ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Control de aforo, peso y altura máxima/mínima en un recinto cerrado.

AUTOR: José Antonio Salmerón Marín

DIRECTOR(ES): José Vera Saura

Septiembre / 2008

Autor	José Antonio Salmerón Marín
E-mail del Autor	jasalmeronmarion@hotmail.com
Director(es)	José Vera Saura
E-mail del Director	jose.veras@upct.es
Codirector(es)	
Título del PFC	Control de aforo, peso y altura máxima/mínima en un recinto cerrado
Descriptores	

Resumen

Sistema que gestione el aforo de un local, con limitación en peso máximo total, altura mínima y/o máxima de individuo:

A. No se permite la entrada a partir de:
    a. Una suma de pesos mayor de un valor prefijado.
    b. Un número máximo prefijado de individuos.
    c. Una talla mínima de individuo.
    d. Una talla máxima de individuo.
B. En caso de cumplirse alguna de las condiciones anteriores, mantener en espera al individuo en la entrada del local.
C. Debe existir un control de Entrada y otro de Salida, con los sensores específicos para cada parámetro a controlar.
D. Los valores de peso máximo y aforo máximo soportados por el local, altura máxima y mínima del individuo y altura de sensor. podrán ser variados por una interfaz gráfica que también muestra los valores actuales de peso y aforo del local y la altura y peso del último individuo que entró.

Titulación	Ingeniero Técnico de Telecomunicación Especialidad Telemática
Departamento	DETCP
Fecha de Presentación	Septiembre-2008

INDICE

TABLAS

ILUSTRACIONES

1. Introducción

1.1. Planteamiento inicial del proyecto

A partir de los conocimientos adquiridos en microcontroladores en la asignatura de Sistemas Electrónicos Digitales, se plantea implementar un sistema de control para gestionar el aforo de personas a un local, teniendo en cuenta parámetros de medida del individuo: peso y altura, así como el nº de individuos en el local.

Los objetivos del proyecto se resumen a continuación:

A. No se permite la entrada a partir de:
    a. Una suma de pesos mayor de un valor prefijado
    b. Un número máximo prefijado de individuos.
    c. Una talla mínima de individuo.
    d. Una talla máxima de individuo.

B. En caso de cumplirse alguna de las condiciones anteriores, mantener en espera al individuo en la entrada del local.

C. Debe existir un control de Entrada y otro de Salida.

D. Los valores prefijados los escribe el usuario a través de una interfaz gráfica y los puede cambiar siempre que desee.

1.2. Resumen de la memoria

La memoria del proyecto se divide en los siguientes capítulos:

1. **Introducción.** Se explica resumidamente el planteamiento inicial del proyecto y los objetivos que se pretenden alcanzar.

2. **Memoria descriptiva.** Se describen los distintos dispositivos que forman el PFC además de explicar su funcionamiento y sus principales características.

3. **Interfaz.** Este es uno de los capítulos más extensos dedicados a explicar cómo se crea una interfaz gráfica, así como la descripción de los componentes utilizados. También se habla de la librería usada para la comunicación del PIC y el software necesario para que el usuario pueda usarla.

4. **Diseño e implementación del software.** En esta parte del proyecto se muestra el esquema eléctrico del sistema y se explica el uso que se da a otros recursos que se utilizan (conversor A/D, uso de los puertos...). Se explica también el intercambio de señales entre la interfaz y el PIC.

5. **Anexos.** Se muestra el código que el PIC tiene grabado y el de la interfaz gráfica.

6. **Bibliografía.** Resumen de la bibliografía usada en la realización del PFC.

2. Memoria descriptiva

2.1. El sensor de peso

2.1.1. Introducción

Los sensores de peso son conocidos en el mundo de la electrónica como celdas de cargas (load cell), cuyo principio básico está basado en el funcionamiento cuatro galgas extensiométricas (strain gauege), dispuestas en una configuración especial.

Se debe aclarar que una vez visto el funcionamiento de una celda de carga, se optó por utilizar un potenciómetro, pues el comportamiento de ambos es parecido y a la hora del montaje es mucho más simple utilizar el potenciómetro.

2.1.2. La galga extensiométrica

La galga extensiométrica es básicamente una resistencia eléctrica. El parámetro variable y sujeto a medida es la resistencia de dicha galga. Esta variación de resistencia depende de la deformación que sufre la galga.

Se parte de la hipótesis inicial de que el sensor experimenta las mismas deformaciones que la superficie sobre la cual está pegada. El sensor está constituido básicamente por una base muy delgada no conductora, sobre la cual va adherido un hilo metálico muy fino, de forma que la mayor parte de su longitud está distribuida paralelamente a una dirección determinada, tal y como se muestra en la figura siguiente:

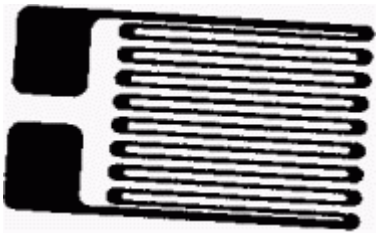


Ilustración 1. Galga extensiométrica

La resistencia eléctrica del hilo es directamente proporcional a su longitud, o lo que es lo mismo, su resistencia aumenta cuando éste se alarga. De este modo las deformaciones que se producen en el objeto, en el cual está adherida la galga, provocan una variación de la longitud y, por consiguiente, una variación de la resistencia.

Otro principio de funcionamiento de las galgas se basa en la deformación de elementos semiconductores. Esta deformación provoca una variación, tanto en la longitud como en la sección.

Este tipo de sensor semiconductor posee un factor de galga más elevado que el constituido por hilo metálico.

2.1.3. Tipos de galgas extensiométricas

Existen dos tipos de galgas:

- **De hilo conductor o lámina conductora:** El sensor está constituido básicamente por una base muy delgada no conductora y muy flexible, sobre la cual va adherido un hilo metálico muy fino. Las terminaciones del hilo acaban en dos terminales a los cuales se conecta el transductor.
- **Semiconductor:** Las galgas semiconductoras son similares a las anteriores. En este tipo de galgas se sustituye el hilo metálico por un material semiconductor. La principal diferencia constructiva de estas galgas respecto a las anteriores se encuentra en el tamaño; las galgas semiconductoras tienen un tamaño más reducido.

2.1.4. Características de las galgas extensiométricas

Las principales características de las galgas son las siguientes:

- **Anchura y Longitud**: Dichos parámetros proporcionan las características constructivas de la galga. Esto permite escoger el tamaño del sensor que más se adecue a nuestras necesidades.
- **Peso**: Esta característica define el peso de la galga. Este suele ser del orden de gramos. En aplicaciones de mucha precisión el peso puede influir en la medida de la deformación.
- **Tensión medible**: Es el rango de variación de longitud de la galga, cuando ésta se somete a una deformación. Este rango viene expresado en un tanto por cien respecto a la longitud de la galga.
- **Temperatura de funcionamiento**: Es aquella temperatura para la cual el funcionamiento de la galga se encuentra dentro de los parámetros proporcionados por el fabricante.
- **Resistencia de la galga**: Es la resistencia de la galga cuando ésta no está sometida a ninguna deformación. Es la resistencia de referencia y suele acompañarse de un porcentaje de variación.
- **Factor de galga**: Factor de galga o factor de sensibilidad de la galga es una constante K característica de cada galga. Determina la sensibilidad de ésta. Este factor es función de muchos parámetros, pero especialmente de la aleación empleada en la fabricación.
- **Coeficiente de temperatura del factor de galga**: La temperatura influye notablemente en las características. A su vez, cualquier variación en estas características influye en el factor de galga. Este coeficiente se mide en %/ºC, que es la variación porcentual del valor nominal del factor de galga respecto al incremento de temperatura.
- **Prueba de fatiga**: Esta característica indica el número de contracciones o deformaciones a una determinada tensión que puede soportar la galga sin romperse.
- **Material de la lámina**: Esta característica define el material del que está hecho el hilo conductor o el material semiconductor.
- **Material de la base**: Esta característica define el material del que está constituida la base no conductora de la galga.

- **Factor de expansión lineal**: Representa un error que se produce en la magnitud de salida en ausencia de señal de entrada, es decir, en ausencia de deformación. Este error depende de la temperatura ambiente a la que está sometida la galga.

2.1.5. Configuración física

El montaje más común utilizado para medir deformaciones mediante galgas es el puente de Wheatstone. Existen tres tipos de montajes básicos: con una, dos y cuatro galgas, pero puesto que se usa el de cuatro, sólo se explica éste.

La medida se suele realizar por deflexión, es decir midiendo la diferencia de tensión existente entre los terminales de salida del sensor. Las principales diferencias de estos montajes se encuentran en la sensibilidad y la capacidad de compensación del efecto de temperatura. Esta compensación consiste en suprimir los efectos de la temperatura en el valor de la resistencia de la galga; cuando en un puente de medida coinciden dos o cuatro galgas de iguales características, los efectos de la temperatura se anulan ya que ésta les afecta por igual.

La utilización de cuatro galgas cuadruplica la sensibilidad del puente respecto al puente de una sola galga. Esto permite que para una misma deformación se tenga una mayor señal de salida para una tensión de alimentación dada.

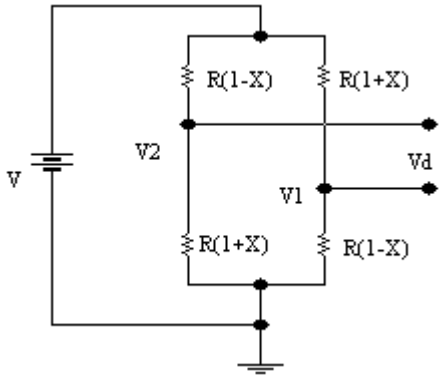


Ilustración 2. Puente de Wheatstone

2.2. El sensor de altura

2.2.1. Introducción

Se trata de un emisor/receptor de ultrasonidos cuyo modelo es el SRF05. Cuenta con un nuevo modo de trabajo que emplea un solo pin para controlar el sensor y hacer la lectura de la medida. Lo que se hace es mandar un impulso para iniciar la lectura y luego poner el pin en modo entrada. Después basta con leer la longitud del pulso devuelto por el sensor, que es proporcional a la distancia medida por el sensor.

El sensor SRF05 incluye un breve retardo después del pulso de eco para dar a los controladores más lentos como Basic Stamp y Picaxe el tiempo necesario para ejecutar sus pulsos en los comandos.

Ilustración 3. Sensor SRF05

2.2.2. Cálculo de la distancia

Se debe suministrar un breve pulso de al menos 10uS para disparar la entrada de comienzo del cálculo de distancia. El SRF05 transmite una ráfaga de 8 ciclos de ultrasonidos a 40khz elevando el nivel lógico de la señal del eco (o la línea de activación en el modo 2). Entonces el sensor "escucha" un eco, y en cuanto lo detecta, vuelve a bajar el nivel lógico de la línea de eco. La línea de eco es por lo tanto un pulso, cuyo ancho es proporcional a la distancia respecto al objeto. Registrando la duración del pulso es posible calcular la distancia en pulgadas/centímetros o en cualquier otra unidad de medida. Si no se detectase nada, entonces el SRF05 baja el nivel lógico de su línea de eco después de 30mS.

El SRF05 proporciona un pulso de eco proporcional a la distancia. Si el ancho del pulso se mide en µs, el resultado se debe dividir entre 58 para saber el equivalente en centímetros, y entre 148 para saber el equivalente en pulgadas. µs/58=cm o µs/148=pulgadas.

El SRF05 puede activarse cada 50mS, o 20 veces por segundo. Debería esperar 50ms antes de la siguiente activación, incluso si el SRF05 detecta un objeto cerca y el pulso del eco es más corto. De esta manera se asegura que el "bip" ultrasónico ha desaparecido completamente y no provoca un falso eco en la siguiente medición de distancia.

2.2.3. Modos de funcionamiento

El sensor SRF05 tiene dos modos de funcionamiento, según se realicen las conexiones:

- *Modo1: Señal de activación y eco independientes.*

Este modo utiliza pines independientes para la señal de inicio de la medición y para retorno del eco, siendo el modo más sencillo de utilizar. Para utilizar este modo, simplemente se debe dejar sin conectar el pin de modo. El SRF05 integra una resistencia pull-up en este pin.

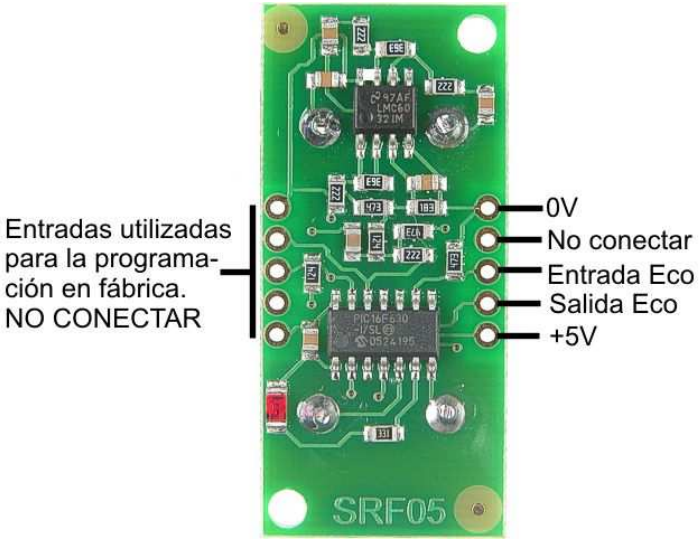


Ilustración 4. Patillaje modo 1

A continuación se muestra el diagrama de tiempos.

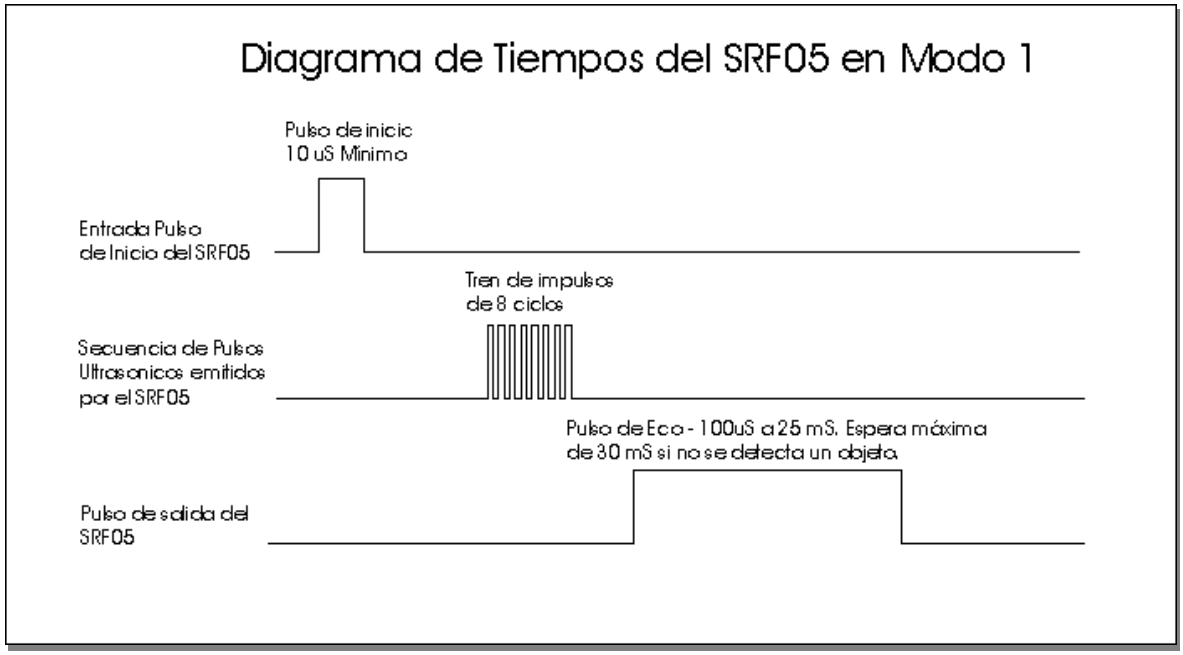


Ilustración 5. Diagrama de tiempos en modo 1

- *Modo2: Pin único para la señal de activación y eco*

Este modo utiliza un único pin para las señales de activación y eco, y está diseñado para reducir el número de pines en los microcontroladores. Para utilizar este modo, conecte el pin de modo al pin de tierra de 0v. La señal de eco aparece en el mismo pin que la señal de activación. El SRF05 no eleva el nivel lógico de la línea del eco hasta 700uS después del final de la señal de activación. Dispone de ese tiempo para cambiar el pin del disparador y convertirlo en una entrada para preparar el código de medición de pulsos. El comando PULSIN integrado en la mayor parte de los controladores del mercado lo hace automáticamente.

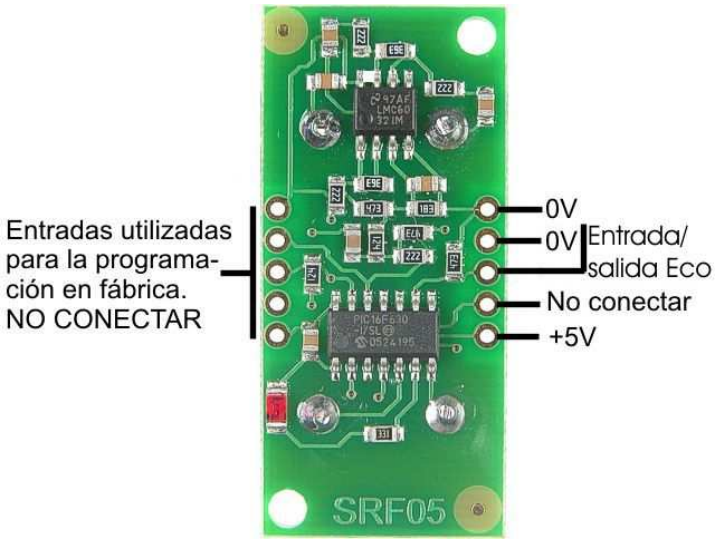


Ilustración 6. Patillaje modo 2

Seguidamente se ilustra el diagrama de tiempos para el modo2.

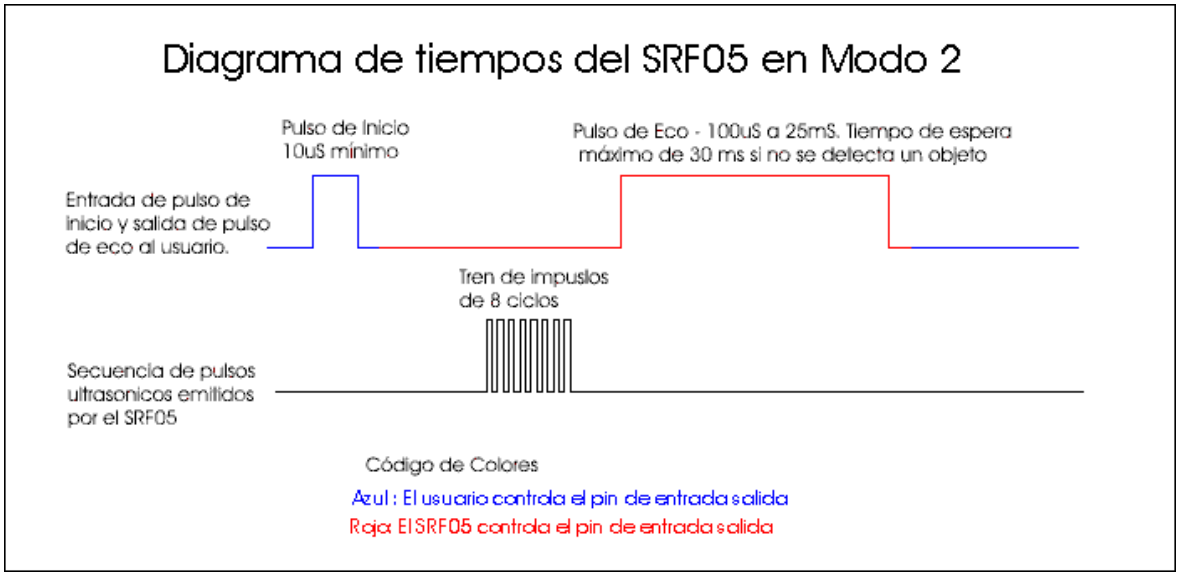


Ilustración 7. Diagrama de tiempos en modo 2

Finalmente se ha optado por la utilización del primer modo debido a su facilidad y a que no hay problemas de escasez de puertos.

2.3. El torno de control de acceso

2.3.1. Introducción

Los tornos electromecánicos fueron desarrollados con la intención de controlar la entrada y salida de personas a áreas de acceso controlado. Son la opción ideal para bancos, edificios de oficinas, complejos deportivos, salones de exposición e instalaciones industriales.

Con su diseño compacto y moderno, ellos pueden ser fácilmente integrados con barreras de seguridad modulares. Están formados por un pedestal o armazón que soporta tres brazos en forma de trípode. Estos brazos realizan el control de personal mediante la rotación de 120° cada vez que pasa un usuario, quedando siempre uno de los brazos en posición para impedir el paso.

Existen diversos modelos con funcionamiento idéntico al descrito o con doble acceso, variando apenas su aspecto exterior.

El control de acceso puede ser efectuado en uno o ambos sentidos.

También se debe aclarar que en el PFC se copia el comportamiento del torno y se simula mediante pulsadores.

Ilustración 8. Torno giratorio

2.3.2. Características

• Accionamiento de trabas a través de electroimanes y no de solenoides, permitiendo confiabilidad mayor.
• Señales de confirmación de giro generados por sensores ópticos de alto rendimiento, sin desgaste mecánico.
• Gabinete fabricado con acero carbono y pintura epoxi en polvo. Posee 3 brazos de acero inoxidable pulido.
• La tapa superior posee una combinación de chapa de acero inoxidable y terminación con piezas de plástico inyectado de alto impacto, proporcionando robustez y líneas redondeadas, además de un excelente diseño.
• No hay tornillos aparentes. Para acceder a la parte interna del torniquete es necesario usar una llave con secreto.

• Dos rodamientos automotores aumentan la vida útil del torniquete evitando mantenimientos desnecesarios.
• Sistema opcional con amortiguador hidráulico para control de retorno de los brazos. Desaceleración para evitar choques a los usuarios.
• Mecanismo de alto desempeño para funcionamiento en gran flujo de pasaje.
• Para caso de emergencia, si falta energía, el torniquete funciona libre para los dos sentidos.
• Puede fácilmente ser adaptado e integrado a cualquier terminal/tecnología.
• Placa controladora opcional (interfaz entre torniquete y el control del integrador) montada en proceso SMD posee sistema de protección de los electroimanes, configurable y ofrece 4 entradas y 1 salida. Permite configuración de las 4 opciones, liberada para los dos sentidos, bloquea los dos sentidos, libera un sentido y bloquea el otro sentido. Permite configurar tiempos para pasaje y señal disponible de retorno informando pasaje del usuario.
• Fuente opcional de alto rendimiento - entrada 90 a 250 VAC y salida 12 VDC/2A. Posee protección contra cortes eléctricos.
• Posee pictogramas opcionales de operación. Señalización visual a través de flecha de color verde indicando acceso permitido y “X” de color rojo indicando acceso negado.
• No-break opcional para funcionamiento del torniquete en el período de hasta 4 horas. El no-break posee una función de la cargar la batería y un dispositivo para apagarse en caso de emergencia.

2.3.3. Funcionamiento

El funcionamiento de este dispositivo es sencillo, pues sólo depende de una señal para ponerse en funcionamiento. Las otras 3 son para la configuración de los brazos.

En este caso se dispone de dos tornos, uno para la entrada y otro para la salida. Por este motivo se le especifica al fabricante que los tornos vengan configurados para girar en un solo sentido.

Por otro lado está la señal que activa el dispositivo para que rote. Una vez que la recibe el torno, a través del PIC que lleva insertado, inicia el proceso para que giren los brazos. Para ello está el sistema de potencia, que toma la información del microcontrolador y la convierte en señales de nivel de potencia requerida por los motores que realizan la rotación.

Una vez que se produce el giro, el torno envía una señal confirmando que ha pasado un usuario.

2.4. El Bus USB

2.4.1. Introducción

El **Universal Serial Bus** (bus universal en serie) es un puerto que sirve para conectar periféricos a una computadora. Fue creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.

El diseño del USB tenía en mente eliminar la necesidad de adquirir tarjetas separadas para poner en los puertos bus ISA o PCI, y mejorar las capacidades plug-and-play permitiendo a esos dispositivos ser conectados o desconectados al sistema sin necesidad de reiniciar. Cuando se conecta un nuevo dispositivo, el servidor lo enumera y agrega el software necesario para que pueda funcionar.

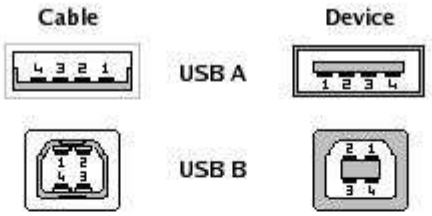
2.4.2. Características y transmisión

Los dispositivos USB se clasifican en cuatro tipos según su velocidad de transferencia de datos:

- Baja Velocidad (1.0): Bitrate de 1.5Mbit/s (192KB/s).
- Velocidad Completa (1.1): Bitrate de 12Mbit/s (1.5MB/s). Estos dispositivos, dividen el ancho de banda de la conexión USB entre ellos basados en un algoritmo FIFO.
- Alta Velocidad (2.0): Bitrate de 480Mbit/s (60MB/s).
- Súper Velocidad (3.0) Actualmente en fase experimental. Bitrate de 4.8Gbit/s (600MB/s).

Las señales del USB son transmitidas en un cable de data par trenzado con impedancia de 90Ω ±15% llamados D+ y D-. Éstos, colectivamente utilizan señalización diferencial en half dúplex para combatir los efectos del ruido electromagnético en enlaces largos. D+ y D- usualmente operan en conjunto y no son conexiones simples. Los niveles de transmisión de la señal varían de 0 a 0.3V para bajos (ceros) y de 2.8 a 3.6V para altos (unos) en las versiones 1.0 y 1.1, y en ±400mV en Alta Velocidad (2.0). El estándar USB, a diferencia de otros estándares también define tamaños para el área alrededor del conector de un dispositivo, para evitar el bloqueo de un puerto adyacente por el dispositivo en cuestión.

Las especificaciones USB 1.0, 1.1 y 2.0 definen dos tipos de conectores para conectar dispositivos al servidor: A y B



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

Ilustración 9. Tipos USB

2.4.3. Transferencia de datos

Todos los dispositivos USB están compuestos por una serie de endpoints y una dirección única asignada por el sistema. Un endpoint es un buffer que almacena datos dentro del dispositivo. Cada endpoint dispone de un identificador único que viene asignado de fábrica y una determinada orientación del flujo de datos.

Las asociaciones entre los distintos endpoints de un dispositivo y el host se llaman Pipes (tuberías). Las Pipes permiten mover datos entre el software del host y el endpoint del dispositivo. Cada Pipe está determinado por su tipo de servicio, número de endpoint, tamaño de paquetes, dirección, etc.

Existen 4 tipos de endpoints distintos (Control, Bulk, Isócronas e Interrupciones), cada uno se utiliza en un tipo de transferencia, y las asociaciones (Pipes) que se producen entre estos endpoints son:

- **Pipe de Control o Mensaje:** Es una vía de comunicación entre dos endpoints de control, uno de Entrada y otro de Salida, de manera que se puede establecer una comunicación bidireccional. Todos los dispositivos poseen dos endpoints de Control en la dirección 0, que se puede establecer antes de configurar el dispositivo. A través de este endpoint el host puede leer información sobre el dispositivo USB antes de iniciar la transferencia.

- **Pipe Stream**: Es una vía de comunicación unidireccional entre el sistema y los endpoints de tipo Bulk, Isócrono e Interrupciones.

Las transferencias pueden ser, según el tipo de endpoint, de tipo:

- **Transferencias de Control**: Proporcionan control de flujo y una entrega de datos garantizada y libre de errores. Todos los dispositivos full, high y lowspeed pueden

incorporar endpoints de Control, y por lo tanto pueden hacer uso de estas transferencias, y implementan al menos dos endpoints (entrada y salida) en la dirección 0 para poder establecer la Pipe de Control por defecto.

- **Transferencias Isócronas**: Se utiliza en transferencias de datos en tiempo real. Los datos se envían a velocidad constante aunque se produzcan errores. Solo los dispositivos high y full-speed pueden incorporar endpoints isócronos.

- **Transferencias de Interrupción**: Se utiliza en dispositivos que precisan el envío de datos de manera no frecuente, y dentro de un periodo máximo. Todos los dispositivos pueden incorporar este tipo de endpoint.

- **Transferencias Bulk**: Se utilizan en dispositivos que precisan enviar grandes cantidades de datos utilizando todo el ancho de banda posible. Solo los dispositivos high y full-speed pueden contener estos endpoints.

La comunicación entre el PIC y el USB se puede separar en tres fases:

a) USB Bus Reset: cuando el dispositivo es conectado, el host lo detecta y provoca una interrupción de reset para que el dispositivo configure los registros y punteros necesarios y para que se pueda proceder a la enumeración. En esta fase el programa debe habilitar y configurar el endpoint 0 para recibir y contestar a transacciones de tipo Setup e inicializar todas las variables que posteriormente se utilizan. Esto se realiza dentro de la rutina de servicio de la interrupción BusReset.

b) Proceso de Enumeración: esta fase se produce cuando el dispositivo es conectado al Bus y después de la fase de Bus Reset. El host debe reunir la información necesaria para que el sistema identifique al dispositivo y configure el tipo de comunicación que se produce entre ambos y encuentre el driver que tiene que utilizar para establecer la comunicación. El proceso consiste primero, en asignar una dirección al device y segundo, el host envía una serie de peticiones para que el dispositivo mande información con el fin de establecer la comunicación

La información que debe mandar el dispositivo se estructura en registros o descriptores que configuran el dispositivo y son transmitidos mediante transferencia de control y siempre por el endpoint 0. Estos descriptores son los siguientes:

Device descriptor: contiene información básica del dispositivo como puede ser número de serie, clase de dispositivo, etc.

Configuración descriptor: contiene información sobre las capacidades y funciones del dispositivo, tipo de alimentación de energía que soporta.

Interface descriptor: contiene información sobre el número de endpoints que soporta y el protocolo utilizado.

Class descriptor: determina la clase del dispositivo.

Endpoint descriptor: excepto para el endpoint 0, cada endpoint debe ser configurado. La configuración de cada endpoint consiste en el número de endpoint, dirección de sus comunicaciones (IN, OUT) y numero de bytes que transmite.
Con estos descriptores y otros del sistema operativo, debe encontrar el driver que necesita para comunicarse con el dispositivo. Para realizar este proceso de enumeración se debe cargar en el programa el código de los descriptores, habilitar y configurar la interrupción del Endpoint 0 para aceptar transacciones de tipo Setup, saber qué tipo de report ha solicitado en cada caso el host y mandarlo por el Endpoint 0.
Cuando se termine esta fase de enumeración se pasa a la fase de gestión del dispositivo y de las comunicaciones.

c) Gestión del dispositivo y de las comunicaciones: después de las dos fases anteriores en las que se configura y se establece la comunicación con el host se pasa a la tercera fase que es la que realmente se centra en gestionar la funcionalidad para la que se ha diseñado el dispositivo.

2.4.4. Driver

Un controlador de dispositivo (llamado normalmente controlador, o, en inglés, *driver*) es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz para usarlo. Se puede esquematizar como un manual de instrucciones que le indica cómo debe controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el hardware.

Normalmente son los fabricantes del hardware quienes escriben sus controladores, ya que conocen mejor el funcionamiento interno de cada aparato, pero también se encuentran controladores libres. En este caso, los creadores no son de la empresa fabricante, aunque a veces hay una cooperación con ellos, cosa que facilita el desarrollo. Si no la hay, el procedimiento necesita de ingeniería inversa y otros métodos difíciles.

2.5. Microcontroladores PIC

2.5.1. Introducción

Un microcontrolador es un circuito integrado o chip que incluye en su interior las tres unidades funcionales de una computadora: CPU, Memoria y Unidades de E/S, es decir, se trata de un computador completo en un solo circuito integrado.

La limitación en la aplicación de los microcontroladores a un desarrollo de ingeniería tiene su límite en la imaginación del desarrollador. Con los diversos modelos disponibles se puede afrontar multitud de diseños distintos desde los más simples hasta los más complejos.

2.5.2. Arquitectura interna

Las partes que constituyen la arquitectura interna de un microcontrolador son:

- **Procesador:** Es la parte encargada del procesamiento de las instrucciones. Debido a la necesidad de conseguir elevados rendimientos en este proceso, se ha desembocado en el empleo generalizado de procesadores de arquitectura Harvard frente a los tradicionales que seguían la *arquitectura de Von Neumann*. Ésta última se caracterizaba porque la CPU se conectaba con una memoria única, donde coexistían datos e instrucciones, a través de un sistema de buses.

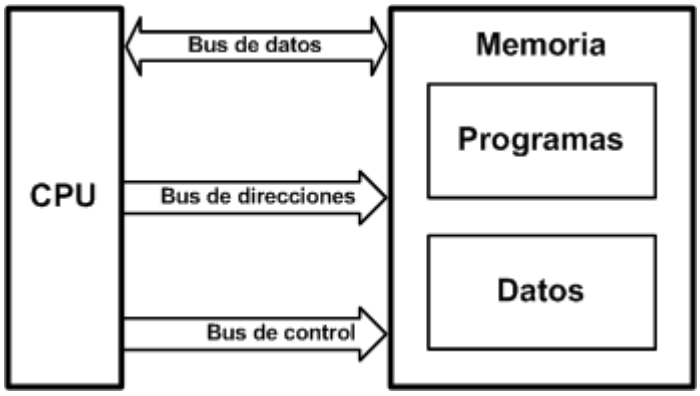


Ilustración 10. Von Neuman

En la *arquitectura Harvard* son independientes la memoria de instrucciones y la memoria de datos y cada una dispone de su propio sistema de buses para el acceso. Esta dualidad, además de propiciar el paralelismo, permite la adecuación del tamaño de las palabras y los buses a los requerimientos específicos de las instrucciones y de los datos.

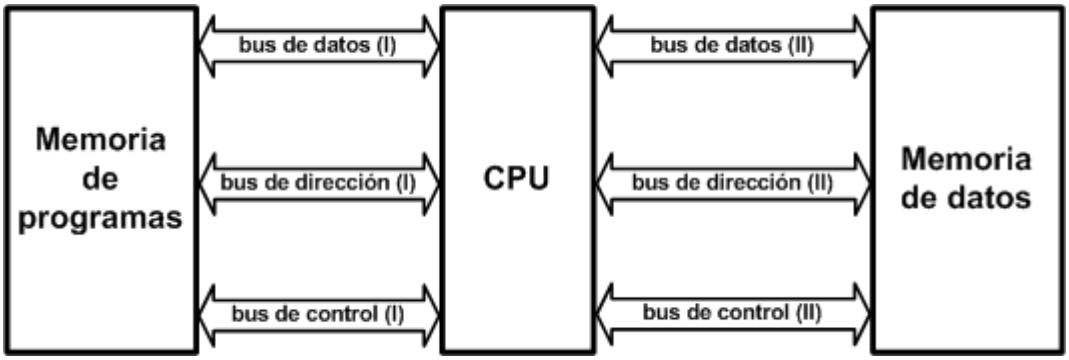


Ilustración 11. Harvard

El procesador de los modernos microcontroladores responde a la arquitectura RISC (Computadores de Juego de Instrucciones Reducido), que se identifica por poseer un repertorio de instrucciones máquina pequeño y simple, de forma que la mayor parte de las instrucciones se ejecutan en un ciclo de instrucción.

Otra aportación frecuente que aumenta el rendimiento del computador es el fomento del paralelismo implícito, que consiste en la segmentación del procesador (pipe-line), descomponiéndolo en etapas para poder procesar una instrucción diferente en cada una de ellas y trabajar con varias a la vez.

- **Memoria:** En este punto hay que distinguir entre:

--*Memoria de programa:* El microcontrolador está diseñado para que en su memoria de programa se almacenen todas las instrucciones del programa de control. Como éste siempre es el mismo, debe estar grabado de forma permanente.
Existen algunos tipos de memoria adecuados para soportar estas funciones, de las cuales se citan las siguientes:

- **ROM** con máscara: se graba mediante el uso de máscaras. Sólo es recomendable para series muy grandes debido a su elevado coste.
- **EPROM**: se graba eléctricamente con un programador controlador por un PC. Disponen de una ventana en la parte superior para someterla a luz ultravioleta, lo que permite su borrado. Puede usarse en fase de diseño, aunque su coste unitario es elevado.
- **OTP:** su proceso de grabación es similar al anterior, pero éstas no pueden borrarse. Su bajo coste las hace idóneas para productos finales.
- **EEPROM:** también se graba eléctricamente, pero su borrado es mucho más sencillo, ya que también es eléctrico. No se pueden conseguir grandes capacidades y su tiempo de de escritura y su consumo es elevado.
- **FLASH:** se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar en circuito al igual que las EEPROM, pero que suelen disponer de mayor capacidad que estas últimas. Son recomendables aplicaciones en las que es necesario modificar el programa a lo largo de la vida del producto. Por sus mejores prestaciones, está sustituyendo a la memoria EEPROM para contener instrucciones.

--*Memoria de datos:* Los datos que manejas los programas varían continuamente, y esto exige que la memoria que los contiene debe ser de lectura y escritura, por lo que la memoria RAM estática (SRAM) es la más adecuada, aunque sea volátil.

Hay microcontroladores que disponen como memoria de datos una de lectura y escritura no volátil, del tipo EEPROM. De esta forma, un corte en el suministro de la alimentación no ocasiona la pérdida de la información, que está disponible al reiniciarse el programa.

- **Líneas de Entrada/Salida:** A excepción de dos patitas destinadas a recibir la alimentación, otras dos para el cristal de cuarzo, que regula la frecuencia de trabajo, y una más para provocar el Reset, las restantes patitas de un microcontrolador sirven para soportar su comunicación con los periféricos externos que controla.

  Las líneas de E/S que se adaptan con los periféricos manejan información en paralelo y se agrupan en conjuntos de ocho, que reciben el nombre de Puertas. Hay modelos con líneas que soportan la comunicación en serie; otros disponen de conjuntos de líneas que implementan puertas de comunicación para diversos protocolos, como el I2C, el USB, etc.

- **Recursos auxiliares:** Según las aplicaciones a las que orienta el fabricante cada modelo de microcontrolador, incorpora una diversidad de complementos que refuerzan la potencia y la flexibilidad del dispositivo. Entre los recursos más comunes se citan los siguientes:

  - Circuito de reloj: se encarga de generar los impulsos que sincronizan el funcionamiento de todo el sistema.

  - Temporizadores: orientados a controlar tiempos.

  - Perro Guardián o WatchDog: se emplea para provocar un reset cuando el programa queda bloqueado.

  - Conversores AD y DA: para poder recibir y enviar señales analógicas.

  - Sistema de protección ante fallos de alimentación

  - Estados de reposos: gracias a los cuales el sistema queda congelado y el consumo de energía se reduce al mínimo.

2.5.3. Familia de microcontroladores

A continuación se muestra una relación de algunos fabricantes y modelos de microcontroladores incluyendo su dirección en Internet, si es CISC o RISC, el número de bits del bus de datos y el núcleo del que deriva (8051, ARM, etc.) así como si está disponible un IDE gratuito:

FABRICANTE	FAMILIA	ARQUITECTURA	IDE
Analog Device	ADUC8xx	CISC 8 bits 8051	-
	ADUC7xx	RISC 32 bits ARM7	-
Atmel	AT89xxx	CISC 8 bits 8051	prog. independientes
	TS87xxx	CISC 8 bits	prog. independientes
	AVR	RISC 8 bits	AVR studio
	AT91xxx	RISC 16 bits ARM7/9	-
Cirrus Logic	EP73xxx	RISC 32 bits ARM7	-
	EP93xxx	RISC 32 bits ARM9	-
Cygnal	C8051F	CISC 8 bits 8051	-
Freescale (Motorola)	HC05	CISC 8 bits 6800	-
	HC08	CISC 8 bits 6809	Code Warrior
	HC11	CISC 8 bits 6809	-
	HC12	CISC 16 bits	-
	HCS12	CISC 16 bits	Code Warrior
	HC16	CISC 16 bits	-
	56800	CISC 16 bits	-
	68K	CISC 32 bits 68000	-
	ColdFire	CISC 32 bits	-
	MAC7100	RISC 32 bits ARM7	-
Fujitsu	F2MC-8	CISC 8 bits	-
	F2MC-16	CISC 16 bits	-
	FR	RISC 32 bits	-
Infineon	C5xxx	CISC 8 bits 8051	-
	C8xxx	CISC 8 bits 8051	-
	C16xxx	CISC 16 bits	-

	XC16xxx	CISC 16 bits	-
	TCxxx	CISC 32 bits	-
Intel	MCS251	CISC 8 bits 8051	-
	MCS96/296	CISC 16 bits	-
Maxim (Dallas)	DS80Cxxx	CISC 8 bits 8051	-
	DS83Cxxx	CISC 8 bits 8051	-
	DS89Cxxx	CISC 8 bits 8051	-
	MAXQ	RISC 16 bits	-
Microchip	**PIC** 10,12,14,16,17,**18**	RISC 8 bits	MPLAB
	dsPIC	RISC 16 bits	MPLAB
NS (NATIONAL SEMICONDUCTOR) (VER ESTO)	COP8xxx	CISC 8 bits	Webench
	CR16Cxxx	CISC 16 bits	-
	CP3000	RISC 16 bits	-
Philips	P8xxx	CISC 8 bits 8051	-
	Xaxxx	CISC 16 bits	-
	LPC2xxx	RISC 32 bits ARM7	-
Rabbit Semiconductor	Rabbit2000	CISC 8 bits	-
	Rabbit3000	CISC 8 bits	-
Renesas	740	CISC 8 bits	-
	H8	CISC 16 bits	HEW
	H8S	CISC 16 bits	HEW
	M16C	CISC 16 bits	-
	7700	CISC 16 bits	-
	H8SX	CISC 32 bits	-
	Super H	CISC 32 bits	HEW
ST (SGS-THOMSON)	ST5	CISC 8 bits	Visual FIVE
	ST6	CISC 8 bits	-
	ST7	CISC 8 bits	STVD 7
	ST9	CISC 8 bits	STVD 9
	ST9	CISC 16 bits	STVD 9 (VER ESTO)

	ST10	CISC 16 bits	-
	ARM7	RISC 32 bits ARM7	-
Texas Instruments	MSC12xxx	CISC 8 bits 8051	-
	MSP430	CISC 16 bits	Eclipse
	TMS470	RISC 32 bits ARM7	-
Toshiba	870	CISC 8 bits	-
	900/900H	CISC 16 bits	-
	900/900H (VER ESTO)	CISC 32 bits	-
Ubicom (Scenix)	SXxx	RISC 8 bits	-
Zilog	Z8xxx	CISC 8 bits Z80	-
	Z8Encore!	CISC 8 bits Z80	-
	eZ80Aclaim	CISC 8 bits Z80	-

Tabla 1. Familia 18

2.5.4. Familia 18

Aunque en principio se empezó a desarrollar el software con el PIC 16F84A, pronto se desechó esa idea debido a que dicho PIC no tenía los recursos necesarios para afrontar periféricos como el sensor de peso o la comunicación USB con la interfaz. Finalmente se optó por la utilización del PIC18F4550 que también procede de la familia Microchip.

Las principales características de esta familia son:

- Existen unos 133 modelos.
- El número de pines respecto al modelo varía desde los 18 a los 100 pines.
- La máxima frecuencia a la que puede trabajar es 64 MHz.
- Dispone hasta 8Kpalabras de memoria de programa.
- Hasta 3968 bytes de memoria de datos de propósito general (RAM).
- Entradas y salidas digitales, temporizadores, conversores analógicos/digitales de hasta 16 bits, PWM (modulación de ancho de pulso), comparadores analógicos, módulos de comunicación serie, memoria de datos EEPROM, USB…
- Hasta 10 vectores de interrupción.
- La mayoría de modelos poseen 75 instrucciones, aunque existe algunos que tienen 83.
- Pila de 32 niveles

Finalmente se ha optado por la elección del PIC 18F4550, debido a su fácil manejo y a que posee soporte para USB, amplia memoria EEPROM y hasta 35 líneas de E/S.

2.5.5. El PIC 18F4550

En esta sección se comenta por encima las principales características del PIC 18F4550 de Microchip ya que al tratarse de la gama alta tiene un datasheet extenso y muchos periféricos no se utilizan en este PFC.

Se empieza mostrando una figura del PIC donde se ve que tiene 40 pines y al lado del número aparece cual puede ser su función:

Ilustración 12. PIC 18F4550

El PIC 18F4550 dispone de 5 memorias:

1) **Memoria de programa**: memoria flash interna de 32.768 bytes
    a. Almacena instrucciones y constantes/datos
    b. Puede ser escrita/leída mediante un programador externo o durante la ejecución programa mediante unos punteros.
2) **Memoria RAM de datos**: memoria SRAM interna de 2048 bytes en la que están incluidos los registros de función especial.
    a. Almacena datos de forma temporal durante la ejecución del programa
    b. Puede ser escrita/leída en tiempo de ejecución mediante diversas instrucciones
3) **Memoria EEPROM de datos:** memoria no volátil de 256 bytes.

   a. Almacena datos que se deben conservar aun en ausencia de tensión de alimentación
   b. Puede ser escrita/leída en tiempo de ejecución a través de registros
4) **Pila**: bloque de 31 palabras de 21 bits
   a. Almacena la dirección de la instrucción que debe ser ejecutada después de una interrupción o subrutina.
5) **Memoria de configuración**: memoria en la que se incluyen los bits de configuración (12 bytes de memoria flash) y los registros de identificación (2 bytes de memoria de sólo lectura).

La arquitectura del microcontrolador es Harvard y por ello posee dos buses diferentes:

- **Bus de la memoria de programa:**
  - 21 líneas de dirección
  - 16/8 líneas de datos (16 líneas para instrucciones/8 líneas para datos)
- **Bus de la memoria de datos:**
  - 12 líneas de dirección
  - 8 líneas de datos

Esto permite ejecutar una instrucción mientras se lee de la memoria de programa la siguiente instrucción (lo que se conoce como proceso pipeline).

Posee dos vectores de interrupción:

a) **Vectorización de las interrupciones de alta prioridad,** cuya dirección en la memoria de programa es la 0008H.
b) **Vectorización de las interrupciones de baja prioridad**, que se encuentra en la 0018H de la memoria de programa.

Otros datos destacables del PIC son:

CARACTERÍSTICAS	**PIC18F4550**
FRECUENCIA DE OPERACIÓN	HASTA 48 MHZ
NÚMERO DE INTERRUPCIONES	20
LÍNEAS DE E/S	35
TEMPORIZADORES	4
MÓDULOS DE COMPARACIÓN/CAPTURA/PWM (CCP)	1
CCP MEJORADO (ECCP)	1
CANALES DE COMUNICACIÓN SERIE	MSSP, EUSART
CANAL USB	1
PUERTO PARALELO DE TRANSMISIÓN DE DATOS	1

CANALES DE CONVERSIÓN A/D DE 10 BITS	13
COMPARADORES ANALÓGICOS	2
JUEGO DE INSTRUCCIONES	75 (83 ext.)

Tabla 2. Otras características PIC 18F4550

Para acabar esta sección se muestra una foto con el diagrama de bloques, donde se puede observar lo anteriormente descrito de forma más detallada:

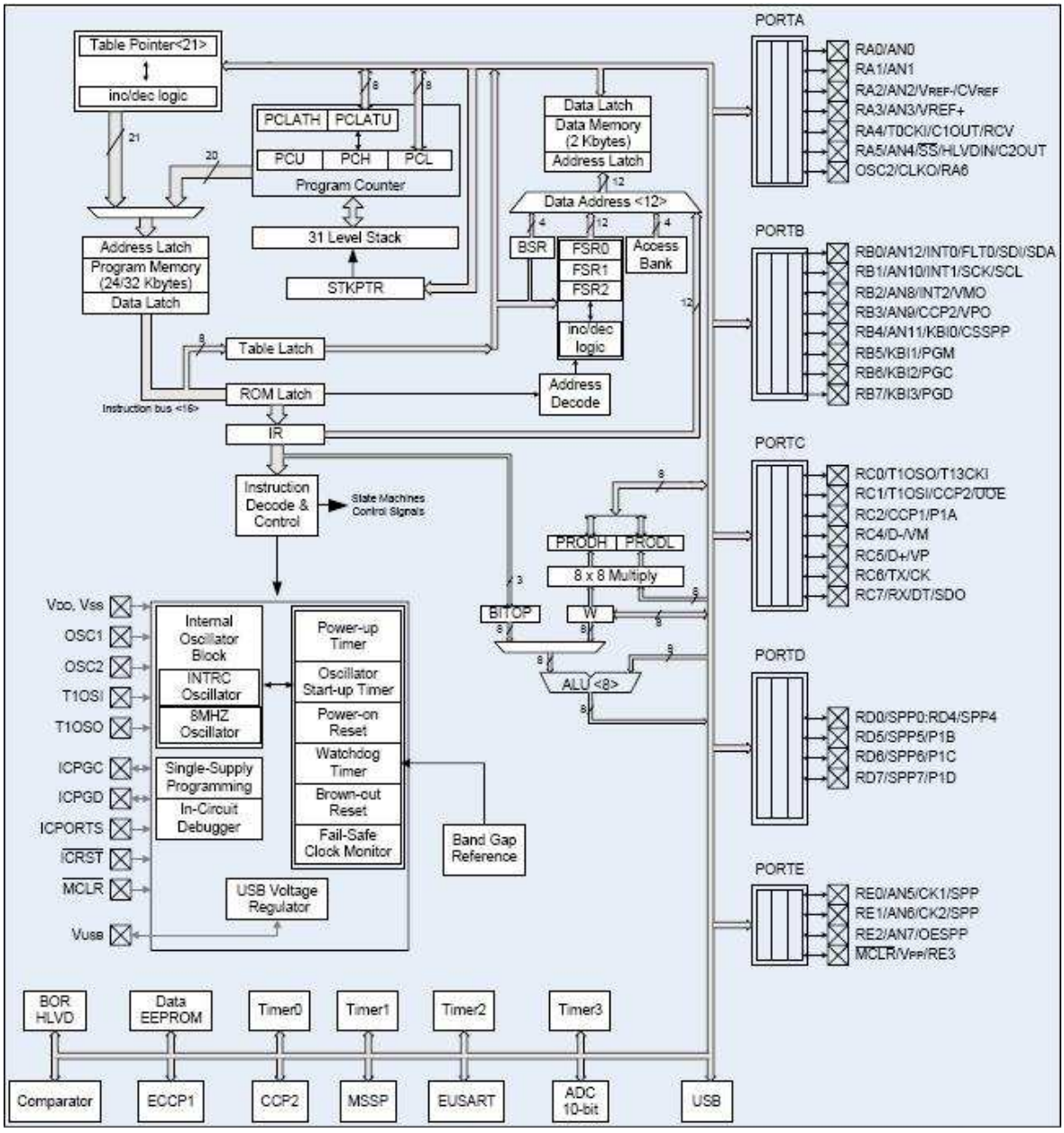


Ilustración 13. Diagrama de bloques PIC 18F4550

2.5.6. Lenguajes de programación del PIC

Para escribir programas para PIC se puede escoger entre varios lenguajes de programación: Ensamblador, Basic, C.

El lenguaje ensamblador es un tipo de lenguaje de bajo nivel utilizado para escribir programas informáticos, y constituye la representación más directa del código máquina específico para cada arquitectura de computadoras legible por un programador. Es un lenguaje difícil de depurar, aunque es generalmente más rápido que los de alto nivel y ocupan menos espacio

Por otro lado está, Basic y C que son de alto nivel. El lenguaje BASIC se caracteriza por su sencillez, siendo un lenguaje apropiado para iniciarse en la programación a alto nivel. Su mayor defecto es que es un lenguaje "no estructurado", lo que significa que se puede saltar a cualquier parte del programa usando la instrucción GOTO, lo que impide la realización de código estructurado. Por otro lado está C que es el lenguaje más popular para la programación de alto nivel en microcontroladores, y uno de los más potentes. Con respecto a BASIC, posee la ventaja de ser estructurado y además tiene gran cantidad de librerías. También permite introducir código en ensamblador dentro del programa. Además existen varios compiladores que poseen abundantes librerías para todo tipo de dispositivos. Su principal desventaja es la implícita a cualquier lenguaje de alto nivel, poco control sobre el código generado lo que no permite código optimizado y además lo hace más difícil de depurar.

Como conclusión, el ensamblador es el lenguaje óptimo para programas de poco alcance, con PICs pequeños, donde se busque la velocidad y sencillez. Además permite depurar el código generado línea por línea. Si se necesita un lenguaje de alto nivel para realizar programas más complejos, puede ser útil empezar con BASIC ya que es más sencillo de aprender, pero proporciona malos hábitos de programación al ser de tipo "no estructurado" y no es tan potente como C, por lo que si se tiene ya conocimientos mínimos de programación lo mejor es el desarrollo en C debido a la facilidad que ofrecen sus instrucciones.

2.5.7. Grabación de los PICs

Para transferir el código de un ordenador al PIC normalmente se usa un dispositivo llamado programador. La mayoría de PICs que Microchip distribuye hoy en día incorporan ICSP (*In Circuit Serial Programming*, programación serie incorporada) o LVP (*Low Voltage Programming*, programación a bajo voltaje), lo que permite programar el PIC directamente en el circuito destino. Muchos programadores incluyen ellos mismos PICs preprogramados como interfaz para enviar las órdenes al PIC que se desea programar. Uno de los programadores más simples es el TE20, que utiliza la línea TX del puerto RS232 como alimentación y las líneas DTR y CTS para mandar o recibir datos cuando el microcontrolador está en modo programación. El software de programación puede ser el ICprog, muy común entre la gente que utiliza este tipo de microcontroladores.

Algunos de los programadores más usados son:

- PICStart Plus (puerto serie y USB)
- Promate II (puerto serie)
- MPLAB PM3 (puerto serie y USB)
- ICD2 (puerto serie y USB)
- PICKit 1 (USB)
- IC-Prog 1.0X
- PICAT 1.25 (puerto USB2.0 para PICs y Atmel)
- WinPic 800 (puerto paralelo, serie y USB)
- Terusb1.0

3. Interfaz

3.1. Introducción a la interfaz

La interfaz es la forma en que los usuarios pueden comunicarse con una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo. Sus principales funciones son:

- Manipulación de archivos y directorios
- Herramientas de desarrollo de aplicaciones
- Comunicación con otros sistemas
- Información de estado
- Configuración de la propia interfaz y entorno
- Intercambio de datos entre aplicaciones
- Control de acceso
- Sistema de ayuda interactivo.

Existen dos tipos de interfaz de usuario:

- Interfaces alfanuméricas (intérpretes de mandatos).
- Interfaces gráficas de usuario (GUI, *Graphics User Interfaces*), las que permiten comunicarse con el ordenador u otros dispositivos de una forma muy rápida e intuitiva.

3.2. Interfaz gráfica

Se trata del artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

La interfaz gráfica de usuario (*Graphical User Interface*, 'GUI') es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora y es pieza fundamental en un entorno gráfico.

Para que la interfaz gráfica funcione correctamente en Windows es necesario tener instalado la herramienta .NET Framework, cuya función se explica en el capítulo **3.3 .NET Framework**.

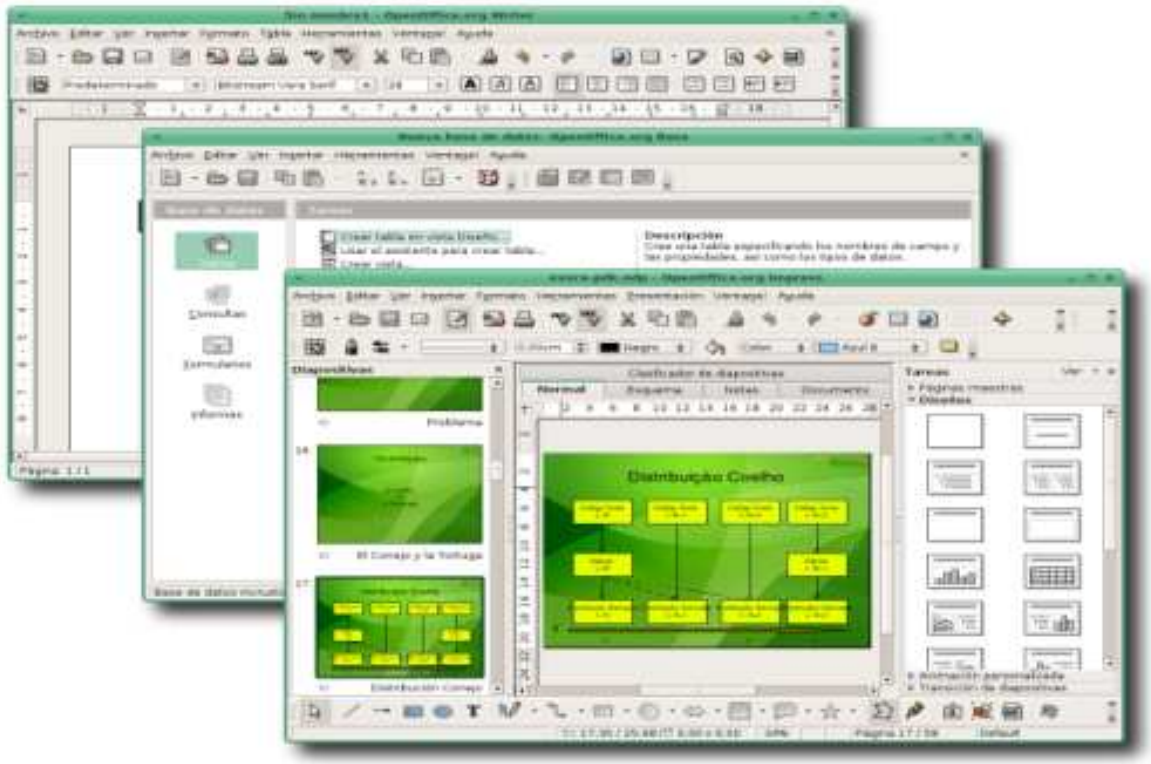


Ilustración 14. Ejemplo de interfaz gráfica

3.2.1. Elección de la herramienta usada para la interfaz

Existen multitud de programas para el desarrollo de interfaces. A continuación se muestran algunos de ellos y sus características.

Delphi: Uno de los usos habituales de Delphi (aunque no el único) es el desarrollo de aplicaciones visuales (interfaz) y de bases de datos cliente-servidor y multicapas. Debido a que es una herramienta de propósito múltiple, se usa también para proyectos de casi cualquier tipo, incluyendo aplicaciones de consola, aplicaciones de web (por ejemplo servicios web, CGI, ISAPI, NSAPI, módulos para Apache), servicios COM y DCOM, y servicios del sistema operativo. Su lenguaje de programación es una versión moderna de Pascal llamada Object Pascal.

NetBeans: plataforma para el desarrollo de aplicaciones de escritorio usando Java.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Visual Basic: Visual Basic es un lenguaje de programación desarrollado por Microsoft. El objetivo de Visual Basic es simplificar la programación utilizando un ambiente de desarrollo completamente gráfico que facilite la creación de interfaces gráficas y en cierta medida también la programación. En 2001 Microsoft propone abandonar el desarrollo basado en la API Win32 y pasar a trabajar sobre un Framework o marco común de librerías independiente de la versión del sistema operativo, .NET Framework, a través de Visual Basic .NET (y otros lenguajes como C-Sharp (C#) de fácil transición de código entre ellos).

Visual Basic constituye un IDE (entorno de desarrollo integrado o en inglés Integrated Development Enviroment), es decir, consiste en un editor de código (programa donde se escribe el código fuente), un depurador (programa que corrige errores en el código fuente para que pueda ser bien compilado), un compilador (programa que traduce el código fuente a lenguaje de máquina), y un constructor de interfaz gráfica o GUI (es una forma de programar en la que no es necesario escribir el código para la parte gráfica del programa, sino que se puede hacer de forma visual). El lenguaje que utiliza para programar es el Basic.

Visual C#: es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. Visual C# es simple, eficaz, con seguridad de tipos y orientado a objetos. Con sus diversas innovaciones, Visual C Express permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

El programa seleccionado para la realización de la interfaz gráfica ha sido Visual C#, debido a los conocimientos que se poseen de su lenguaje y a los numerosos ejemplos encontrados en la red que ayudaron a desarrollar el código de la interfaz de este PFC.

3.2.2. Introducción a la interfaz deseada

La interfaz debe tener como mínimo las siguientes características:

- Dos ventanas: Una para conocer los datos actuales del local y otra para que el usuario introduzca los valores de peso y aforo máximos del edificio, así como la altura máxima/mínima del individuo que desea acceder y altura a la que está colocada el sensor.
- No se puede escribir en la ventana donde se actualizan los datos.
- Dicha ventana debe actualizarse sola.
- La ventana donde el usuario introduce los datos deseados debe poseer un botón para enviarlos
- No se pueden cambiar los valores mientras existan personas dentro del local
- Se diferencia entre los datos propios del local y los de las personas.
- Debe indicar el estado en el que se encuentra el sistema.
- Dispondrá de menú.

3.2.3. Componentes utilizados en Visual C para la creación de la interfaz

A partir de los datos de la sección anterior se hace uso de Visual C# para crear la interfaz deseada. Existe una amplía gama de componente en este programa pero la aplicación desarrollada utiliza los siguientes:

En primer lugar se elige la opción **Windows forms**, que es la base de todos los componentes que se colocaran.

1. **Label**: Es utilizado para colocar los nombres de los Textbox y que éstos queden identificados, además de indicar la unidad de medida.
2. **Textbox**: Dentro de ellos están los datos que introduce el usuario y los que llegan del PIC
3. **Button**: Si se clickea sobre él se envían los datos introducidos por el usuario al PIC
4. **Menú strip**: Se trata de un menú que tiene tres contenidos:
   i. Archivo
      1. Resetear: Borra los valores (aforo y peso total del local) del PIC.
      2. Salir: Cierra la ventana.
   ii. Modo
      1. Valores actuales: Se coloca en la pestaña de Valores actuales y lo indica.

2. Cambiar valores: Se coloca en la pestaña de Cambiar valores y lo indica.

iii. Ayuda

1. Acerca del microcontrolador: Muestra en una ventana nueva las instrucciones de cómo funciona todo el dispositivo.
2. Información: Abre una ventana donde se visualiza los último valores introducidos por el usuario

5. **Status strip**: Este elemento es una barra de estado, en ella existen dos etiquetas.
   i. En la primera de ellas se indica el estado en que se encuentra el PIC, es decir, si está actualizando datos, entrando persona…
   ii. En la segunda se observa el estado actual del USB (conectado o no).
6. **Timer:** Son temporizadores que se activan cada cierto tiempo ejecutando una serie de instrucciones. Existen dos timers.
   i. El primero de ellos envía una señal al PIC para que éste le envíe los últimos datos que tenga en la EEPROM (se activa cada 300 ms.)
   ii. El segundo de ellos está pendiente de la entrada o salida de personas, así como de las señales de confirmación que le llegan del PIC. (se activa cada 200 ms.).

El resultado se puede observar aquí:

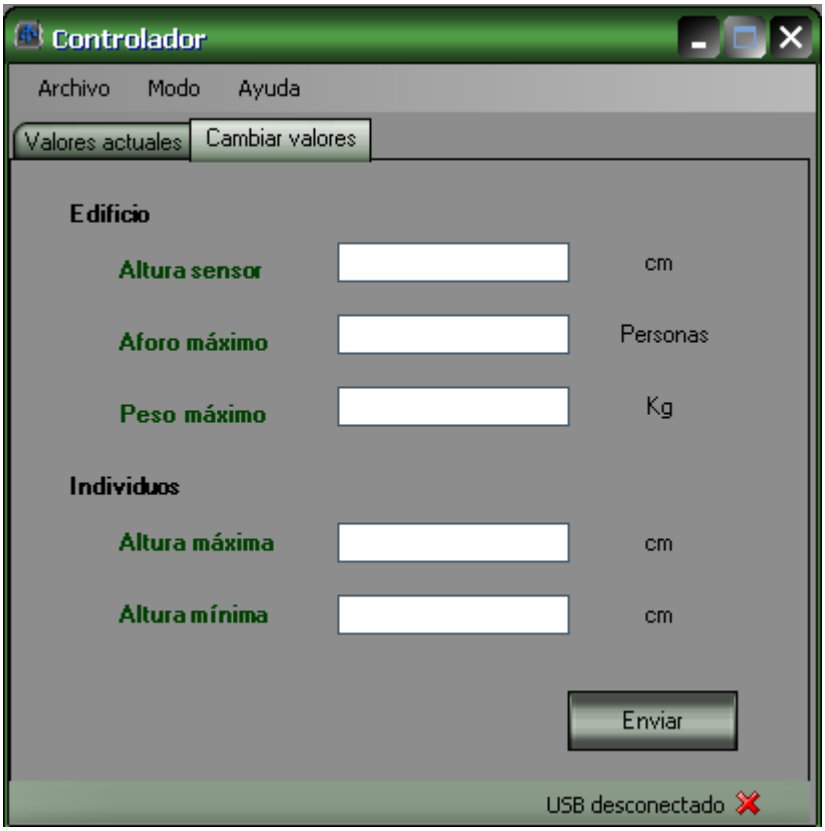


Ilustración 15. Pestaña "Cambiar valores"

En esta pestaña, el usuario introduce los datos, que posteriormente se utilizan por el PIC para descartar a las personas que no cumplan sus condiciones.

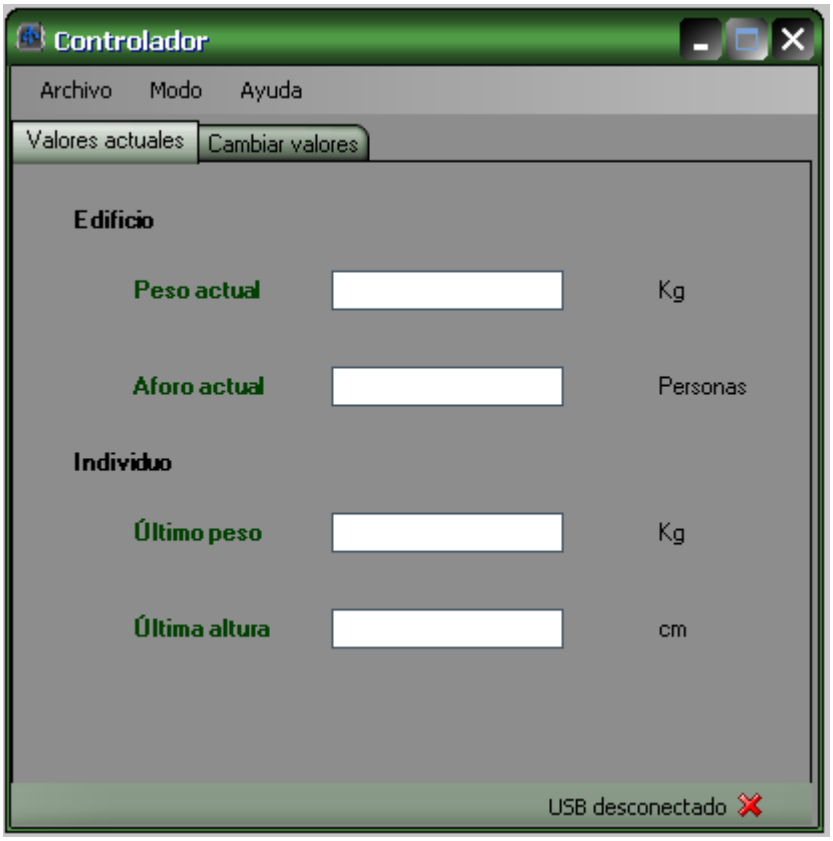


Ilustración 16. Pestaña "Valores actuales"

En la pestaña de “Valores actuales”, el usuario tiene los valores de peso y aforo, que en ese momento, se encuentran en el local o edificio.

La actualización de los datos se realiza cada 300 ms.

3.2.4. Librería MPUSBAPI.dll

Microchip proporciona el driver mpusbapi.dll que contiene las funciones necesarias para establecer la comunicación con el dispositivo USB.

Las funciones principales proporcionada por el driver son:

DWORD _MPUSBGetDLLVersion() → Lee la versión del mpusbapi.dll.
DWORD _MPUSBGetDeviceCount(string pVID_PID) → Devuelve el numero de dispositivos con el mismo VID&PID.
void* _MPUSBOpen(DWORD instance, string pVID_PID, string pEP, DWORD dwDir, DWORD dwReserved) → Abre una Pipe con un dispositivo al endpoint seleccionado.

DWORD _MPUSBRead(void* handle, void* pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds) → Lee los datos del buffer.
DWORD _MPUSBWrite(void* handle, void* pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds) → Escribe datos en el buffer.
bool _MPUSBClose(void* handle) → Cierra el Pipe.

3.3. Software de la interfaz gráfica: NET Framework

3.3.1. Introducción

.NET Framework es un componente integral de Windows que admite la creación y la ejecución de la siguiente generación de aplicaciones y servicios Web XML. El diseño de .NET Framework está enfocado a cumplir los objetivos siguientes:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.
- Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.
- Ofrecer un entorno de ejecución de código que fomente la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos.
- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes.
- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

.NET Framework contiene dos componentes principales: Common Language Runtime y la biblioteca de clases de .NET Framework. Common Language Runtime es el fundamento de la tecnología. El motor en tiempo de ejecución se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la interacción remota, al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que fomentan su seguridad y solidez. De hecho, el concepto de administración de código es un principio básico del motor en tiempo de ejecución. El código destinado al motor en tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado. La biblioteca de clases, el otro componente principal de .NET Framework, es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI) o de línea de comandos hasta las aplicaciones basadas en las innovaciones más recientes proporcionadas por ASP.NET, como los formularios Web Forms y los servicios Web XML.

.NET Framework puede alojarse en componentes no administrados que cargan Common Language Runtime en sus procesos e inician la ejecución de código administrado, con lo que se crea un entorno de software en el que se pueden utilizar características administradas y no administradas. En .NET Framework no sólo se ofrecen varios hosts de motor en tiempo de ejecución, sino que también se admite el desarrollo de estos hosts por parte de terceros.

3.3.2. NET Framework en contexto

En la ilustración siguiente se muestra la relación de Common Language Runtime y la biblioteca de clases con las aplicaciones y el sistema en su conjunto. En la ilustración se representa igualmente cómo funciona el código administrado dentro de una arquitectura mayor.

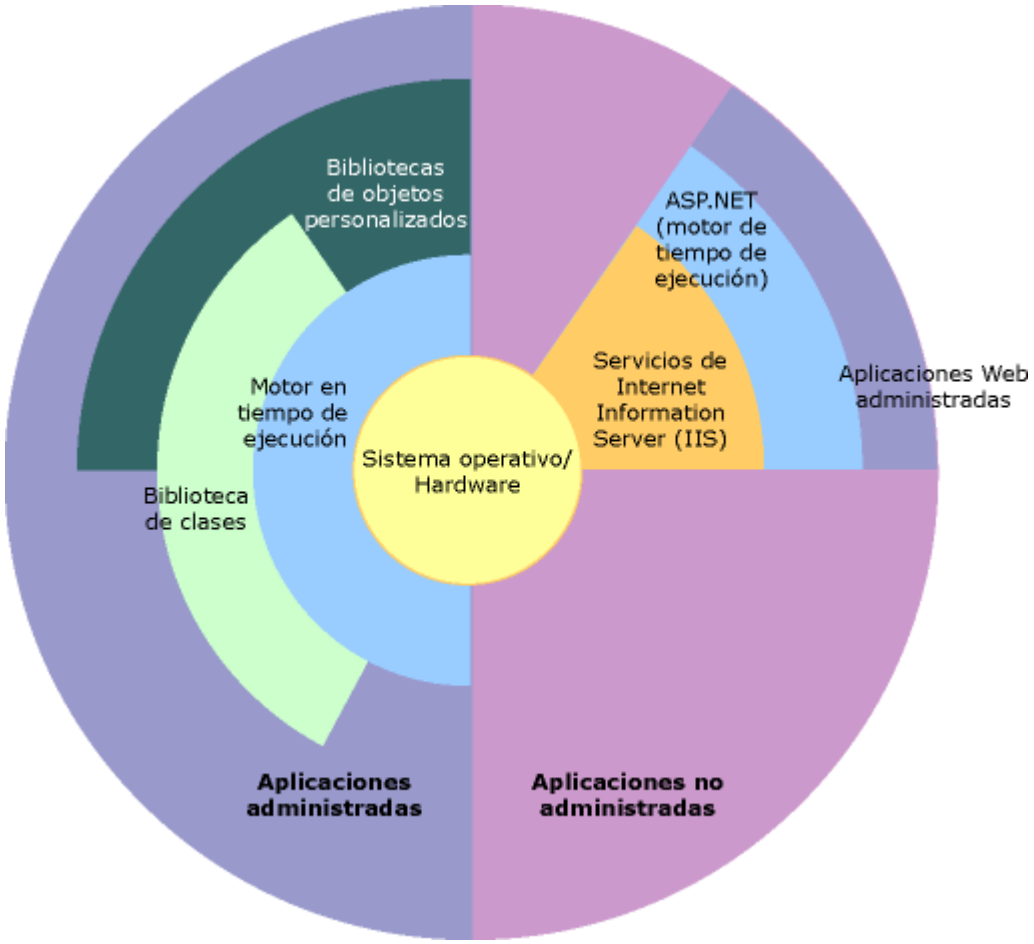


Ilustración 17. .NET Framework

3.3.3. Características de Common Language Runtime

Common Language Runtime administra la memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema. Estas características son intrínsecas del código administrado que se ejecuta en Common Language Runtime.

Con respecto a la seguridad, los componentes administrados reciben grados de confianza diferentes, en función de una serie de factores entre los que se incluye su origen (como Internet, red empresarial o equipo local).

El motor en tiempo de ejecución impone seguridad en el acceso al código. Además, el motor en tiempo de ejecución impone la solidez del código mediante la implementación de una infraestructura estricta de comprobación de tipos y código denominado CTS (Common Type System, Sistema de tipos común). CTS garantiza que todo el código administrado es auto descriptivo. Los diferentes compiladores de lenguajes de Microsoft y de terceros generan código administrado que se ajusta a CTS. Esto significa que el código administrado puede usar otros tipos e instancias administrados, al tiempo que se aplica inflexiblemente la fidelidad y seguridad de los tipos.

Además, el entorno administrado del motor en tiempo de ejecución elimina muchos problemas de software comunes y aumenta la productividad del programador. Esta administración automática de la memoria soluciona los dos errores más comunes de las aplicaciones: la pérdida de memoria y las referencias no válidas a la memoria.

Los compiladores de lenguajes que se destinan a .NET Framework hacen que las características de .NET Framework estén disponibles para el código existente escrito en dicho lenguaje, lo que facilita enormemente el proceso de migración de las aplicaciones existentes.

3.3.4. Bibliotecas de clases de .NET Framework

La biblioteca de clases de .NET Framework es una colección de tipos reutilizables que se integran estrechamente con Common Language Runtime. La biblioteca de clases está orientada a objetos, lo que proporciona tipos de los que su propio código administrado puede derivar funciones. Esto ocasiona que los tipos de .NET Framework sean sencillos de utilizar y reduce el tiempo asociado con el aprendizaje de las nuevas características de .NET Framework. Además, los componentes de terceros se pueden integrar sin dificultades con las clases de .NET Framework.

Como en cualquier biblioteca de clases orientada a objetos, los tipos de .NET Framework permiten realizar diversas tareas de programación comunes, como son la administración de cadenas, recopilación de datos, conectividad de bases de datos y acceso a archivos. Además de estas tareas habituales, la biblioteca de clases incluye tipos adecuados para diversos escenarios de desarrollo especializados. Por ejemplo, puede utilizar .NET Framework para desarrollar los siguientes tipos de aplicaciones y servicios:

- Aplicaciones de consola
- Aplicaciones GUI de Windows (formularios Windows Forms)
- Aplicaciones de ASP.NET
- Servicios Web XML
- Servicios de Windows

Por ejemplo, las clases de formularios Windows Forms son un conjunto completo de tipos reutilizables que simplifican enormemente el desarrollo de interfaces GUI para Windows.

4. Diseño e implementación del software

4.1. Adquisición de datos: El convesor A/D

Un conversor analógico-digital (CAD ó ADC) es un dispositivo electrónico capaz de convertir un voltaje determinado en un valor binario, es decir, éste se encarga de transformar señales analógicas a digitales.

4.1.1. La conversión A/D

Una conversión analógica-digital (ADC) consiste en la transcripción de señales analógicas en señales digitales, con el propósito de facilitar su procesamiento (codificación, compresión, etc.) y hacer la señal resultante (la digital) más inmune al ruido y otras interferencias a las que son más sensibles las señales analógicas.

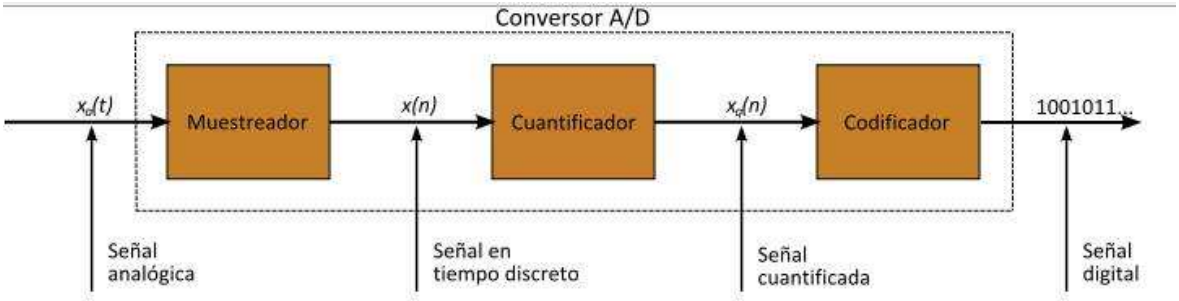


Ilustración 18. Conversor A/D

4.1.2. Funcionamiento

Estos conversores poseen dos señales de entrada llamadas Vref+ y Vref- y determinan el rango en el cual se convierte una señal de entrada.

El dispositivo establece una relación entre su entrada (señal analógica) y su salida (digital) dependiendo de su resolución. Esta resolución se puede saber, siempre y cuando se conozca el valor máximo que la entrada de información utiliza y la cantidad máxima de la salida en dígitos binarios. Para este proyecto se utiliza el convertidor análogo digital del PIC 18F4550 que convierte una muestra analógica de entre 0 y 5 voltios que le llega del sensor de peso y su resolución es:

Resolución = valor analógico / (2^8) (el 8 es debido a que la salida es de 8 bits).

Resolución = 5 V / 256

Resolución = 0.0195v o 19.5mv.

Resolución = LSB (Byte menos significativo)

Lo anterior quiere decir que por cada 19.5 milivoltios que aumente el nivel de tensión entre las entradas denominadas como "Vref+" y "Vref-" (que en este caso tiene los valores +5V y 0V respectivamente) que ofician de entrada al conversor, éste aumenta en una unidad su salida (siempre sumando en forma binaria bit a bit). Por ejemplo:

ENTRADA	SALIDA
0 V	00000000
0.02 V	00000001
0.04 V	00000010
1 V	00110011
5 V	11111111

Tabla 3. Ejemplo conversión A/D

Una vez explicado el funcionamiento se pasa a describir los SFR's que sirven para configurar el ADC:

-- **ADCON0** depende de estos bits de control.

- **CHS3…CHS0**: Bits selección del canal de conversión A/D (13 canales).
- **GO/DONE**: Bit de inicio y de monitorización del estado de la conversión A/D:
  - GO/DONE='0': Proceso de conversión parado.
  - GO/DONE='1': Proceso de conversión en marcha.
- **ADON**: Bit de habilitación del convertidor A/D:
  - ADON='0': Convertidor A/D desactivado.
  - ADON='1': Convertidor A/D activado.

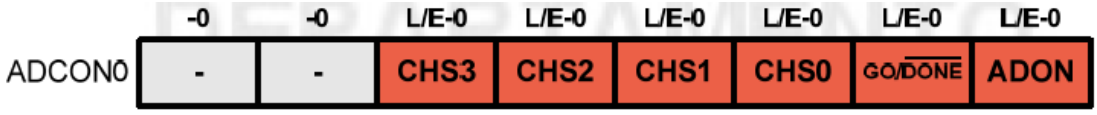


Ilustración 19. ADCON 0

Para elegir los canales que han sido configurados como entradas analógicas, y así ver su valor, hay que fijarse en esta tabla:

CHS3	CHS2	CHS1	CHS0	CANAL SELECCIONADO
0	0	0	0	CANAL AN0 (RA0)
0	0	0	1	CANAL AN1 (RA1)
0	0	1	0	CANAL AN2 (RA2)
0	0	1	1	CANAL AN3 (RA3)
0	1	0	0	CANAL AN4 (RA5)
0	1	0	1	CANAL AN5 (RE0)
0	1	1	0	CANAL AN6 (RE0)
0	1	1	1	CANAL AN7 (RE2)
1	0	0	0	CANAL AN8 (RB2)
1	0	0	1	CANAL AN9 (RB3)

1	0	1	0	CANAL AN10 (RB1)
1	0	1	1	CANAL AN11 (RB4)
1	1	0	0	CANAL AN12 (RB0)
1	1	0	1	No implementado
1	1	1	0	No implementado
1	1	1	1	No implementado

Tabla 4. Bits CHS3...CHS0

-- **ADCON1** cuyos bits de control son:

- **VCFG1**: Bit de configuración de la tensión de referencia VREF-:
  - VCFG1='0': VREF- se conecta a VSS.
  - VCFG1='1': VREF- se conecta a la línea física RA2.
- **VCFG0**: Bit de configuración de la tensión de referencia VREF+:
  - VCFG1='0': VREF+ se conecta a VDD.
  - VCFG1='1': VREF+ se conecta a la línea física RA2.
- **PCFG3…PCFG0**: Bits configuración de los puertos de conversión A/D. Mediante estos bits se establecen que líneas físicas (RA5…RA0, RB4…RB0, RE1 y RE0) van a trabajar como entradas del convertidor A/D (Ver Tabla de configuración de líneas de conversión A/D).

Ilustración 20. ADCON 1

Para utilizar el conversor A/D, previamente deben configurarse los bits PCFG3…PCFG0 para seleccionar que puertos actúan como entradas analógicas. La tabla que viene a continuación muestra las opciones de configuración que existen:

PCFG3…PCFG0	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A

1111	D	D	D	D	D	D	D	D	D	D	D	D	D

Tabla 5. Bits PFG3...PFG0

-- **ADCON2** gobernado por estos bits de control:

- **ADFM**: Bit de configuración del tipo de almacenamiento del resultado de la conversión en los registros ADRESH y ADRESL:
  - ADFM='0': El resultado de la conversión se almacena con justificación a izquierdas.
  - ADFM='1': El resultado de la conversión se almacena con justificación a derechas
- **ACQT2…ACQT0**: Bits de configuración del tiempo de adquisición
- **ADCS2…ADCS0**: Bits selección de la señal de reloj del convertidor A/D

Ilustración 21. ADCON 2

En este registro se habla de tiempo de adquisición y tiempo de conversión, los cuales se definen para entender la función que cumplen.

En primer lugar, se habla del *tiempo de adquisición*, que se define como el tiempo total que hay que esperar hasta que se carga el condensador interno (Chold). Dicho tiempo se puede establecer de dos formas:

- **Por programa**: se implementa un retardo software entre la selección del nuevo canal y el inicio de la conversión.
- Estableciendo un **tiempo de adquisición automático**: se programa un tiempo de adquisición que se establece de forma automática entre la orden de inicio de conversión y el muestreo de la señal para iniciar la conversión. Dicho tiempo se programa mediante los bits ACQT2…ACQT0 del registro ADCON2:

ACQT2	ACQT1	ACQT0	TIEMPO DE ADQUISICIÓN
0	0	0	$0*T_{AD}$
0	0	1	$2*T_{AD}$
0	1	0	$4*T_{AD}$
0	1	1	$6*T_{AD}$
1	0	0	$8*T_{AD}$
1	0	1	$12*T_{AD}$
1	1	0	$16*T_{AD}$
1	1	1	$20*T_{AD}$

Tabla 6.Bits ACQT2...ACQT0

Por último está el tiempo de conversión (T_{AD}), definido como el tiempo de conversión de 1 bit. Una operación completa de conversión requiere un total de 11 T_{AD} para 10 bits. El valor de T_{AD} debe ser lo menor posible, pero siempre superior al T_{AD} mínimo indicado en las hojas de datos de PIC18F4550, siendo en este caso, 0.7us.

La señal de reloj que genera las temporizaciones T_{AD} puede ser establecida mediante los bits ADCS2…ADCS0 del registro ADCON2. Existen dos fuentes para dicha señal de reloj:

- **El oscilador principal.**
- Una **red RC** interna que incorpora el propio convertidor A/D. Esta red puede utilizarse cuando se deseen realizar conversiones en modos de bajo consumo. Esta red RC permite que se puedan llevar a cabo conversiones con el oscilador principal desactivado.

Por último se muestra la tabla que regula el tiempo de conversión:

ADCS2	ADCS1	ADCS0	SEÑAL DE RELOJ DE CONVERSIÓN
0	0	0	$F_{OSC}/2$
0	0	1	$F_{OSC}/8$
0	1	0	$F_{OSC}/32$
0	1	1	F_{RC}(oscilador RC interno)
1	0	0	$F_{OSC}/4$
1	0	1	$F_{OSC}/16$
1	1	0	$F_{OSC}/64$
1	1	1	F_{RC}(oscilador RC interno)

Tabla 7. Bits ADCS2...ADCS0

En cuanto a la configuración y lectura de datos, se utilizan las siguientes funciones de CCS:

- undefinied setup_adc(mode)_→ Habilita el modulo A/D, configure el reloj del conversor, etc.
- undefinied setup_adc_ports(value) → Configura los pins que son analógicos o digitales.
- undefinied set_adc_channel(channel) → Especifica que canal es leído por el módulo A/D.
- int read_adc(mode) → Inicia la conversión y lee el valor. Mode es un parámetro opcional que controla la forma de leer el valor del conversor.

Resumiendo, estos serían los pasos a seguir para el proceso de conversión A/D:

1. **PASO**: Configuración del convertidor A/D:
   a. Configuración como canales A/D de las líneas que vayan a ser utilizadas (bits PCFG3...PCFG0 del registro ADCON1).
   b. Configuración de las tensiones de referencia VREF+ y VREF- (bits VCFG0 y VCFG1 del registro ADCON1).

   c. Configuración del reloj de conversión TAD (bits ADCS2…ADCS0 del registro ADCON2).
   d. Configuración del tiempo de adquisición (bits ACQT2…ACQT0 del registro ADCON2).
   e. Configuración del modo de almacenamiento de la conversión (bit ADFM del registro ADCON2).
   f. Activación del conversor (bit ADON del registro ADCON0).
2. **PASO**: Selección del canal (bits CHS3…CHS0 del registro ADCON0).
3. **PASO**: Retardo de espera del tiempo de adquisición (solo en caso de no hacer uso del tiempo de adquisición automático).
4. **PASO**: Inicio de la conversión poniendo a '1' el bit GO/DONE del registro ADCON0.
5. **PASO**: Bucle de espera del final de conversión (comprobación del bit GO/DONE hasta que se ponga a '0').
6. **PASO**: Lectura del resultado de la conversión de los registros ADRESH y ADRESL.

4.2. Almacenamiento permanente de datos: La memoria EEPROM

4.2.1. Introducción

EEPROM son las siglas de *electrically-erasable programmable read-only memory* (ROM programable y "borrable" eléctricamente), en español o castellano se suele referir al hablar como E²PROM y en inglés "E-Squared-PROM". Es un tipo de memoria ROM que puede ser programado, borrado y reprogramado eléctricamente, a diferencia de la EPROM que ha de borrarse mediante rayos ultravioletas. Aunque una EEPROM puede ser leída un número ilimitado de veces, sólo puede ser borrada y reprogramada entre 100.000 y 1.000.000 de veces.

Estos dispositivos suelen comunicarse mediante protocolos como I²C, SPI y Microwire. En otras ocasiones se integra dentro de chips como microcontroladores y DSPs para lograr una mayor rapidez.

4.2.2. Tamaño y memoria utilizada

El PIC 18F4550 dispone de una memoria EEPROM de datos de 256 bytes. Al ser una memoria no volátil los datos almacenados en ella se mantienen aún en ausencia de tensión de alimentación.

El acceso a esta memoria se realiza mediante los SFR´s (Special Function Registers):

-- EECON1 cuyos bits de control son:

- **EEPGD**: Bit de selección de acceso a memoria Flash/EEPROM:
  - EEPG =’0’: Acceso a memoria de datos EEPROM.
  - EEPGD=‘1’: Acceso a memoria Flash de programa.
- **CFGS**: Bit de selección de acceso a memoria (Flash programa-EEPROM daros)/Configuración:
  - CFGS=’0’: Acceso a memoria de Flash de programa o a memoria de datos EEPROM.
  - CFGS=’1’: Acceso a los registros de configuración de la memoria Flash/EEPROM se ha llevado a cabo correctamente.
- **FREE:** Bit de habilitación del borrado de una fila en memoria Flash:
  - FREE=‘0’: Activada únicamente la opción de lectura
  - FREE=‘1’: Borrado de la fila de la memoria Flash de programa apuntada por TBLPTR en el siguiente comando de escritura (el bit se pone a ‘0’ cuando la operación de borrado se haya completado)

- **WRERR**: Bit de error de escritura en memoria Flash/EEPROM:

- WERR=’0’: La operación de escritura en la memoria EEPROM se ha llevado a cabo correctamente;
- WERR=’1’: Se ha producido un error en la operación de escritura en la memoria Flash/EEPROM.

o **WREN**: Bit de habilitación de la operación de escritura en la memoria Flash/EEPROM:
- WREN=’0’: Operación de escritura en la memoria Flash/EEPROM deshabilitada.
- WREN=’1’: Operación de escritura en la memoria Flash/EEPROM habilitada.

o **WR**: Bit de control de escritura en la memoria Flash/EEPROM:
- WR=’0’: La operación de escritura en la memoria Flash/EEPROM se ha completado.
- WR=’1’: Inicio de una operación de borrado/escritura en memoria Flash/EEPROM o de las operaciones de borrado y/o escritura en memoria Flash (cuando la operación termina el bit se pone automáticamente a ‘0’;por programa solo puede ponerse a ‘1’).

o **RD**: Bit de control de lectura en memoria EEPROM:
- RD=’0’: La operación de lectura en la memoria EEPROM se ha completado.
- RD=’1’: Inicio de una operación de lectura en memoria EEPROM (cuando la operación termina el bit se pone automáticamente a ‘0’; por programa sólo puede ponerse a ‘1’).

Ilustración 22. EECON 1

-- **EECON2**: No es un registro físico. Es usado exclusivamente en las secuencias de escritura y borrado de datos. La lectura de EECON2 es ‘0’ en todos sus bits.

Respecto a las funciones en CCS usadas para este proceso, están descritas abajo:

- int read_program_eeprom(address) → Lee la dirección indicada en el paréntesis y devuelve su valor.
- undefinied write_program_eeprom(address, value) → Escribe el valor escrito por el usuario en la dirección indicada.
- undefinied erase_program_eeprom(address) → Borra la memoria de la dirección escrita. Esta función no es utilizada en este proyecto.

-
- Por último, la tabla situada abajo informa de las direcciones usadas y que valor contiene:

DIRECCÍON	CONTENIDO
0x00	Byte que guarda el peso de la última persona que entró
0x02	Byte de apoyo para iniciar el resto de la memoria EEPROM a 0 sólo 1 vez
0x03	Low aforo ACTUAL
0x04	High aforo ACTUAL
0x06	Low aforo MÁXIMO
0x07	High aforo MÁXIMO
0x13	Low peso ACTUAL
0x14	High peso ACTUAL
0x16	Low peso MÁXIMO
0x17	High peso MÁXIMO
0x20	Byte que guarda la altura de la persona
0x25	Byte que guarda la distancia a la que está colocada el sensor de altura
0x26	Byte que guarda la altura mínima para poder acceder
0x27	Byte que guarda la altura máxima para poder acceder
0x30	Low altura acumulada para hacer la media
0x31	High altura acumulada para hacer la media
0x32	Low peso acumulado para hacer la media
0x33	High peso acumulado para hacer la media

Tabla 8. Memoria EEPROM usada

4.3. Comunicación PIC / Interfaz

4.3.1. Comunicación PIC

La comunicación entre el PIC y el PC a través del bus USB se realiza mediante las funciones proporcionadas por el driver del compilador PCWH. Estas funciones vienen definidas e implementadas en las librerías pic18_usb.h, usb.h y usb.c.

El driver proporciona las siguientes funciones:

- void usb_init () → Inicializa el USB. La función entra en un bucle infinito hasta conseguir respuesta del dispositivo.
- void usb_init_cs () → Lo mismo que el anterior pero no espera infinitamente al dispositivo. Útil en dispositivos cuyo funcionamiento no depende exclusivamente del USB.
- void usb_task () → Si se ha usado usb_init_cs () para inicializar la conexión, periódicamente se observa el estado con la función usb_task ().
- Esta función prepara el periférico cuando detecta conexión, y habilita las interrupciones.
- void usb_attach () / usb_detach () → Enlaza/Desenlaza el PIC al bus USB. Estas funciones son usadas por la función usb_task ();
- boolean usb_enumerated () → Devuelve TRUE si el dispositivo está conectado y reconocido por el PC.
- boolean usb_kbhit () → Devuelve TRUE si hay datos esperando ser leídos.
- void usb_put_packet () y void usb_get_packet () → Se usan para leer/enviar datos al PC.

4.3.2. Comunicación interfaz

Recordando el capítulo **3.3.3 La librería MPUSBAPI.dll** donde se hablaba del driver necesario para la comunicación entre la interfaz y el PIC, pues a partir de ese driver, utilizando sus funciones, se consigue crear una API para enviar y recibir paquetes.

Las nuevas funciones son:

- public void OpenPipes() {} → Abre un Pipe de salida y uno de entrada. Utiliza la función _MPUSBOpen().
- public void ClosePipes() {} → Cierra las Pipes creadas. Utiliza la función _MPUSBClose.
- private void SendPacket(byte* SendData, DWORD SendLength) → Abre un Pipe, envia los datos a través del Pipe de salida y cierra el Pipe. Utiliza las funciones _MPUSBOpen(), _MPUSBClose(), y _MPUSBWrite().
- private void ReceivePacket(byte* SendData, DWORD SendLength) → Abre un Pipe, recibe los datos a través del Pipe de entrada y cierra el Pipe. Utiliza las funciones _MPUSBOpen(), _MPUSBClose(), y _MPUSBWrite().

Una vez creado la API, el siguiente paso es desarrollar las funciones específicas del proyecto, cuya labor es la de recoger los datos que introduce el usuario y enviarlos al PIC y recibir los datos del PIC, para que en todo momento se tengan los datos más actuales.

Las nuevas funciones son:

- public void RecibirDato() {} → Se envía para que el PIC transmita los datos actuales.
- public void Entrar() {} → Señal de confirmación para saber que una persona ha entrado y así indicarlo en la interfaz.
- public void Salir() {} → Señal de confirmación para saber que una persona ha salido y así indicarlo en la interfaz.
- public void Solosalir() {} → Señal para indicar en la interfaz que sólo puede salir las personas.
- public void valores() {} → Se envía una señal al PIC para que éste transmita los últimos datos introducidos por el usuario.
- public void Resetear() {} → Esta señal resetearía los valores del PIC (aforo actual, peso actual, último peso y última altura).
- public uint ValorGuardar1() {} → Recibe los valores actuales del PIC por el pipe devolviendo el peso actual del local.
- public uint ValorGuardar2() {} → Puesto que ValorGuardar1 sólo puede devolver un resultado, el resto se guardan en variables locales. En esta función se devuelve el valor del aforo actual para que se muestre por la interfaz.
- public uint ValorGuardar3() {} → devuelve el peso de la última persona que entró al edificio.
- public uint ValorGuardar4() {} → devuelve la altura de la última persona que entró.
- public uint ValorGuardar5() {} → Utilizado para saber cambiar las etiquetas en la barra de estado de la interfaz, dependiendo el modo que devuelva el PIC muestra etiquetas diferentes.
- public uint EnviarDatos(uint dato,uint dato2, uint dato3, uint dato4, uint dato5) {} → Como su propio nombre indica, envía los datos introducidos por el usuario al PIC. Comprueba que los datos cumplan unas condiciones (ejemplo: que la altura máxima no sea menor que la mínima, que el aforo no sobrepase un cierto límite…). En caso de que algún dato no esté correcto devuelve 0 y no envía los datos.

4.4. Los puertos del PIC

El PIC18F4550 dispone 5 puertos de E/S que incluyen un total de 35 líneas digitales de E/S. El número de entradas/salidas de cada puerto lo muestra la siguiente tabla:

PUERTO	LÍNEAS DE ENTRADA SALIDA
PORTA	7 LÍNEAS DE ENTRADA/SALIDA
PORTB	8 LÍNEAS DE ENTRADA/SALIDA
PORTC	6 LÍNEAS DE ENTRADA/SALIDA + 2 LÍNEAS DE ENTRADA
PORTD	8 LÍNEAS DE ENTRADA/SALIDA
PORTE	3 LÍNEAS ENTRADA/SALIDA + 1 LÍNEA DE ENTRADA

Tabla 9. Distribución de los puertos

Todas las líneas digitales de E/S disponen de al menos una función alternativa asociada a alguna circuitería especifica del microcontrolador. Cuando una línea trabaja en el modo alternativo no puede ser utilizada como línea digital de E/S estándar. Por ejemplo, si se utiliza el PORTA, 0 como canal analógico, éste ya no se puede utilizar como entrada/salida digital.

Cada puerto de E/S tiene asociado 3 registros:

- **Registro** TRIS: mediante este registro se configuran cada una de las líneas de E/S del puerto como ENTRADA (bit correspondiente a ‘1’) o como SALIDA (bit correspondiente a ‘0’).
- **Registro** PORT: mediante este registro se puede leer el nivel de pin de E/S y se puede establecer el valor del latch de salida.
- **Registro** LAT: mediante este registro se puede leer o establecer el valor del latch de salida. El registro existe por cuestiones de rapidez al cambiar un flanco en un pin.

4.4.1. PORTA

Dispone de 7 líneas de E/S. Las funciones alternativas son:

- ✓ **RA0**: entrada analógica (AN0)/ entrada de comparación (C1IN-)
- ✓ **RA1**: entrada analógica (AN1)/ entrada de comparación (C2IN-)
- ✓ **RA2**: entrada analógica (AN2)/ entrada de comparación (C2IN+)
- ✓ **RA3**: entrada analógica (AN3)/ entrada de comparación (C1IN+)
- ✓ **RA4**: entrada de reloj del Temporizador 0 (T0CKI)/salida de comparación (C1OUT)
- ✓ **RA5**: entrada analógica (AN4)/ salida de comparación (C2OUT)/HLVDIN entrada de detección de tensión alta/baja.
- ✓ **RA6**: entrada del oscilador principal (OSC2)/salida de señal de reloj (CLK0).

En el reset las líneas RA0, RA1, RA2, RA3 y RA5 se configuran como líneas de entrada analógicas. Para poder utilizarlas como líneas digitales de E/S hay que desactivar la función analógica.

4.4.2. PORTB

Dispone de 8 líneas de E/S. Las funciones alternativas son:

- ✓ **RB0**: entrada analógica (AN12)/ interrupción externa 0 (INT0)/entrada de fallo del ECCP (FLT0)/entrada de datos del SPI (SDI)/línea de datos del I2C (SDA).
- ✓ **RB1**: entrada analógica (AN10)/ interrupción externa 1 (INT1)/línea de reloj del SPI (SDI)/línea de reloj del I2C (SDA).
- ✓ **RB2**: entrada analógica (AN8)/ interrupción externa 2 (INT2)/salida de datos del USB (VCMO).
- ✓ **RB3**: entrada analógica (AN9)/ línea de E/S del CCP2 (CCP2)/salida de datos del USB (VPO).
- ✓ **RB4**: entrada analógica (AN11)/ interrupción por cambio en pin (KBI0)/ salida de CS del SSP (CSSP).
- ✓ **RB5**: interrupción por cambio en pin (KBI1)/ línea de programación (PGM).
- ✓ **RB6**: interrupción por cambio en pin (KBI2)/ línea de programación (PGC).
- ✓ **RB7**: interrupción por cambio en pin (KBI3)/ línea de programación (PGD).

Por defecto, en el reset las líneas RB4...RB0 están programadas como entradas analógicas, aunque existen dos formas de configurar RB4…RB0 como líneas de E/S digitales:

- Poniendo a ‘0’ el bit PBADEN del registro de configuración CONFIG3H (en el reset RB4…RB0 se configuran como líneas de E/S digitales).
- Si PBADEN=‘1’ (valor por defecto) se pueden configurar RB4...RB0 como líneas de E/S digitales, desactivando la función analógica.

Todas las líneas del puerto B disponen de *resistencias de pullup* internas que pueden ser activadas poniendo el bit RBPU del registro INTCON2 a ‘0’ (RPBU=‘1’ después de un reset). Si una línea del puerto B se configura como salida, la resistencia de pull-up correspondiente se desactiva automáticamente.

4.4.3. PORTC

Dispone de 5 líneas de E/S (RC0, RC1, RC2, RC6 y RC7) y 2 líneas de solo entrada
(RC4 y RC5). Las funciones alternativas son:

- ✓ **RC0**: salida del oscilador del Temp. 1 (T1OSO)/ entrada de contador de los
- ✓ Temp. 1 y 3 (T13CKI).
- ✓ **RC1**: entrada del oscilador del Temp. 1 (T1OSI)/ línea de E/S del CCP2 (CCP2)/ salida OE del transceiver del USB (UOE).
- ✓ **RC2**: línea de E/S del CCP1 (CCP1)/ salida PWM del ECCP1 (P1A).

- ✓ **RC4**: línea menos del bus USB (D-) / línea de entrada del USB (VM).
- ✓ **RC5**: línea más del bus USB (D-) / línea de entrada del USB (VP).
- ✓ **RC6**: salida de transmisión del EUSART (TX)/ línea de reloj del EUSART (CK).
- ✓ **RC7**: entrada de recepción del EUSART (RX)/ línea de datos síncrona del EUSART (DT)/ salida de datos del SPI (SDO).

En el reset todas las líneas del puerto C quedan configuradas como entradas digitales.

4.4.4. PORTD

Dispone de 8 líneas de E/S. Las funciones alternativas son:

- ✓ **RD0**: línea de datos del SPP (SPP0).
- ✓ **RD1**: línea de datos del SPP (SPP1).
- ✓ **RD2**: línea de datos del SPP (SPP2).
- ✓ **RD3**: línea de datos del SPP (SPP3).
- ✓ **RD4**: línea de datos del SPP (SPP4).
- ✓ **RD5**: línea de datos del SPP (SPP5) / salida PWM del ECCP1 (P1B).
- ✓ **RD6**: línea de datos del SPP (SPP6) / salida PWM del ECCP1 (P1C).
- ✓ **RD7**: línea de datos del SPP (SPP7) / salida PWM del ECCP1 (P1D).

Todas las líneas del puerto D disponen de *resistencias de pull-up* internas que pueden ser activadas poniendo el bit RDPU del registro PORTE a '1' (RPDU='0' después de un reset). Si una línea del puerto D se configura como salida la resistencia de pull-up correspondiente se desactiva automáticamente.

4.4.5. PORTE

Dispone de 3 líneas de E/S (RE0, RE1 y RE2) y 1 línea de solo entrada (RE3). Las funciones alternativas son:

- ✓ **RE0**: entrada analógica (AN5)/ salida de reloj 1 del SPP (CK1SPP).
- ✓ **RE1**: entrada analógica (AN6)/ salida de reloj 2 del SPP (CK2SPP).
- ✓ **RE2**: entrada analógica (AN7)/ salida de habilitación del SPP (OESPP).
- ✓ **RE3**: Línea de reset externo (MCLR) / línea de programación (VPP).

En el reset todas las líneas RE2...RE0 se configuran como entradas analógicas. Para poder utilizarlas como líneas digitales de E/S hay que desactivar la función analógica.

La línea RE3 por defecto tiene la función de Reset del microcontrolador. Si se desea desactivar la función de Reset y utilizar RE3 como línea de entrada digital hay que poner a '0' el bit MCLRE del registro de configuración CONFIG3H.

4.4.6. Configuración de los puertos

Se adjunta la tabla donde se ve la configuración de los puertos usada para este PFC:

PUERTO,BIT	I/O	FUNCIÓN
PORTD,7	O	ENTRADA AUTORIZADA
PORTD,6	O	SALIDA AUTORIZADA
PORTD,5	O	INICIACIÓN MEDIDA SENSOR DE ALTURA
PORTD,4	O	ALTURA MÁXIMA/MÍNIMA EXCEDIDA
PORTD,3	O	AFORO COMPLETADO
PORTD,2	O	PESO EXCEDIDO
PORTD,1	O	SENSOR ALTURA ESTÁ MIDIENDO
PORTD,0	O	NO HA ENTRADO (TORNO SIN GIRAR)
PORTB,7	I	HA SALIDO (GIRA TORNO)
PORTB,6	I	HA ENTRADO (GIRA TORNO)
PORTB,5	O	NO HA SALIDO(TORNO SIN GIRAR)
PORTA,4	O	PERMITE EL GIRO DEL TORNO PARA ENTRAR
PORTA,3	O	PERMITE EL GIRO DEL TORNO PARA SALIR
PORTA,2	I	MIDE PULSO DEVUELTO POR EL SENSOR DE ALTURA
PORTA,1	I	CONVERSOR A/D SALIDA
PORTA,0	I	CONVERSOR A/D ENTRADA

Tabla 10. Configuración de los puertos

Por último se muestra las funciones utilizadas para la configuración de puertos:

- undefinied output_x(address, value) → Muestra por el puerto correspondiente el valor introducido. Hay que tener en cuenta que cuando se usa configura todo el puerto como salida.
- undefinied set_tris_x(value) → Configura el puerto como entrada o salida.

4.5. Funcionamiento del sistema

En primer lugar se muestra el esquema eléctrico del sistema.

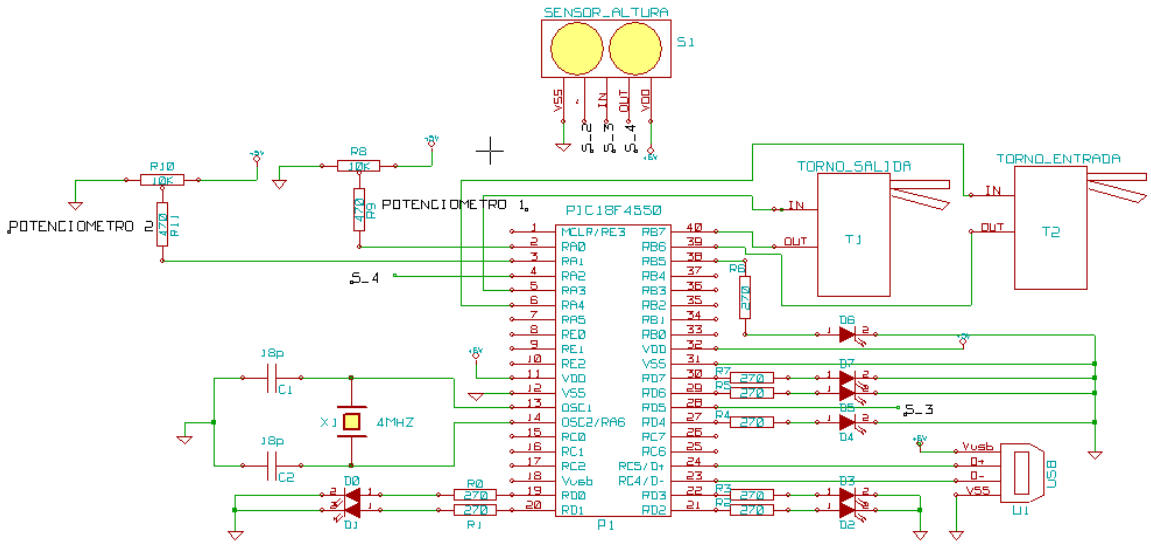


Ilustración 23. Esquema eléctrico del sistema

A continuación se muestra la configuración correcta del PIC utilizando todo lo anteriormente explicado. El PIC queda configurado para realizar las funciones de la siguiente manera:

- Conversor A/D de 8 bits utilizando los canales AN0 (RA0) y AN1 (RA1)
- Timer0 configurado como contador de 16 bits con pre-escalar desactivado, usado por el sensor de altura para contar la duración del pulso que devuelve el sensor, y así calcular la altura
- 18 bytes de la memoria EEPROM usados.
- Utilización del USB
- Puertos adaptados para la iluminación de led's y torno giratorio

Una vez hecho lo anterior, se pasa a la conexión del USB. Cuando se conecta el USB se produce una negociación entre el PC y el PIC, donde hay que instalar el driver que viene con el dispositivo para que se reconozca y empiece a funcionar.

Puesto que la memoria por programación se inicializa a 0, lo primero que debe hacer el usuario es introducir las condiciones de su local (aforo, peso, altura que se encuentra el sensor y altura máxima/mínima del individuo), sino no permite el paso de nadie, puesto que no se activa ningún periférico.

Una vez que el usuario mete los datos por la interfaz y recibe la confirmación de que los datos se han enviado, el sistema ya está dispuesto para funcionar.

El sistema se activa cando la persona que desea entrar al local se coloca sobre la báscula comenzando a generar una señal analógica que convierte el ADC (dicha señal debe

ser como mínimo de 10 kgs.) en señal digital que queda guardada en un registro de la EEPROM.

El paso siguiente es medir la altura de la persona que se encuentra sobre la báscula. Para ello se envía una señal de 15 µs que activa el sensor de altura, así comienza a medir. Se realizan 15 medidas y se hace una media para que sea la altura más exacta. Si el resultado de esa media no cumple las condiciones de altura máxima/mínima impuestas por el usuario, se enciende un led rojo que le indica a la persona que intenta entrar que su altura no está en el rango permitido.

Verificada la altura, se pasa a comprobar si la suma entre el peso calculado por la báscula al peso que hay en el local no sobrepasa el peso máximo. Para ello, se extrae el registro de la EEPROM 0x00, donde se encuentra guardado el peso de la persona y se suma a los registros 0x13 y 0x14, que guardan el peso actual del local. El resultado de dicha suma se resta a los valores 0x16 y 0x17 (que guardan el peso máximo soportado por el local). Si de esa resta sale un valor negativo, evidentemente sobrepasa el peso máximo, por lo que se ilumina un led rojo indicando a la persona que el peso máximo ha sido alcanzado y por lo tanto, debe esperar la salida de alguien. Puesto que se sobrepasa los límites de peso, los valores no se actualizan. En el caso de que la resta sea positiva, se actualizan los registros de la EEPROM que guardan el peso actual del local (0x06 y 0x07).

Si el peso del usuario sumado al del local actual está en el margen permitido, la siguiente comprobación que se hace es la del aforo. Para ello se usan los registros 0x03, 0x04 (que contienen el aforo actual), 0x13 y 0x14 (que guardan el aforo máximo soportado). A los registros 0x03 y 0x04 se les suma 1 y el resultado de dicha suma se resta al valor guardado por los registros 0x13 y 0x14. Igual que en el caso anterior donde si la resta era negativa no se permitía el paso, aquí ocurre lo mismo, ya que se rebasa el aforo máximo permitido, por lo que no se varía los registros 0x03 y 0x04, además de indicarle a la persona mediante un led rojo que debe esperar la salida de otra persona debido a que el aforo está al máximo. Por el contrario, si la suma ha sido positiva se le indica mediante led verde que puede acceder al local, enviando una señal correspondiente al torno para que se pueda producir el giro. Dicha señal sólo permanece durante 5 segundos, tiempo que tiene la persona para girar el torno, En el caso de que no lo gire (es decir que no llegue la señal de giro al PIC que el torno envía), todos los registros utilizados anteriormente y que habían sido modificados, se cambian de nuevo, restándoles los valores de la persona que finalmente no ha entrado.

En el caso de salida de personas del local todo se simplifica más, ya que no se utiliza el sensor de altura y tampoco se necesitan comprobaciones, simplemente con que la persona se coloque encima de la báscula, guarde su peso (que se le resta al peso actual del local) y resté uno al aforo, es suficiente, aunque hay que indicar, que como en el caso anterior, donde una vez que se ilumina el led verde que permite la salida, si al pasar 5 segundos la persona no ha girado el torno, todos los valores que se habían restado deben volver a sumarse.

Se debe aclarar que no se puede atender a la vez las dos básculas, es decir, mientras una esté en funcionamiento, por más que haya otra persona subida a la otra báscula no toma sus datos hasta que no acabe con la primera, aunque está posibilidad se podría cambiar añadiendo otro PIC. Además también está pendiente de las señales que le llegan de la interfaz para actualizar el sistema o incluso para resetearlo, siempre y cuando no esté

atendiendo una petición, ya que se generaría un conflicto. También se debe tener en cuenta que si en pleno proceso de mediciones de altura y peso, la persona se baja de la báscula el sistema se interrumpe, a la espera de que otra persona quiera entrar o salir del local.

Puesto que no siempre se puede tomar dos veces el mismo peso en la misma persona, se puede dar el caso de que el aforo sea 0 pero que el peso no lo sea. Para ello se crea una función que coloca a 0 el peso una vez que el aforo está a 0.

5. PLANOS

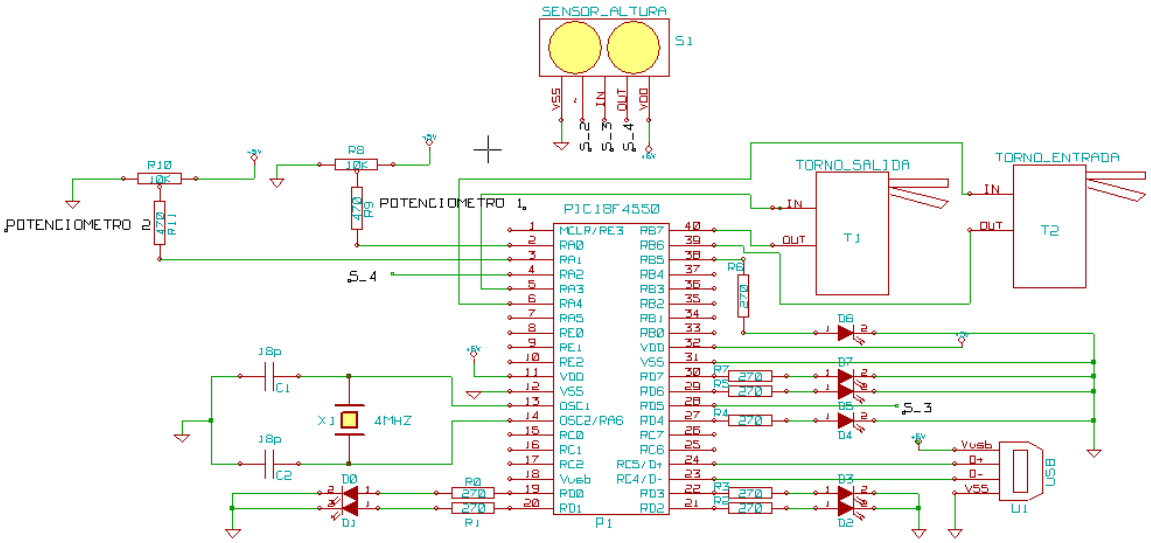


Ilustración 24. Plano del sistema

SIGLAS	DESCRIPCIÓN	CANTIDAD	PRECIO UNITARIO	TOTAL
S1	SENSOR ALTURA SRF05	1	20,35 €	20,35 €
POTENCIÓMETRO 1 Y 2	CELDAS DE CARGA CEL HBB 250	2	445,00 €	890 €
D0…D7	LEDS	8		
T1...T2	TORNO GIRATORIO TR8208	2	2002,00 €	4004 €
R0…R7	RESISTENCIAS 270 OHMIOS	8	0,035 €	0,28 €
P1	PIC 18F4550	1	63,00 €	63 €
			PRESPUESTO TOTAL MATERIAL PFC	4977,63 €

Tabla 11. Tabla de componentes

6. Anexos

6.1. Código del PIC

Principal.c

```
#include <18F4550.h>
#include <libreria.h>
#fuses
XTPLL,NOMCLR,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL1,CPUDIV1,VREGEN,NOPBADEN
#use delay(clock=4000000)

//////////////////////////////////////////////////////////////////
#define USB_HID_DEVICE      FALSE           //deshabilitamos el uso de las directivas HID
#define USB_EP1_TX_ENABLE   USB_ENABLE_BULK  //Creación del endpoint que transmite datos
#define USB_EP1_RX_ENABLE   USB_ENABLE_BULK  //Creación del endpoint que recibe los datos
#define USB_EP1_TX_SIZE     7               //Tamaño del endpoint que transmite los datos
#define USB_EP1_RX_SIZE     8               //Tamaño del endpoint que recibe los datos
//////////////////////////////////////////////////////////////////

#include <pic18_usb.h>   //librerías y drivers utilizadas por el USB
#include <PicUSB.h>
#include <usb.c>

int envia[7];            //array que guarda los datos a enviar
int recibe[8];           //array que guarda los datos recibidos

#define modo    recibe[0]
#define param1  recibe[1]
#define param2  recibe[2]
#define param3  recibe[3]
#define param4  recibe[4]
#define param5  recibe[5]
#define param6  recibe[6]
#define param7  recibe[7]

int     sensor_altura();  //prototipos de las funciones
int     sensor_peso_entrada();
int     suma_peso();
int     sumar_aforo();
void    usb_iniciar();
void    modo1();
void    peticion();
int     torno();
int     torno2();
int     sensor_peso_salida();
int     bascula_on();
void    resetear();

void main(){             //configuración de la palabra, de los
    int memoria;         // periféricos y reseteo de
    set_tris_a(0x07); //de las variables utilizadas de la EEPROM
```

```
		set_tris_b(0xC0);
		set_tris_d(0x00);
		setup_adc(ADC_CLOCK_DIV_8);
		setup_adc_ports(AN0_TO_AN1 || vss_vDD);
		setup_timer_0(RTCC_INTERNAL | RTCC_DIV_2);
		#asm
		bsf	0xfd5,4
		bsf	0xfd5,3
		#endasm
		memoria=read_eeprom(0x02);
		if(memoria==255){
			write_eeprom(0x03,0);
			write_eeprom(0x04,0);
			write_eeprom(0x06,0);
			write_eeprom(0x07,0);
			write_eeprom(0x13,0);
			write_eeprom(0x14,0);
			write_eeprom(0x16,0);
			write_eeprom(0x17,0);
			write_eeprom(0x00,0);
			write_eeprom(0x20,0);
			write_eeprom(0x02,0);
		}
		usb_iniciar();		//conexión con USB
	}

	void usb_iniciar(){
		usb_init();			//inicializamos el USB
		usb_wait_for_enumeration();

	 while(TRUE){
	   if(usb_enumerated())		//si el PicUSB está configurado
	     {
	      if (usb_kbhit(1))	//si el endpoint de salida contiene datos
	        {
	          usb_get_packet(1, recibe, 8); //cogemos el paquete de tamaño 8 bytes del EP1 y almacenamos en recibe

			if(modo == 0){		//se guardan los datos enviados por //el usuario en la EEPROM
				write_eeprom(0x25,param1);
				write_eeprom(0x27,param2);
				write_eeprom(0x26,param3);
				write_eeprom(0x16,param4);
				write_eeprom(0x17,param5);
				write_eeprom(0x06,param6);
				write_eeprom(0x07,param7);
				write_eeprom(0x03,0);
				write_eeprom(0x04,0);
				write_eeprom(0x13,0);
				write_eeprom(0x14,0);
				write_eeprom(0x00,0);
				write_eeprom(0x20,0);
				modo1();	//envia la señal de confirmación
				usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
				delay_ms(1000);
		}

			if(modo==1){	//envía datos actuales de la EEPROM
				envia[0]=read_eeprom(0x03);
```

```
                envia[1]=read_eeprom(0x04);
                envia[2]=read_eeprom(0x13);
                envia[3]=read_eeprom(0x14);
                envia[4]=read_eeprom(0x00);
                envia[5]=read_eeprom(0x20);
                envia[6]=2;
                usb_put_packet(1, envia, 7, USB_DTS_TOGGLE);
            }

            if(modo==2){    //resetea los valores de la EEPROM
                write_eeprom(0x03,0);
                write_eeprom(0x04,0);
                write_eeprom(0x13,0);
                write_eeprom(0x14,0);
                write_eeprom(0x00,0);
                write_eeprom(0x20,0);
            }

            if(modo==6){            //envia los ultimos datos //introducidos por el usuario
                envia[0]=read_eeprom(0x06);
                envia[1]=read_eeprom(0x07);
                envia[2]=read_eeprom(0x16);
                envia[3]=read_eeprom(0x17);
                envia[4]=read_eeprom(0x26);
                envia[5]=read_eeprom(0x27);
                envia[6]=read_eeprom(0x25);
                usb_put_packet(1, envia, 7, USB_DTS_TOGGLE);
            }

        }
        resetear();
        peticion();
    }

 }
}

void modo1(){
        envia[0]=1;
        envia[1]=1;
        envia[2]=1;
        envia[3]=1;
        envia[4]=1;
        envia[5]=1;
        envia[6]=1;
return;
}

void peticion() {       //comprueba si alguien ha subido a
        int8 entrada=0;         //alguna báscula (salida o entrada)
        int8 salida=0;          //guarda ese peso y comprueba si el
        int8 bascula=0;         //usuario cumple las condiciones
        int8 pentrada=0;        //para entrar al local. En caso que
        int8 psalida=0;         //salga se decrementa el peso y el
        int1 altura=0;          //aforo
        int1 peso=0;
        int1 aforo=0;
        int8 p1;
        int8 p2;
        p1=read_eeprom(0x16);
```

```
p2=read_eeprom(0x17);
if(p1==0 && p2==0) return;
bascula=bascula_on();
if(bascula==1){              //entrada de persona al local
        envia[6]=3;
        usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        usb_get_packet(1,recibe,8);
        while(modo!=3){
        usb_get_packet(1,recibe,8);
        usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        }
        output_d(0x02);
        altura=sensor_altura();
output_d(0x00);
        if(altura==1){
        pentrada=sensor_peso_entrada();
                if (pentrada==0) return;
                peso=suma_peso();
                if(peso==1){
                        aforo=sumar_aforo();
                        if(aforo==1){
                                output_d(0x80);
                                entrada=torno();
                                if(entrada==1){
                                        delay_ms(2000);
                                        output_d(0x00);
                                        return;
                                }
                                else{
                                output_d(0x01);
                                delay_ms(2000);
                                output_d(0x00);
                                restar_peso();
                                restar_aforo();
                                return;
                                }
                        }
                        else{                                   //exceso de aforo.
                        output_d(0x08);
                                delay_ms(2000);
                                output_d(0x00);
                                goto solo_salida;
                        }

                }
                else{
                        output_d(0x04);             //exceso del peso
        delay_ms(2000);
                        output_d(0x00);
                        goto solo_salida;;
                }
        }
        else{                           //altura máxima/minima superada
        output_d(0x10);
                delay_ms(2000);
                output_d(0x00);
                return;
        }

}
```

```
else if(bascula==2){                    //Salida de personas
        envia[6]=4;
        usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        usb_get_packet(1, recibe, 8);
        while(modo!=4){
                usb_get_packet(1, recibe, 8);
                usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        }
        psalida=sensor_peso_salida();
        if(psalida==0) return;
        restar_peso();
        restar_aforo();
        output_d(0x40);
        salida=torno2();
        if(salida==1){
                delay_ms(2000);
                output_d(0x00);
                return;
        }
        else{
                output_d(0x00);
                output_bit(PIN_B5,1);
                delay_ms(2000);
                output_bit(PIN_B5,0);
                suma_peso();
                sumar_aforo();
                return;
        }
}
else{
        return;
}

solo_salida:                    //si el local supera el aforo o el peso //total solo se admite salida
        envia[6]=5;         //señales para indicar que sólo puede salir
        usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        usb_get_packet(1, recibe, 8); //se permite salida
        while(modo!=5){
                usb_get_packet(1, recibe, 8);
                usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        }
        while(bascula!=2){bascula=bascula_on();}
        envia[6]=4;
        usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        usb_get_packet(1, recibe, 8);
        while(modo!=4){
                usb_get_packet(1, recibe, 8);
                usb_put_packet(1,envia,7,USB_DTS_TOGGLE);
        }
        sensor_peso_salida();
        restar_peso();
        restar_aforo();
        output_d(0x40);
        salida=torno2();
        if(salida==1){
                delay_ms(2000);
                output_d(0x00);
                return;
        }
        else{
```

```
                        output_d(0x00);
                        output_bit(PIN_B5,1);
                        delay_ms(2000);
                        output_bit(PIN_B5,0);
                        suma_peso();
                        sumar_aforo();
                        goto solo_salida;
                    }
    }
```

Librería.c

```
#DEVICE         ADC=8
#use delay (clock=4M)

int sensor_altura(){
        int8 valor=0;
        int8 altura_minima=0;
        int8 altura_maxima=0;
        int8 altura_l=0;
        int8 altura_h=0;
        int16 altura=0;
        int8 altura_sensor=0;
        int8 altura_ac_l=0;
        int8 altura_ac_h=0;
        int8 contador=15;ç

        double altura_acumulada=0;
        write_eeprom(0x30,0);
        write_eeprom(0x31,0);
        while(contador>0){
                output_bit(PIN_D5,1);
                delay_us(15);
                output_bit(PIN_D5,0);
                valor=input_state(PIN_A2);
                while(valor==0){valor=input_state(PIN_A2);}
                set_timer0(0);
                valor=input_state(PIN_A2);
                while(valor==1){valor=input_state(PIN_A2);}
                altura=get_timer0();
                altura_sensor=altura/58;
                altura=read_eeprom(0x25);
                altura_l=altura-altura_sensor;
                write_eeprom(0x20,altura_l);
                altura_ac_l=read_eeprom(0x30);
                altura_ac_h=read_eeprom(0x31);
                altura_acumulada=altura_ac_h*256+altura_ac_l;
                altura=read_eeprom(0x20);
                altura_acumulada+=altura;
                if(altura_acumulada<255){
                        altura_ac_l=altura_acumulada;
                        write_eeprom(0x30,altura_ac_l);
                        contador--;
                        delay_ms(55);
                }
                else{
                        altura_ac_h=altura_acumulada/256;
                        altura_ac_l=altura_acumulada - (256*altura_ac_h);
                        write_eeprom(0x30,altura_ac_l);
                        write_eeprom(0x31,altura_ac_h);
                        contador--;
                        delay_ms(55);
            }
        }
        altura_ac_l=read_eeprom(0x30);
        altura_ac_h=read_eeprom(0x31);
        altura_acumulada=(altura_ac_h*256)+altura_ac_l;
        altura_acumulada/=15;
        write_eeprom(altura_acumulada,0x20);
        altura_minima=read_eeprom(0x26);
```

```
	altura_maxima=read_eeprom(0x27);
	if(altura_acumulada>altura_maxima){return 0;}
	else if(altura_acumulada<altura_minima){return 0;}
	else
	return 1;
}

int bascula_on(){
	int8 peso=0;
	set_adc_channel(0);
	peso=read_adc();
	if(peso>10){
		write_eeprom(0x12,peso);
		return 2;
	}
	set_adc_channel(1);
	peso=read_adc();
	if(peso>10){
		write_eeprom(0x12,peso);
		return 1;
	}
	else return 0;
}

int sensor_peso_entrada(){
	int8 contador=0;
	int8 peso=0;
	int8 ult_peso=0;		//primer peso que se mide para
	int8 aux=0;				//saber si la báscula está on
	int8 peso_actual_l=0;
	int8 peso_actual_h=0;
	double peso_total_actual=0;

	write_eeprom(0x32,0);
	write_eeprom(0x33,0);
	set_adc_channel(1);

cuenta:
	if(contador !=15){
		peso=read_adc();
		if(peso<10) return 0;
		ult_peso=read_eeprom(0x12);
		ult_peso=peso-ult_peso;
		if(ult_peso>=2){
			write_eeprom(0x12,peso);
			delay_ms(150);
			goto cuenta;
		}
		write_eeprom(0x12,peso);
		peso_actual_l=read_eeprom(0x32);
		peso_actual_h=read_eeprom(0x33);
		peso_total_actual=(peso_actual_h*256)+peso_actual_l;
		peso_total_actual=peso_total_actual+peso;
		peso_actual_h=peso_total_actual/256;
		peso_actual_l=peso_total_actual-(256*peso_actual_h);
		write_eeprom(0x32,peso_actual_l);
		write_eeprom(0x33,peso_actual_h);
		contador++;
		delay_ms(150);
		goto cuenta;
```

```
        }
        peso_actual_l=read_eeprom(0x32);
        peso_actual_h=read_eeprom(0x33);
        peso_total_actual=(peso_actual_h*256)+peso_actual_l;
        peso_total_actual/=15;
        aux=peso_total_actual;
        write_eeprom(0x00,aux);
}

int sensor_peso_salida(){
        int8 contador=0;
        int8 peso=0;
        int8 aux=0;
        int8 ult_peso=0;
        int8 peso_actual_l=0;
        int8 peso_actual_h=0;
        double peso_total_actual=0;

        write_eeprom(0x32,0);
        write_eeprom(0x33,0);
        set_adc_channel(0);

cuenta:
        if(contador !=15){
                peso=read_adc();
                if(peso<10) return 0;
                ult_peso=read_eeprom(0x12);
                ult_peso=peso-ult_peso;
                if(ult_peso>=2){
                        write_eeprom(0x12,peso);
                        delay_ms(150);
                        goto cuenta;
                }
        write_eeprom(0x12,peso);
        peso_actual_l=read_eeprom(0x32);
        peso_actual_h=read_eeprom(0x33);
        peso_total_actual=(peso_actual_h*256)+peso_actual_l;
        peso_total_actual=peso_total_actual+peso;
        peso_actual_h=peso_total_actual/256;
        peso_actual_l=peso_total_actual-(256*peso_actual_h);
        write_eeprom(0x32,peso_actual_l);
        write_eeprom(0x33,peso_actual_h);
        contador++;
        delay_ms(150);
        goto cuenta;
        }
        peso_actual_l=read_eeprom(0x32);
        peso_actual_h=read_eeprom(0x33);
        peso_total_actual=(peso_actual_h*256)+peso_actual_l;
        peso_total_actual/=15;
        aux=peso_total_actual;
        write_eeprom(0x00,aux);
}

int suma_peso(){
        double peso_total=0;
        int8 peso=0;
        int8 peso_total_l=0;
        int8 peso_total_h=0;
        int8 peso_actual_l=0;
```

```
	int8 peso_actual_h=0;
	double peso_total_actual=0;

	peso=read_eeprom(0x00);
	peso_total_l=read_eeprom(0x16);
	peso_total_h=read_eeprom(0x17);
	peso_total=(peso_total_h*256)+peso_total_l;
	peso_actual_l=read_eeprom(0x13);
	peso_actual_h=read_eeprom(0x14);
	peso_total_actual=(peso_actual_h*256)+peso_actual_l;
	peso_total_actual=peso_total_actual+peso;
	peso_total=peso_total-peso_total_actual;
		if(peso_total<0)
			return 0;
		else{
			peso_actual_h=peso_total_actual/256;
		peso_actual_l=peso_total_actual(256*peso_actual_h);
			write_eeprom(0x13,peso_actual_l);
			write_eeprom(0x14,peso_actual_h);
			return 1;
		}

}


int sumar_aforo(){
	double personas_totales=0;
	double personas_actuales=0;
	int8 personas_actuales_l=0;
	int8 personas_actuales_h=0;
	int8 personas_totales_l=0;
	int8 personas_totales_h=0;

	personas_actuales_l=read_eeprom(0x03);
	personas_actuales_h=read_eeprom(0x04);
personas_actuales=personas_actuales_l+(256*personas_actuales_h);
	personas_actuales=personas_actuales+1;
	personas_totales_l=read_eeprom(0x06);
	personas_totales_h=read_eeprom(0x07);
	personas_totales=personas_totales_l+(256*personas_totales_h);
	personas_totales=personas_totales-personas_actuales;
	if(personas_totales<0){return 0;}
	else{
		personas_actuales_h=personas_actuales/256;
personas_actuales_l=personas_actuales(256*personas_actuales_h);
		write_eeprom(0x03,personas_actuales_l);
		write_eeprom(0x04,personas_actuales_h);
		return 1;
	}
}

void restar_peso(){
	int8 peso;
	int8 peso1;
	int8 peso2;
	double peso_acumulado;

	peso=read_eeprom(0x00);
	peso1=read_eeprom(0x13);
	peso2=read_eeprom(0x14);
```

```
		peso_acumulado=(peso2*256)+peso1;
		peso_acumulado=peso_acumulado-peso;
		if(peso_acumulado<0){
			write_eeprom(0x13,0);
			write_eeprom(0x14,0);
		}
		else{
			peso2=peso_acumulado/256;
			peso1=peso_acumulado-(256*peso2);
			write_eeprom(0x13,peso1);
			write_eeprom(0x14,peso2);
		}
}

void restar_aforo(){
		int8 aforo1=0;
		int8 aforo2=0;
		double aforo_acumulado=0;

		aforo1=read_eeprom(0x03);
		aforo2=read_eeprom(0x04);
		aforo_acumulado=(aforo2*256)+aforo1;
		aforo_acumulado=aforo_acumulado-1;
		if(aforo_acumulado<0){
			write_eeprom(0x03,0);
			write_eeprom(0x04,0);
		}
		else{
			aforo2=aforo_acumulado/256;
			aforo1=aforo_acumulado-(256*aforo2);
			write_eeprom(0x03,aforo1);
			write_eeprom(0x04,aforo2);
		}
}

int torno(){
		int8 segundos=0;

		output_bit(PIN_A3,1);

esperando:
		if(input_state(PIN_B6)==1){
			segundos++;
			if(segundos==4){
				output_bit(PIN_A3,0);
				return 0;
			}
			else{
				delay_ms(1000);
				goto esperando;
			}
		}
		else{
			output_bit(PIN_A3,0);
			return 1;
		}
}


int torno2(){
```

```
	int8 segundos=0;

	output_bit(PIN_A4,1);

esperando:
	if(input_state(PIN_B7)==1){
		segundos++;
		if(segundos==4){
			output_bit(PIN_A4,0);
			return 0;
		}
		else{
			delay_ms(1000);
			goto esperando;
		}
	}
	else{
		output_bit(PIN_A4,0);
		return 1;
	}
}

void resetear(){
	int8 af1;
	int8 af2;

	af1=read_eeprom(0x03);
	af2=read_eeprom(0x04);
	if(af1==0 && af2==0){
		write_eeprom(0x13,0);
		write_eeprom(0x14,0);
	}
	else return;
}
```

6.2. Código de la interfaz gráfica

Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Runtime.InteropServices; // Clase para importar DLL

using PVOID = System.IntPtr;
using DWORD = System.UInt32;

namespace PicUSB
{
   unsafe public class PicUSBAPI
   {
      #region Definición de los Strings: EndPoint y VID_PID
      string vid_pid_norm = "vid_04d8&pid_0011";

      string out_pipe = "\\MCHP_EP1";
      string in_pipe = "\\MCHP_EP1";
      #endregion

      #region Funciones importadas de la DLL: mpusbapi.dll
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBGetDLLVersion();
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBGetDeviceCount(string pVID_PID);
      [DllImport("mpusbapi.dll")]
      private static extern void* _MPUSBOpen(DWORD instance, string pVID_PID, string pEP, DWORD
dwDir, DWORD dwReserved);
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBRead(void* handle, void* pData, DWORD dwLen, DWORD*
pLength, DWORD dwMilliseconds);
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBWrite(void* handle, void* pData, DWORD dwLen, DWORD*
pLength, DWORD dwMilliseconds);
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBReadInt(void* handle, DWORD* pData, DWORD dwLen,
DWORD* pLength, DWORD dwMilliseconds);
      [DllImport("mpusbapi.dll")]
      private static extern bool _MPUSBClose(void* handle);
      #endregion

      void* myOutPipe;
      void* myInPipe;
      uint peso1;
      uint peso2;
      uint aforo1;
      uint aforo2;
      uint altura;
      uint ult_peso;
      uint modo;
      uint pes_loc;
      uint afo_loc;
      static void Main()
      {
         Application.EnableVisualStyles();
```

```
        Application.Run(new PicUSB());
    }

    public void OpenPipes()
    {
        DWORD selection = 0;

        myOutPipe = _MPUSBOpen(selection, vid_pid_norm, out_pipe, 0, 0);
        myInPipe = _MPUSBOpen(selection, vid_pid_norm, in_pipe, 1, 0);
    }

    public void ClosePipes()
    {
        _MPUSBClose(myOutPipe);
        _MPUSBClose(myInPipe);
    }
    public DWORD dispositivo()
    {
        DWORD dis;
        dis = _MPUSBGetDeviceCount(vid_pid_norm);
        return dis;
    }
    private void SendPacket(byte* SendData, DWORD SendLength)
    {
        uint SendDelay = 1000;

        DWORD SentDataLength;

        OpenPipes();
        _MPUSBWrite(myOutPipe, (void*)SendData, SendLength, &SentDataLength, SendDelay);
        ClosePipes();
    }

    private void ReceivePacket(byte* ReceiveData, DWORD* ReceiveLength)
    {
        uint ReceiveDelay = 1000;

        DWORD ExpectedReceiveLength = *ReceiveLength;

        OpenPipes();
        _MPUSBRead(myInPipe, (void*)ReceiveData, ExpectedReceiveLength, ReceiveLength,
ReceiveDelay);
        ClosePipes();
    }

    public void RecibirDato()
    {
        byte* send_buf = stackalloc byte[1];

        send_buf[0] = 1;
        SendPacket(send_buf, 1);
    }
    public void Entrar()
    {
        byte* send_buf = stackalloc byte[1];

        send_buf[0] = 3;
        SendPacket(send_buf, 1);
    }
    public void Salir()
```

```
{
   byte* send_buf = stackalloc byte[1];

   send_buf[0] = 4;
   SendPacket(send_buf, 1);
}
public void Solosalir()
{
   byte* send_buf = stackalloc byte[1];

   send_buf[0] = 5;
   SendPacket(send_buf, 1);
}
public void valores()
{
   byte* send_buf = stackalloc byte[1];

   send_buf[0] = 6;
   SendPacket(send_buf, 1);
}
public void Resetear()
{
   byte* send_buf = stackalloc byte[1];

   send_buf[0] = 2;
   SendPacket(send_buf, 1);
}

public uint ValorGuardar1()
{
   uint peso;
   byte* receive_buf = stackalloc byte[7];

   DWORD RecvLength = 7;

   ReceivePacket(receive_buf, &RecvLength);
   aforo1 = receive_buf[0];
   aforo2 = receive_buf[1];
   peso1 = receive_buf[2];
   peso2 = receive_buf[3];
   ult_peso = receive_buf[4];
   altura = receive_buf[5];
   modo = receive_buf[6];
   peso = (256 * peso2) + peso1;
   pes_loc = peso;
   return peso;
}
public uint ValorGuardar2()
{
   uint aforo;
   aforo = (256 * aforo2) + aforo1;
   afo_loc = aforo;

   return aforo;
}
public uint ValorGuardar3()
{
   return ult_peso;
}
public uint ValorGuardar4()
```

```
    {
        return altura;
    }
    public uint ValorGuardar5()
    {
        return modo;
    }
    public uint EnviarDatos(uint dato,uint dato2, uint dato3, uint dato4, uint dato5)
    {
        uint valor_h_p;
        uint valor_l_p;
        uint valor_h_a;
        uint valor_l_a;
        uint aforo_recomendado;

       if(dato4<1000 || dato4>65535){
          MessageBox.Show("El peso total debe estar entre 1000 kg. y 65535 kg.");
          return 0;
       }
        aforo_recomendado=dato4/80;
        if(dato5>aforo_recomendado){
           MessageBox.Show("El aforo máximo recomendado es "+aforo_recomendado);
           return 0;
        }
        if(dato>255){
           MessageBox.Show("La altura máxima del sensor debe ser de 255 cm.");
           return 0;
        }
        if(dato2>dato||dato3>dato){
           MessageBox.Show("Una de las alturas (máxima o mínima) son superiores a la altura del sensor");
           return 0;
        }
        if(dato2<dato3){
           MessageBox.Show("La altura (máxima o mínima) tiene un valor erróneo");
           return 0;
        }
           byte* send_buf2 = stackalloc byte[8];

           valor_h_a = dato5 / 256;
           valor_l_a = dato5 - (valor_h_a * 256);
           valor_h_p = dato4 / 256;
           valor_l_p = dato4 - (valor_h_p * 256);

           send_buf2[0] = 0;
           send_buf2[1] = (byte)dato;
           send_buf2[2] = (byte)dato2;
           send_buf2[3] = (byte)dato3;
           send_buf2[4] = (byte)valor_l_p;
           send_buf2[5] = (byte)valor_h_p;
           send_buf2[6] = (byte)valor_l_a;
           send_buf2[7] = (byte)valor_h_a;
           SendPacket(send_buf2, 8);
           return 1;
       }
    }

}
```

PicUSB.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace PicUSB
{
   public partial class PicUSB : Form
   {
      PicUSBAPI usbapi = new PicUSBAPI();
      uint afo=0; //aforo actual del local
      uint a;     //variable usada para ver si está conectado el USB
      public PicUSB()
      {
         InitializeComponent();
      }
      //envía los datos cuando se clickea el botón
      private void button2_Click(object sender, EventArgs e)
      {
         uint aux;
         uint valores;

            if (a == 0)
            {
               MessageBox.Show("Imposible transmitir datos. USB desconectado");
               return;
            }
            else
            {
               timer1.Enabled = false;
               timer2.Enabled = false;
               if (afo == 0)
               {
                  try
                  {
                     toolStripStatusLabel1.Text = "Enviando datos...";
                     toolStripStatusLabel1.Enabled = true;
                     toolStripStatusLabel1.Visible = true;
                     valores=usbapi.EnviarDatos(uint.Parse(textBox3.Text), uint.Parse(textBox4.Text),
uint.Parse(textBox5.Text), uint.Parse(textBox6.Text), uint.Parse(textBox7.Text));
                     if (valores == 0) return;
                     usbapi.ValorGuardar1();
                     aux = usbapi.ValorGuardar5();
                     while (aux != 1)
                     {
                      usbapi.EnviarDatos(uint.Parse(textBox3.Text), uint.Parse(textBox4.Text),
uint.Parse(textBox5.Text), uint.Parse(textBox6.Text), uint.Parse(textBox7.Text)); toolStripStatusLabel1.Text
= "Datos enviados";
                        usbapi.ValorGuardar1();
                        aux = usbapi.ValorGuardar5();
                     }
```

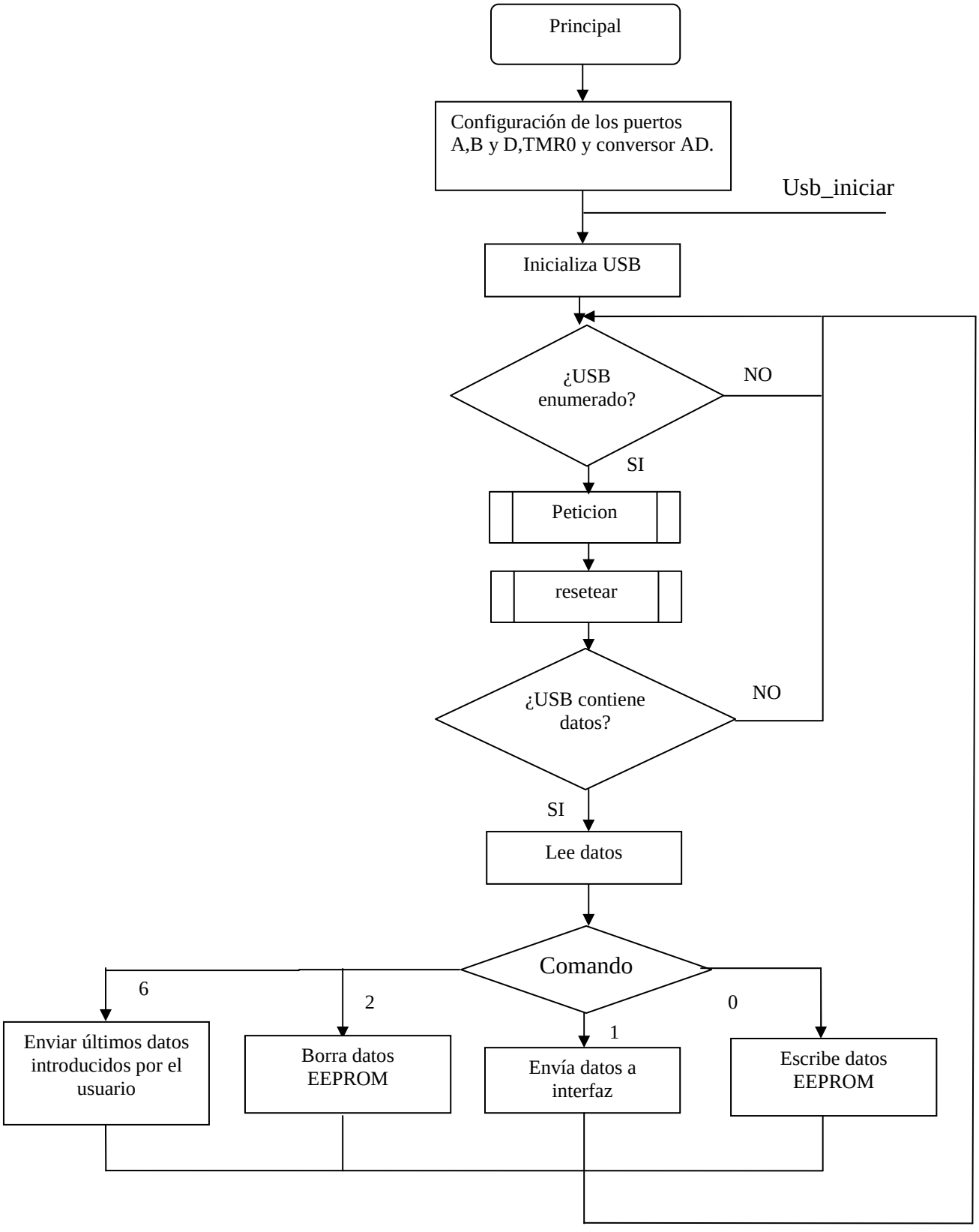
```
                toolStripStatusLabel1.Text = "Datos enviados";
                timer1.Enabled = true;
                timer2.Enabled = true;
                return;
            }
            catch
            {
                MessageBox.Show("Introduzca valores numéricos.");
            }
        }
        else
        {
            MessageBox.Show("Imposible cambiar valores. Hay personas dentro");
            timer2.Enabled = true;
            timer1.Enabled = true;
            return;

        }
    }
}
//timer para actualizar datos
private void timer1_Tick(object sender, EventArgs e)
{
    uint var;
    uint var1;
    uint var2;
    uint var3;
    uint var4;

    usbapi.RecibirDato();
    var = usbapi.ValorGuardar1();
    var1 = usbapi.ValorGuardar2();
    var2 = usbapi.ValorGuardar3();
    var3 = usbapi.ValorGuardar4();
    var4 = usbapi.ValorGuardar5();
    afo = var;
    if (var4 == 2)
    {
        timer2.Enabled = false;
        toolStripStatusLabel1.Text = "Actualizando datos...";
        toolStripStatusLabel1.Enabled = true;
        toolStripStatusLabel1.Visible = true;
        textBox1.Text = (var).ToString();
        textBox2.Text = (var1).ToString();
        textBox8.Text = (var2).ToString();
        textBox9.Text = (var3).ToString();
        timer2.Enabled = true;
        return;
    }
    return;

}

//ventana de ayuda
private void verAyudaToolStripMenuItem_Click(object sender, EventArgs e)
{
    Ayuda ventana = new Ayuda();
    ventana.Show();
}
//cambia a la pestaña de "valores actuales"
```

```
        private void valoresActualesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            cambiarValoresToolStripMenuItem.Checked = false;
            valoresActualesToolStripMenuItem.Checked = true;
            tabControl1.SelectedTab = tabPage1;
        }
        //cambia a la pestaña de "cambiar valores"
        private void cambiarValoresToolStripMenuItem_Click(object sender, EventArgs e)
        {
            valoresActualesToolStripMenuItem.Checked = false;
            cambiarValoresToolStripMenuItem.Checked=true;
            tabControl1.SelectedTab = tabPage2;
        }
        //envía una señal que resetea los valores del PIC
        private void resetearToolStripMenuItem_Click(object sender, EventArgs e)
        {
          DialogResult r= MessageBox.Show("¿Está seguro que quiere borrar los valores?", "Resetar",
MessageBoxButtons.OKCancel);
         if (r == System.Windows.Forms.DialogResult.OK)
             usbapi.Resetear();
            return;
        }
        //recibe las señales del PIC y cambia las etiquetas de la
        //barra de estado. Además comprueba si está conectado el
        //USB
        private void timer2_Tick(object sender, EventArgs e)
        {
            uint entra;
            a = usbapi.dispositivo();
            if (a == 0)
            {
                toolStripStatusLabel2.Enabled = true;
                toolStripStatusLabel2.Visible = true;
                toolStripStatusLabel2.Text = "USB desconectado";
                toolStripStatusLabel1.Enabled = false;
                toolStripStatusLabel1.Visible = false;
                return;
            }
            else
            {
                toolStripStatusLabel1.Enabled = true;
                toolStripStatusLabel1.Visible = true;
                toolStripStatusLabel2.Visible = false;
                toolStripStatusLabel2.Enabled = false;
                toolStripStatusLabel3.Enabled = true;
                toolStripStatusLabel3.Visible = true;
               toolStripStatusLabel3.Text = "USB conectado";
                usbapi.ValorGuardar1();
                entra = usbapi.ValorGuardar5();
                if (entra == 3)
                {
                    usbapi.Entrar();
                    toolStripStatusLabel1.Enabled = true;
                    toolStripStatusLabel1.Visible = true;
                    toolStripStatusLabel1.Text = "Entrando persona...";
                    return;
                }
                else if (entra == 4)
                {
                    usbapi.Salir();
```

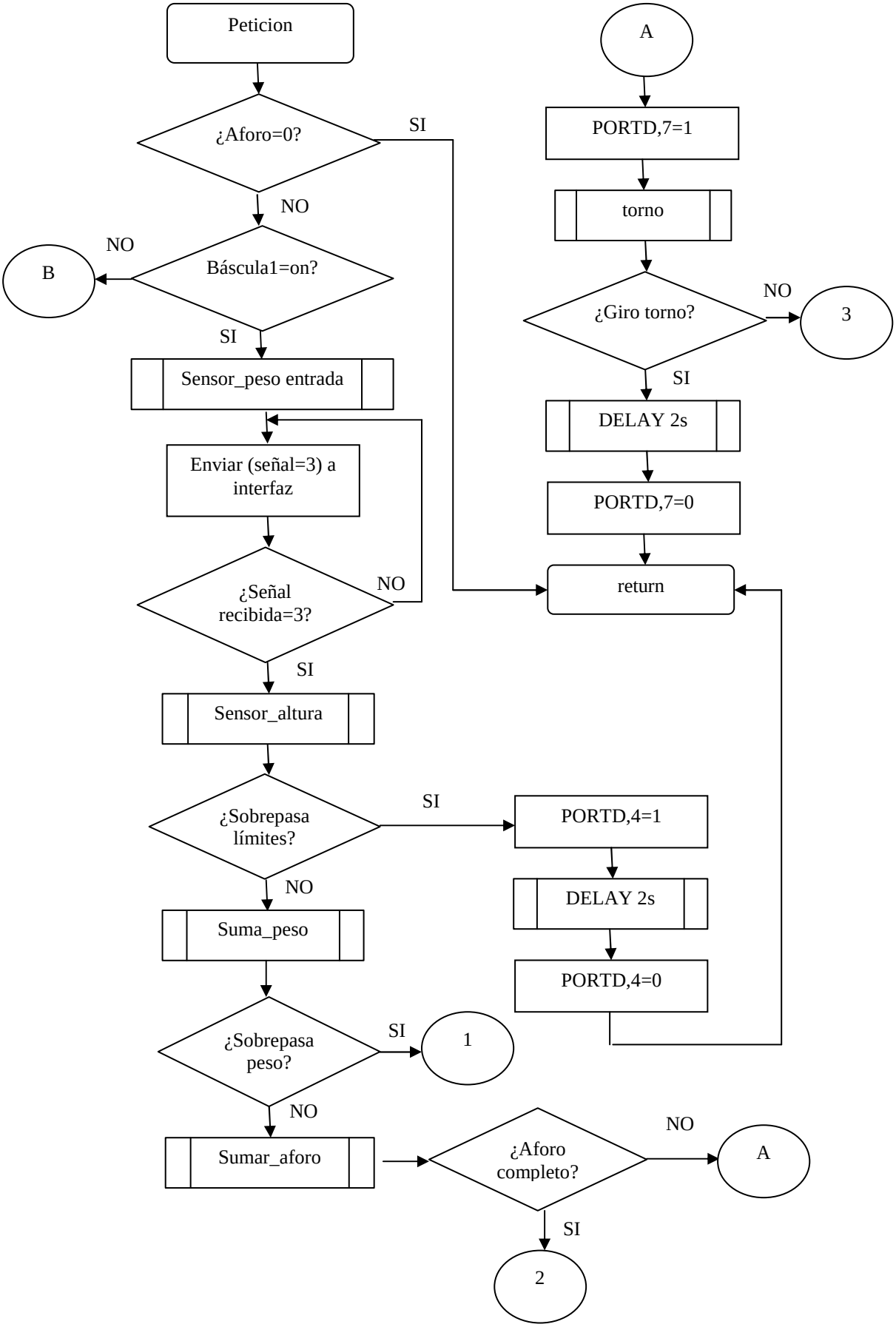
```
                toolStripStatusLabel1.Enabled = true;
                toolStripStatusLabel1.Visible = true;
                toolStripStatusLabel1.Text = "Saliendo persona...";
                return;
            }
            else if (entra == 5)
            {
                usbapi.Solosalir();
                toolStripStatusLabel1.Enabled = true;
                toolStripStatusLabel1.Visible = true;
                toolStripStatusLabel1.Text = "Limite peso/aforo alcanzado.SOLO SALIDA";
                return;
            }
            else
                return;
        }
    }
    //cierra la aplicación
    private void salirToolStripMenuItem_Click_2(object sender, EventArgs e)
    {
        Close();
    }
    //muestra los últimos datos introducidos por el usuario
    private void informaciónToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Informacion ventana = new Informacion();
        ventana.Show();

    }

  }
}
```

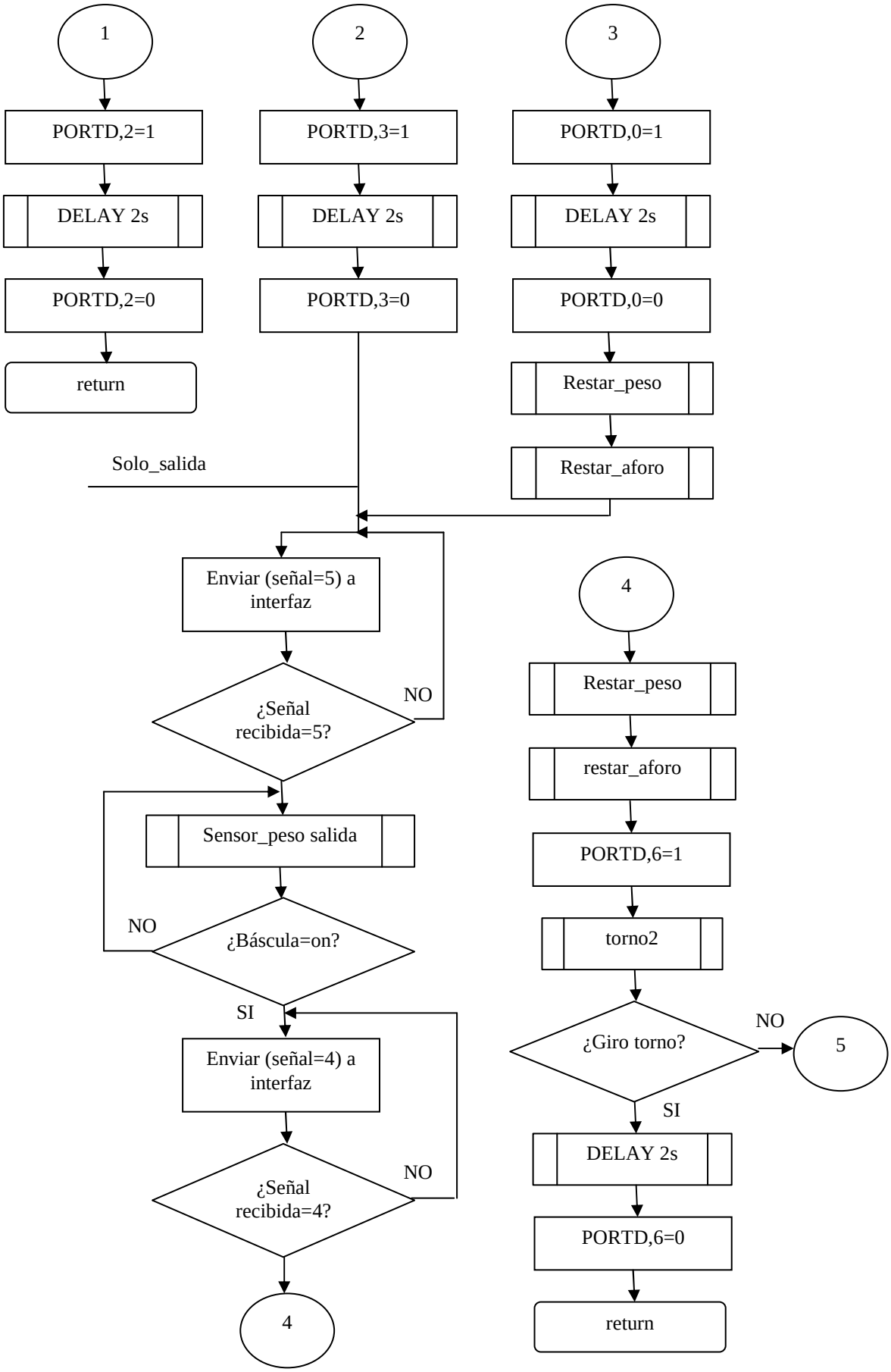
6.3. Flujograma PIC

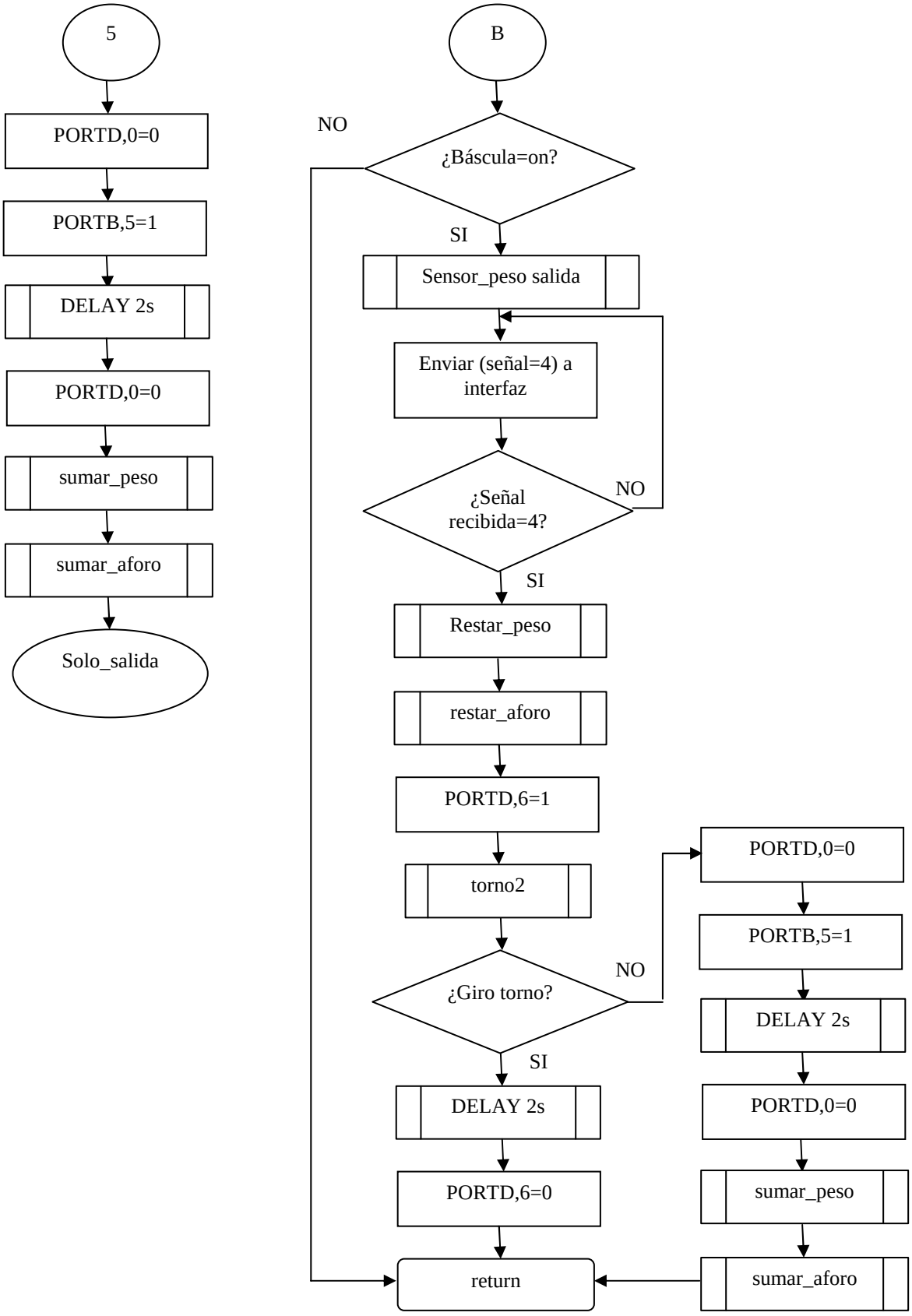
Programa principal e inicio del USB:



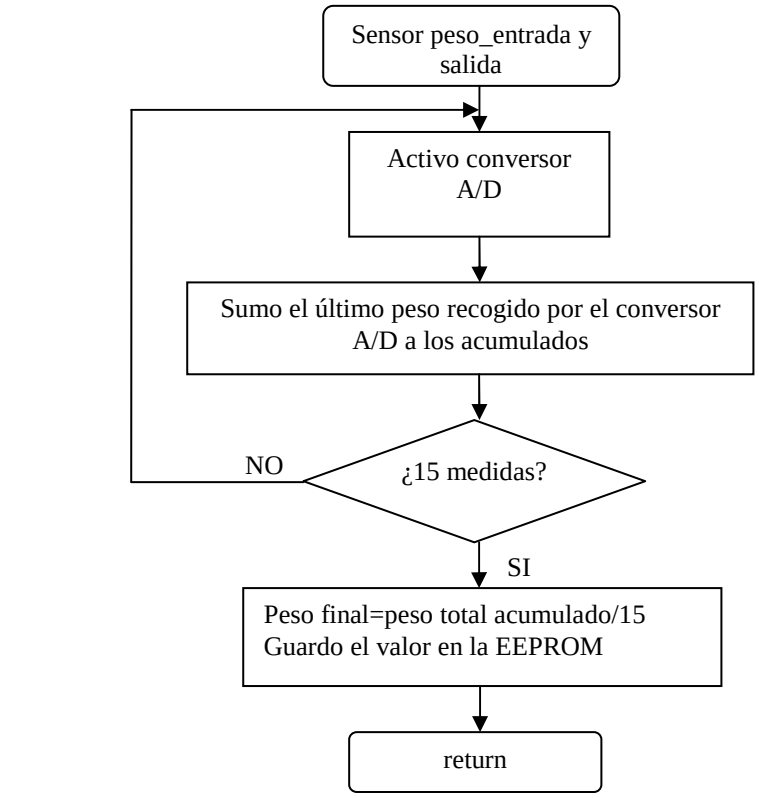
Función petición que tramita las entradas y salidas de personas:

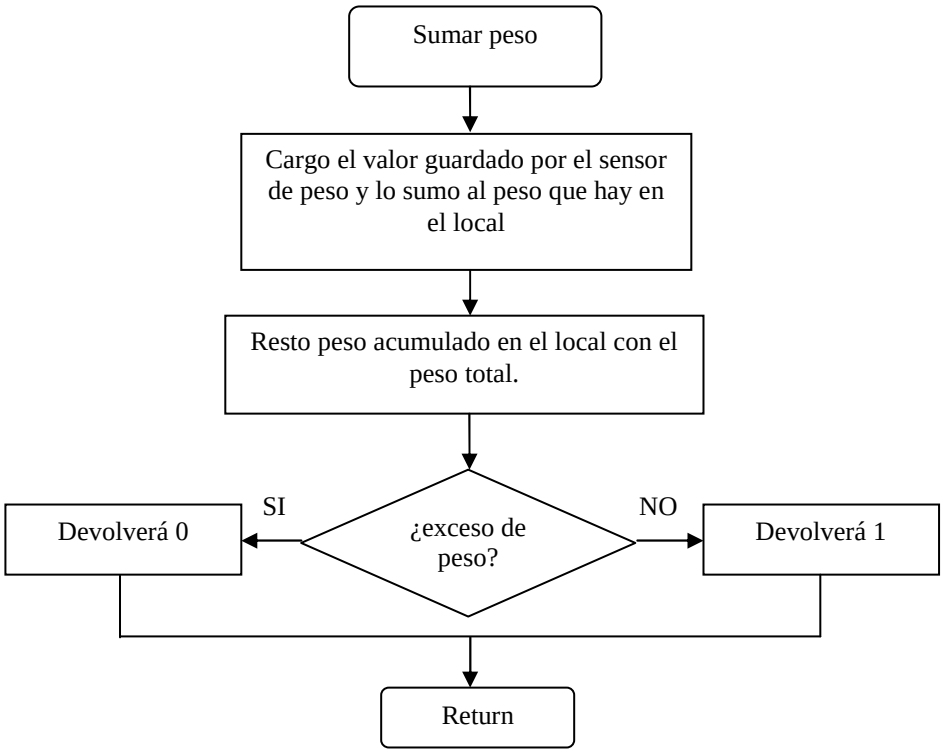


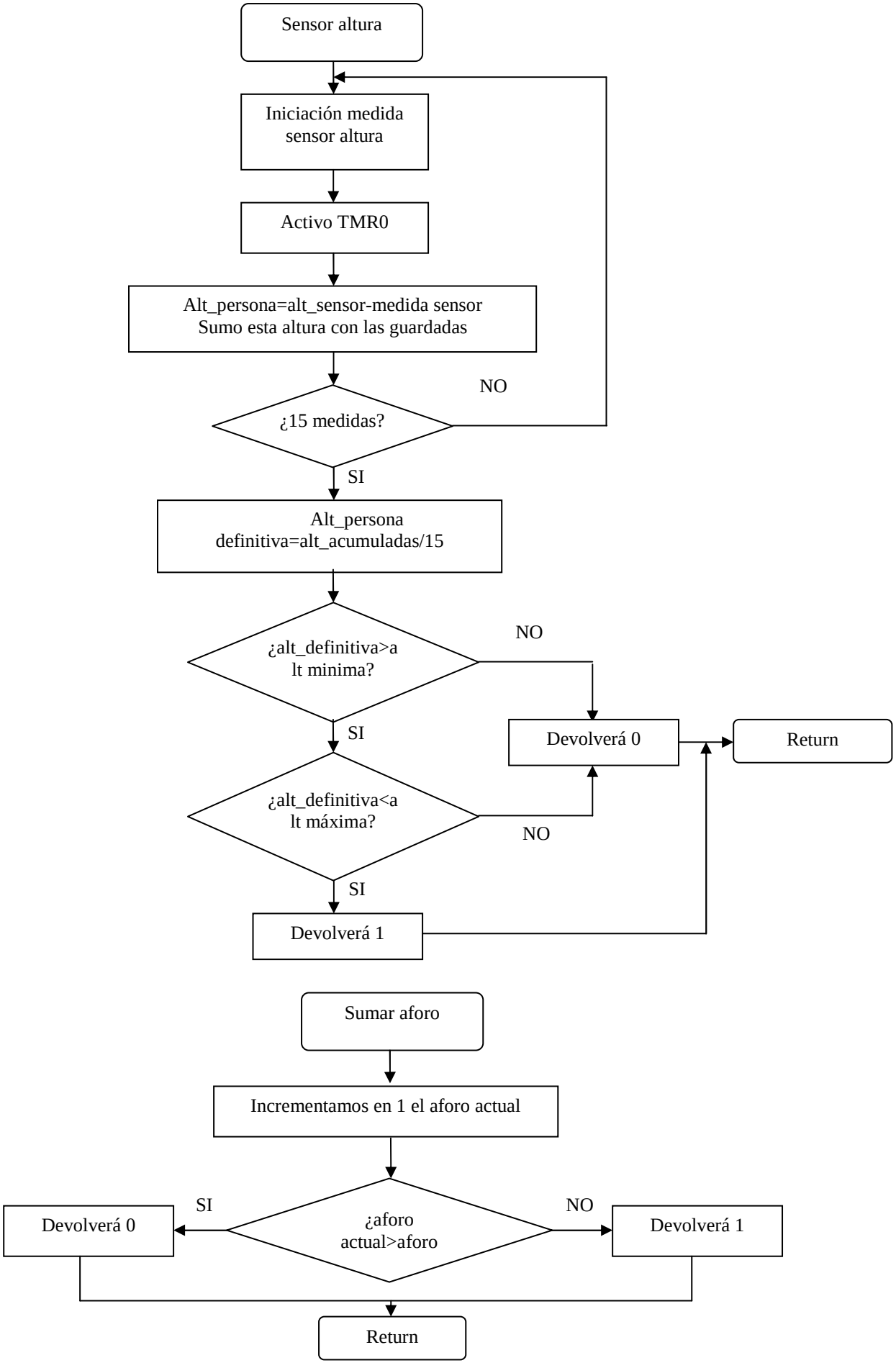

1
PORTD,2=1
DELAY 2s
PORTD,2=0
return
2
PORTD,3=1
DELAY 2s
PORTD,3=0
3
PORTD,0=1
DELAY 2s
PORTD,0=0
Restar_peso
Restar_aforo
Solo_salida
Enviar (señal=5) a interfaz
¿Señal recibida=5?
NO
Sensor_peso salida
¿Báscula=on?
NO
SI
Enviar (señal=4) a interfaz
¿Señal recibida=4?
NO
4
4
Restar_peso
restar_aforo
PORTD,6=1
torno2
¿Giro torno?
NO
5
SI
DELAY 2s
PORTD,6=0
return


5
PORTD,0=0
PORTB,5=1
DELAY 2s
PORTD,0=0
sumar_peso
sumar_aforo
Solo_salida
B
NO
¿Báscula=on?
SI
Sensor_peso salida
Enviar (señal=4) a interfaz
¿Señal recibida=4?
NO
SI
Restar_peso
restar_aforo
PORTD,6=1
torno2
¿Giro torno?
NO
SI
DELAY 2s
PORTD,6=0
PORTD,0=0
PORTB,5=1
DELAY 2s
PORTD,0=0
sumar_peso
sumar_aforo
return

Los siguientes flujogramas son los de la librería.h.

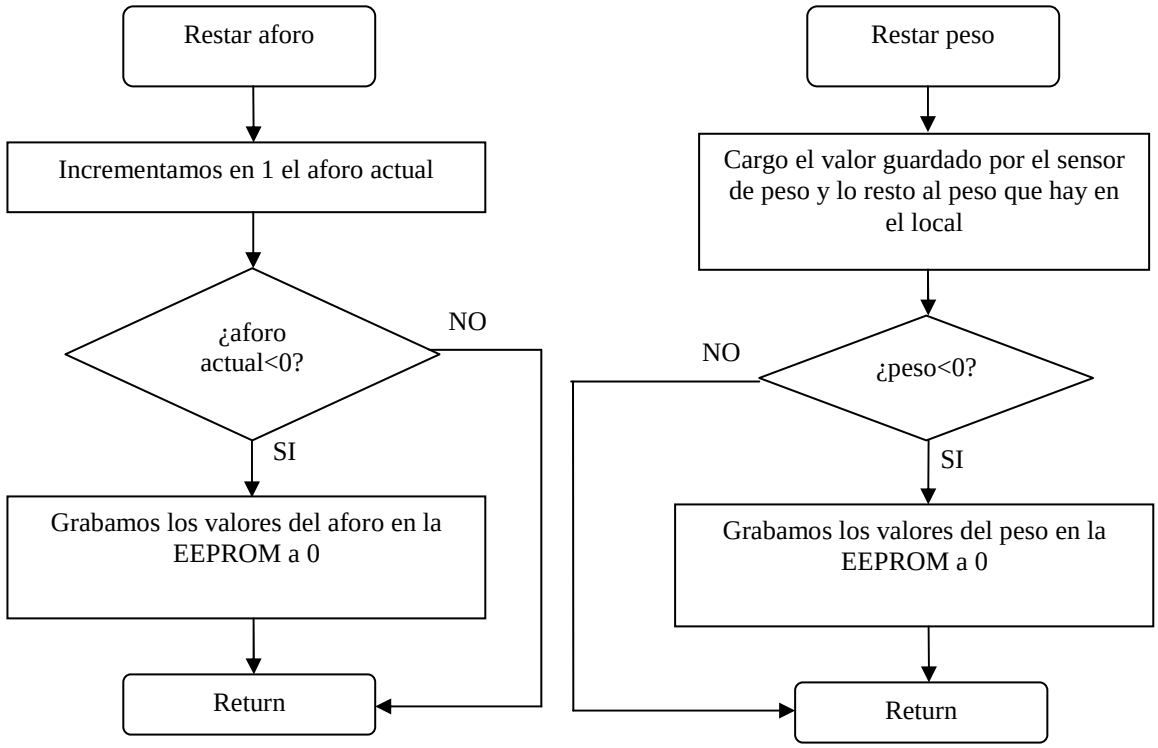


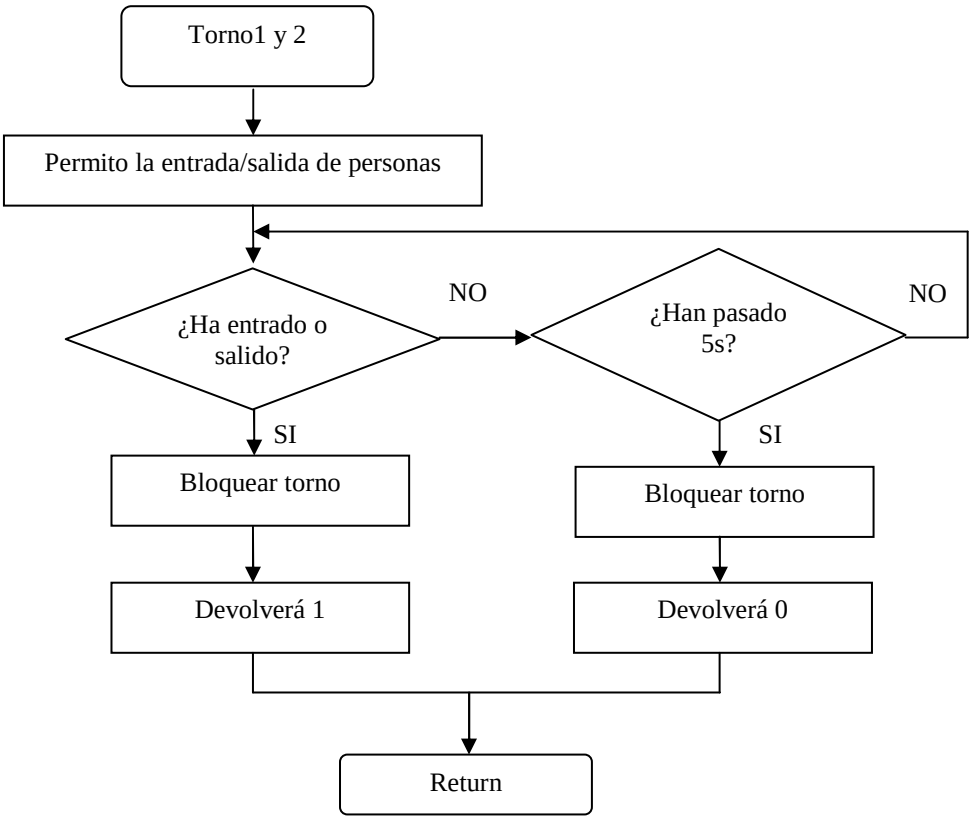


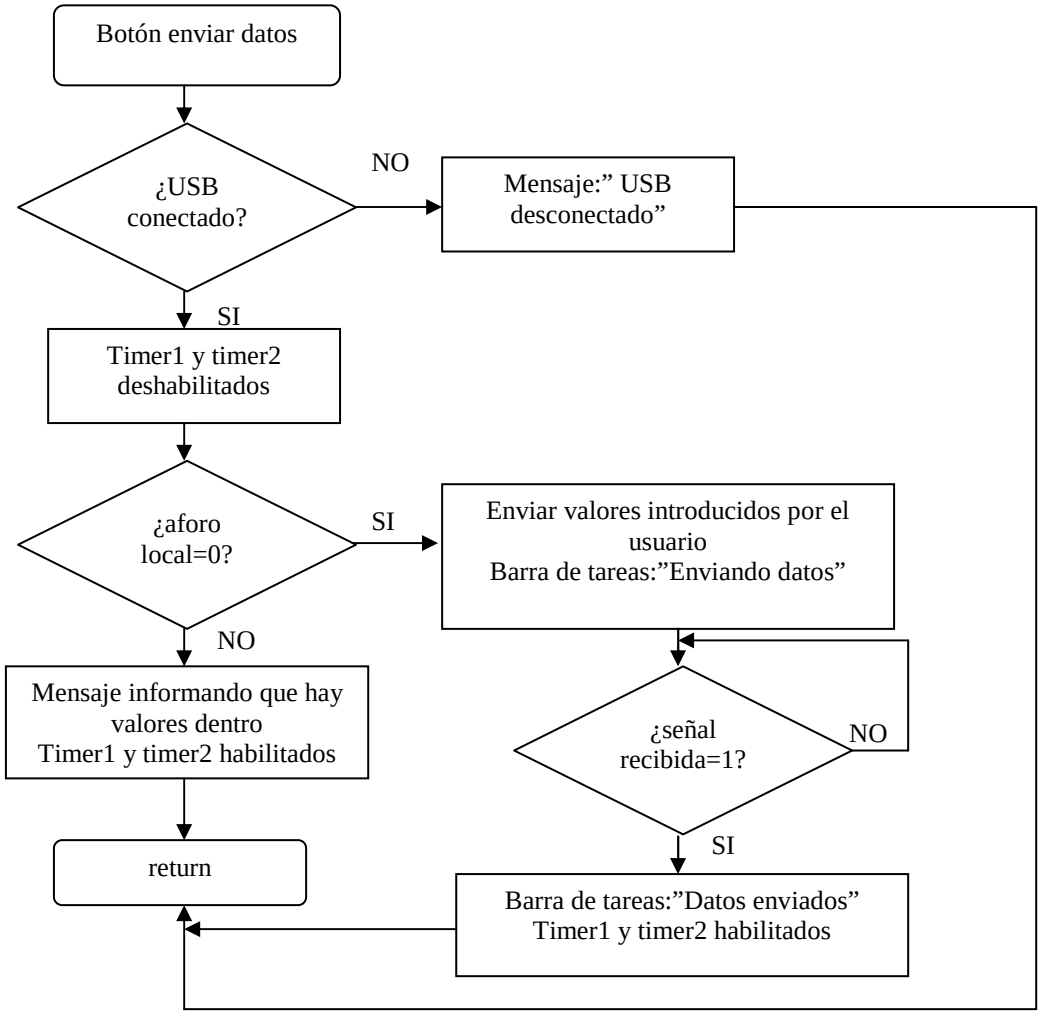

Sensor altura
Iniciación medida sensor altura
Activo TMR0
Alt_persona=alt_sensor-medida sensor
Sumo esta altura con las guardadas
¿15 medidas?
NO
SI
Alt_persona definitiva=alt_acumuladas/15
¿alt_definitiva>alt minima?
NO
SI
Devolverá 0
Return
¿alt_definitiva<alt máxima?
NO
SI
Devolverá 1
Sumar aforo
Incrementamos en 1 el aforo actual
¿aforo actual>aforo
SI
NO
Devolverá 0
Devolverá 1
Return

6.4. Flujograma interfaz

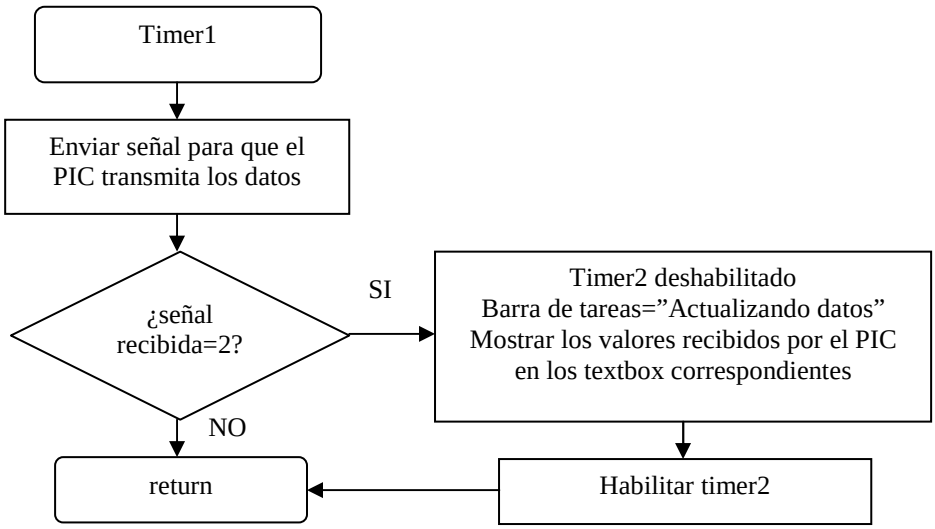
El primero es el del botón enviar datos. Este evento sólo se produce si se **PULSA** el botón “Enviar” de la interfaz gráfica.



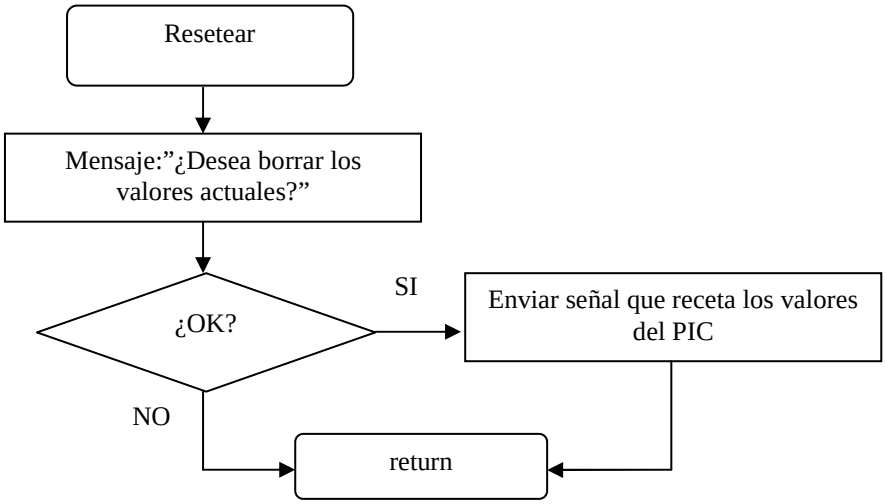




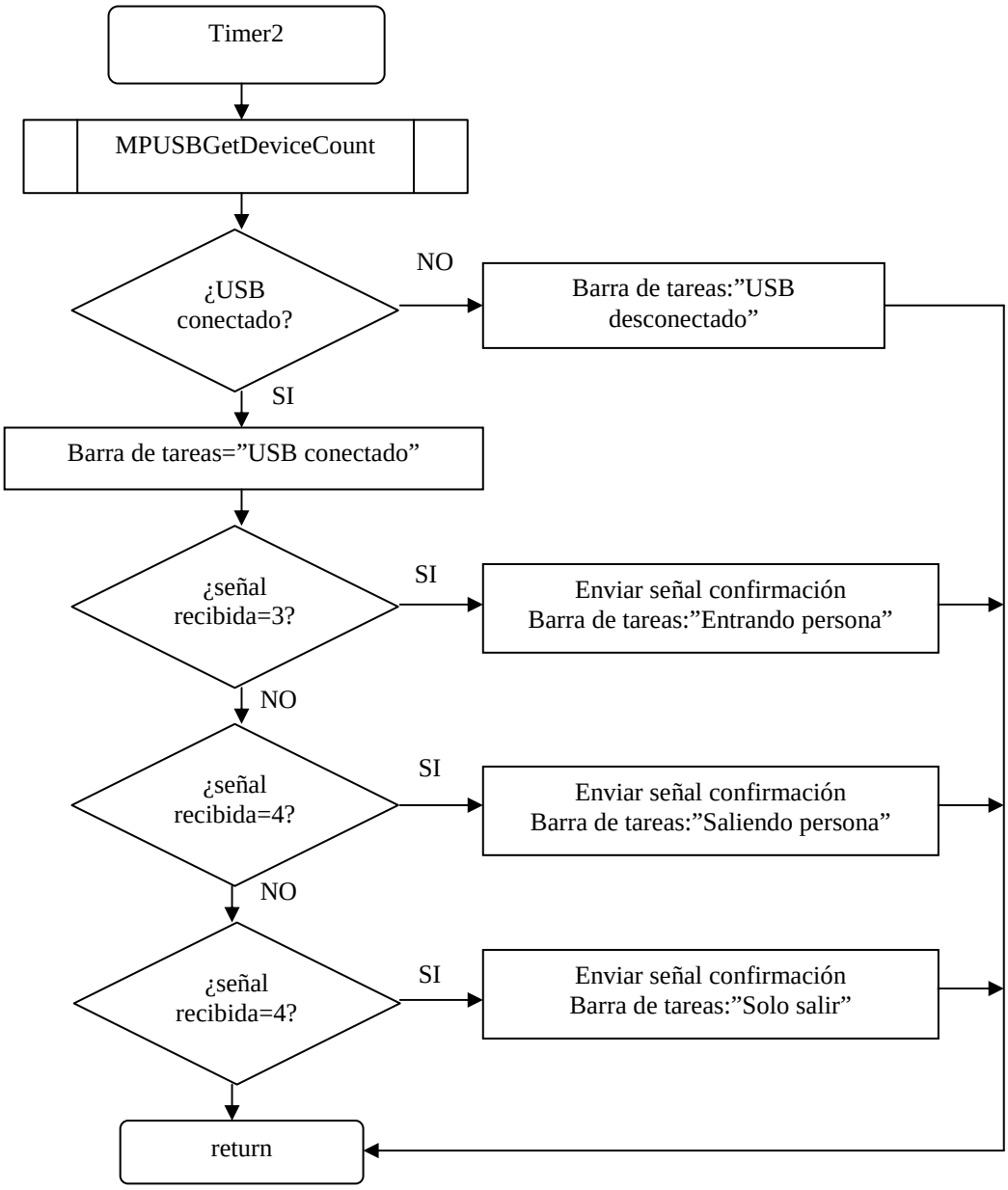
El siguiente flujograma muestra el comportamiento del timer1, que se activa cada 300ms. Su principal función es actualizar los datos y cambiar los mensajes que aparecen en la barra de tareas.



El siguiente flujograma muestra el comportamiento del botón “Resetear” incluido en el menú.



Por último se muestra el comportamiento del timer2 que se activa cada 200ms. Sus principales utilidades son: Estar pendiente de las señales que envía el PIC cuando entra o sale alguien, ver si el USB está conectado o desconectado y permitir si el label de la barra estado se muestre.


Timer2
MPUSBGetDeviceCount
¿USB conectado?
NO
Barra de tareas:"USB desconectado"
SI
Barra de tareas="USB conectado"
¿señal recibida=3?
SI
Enviar señal confirmación
Barra de tareas:"Entrando persona"
NO
¿señal recibida=4?
SI
Enviar señal confirmación
Barra de tareas:"Saliendo persona"
NO
¿señal recibida=4?
SI
Enviar señal confirmación
Barra de tareas:"Solo salir"
return

7. Bibliografía

- Datasheet PIC 16F84A
- Datasheet PIC 18F4550
- http://www.todopic.com
- http://www.superrobotica.com
- Manual de Usuario del compilador PCWH de CCS. CCS Inc.
- Microsoft Visual C# .net Aprenda ya, John Sharp/Jon JaggerMac Graw Hill
- http://www.forosdeelectronica.com
- Manual de formación de USB, Servicio técnico OEM
- http://www.hobbypic.com
- Desarrollo de un servidor DHCP-PIC controlado vía USB, Antonio Martínez García