path="portA4" value="true" /> port_A4</td>
				<td><form:checkbox path="portB3" value="true" /> port_B3</td>
			</tr>
			<tr>
				<td><form:checkbox path="portA5" value="true" /> port_A5</td>
				<td><form:checkbox path="portB2" value="true" /> port_B2</td>
			</tr>
			<tr>
				<td></td>
				<td><form:checkbox path="portB1" value="true" /> port_B1</td>
			</tr>
			<tr>
				<td>

					<table>
						<tr>
							<td width="50%">Temperature:</td>
							<td width="50%"
id="tempRefreshData">loading...</td>
						</tr>
					</table>

				</td>
				<td><form:checkbox path="portB0" value="true" /> port_B0</td>
			</tr>
		</table>
		<br>
		<input type="submit" class="blue_button" align="center" value="Execute command"
/>
	</form:form>
```

```
</body>
</html>

<script language="JavaScript">
	$.ajaxSetup({
		// Disable caching of AJAX responses
		cache : false
	});

	setInterval("updateContent();", 200);

	function updateContent() {
		$('#tempRefreshData').load("TempMonitorServlet");
	}
</script>
```

temperatureReader.jsp

```
<%@ include file="include.jsp"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@page contentType="text/html;charset=UTF-8"%>

<html>
<head>
	<title><fmt:message key="title" /></title>
</head>

<body>

	<br />

	<h1>
		<fmt:message key="TemperatureReader.heading" />
	</h1>


	<table class="foggycountry">
		<tr>
			<td width="50%">Temperature</td>
			<td width="50%" id="tempRefreshData">loading...</td>
		</tr>
	</table>

</body>
</html>

<script language="JavaScript">
	$.ajaxSetup({
		// Disable caching of AJAX responses
		cache : false
	});

	setInterval("updateContent();", 250);

	function updateContent() {
		$('#tempRefreshData').load("TempMonitorServlet");
	}
</script>
```

baseLayout.jsp

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<link href="css/styles.css" rel="stylesheet" type="text/css" />

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><tiles:insertAttribute name="title" ignore="true" /></title>
</head>
<body>
	<table border="1" cellpadding="0" cellspacing="0" align="left" width="80%"
height="100%">

		<tr height="72px">
			<td width="100%" valign="middle" align="left"
colspan="2"><tiles:insertAttribute name="header" /></td>
		</tr>

		<tr height="85%">
			<td width="20%" valign="top" align="left"><tiles:insertAttribute
name="menu" /></td>
			<td width="80%" valign="top" align="left"><tiles:insertAttribute
name="body" /></td>
		</tr>

		<tr height="5%">
			<td width="100%" valign="middle" align="left"
colspan="2"><tiles:insertAttribute name="footer" /></td>
		</tr>

	</table>
</body>
</html>
```

body.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-
1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<table border="0" width="100%" height="100%">
		<tr>
			<td>BODY</td>
		</tr>
	</table>
</body>
</body>
</html>
```

footer.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-
1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<table border="0" width="100%" height="100%">
		<tr>
			<td align="right" style="font: normal 14px verdana, arial, helvetica,
sans-serif;">
			<script language="javascript">
				today = new Date();
				document.write(today.getDate(), "/", today.getMonth() + 1, "/",
today.getYear() + 1900);
			</script>
			</td>
		</tr>
	</table>
</body>
</html>
```

header.jsp

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<script type="text/javascript" src="scripts/general.js">
</script>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>

<body>
	<div id="header">
		<table width="100%" border="0" cellspacing="0" cellpadding="0">
			<tr>
				<td width="1%" height="83">
					<img
src="${pageContext.request.contextPath}/images/imgSpace.gif" alt="" width="1" height="1" />
					<img
src="${pageContext.request.contextPath}/images/imgSpace.gif" width="15" height="15" />
				</td>
				<td width="170" align="center"><img
src="${pageContext.request.contextPath}/images/pic.jpg" width="110" height="60"/></td>
				<td width="1%"><img
src="${pageContext.request.contextPath}/images/imgPicControlWebApp.jpg" width="182"
height="25" /></td>
				<td width="85%"> </td>
				<td width="0%" align="left" valign="bottom"><img
src="<%=request.getContextPath()%>/images/imgCorner.jpg" width="32" height="24" /></td>
				<td width="1%" valign="bottom" class="hedaderbar">

					<table width="100%" border="0" cellspacing="0"
cellpadding="0">
```

```
                                        <tr>
                                              <td valign="top"> </td>
                                        </tr>
                                        <tr>
                                              <td>
                                                    <div class="boxheaderbar">
                                                          <a href="javascript:logout()"
onmouseout="MM_swapImgRestore()"
onmouseover="MM_swapImage('Image6','','${pageContext.request.contextPath}/images/btn_logout_
on.jpg',1)" style="float: right;">
                                                                                <img
src="${pageContext.request.contextPath}/images/btn_logout_off.jpg" name="Image6" width="74"
height="24" border="0" align="absbottom" id="Image6" />
                                                          </a>
                                                          <span class="txtgris">Version
<fmt:message key="version.number" />  </span>
                                                          <span
class="txtgris"> <fmt:message key="home.welcome" /></span>

                                                    </div>
                                              </td>
                                        </tr>
                                  </table>

                            </td>
                      </tr>
                </table>
          </div>
</body>
</html>
```

menu.jsp

```
---------------------------------------------------------------------------------------------
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-
1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<head>
<link href="css/menu.css" rel="stylesheet" type="text/css" />

<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>
<script type="text/javascript" src="scripts/menu/ddaccordion.js">
      /***********************************************
       * Accordion Content script- (c) Dynamic Drive DHTML code library
(www.dynamicdrive.com)
       * Visit http://www.dynamicDrive.com for hundreds of DHTML scripts
       * This notice must stay intact for legal use
       ***********************************************/
</script>

<script type="text/javascript">
      ddaccordion.init({
            headerclass : "submenuheader", //Shared CSS class name of headers group
            contentclass : "submenu", //Shared CSS class name of contents group
            revealtype : "click", //Reveal content when user clicks or onmouseover the
header? Valid value: "click", "clickgo", or "mouseover"
            mouseoverdelay : 200, //if revealtype="mouseover", set delay in milliseconds
before header expands onMouseover
```

```
            collapseprev : true, //Collapse previous content (so only one open at any
time)? true/false
            defaultexpanded : [], //index of content(s) open by default [index1, index2,
etc] [] denotes no content
            onemustopen : false, //Specify whether at least one header should be open
always (so never all headers closed)
            animatedefault : false, //Should contents open by default be animated into
view?
            persiststate : true, //persist state of opened contents within browser session?
            toggleclass : [ "", "" ], //Two CSS classes to be applied to the header when
it's collapsed and expanded, respectively ["class1", "class2"]
            togglehtml : [ "suffix",
                        "<img src='images/menu/plus.gif' class='statusicon' />",
                        "<img src='images/menu/minus.gif' class='statusicon' />" ],
//Additional HTML added to the header when it's collapsed and expanded, respectively
["position", "html1", "html2"] (see docs)
            animatespeed : "fast", //speed of animation: integer in milliseconds (ie: 200),
or keywords "fast", "normal", or "slow"
            oninit : function(headers, expandedindices) { //custom code to run when headers
have initalized
                  //do nothing
            },
            onopenclose : function(header, index, state, isuseractivated) { //custom code
to run whenever a header is opened or closed
                  //do nothing
            }
      })
</script>

</head>

<body>
      <div class="glossymenu">
            <a class="menuitem submenuheader">Menu</a>
            <div class="submenu">
                  <ul>
                        <li><a
href="${pageContext.request.contextPath}/iOPortOperations.htm">IOPort Operations</a></li>
                        <li><a
href="${pageContext.request.contextPath}/temperatureReader.htm">Temperature Reader</a></li>
                  </ul>
            </div>
      </div>
</body>
</html>
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">
```

```
	<bean id="iOPortService" class="jvs.spring.pic.serviceImpl.IOPortServiceImpl">
		<property name="iOPortDAO" ref="iOPortDAO" />
	</bean>

	<bean id="iOPortDAO" class="jvs.spring.pic.daoImpl.IOPortDAOImpl">
		<property name="serialComm" ref="serialComm" />
	</bean>

	<bean id="serialComm" class="jvs.spring.pic.util.SerialComm" />


	<!-- TempMonitorServlet implements HttpRequestHandler so that Spring Beans can be
injected, since it is a Servlet -->
	<bean id="TempMonitorServlet" class="jvs.spring.pic.controller.TempMonitorServlet">
		<property name="iOPortService" ref="iOPortService" />
	</bean>
</beans>
```

springapp-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">


	<!-- the application context definition for the springapp DispatcherServlet -->
	<bean name="/iOPortOperations.htm"
class="jvs.spring.pic.controller.IOPortController">
		<property name="iOPortService" ref="iOPortService" />

		<property name="commandName" value="picPorts" />
		<property name="commandClass" value="jvs.spring.pic.form.PicPorts" />
		<property name="sessionForm" value="true" />

		<property name="formView" value="tileIOPortOperations" />
		<property name="successView" value="tileIOPortOperations" />
	</bean>

	<bean name="/temperatureReader.htm"
class="jvs.spring.pic.controller.TempMonitorController" />


	<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
		<property name="basename" value="messages" />
	</bean>

	<bean id="tilesConfigurer"
class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
		<property name="definitions">
			<list>
				<value>/WEB-INF/tiles-defs.xml</value>
			</list>
		</property>
	</bean>

	<!-- Declare the Resolver -->
```

```
	<bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
		<property name="viewClass"
value="org.springframework.web.servlet.view.tiles2.TilesView" />
	</bean>

</beans>
```

tiles-defs.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
"http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
	<definition name="tileBaseLayout" template="/tiles/baseLayout.jsp">
		<put-attribute name="title" value="Template" />
		<put-attribute name="header" value="/tiles/header.jsp" />
		<put-attribute name="menu" value="/tiles/menu.jsp" />
		<put-attribute name="body" value="/tiles/body.jsp" />
		<put-attribute name="footer" value="/tiles/footer.jsp" />
	</definition>

	<definition name="tileIOPortOperations" extends="tileBaseLayout">
		<put-attribute name="title" value="IOPort Operations" />
		<put-attribute name="body" value="/jsp/iOPortOperations.jsp" />
	</definition>

	<definition name="tileTemperatureReader" extends="tileBaseLayout">
		<put-attribute name="title" value="Temperature Reader" />
		<put-attribute name="body" value="/jsp/temperatureReader.jsp" />
	</definition>
</tiles-definitions>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.5">
  <display-name>Archetype Created Web Application</display-name>

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>

  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
```

```
  </servlet-mapping>

  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <description></description>
    <display-name>TempMonitorServlet</display-name>
    <servlet-name>TempMonitorServlet</servlet-name>
    <servlet-
class>org.springframework.web.context.support.HttpRequestHandlerServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>TempMonitorServlet</servlet-name>
    <url-pattern>/TempMonitorServlet</url-pattern>
  </servlet-mapping>

</web-app>
```

index.jsp

```
<%@ include file="jsp/include.jsp" %>

<%-- Redirected because we can't set the welcome page to a virtual URL. --%>
<c:redirect url="/iOPortOperations.htm"/>
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

      <modelVersion>4.0.0</modelVersion>
      <groupId>jvs.spring.pic</groupId>
      <artifactId>PicControlWebApp</artifactId>
      <packaging>war</packaging>
      <version>0.0.1-SNAPSHOT</version>
      <name>PicControlWebApp Maven Webapp</name>
      <url>http://maven.apache.org</url>

      <properties>
            <spring.version>2.5.6.SEC03</spring.version>
            <java.version>1.6</java.version>
            <servlet.version>2.5</servlet.version>
            <jstl.version>1.1.2</jstl.version>
            <jUnit.version>3.8.1</jUnit.version>
            <tiles.version>2.0.7</tiles.version>
            <RXTXcomm.version>2.1.7</RXTXcomm.version>
```

```
</properties>

<dependencies>

	<!-- Spring -->
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-core</artifactId>
		<version>${spring.version}</version>
	</dependency>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-web</artifactId>
		<version>${spring.version}</version>
	</dependency>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-webmvc</artifactId>
		<version>${spring.version}</version>
	</dependency>

	<!-- Servlet -->
	<dependency>
		<groupId>javax.servlet</groupId>
		<artifactId>servlet-api</artifactId>
		<version>${servlet.version}</version>
		<scope>provided</scope>
	</dependency>

	<!-- JSTL -->
	<dependency>
		<groupId>javax.servlet</groupId>
		<artifactId>jstl</artifactId>
		<version>${jstl.version}</version>
	</dependency>
	<dependency>
		<groupId>taglibs</groupId>
		<artifactId>standard</artifactId>
		<version>${jstl.version}</version>
	</dependency>

	<!-- Tiles -->
	<dependency>
		<groupId>org.apache.tiles</groupId>
		<artifactId>tiles-jsp</artifactId>
		<version>${tiles.version}</version>
	</dependency>

	<!-- JUnit -->
	<dependency>
		<groupId>junit</groupId>
		<artifactId>junit</artifactId>
		<version>${jUnit.version}</version>
		<scope>test</scope>
	</dependency>

	<!-- RXTXcomm -->
	<dependency>
		<groupId>org.rxtx</groupId>
```

```
                <artifactId>rxtx</artifactId>
                <version>${RXTXcomm.version}</version>
            </dependency>

        </dependencies>

        <build>
            <finalName>PicControlWebApp</finalName>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
                        <source>${java.version}</source>
                        <target>${java.version}</target>
                    </configuration>
                </plugin>
            </plugins>
        </build>

</project>
```

Glosario

CAN	Controller Area Network. Protocolo de comunicaciones basado en una topología bus para la transmisión de mensajes en entornos distribuidos.
CMOS	Complementary Metal Oxide Semiconductor. Es una de las familias lógicas empleadas en la fabricación de circuitos integrados
CPU	Central Processing Unit. Es el componente principal de una computadora y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.
DIP	Dual In-Line Package. Es una forma de encapsulamiento común en la construcción de circuitos integrados. La forma consiste en un bloque con dos hileras paralelas de pines, la cantidad de éstos depende de cada circuito.
DSC	Digital Signal Controller. Es un híbrido entre un microcontrolador y un DSP (Digital Signal Processor) que incluye características para el procesamiento de señales.
dsPIC	Digital Signal Peripheral Interface Controller. Familia de microcontroladores híbrido para el especializados en el procesamiento de señales, fabricados por Microchip Technology Inc.
EEPROM	Electrically Erasable Programmable Read-Only Memory. Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente.
Firmware	El firmware es un bloque de instrucciones máquina para propósitos específicos, grabado en una memoria, normalmente de lectura/escritura (ROM, EEPROM, flash, etc.), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.
HTML	Hyper Text Markup Language. Lenguaje estándar de marcado para la elaboración de páginas Web.

HTTP	Hyper Text transfer Protocol. Es un protocolo de comunicación sin estado, es decir, no guarda ninguna información sobre conexiones anteriores, y es el utilizado en cada transacción de la World Wide Web o Internet.
I2C	Inter Integrated Circuit. Es un bus de comunicaciones en serie muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados.
ICSP	In Circuit Serial Programming. Es un método para la programación directa de microcontroladores.
IDE	Integrated Development Environment. Es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.
JEE	Java Enterprise Edition. Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.
JDK	Java Development Kit. Es un software que provee herramientas de desarrollo para la creación de programas en Java.
LAN	Local Area Network. Una red de área local es la interconexión de uno o varios dispositivos.
LCD	Liquid Crystal Display. Es una pantalla de cristal líquido delgada y plana, formada por un número de pixeles en color o monocromos colocados delante de una fuente de luz o reflectora.
MCU	Microcontroller Unit. Es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.
MVC	Model View Controller. Es un patrón de arquitectura de software que separa

	los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.
PIC	Peripheral Interface Controller. Familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc.
POT	Abreviación de potenciómetro, que es un resistor cuyo valor de resistencia es variable.
PWM	Pulse Wide Modulation. La modulación por ancho de pulsos de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.
RAM	Random Access Memory. La memoria de acceso aleatorio se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es en donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo.
ROM	Read Only Memory. La memoria de solo lectura, es un medio de almacenamiento utilizado en computadoras y dispositivos electrónicos, que permite sólo la lectura de la información y no su escritura, independientemente de la presencia o no de una fuente de energía.
SCRUM	Scrum es un marco de trabajo para la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software.
SPI	Serial Peripheral Interface. El Bus SPI es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
TOKEN	Cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.

USART	Universal Synchronous Asynchronous Receiver Transmitter.
USB	Universal Serial Bus. Es un estándar industrial desarrollado a mediados de los años 1990 que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras y periféricos o dispositivos electrónicos.
WAN	World Area Network. Es una red de computadoras que abarca varias ubicaciones físicas, proveyendo servicio a una zona, un país, incluso varios continentes.