UNIVERSIDAD NACIONAL AUTÓNOMA

DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

"DISEÑO E IMPLEMENTACIÓN DE UN CONTROL DE VELOCIDAD PROGRAMABLE CON RESOLUCIÓN MICROMÉTRICA PARA LA ESTRUCTURACIÓN DE SUPERFICIES"

TESIS

PARA OBTENER EL TÍTULO DE:

LICENCIADO EN TECNOLOGÍA

PRESENTA:

RICARDO RODRÍGUEZ GONZÁLEZ

TUTOR:

DRA. CITLALI SÁNCHEZ AKÉ

COASESOR:

DR. CRESCENCIO GARCÍA SEGUNDO

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO 2015

VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
UNIDAD DE ADMINISTRACIÓN ESCOLAR
DEPARTAMENTO DE EXÁMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: M. en A. ISMAEL HERNÁNDEZ MAURICIO
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán.

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos **La Tesis:**

DISEÑO E IMPLEMENTACIÓN DE UN CONTROL DE VELOCIDAD PROGRAMABLE CON RESOLUCIÓN MICROMÉTRICA PARA LA ESTRUCTURACIÓN DE SUPERFICIES

Que presenta el pasante: **RICARDO RODRÍGUEZ GONZÁLEZ**
Con número de cuenta: **41001573-9** para obtener el Título de: **Licenciado en Tecnología**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 22 de junio de 2015.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	M. en I. Ramón Osorio Galicia	
VOCAL	M. en I. Felipe Díaz del Castillo Rodríguez	
SECRETARIO	Dra. Citlali Sánchez Ake	
1er SUPLENTE	Ing. Noemí Hernández Domínguez	
2do SUPLENTE	M. en C. José Isaac Sánchez Guerra	

NOTA: Los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).
En caso de que algún miembro del jurado no pueda asistir al examen profesional deberá dar aviso por anticipado al departamento. (Art 127 REP)
IHM/yrf

Dedicatorias y Agradecimientos

Este trabajo se lo dedico a una persona en especial, que me vio crecer, que creyó y dio toda su vida por mí, hasta en los últimos momentos, donde ni yo me reconocía, porque tú y mi familia son lo más importante de mi vida, y sin tí, los sueños no saben igual, y las victorias saben a tragos amargos porque tú fuiste mi universo y mi esperanza. Por tí seguiré adelante, para que veas que tanto esfuerzo y apoyo que me diste valió la pena.

Gracias mamá y hermano, por estar y apoyarme en los momentos más importantes de mi vida y hacerme sentir bien día a día, porque son mi motivación e inspiración.

Mis más profundos agradecimientos a la Dra. Citlalli y al Dr. Crescencio por apoyarme de manera personal y profesional. Porque, para mí, continuar ha sido una odisea en todos los sentidos, pero he aguantado y aún sigo luchando para cumplir mis sueños.

También quiero agradecer de manera general a todas las personas que me han ayudado, dado ánimo, o escuchado a lo largo de toda mi vida, gracias a que nuestros caminos se cruzaron un momento, me sirvió para comprender, aprender y entender el significado de la vida, y ser la persona que soy hoy en día.

Por último agradezco al proyecto PAPIIT-DGAPA IG100415 por la beca otorgada.

Índice

Sección Página

RESUMEN

El presente trabajo consiste en la implementación de un sistema de movimiento con velocidad controlada en un intervalo de 400 μm/s a 10,000 μm/s, que se utilizará posteriormente en la técnica de recubrimiento por inmersión para la estructuración de superficies de tamaños micro y nano-métrico. Este dispositivo fue bautizado con el nombre de EEMGRI que significa Elevador Electro-Mecánico para la Generación Recubrimientos por Inmersión.

El sistema de movimiento con alto control en velocidades bajas es la pieza clave para tal aplicación, por lo que esta tesis constituye el primer paso para alcanzar este fin. La técnica de recubrimiento por inmersión consiste básicamente en sumergir un sustrato en una suspensión coloidal, para después sacarlo a una velocidad controlada generando una estructura ordenada sobre la superficie del sustrato. Tiene aplicaciones que se extienden en todos los campos de la ciencia, que van desde una rejilla de difracción, membranas filtrantes o sensores, cristales fotónicos, máscaras litográficas y plantillas, hasta elementos de memoria óptica y la fabricación de materiales.

Para construir el EEMGRI se utilizó un carro de impresora para el sistema físico y se diseñó e imprimió en una placa el circuito electrónico para el sistema de control del mismo. La placa de este circuito electrónico es un sistema de control de motores paso a paso que opera con un microcontrolador PIC-18F4550 de microchip a 48 MHz, contiene un controlador de potencia L298, para el manejo de motores unipolares y bipolares paso a paso el cual soporta motores de 3 V a 20 V con corrientes máximas de 1 A, cuenta con dos sensores infrarrojo IR333c, dos fototransistores PT331c, y dos micro-*switch*, para el control de la posición, e incluye una interfaz de comunicación USB de clase HID (*Human Interface Device*) con el protocolo USB 2.0 para comunicarse con el ordenador. Una de las ventajas de la clase HID es que no requiere de controladores adicionales para su instalación, y es compatible con los principales sistemas operativos (Linux, Windows y Mac).

También cabe señalar que el EEMGRI será ocupado principalmente en aplicaciones para la investigación en la estructuración de superficies y películas delgadas, además el presente trabajo servirá como manual para el desarrollo de circuitos con interface USB, también puede ser adaptado para una gran cantidad de aplicaciones en sistemas ópticos y síntesis de materiales diversos, debido a que usa un motor de pasos y posee control de la velocidad y posición del mismo. Como ejemplo de aplicaciones está la alineación de espejos y soporte de sustratos para el depósito de películas delgadas. Por esta razón, el sistema desarrollado en esta tesis se implementará en diversos dispositivos en el Laboratorio de Fotofísica y Películas Delgadas del Centro de Ciencias Aplicadas y Desarrollo Tecnológico de la UNAM

OBJETIVOS

OBJETIVO GENERAL

Diseñar y crear un sistema de control electrónico de movimiento con velocidad controlada para su implementación en un dispositivo que en el futuro será aplicado a la estructuración de superficies.

OBJETIVOS ESPECÍFICOS:

- Implementar la comunicación USB para manipular el dispositivo desde múltiples sistemas operativos.
- Controlar de manera programable las velocidades del dispositivo en un rango de 400 μm/s a 10,000 μm/s.
- Controlar la dirección del movimiento del sistema mecánico.

HIPÓTESIS

Se plantea generar un circuito electrónico que controle la velocidad de un motor paso a paso como herramienta para el depósito de estructuras superficiales que requiere velocidades constantes sin efectos de inercia, ni vibraciones. La hipótesis principal es que los elementos del circuito permitirán el control del motor a pasos a una velocidad mínima de centenas de μm/s. También se establece que la interfaz USB permitirá el control del dispositivo desde diferentes sistemas operativos aumentando así su versatilidad.

JUSTIFICACIÓN

Una de las líneas de investigación del laboratorio de Fotofísica y Películas Delgadas del Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la UNAM, es la síntesis de películas delgadas y materiales nanoestructurados principalmente mediante métodos físicos como pulverización catódica y ablación láser. Tales métodos otorgan gran control y reproducibilidad. Se ha propuesto un proyecto a futuro, que consiste en complementar las técnicas ya en funcionamiento generando estructuras superficiales de materiales diversos que funcionen como máscaras para la ayuda de la formación de películas delgadas o nanoestructuras. Para poder crear estas estructuras se optó por ocupar la técnica de recubrimiento por inmersión, debido a su alta calidad de estructuración y bajo costo. Sin embargo, los equipos comerciales que realizan este proceso tienen costos muy elevados cercanos a los 10,000 USD. Por ello, en esta tesis se propuso desarrollar un sistema de control de movimiento que pueda controlarse con una computadora de manera sencilla y con velocidades inferiores a los 500 μm/s. De este modo esta tesis constituye el primer paso indispensable para implementar la técnica a futuro. Más aún, el sistema de movimiento desarrollado en este trabajo podrá también utilizarse en técnicas ópticas para el análisis de materiales y plasmas.

Para cumplir las características necesarias en cuanto a control de velocidad, posición y versatilidad en el sistema, se propuso utilizar la interface de comunicación USB versión 2.0 de clase HID, para una mejor manipulación con el usuario, además de que esta interfaz es la más popular en el mercado facilitando la conexión desde cualquier ordenador, y además la clase HID no requiere de drivers adicionales para su instalación. También se propuso usar un motor a paso a paso, debido a su control de posición y su gran precisión, reduciendo efectos de inercia.

CAPITULO 1

FUNDAMENTOS TEÓRICOS

1. 1 Recubrimiento por Inmersión

El recubrimiento por inmersión ó *dip coating* es un proceso altamente usado, tanto en laboratorios con fines de investigación como en la industria, introducido por Dimitrov y Nagayama en 1996, para producir estructuras superficiales sobre un sustrato plano o cilíndrico. [1] Esta técnica consiste en sumergir un sustrato en una suspensión coloidal, el cual se retira cuidadosamente para controlar la estructuración de la superficie con la velocidad de extracción, como se puede ver en la figura 1.1. El resultado es el depósito sobre el substrato de capas de esferas que se encontraban originalmente en el coloide, bajo ciertas condiciones puede depositarse incluso una sola capa.

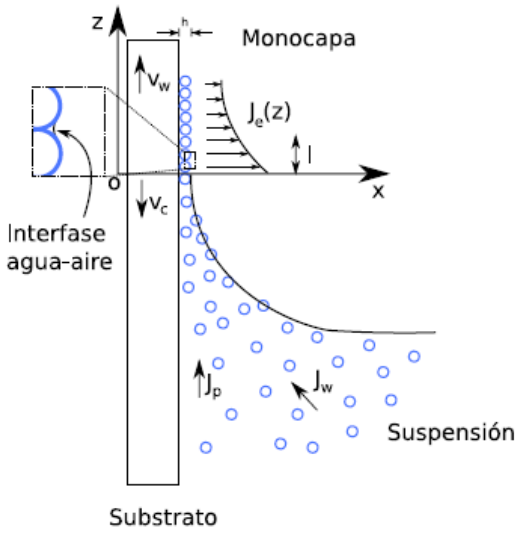

Figura 1.1: Esquema de los flujos y velocidades que intervienen en la técnica de dip-coating. El sustrato es elevado con velocidad V_w, y la tasa de crecimiento del depósito es V_c. Los flujos marcados corresponden al flujo de agua de la suspensión hacia el mecanismo (J_w), el de las partículas (J_p) y el del agua que se evapora en la estructura superficial (J_e). La altura h denota el espesor del depósito [2].

Esta técnica es muy útil en la adición de capas protectoras contra la corrosión, en el incremento del coeficiente de fricción de un material, aislamiento de calor, frío, tensiones, corrientes eléctricas, como mejoramiento estético con la implementación de colores, acabados y texturas, en la fabricación de celdas solares, películas anti-reflejantes y sensores. Con la ventaja de ser de fácil implementación de una manera económica y de poderse adaptar a grandes volúmenes de producción.

En la actualidad, el recubrimiento por inmersión no es difícil de implementar. Básicamente consiste en cinco pasos que se ilustran en la figura 1.2 y se resumen a continuación:

- Inmersión: El sustrato es introducido dentro de la solución del material de recubrimiento a una velocidad constante, evitando sacudidas.

- Comienzo: Es cuando el sustrato permanece cierto tiempo en la solución hasta que se le comienza a elevar.

- Depósito: La capa delgada se deposita en el sustrato mientras se le extrae de la solución. La extracción se realiza a una velocidad constante para evitar sacudidas. La velocidad determina el espesor de la capa del recubrimiento (una velocidad de extracción mayor produce una capa de recubrimiento más gruesa).

- Drenado: El exceso de líquido drena la superficie.

- Evaporación: El solvente se evapora del líquido, formando la capa delgada.

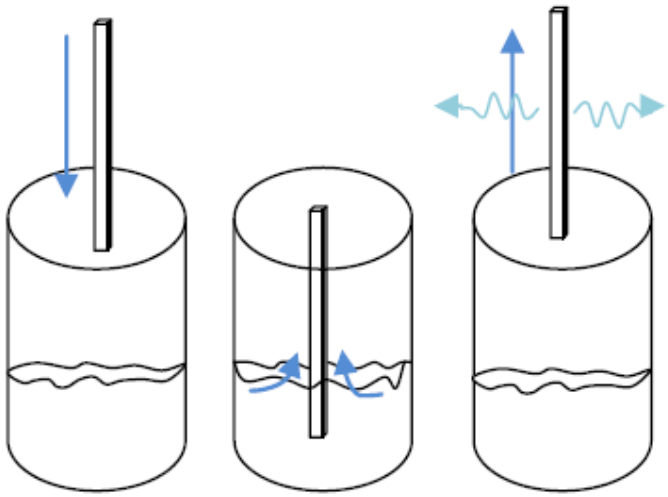

Figura 1.2: Inmersión, depósito y evaporación. Se puede apreciar como el substrato es sumergido en la solución coloidal y después es extraído a velocidad constante para una buena calidad en el depósito. [3]

Una de las ventajas del recubrimiento de inmersión es que se tiene un buen aprovechamiento de la solución coloidal, debido a que el material que no se adhiere al sustrato se regresa al recipiente de la solución por el efecto de la gravedad y puede ser usado en subsecuentes crecimientos. La reproducibilidad en el crecimiento de las películas se ve beneficiada por el control preciso de la velocidad de extracción, las pausas en el proceso de inmersión y la extracción de los sustratos, por ello se fabrican aparatos comerciales que puedan manejar estas variables.

1.2 Sistemas de Control

Hoy en día los sistemas de control asumen un papel muy importante en el desarrollo y avance de la civilización moderna y la tecnología, debido a que nos permiten manipular las variables de un sistema y a su vez controlar todo el proceso, permitiendo la manipulación de nuestras variables de salida, obteniendo una mayor calidad en cuanto a los resultados. Los sistemas de control son aplicados en casi todos los sectores de la industria, tales como el control de calidad de los productos manufacturados, líneas de ensamble automático, control de máquinas-herramienta, tecnología espacial, sistemas de armas, control por computadora, sistemas de transporte, sistemas de potencia, robótica, y muchos otros.

Los sistemas de control se definen como un conjunto de elementos que actúan colectivamente para controlar un proceso o sistema y cumplir un determinado objetivo. [4] Los componentes básicos de un sistema se pueden describir mediante:

1. Objetivos de Control
2. Componentes del sistema de control
3. Resultados o salidas

La relación básica entre estos tres elementos se ilustra en la figura 1.3:

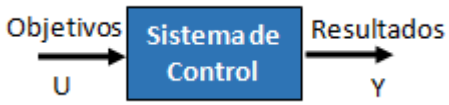

Figura 1.3: Esquema básico de Control.

Los sistemas de control se dividen en:

Sistemas de control de Lazo Abierto: los cuales se caracterizan por no tener retroalimentación, es decir no se compara la variable de salida con el valor deseado de la misma, suelen aparecer en dispositivos con control secuencial, en el cual no hay una regulación de variables, si no que se realizan una serie de operaciones de manera predeterminada (ver Fig. 1.4).

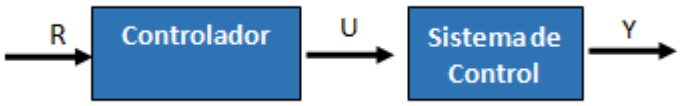

Figura 1.4: Sistema básico de control de lazo abierto.

Sistemas de control de Lazo Cerrado: Son aquellos en los que la variable de salida es controlada, es decir hay una señal de retroalimentación la cual compara una variable de salida con una entrada de referencia, el cual tiende a reducir los errores del proceso (Figura 1.5).

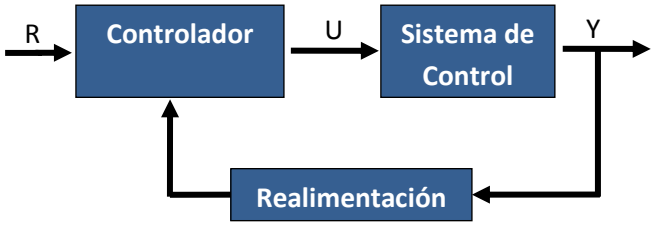

Figura 1.5: Sistema básico de control de lazo cerrado.

1.2.1 Sistemas de control de un motor paso a paso (PAP)

Para la implementación de los motores a pasos es necesario un sistema de control el cual se compone básicamente de un controlador, un sistema de potencia o driver y un actuador que en este caso es el motor de PAP, como se muestra en la figura 1.6.

Estos tipos de sistemas de control suelen ser de lazo abierto debido a la confiabilidad y características que presentan los motores PAP, como se verá más adelante. Sin embargo, en ocasiones donde la velocidad que se ocupa es relativamente alta se ocupan sistemas de control cerrado.

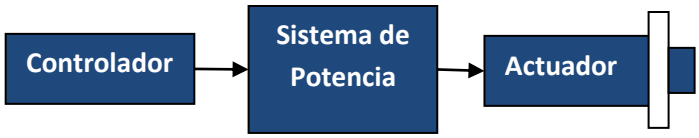

Figura 1.6: Sistema básico de control de pasos. El controlador se encarga de generar un tren de pulsos, que serán enviados al sistema de potencia donde serán amplificados con la corriente necesaria para accionar las bobinas del motor a pasos, inmediatamente éste se moverá un cierto ángulo en cada pulso enviado.

Motores Paso a Paso (PAP)

Los motores PAP se caracterizan por traducir una señal de pulsos eléctricos en incrementos de movimiento definidos con precisión en la posición del rotor, conocidos como pasos. Este tipo de motores se usan habitualmente en sistemas de control digital en los que el motor recibe órdenes de lazo abierto. Por ello son usados en el campo de la robótica, tecnología aeroespacial, discos duros, impresoras, manipulación y posicionamiento de herramientas y piezas en general (maquinas CNC). [5]

Los motores PAP se caracterizan por las siguientes ventajas:

- Precisión y receptibilidad. Capacidad para posicionar con precisión además de que el error de resolución del motor no es acumulable.
- Capacidad de respuesta y aceleración rápida. Los motores de pasos tienen una baja inercia del rotor, lo que les permite ponerse en marcha rápidamente. Esto hace que los motores paso a paso sean una excelente opción para distancias cortas y movimientos rápidos.
- Posicionamiento de Estabilidad. A diferencia de otros tipos de motores, los motores paso a paso se hacen completamente inmóviles en su posición de parada.
- Lazo de control. El control de lazo abierto es más simple, más fiable y menos costoso que el control de retroalimentación basada (circuito cerrado). En los sistemas de circuito cerrado, los codificadores se utilizan para contar el número de pasos dados por el motor. El número de pasos dados se compara con el número de comandos paso determinado. Esta retroalimentación se utiliza para hacer correcciones de la posición o iniciar señales de alarma. Los codificadores y sus componentes electrónicos asociados agregan costo adicional a un sistema de control de movimiento. Suponiendo que un motor paso a paso está dimensionado adecuadamente para su carga, nunca debe faltar un paso, haciendo un codificador innecesario.
- Costo y Confiabilidad. La tecnología de motor PAP es fiable y probada. Es el método más rentable de control de posición de precisión.

Motor PAP de imán permanente

Existen tres tipos de motores PAP clasificados por su funcionamiento y las características de su estator y rotor: de reluctancia variable, de imán permanente e híbridos. En este texto solo nos centraremos en los motores de imán permanente, ya que fue el que se utilizó en este proyecto.

Los motores de imán permanente tienen un rotor de imán permanente sin dientes, los cuales son magnetizados perpendicularmente por las bobinas que contiene el estator (figura 1.7). Energizando las cuatro fases en secuencia, el rotor gira conforme es atraído por los polos magnéticos, se caracterizan por la velocidad de pasos es relativamente baja, y presentan un alto par o torque y unas buenas características de amortiguamiento. [6]

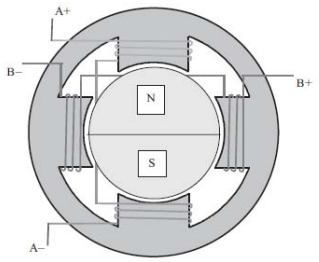

Figura 1.7: Sección transversal de un PAP de imán permanente.

Para los motores a paso a paso de imán permanente se puede conectar su devanado de manera unipolar y bipolar.

Motor Unipolar: Estos motores son 6 y 5 hilos de entrada para comunicarse con el controlador, dependiendo de sus conexiones internas, se caracteriza por ser el más simple de controlar.

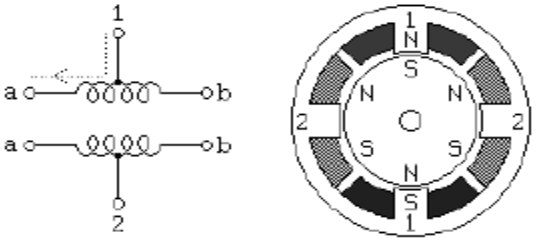

Figura 1.8: Motor unipolar de 4 fases

Motor bipolar: Estos motores tienen generalmente 4 hilos de entrada para comunicarse con el controlador. Necesitan cierta manipulación para ser controlados, debido a que requieren del cambio de dirección del flujo de corriente a través de las bobinas en la secuencia apropiada para realizar un movimiento.

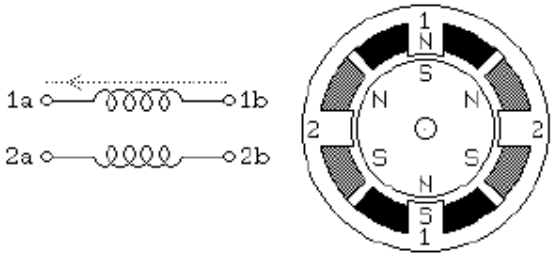

Figura 5.9: Motor bipolar de 2 fases

Pasos de un Motor PAP

Como se vio anteriormente un motor PAP funciona mediante pulsos eléctricos, a este conjunto de pulsos los podemos llamar tren de pulsos. Éstos definen el movimiento de un motor PAP, cada pulso que entra al motor PAP provoca un pequeño giro al eje del motor, el cual se puede definir como un paso. Para el control de los motores PAP existen cuatro tipos diferentes de pasos.

Paso simple: El paso simple es el más común, en donde la secuencia lo único que hace es activar bobina por bobina de manera secuencial para así producir movimiento (Figura 1.10).

Paso	L1	L2	L3	L4
0x08	1	0	0	0
0x04	0	1	0	0
0x02	0	0	1	0
0x01	0	0	0	1

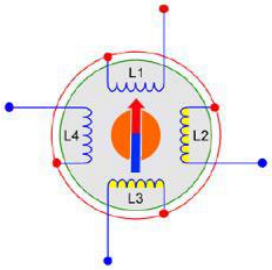

Figura 1.10: Secuencia del paso simple de un motor PAP

Paso completo: Esta secuencia de pasos lo que hace es activar dos bobinas a la vez, incrementando al doble el torque del motor como se puede ver en la siguiente figura 1.11.

Paso	L1	L2	L3	L4
0x09	1	0	0	1
0x0C	1	1	0	0
0x06	0	1	1	0
0x03	0	0	1	1

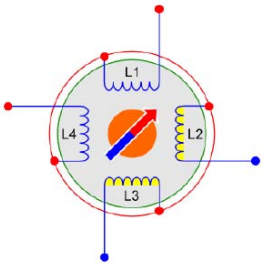

Figura 1.11: Secuencia del paso completo de un motor PAP

Medio Paso: Esta secuencia combina a los dos anteriores pasos incrementando la resolución del motor al doble, como se puede ver en la figura 1.12.

Paso	L1	L2	L3	L4
0x09	1	0	0	1
0x08	1	0	0	0
0x0C	1	1	0	0
0x04	0	1	0	0
0x06	0	1	1	0
0x02	0	0	1	0
0x03	0	0	1	1
0x01	0	0	0	1

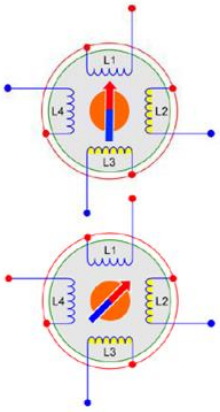

Figura 1.12: Secuencia de medio paso de un motor PAP.

Micro-pasos: Los micro-pasos usan un controlador diferente a los anteriores tipos de pasos, debido a que estos utilizan ondas senoidales para su control, dividiendo cada paso en 2, 4, 8 o hasta más, aumentando la resolución del motor. Estos pasos se caracterizan por evitar el cambio brusco de torque entre paso y paso, suavizando el movimiento y aumentando la resolución del motor PAP.

Etapa de Potencia

Existe una gran diversidad de circuitos para la conmutación de corrientes para los motores PAP. Estos son encargados de la etapa de potencia, es decir, proporcionan la suficiente corriente eléctrica para mover un motor PAP. Estos circuitos también dependen del tipo de motor y del tipo de paso.

Controladores

Para un sistema de control de un motor PAP es necesario un controlador, el cual manipula la secuencia y tiempo de los pulsos. Los microcontroladores son una buena opción debido a su versatilidad y bajo costo.

Microcontrolador PIC-18F4550 de Microchip

El microcontrolador PIC 18F4550 de Microchip es una buena opción para el desarrollo de estos sistemas de control debido a su versatilidad, bajo precio y fácil uso. Además el microchip proporciona el software para su aplicación y este microcontrolador cuenta con modulo para la comunicación USB.

Entre las características que destacan del microcontrolador PIC 18F4550 están:

- Microcontrolador de gama media
- Programación de alto nivel en C
- Comunicación USB de baja y alta velocidad
- Múltiples salidas digitales
- Bajo Costo
- Fácil adquisición

Una de las funciones especiales del PIC 18F4550 es el modulo USB el cual requiere de 48 MHz para poder trabajar adecuadamente, además este módulo tiene compatibilidad con USB versión 2.0, velocidad de trasmisión *Low Speed* (1.5 Mb/s) y *Full Speed* (12 Mb/s), soporta transferencias de control, interrupción, isocrónicas y *Bulk* (masivas), soporta hasta 32 *endpoint* o *buffers* de memoria RAM(16 bidireccionales) entre otras. [7]

1.3 USB 2.0 (Universal Serial Bus)

El USB es un puerto que sirve para conectar periféricos a un ordenador. Existen cuatro versiones del protocolo (1.0, 1.1, 2.0 y 3.0). La versión 2.0 soporta tasas de transferencia de altas velocidades, comparables a la de un disco duro. Además esta versión es compatible con todas las versiones anteriores.

El USB 2.0 nos permite interconectar una variedad de dispositivos con el ordenador, usando un simple cable de cuatro hilos; dos de alimentación (5 V y GND) y dos para datos (D+ y D-) utilizando señalización diferencial half dúplex, minimizando el ruido electromagnético, además su diseño permite un largo máximo de 5 metros. También funciona en un intervalo de velocidades de 1.5 Mb/s a 480 Mb/s.[8]

Tipo	Velocidad
Baja Velocidad (*low speed*)	183 Kbytes/s (1.5 Mbits/s)
Velocidad completa (*full speed*)	1.4 Mbytes/s (12 Mbits/s)
Alta velocidad (*high speed*)	57 Mbytes/s (480 Mbits/s)

Figura 1.13: Tabla de velocidades de comunicación del USB.

Existen dos tipos de conectores: Estándar y mini. Los estándar los encontramos típicamente en un ordenador y vienen en dos tipos A y B (Figura 1.14). El tipo A es el que es plano y se encuentra del lado del host controlador, mientras el tipo B se encuentra del lado del dispositivo esclavo.

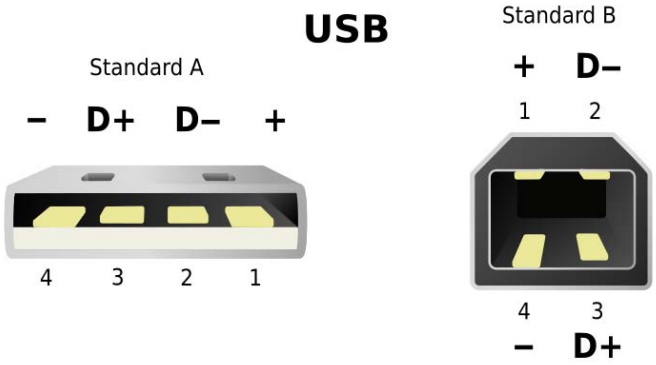

Figura 1.14: Conectores estándar A y conector estándar B

Las clases de USB son como una plantilla que ayuda a los desarrolladores debido a que estas definen el comportamiento y los protocolos para varios dispositivos que sirven de forma similar. Existen muchas clases para la comunicación USB, algunas de las más usadas en microcontroladores son: HID (Human Interface Device), MSDC (Mass Storage Device

Class),CDC (Comunications Device Class) y Custom Class la cual es una clase genérica en caso de que el dispositivo no se asemeje a ninguna de las clases existentes.

1.3.1 Clase HID (Human Interface Device)

La clase HID consiste principalmente de dispositivos que se usan por los seres humanos para controlar el funcionamiento de los sistemas informáticos como ratones, teclados y controles. Aunque también pueden ser usados en dispositivos que no requieren la interacción humana, como termómetros, voltímetros, lectores de código de barras, etc.

Una de las ventajas que tienen los dispositivos que implementan en su firmware la clase HID es que la mayoría de los Sistemas Operativos modernos implementan los controladores necesarios para poder comunicarse con ellos sin ser necesaria la instalación de ningún driver adicional, bastando con los driver del sistema. Además trabaja con las velocidades bajas del USB. [9]

1.3.2 API (Application Programming Interface)

Es una aplicación que se encargara de manejar el Host del ordenador para comunicar con la interfaz USB con un dispositivo HID.

HIDAPI

Es una librería multiplataforma que permite que una aplicación interactúe con dispositivos de clase HID en Windows, Linux y Mac OS X. El desarrollo de estas aplicaciones es más fácil ya que no se necesita conocer los detalles de las bibliotecas HID y las interfaces de cada plataforma. [10]

Hay varias formas de utilizar HIDAPI dependientes del sistema operativo utilizado:

- Windows (usando hid.dll)
- Linux/hidraw (usando el driver hidraw del kernel)
- Linux/libusb (usando libusb-1.0)
- Mac (usando IOHidManager)

Las funciones proporcionadas por la librería HIDAPI facilitan la programación de la API para controlar los procesos básicos de los dispositivos de clase HID. A continuación mencionamos las más relevantes para nuestro proceso de comunicación:

Hid_init (void): Inicializa la biblioteca HIDAPI, por ello debe ser llamada al inicio de la ejecución.

Hid_exit (void): Finaliza la biblioteca HIDAPI, liberando todos los datos estáticos asociados.

Hid_enumerate (vendor_id, product_id): Enumera los dispositivos HID, devolviendo una lista enlazada de todos los dispositivos HID conectados al sistema que coincidan con el número del proveedor y el producto.

Hid_free_enumeracion (struct hid_device_info): Libera una lista de enumeración de todos los dispositivos HID, creada por hid_enumerate.

Hid_open (vendor_id, product_id , serial number): Abre un dispositivo HID utilizando el VID, PID y opcionalmente el número de serie.

Hid_write(hid_device, unsigned char, size_t): Escribe un informe de salida para un dispositivo HID. El primer byte de datos debe de contener el informe de identificación. Para los productos que solo soportan un único informe, esto se debe de establecer en 0x0. Los bytes restantes contienen los datos del informe. Desde el informe de identificación es obligatorio incluir un byte más que los que el informe contiene.

Hid_read (hid_device, unsigned char,size_t): Lee un informe de entrada del dispositivo HID. Los informes de entrada se devuelven al Host a través de la interrupción del endpoint. El primer byte contiene el número de informes.

Hid_close (hid_device): Cierra un dispositivo HID.

CAPITULO 2

DISEÑO E IMPLEMENTACIÓN DEL EEMGRI

Ahora que ya conocemos los conceptos necesarios para el entendimiento de este proyecto nos dispondremos en describir cómo se diseñó y desarrolló el sistema de control de movimiento del EEMGRI.

Como se mencionó en la introducción, la técnica de recubrimiento por inmersión consiste en subir y bajar un sustrato en una solución de manera controlada, tanto en la posición como en la velocidad. Visualizando el problema desde el punto de vista de la ingeniería lo que necesitamos es un sistema mecatrónico capaz de desplazar una placa con velocidades constantes y tiempos controlados. Este problema lo podemos dividir en dos partes, una es generar la solución mecánica, y la otra es crear el sistema de control capaz de manipular el sistema mecánico. Nosotros decidimos para el sistema de control tomar como ejemplo de un sistema mecánico ya existente: una base de un carro de impresora, la cual se pretende acoplar con el sistema de control para el desarrollo de nuestras aplicaciones.

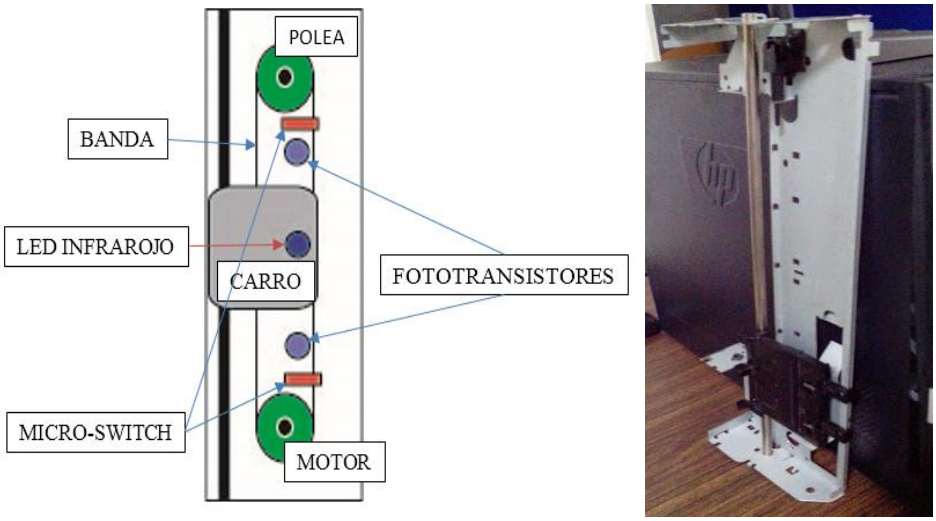

Figura 2.1: En la parte izquierda se puede apreciar el esquema del carro de la impresora adaptado a nuestro sistema y del lado derecho el carro de impresora a adaptar.

Entonces lo primero que nos planteamos fue determinar las características que deseábamos para nuestro sistema de control, las cuales se mencionan a continuación:

- Posicionamiento y precisión.
- Velocidades bajas y controladas.
- Conexión con el ordenador.

En este contexto empezamos con la selección del motor, ya que será el encargado de definir nuestro sistema de control. Se decidió ocupar los motores PAP, porque pueden ser controlados por medio de un microcontrolador y además son usados en sistemas de control de lazo abierto, disminuyendo la complejidad y costos del sistema.

Como controlador en nuestro sistema utilizamos el microcontrolador PIC18F4550 por la flexibilidad para adaptarlo a nuestras aplicaciones además de que incluye un módulo USB para la interacción con el ordenador.

En este proyecto usamos un motor semi-usado unipolar de 6 hilos con 1.8° de precisión de 3 V a 5 V y 500 mA. (Figura 2.2)

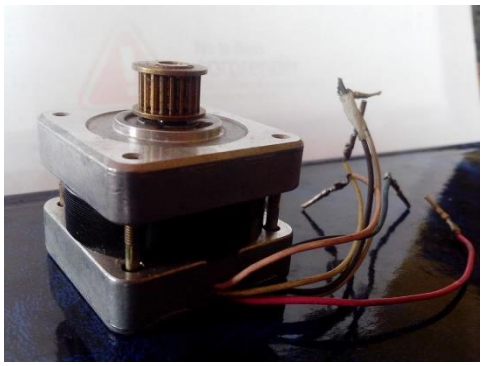

Figura 2.2: Motor Sanyo Denky 103-550-0149.

2.1 Circuito Electrónico

Usamos un microcontrolador PIC18F4550 por su flexibilidad para adaptarlo a nuestras aplicaciones además que incluye un módulo USB para la interacción con un ordenador. El circuito lo podemos dividir en cuatro partes:

Circuito básico del PIC: La primera sección es la del control básico del PIC, la cual contiene un circuito para el control del *master clean* como se puede ver en la figura 2.3 y otro para el control del reloj externo que servirá para la frecuencia de trabajo del PIC y del módulo USB.

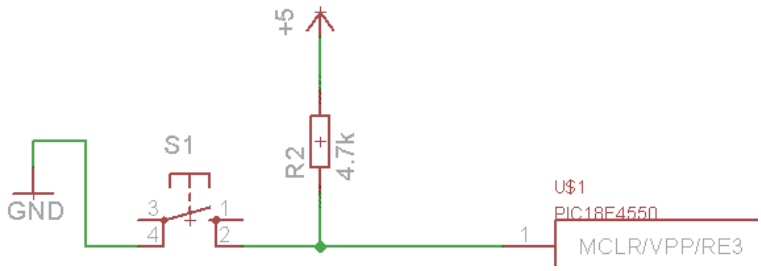

Figura 2.3: Circuito de control del MCLR, el cual sirve para reiniciar las acciones del microcontrolador. Es necesario conectar la patilla del MCRL para el funcionamiento del PIC.

Circuito de la interfaz USB: Esta parte importante para la comunicación de la interfaz USB (Figura 2.4 y Figura 2.5), ya que se debe considerar la frecuencia de trabajo del módulo USB del microcontrolador, que nuestro caso es de 48MHz, para realizar los cálculos pertinentes en la programación y así determinar el valor de nuestro oscilador externo. En caso de que el microcontrolador no le proporcione los 48MHz de frecuencia de trabajo al modulo USB, éste no funcionará. Además, para facilitar la visualización del buen funcionamiento de la interface agregamos dos leds que hacen una rutina de parpadeo cuando se conecta el dispositivo al ordenador y se detienen cuando el dispositivo este enumerado.

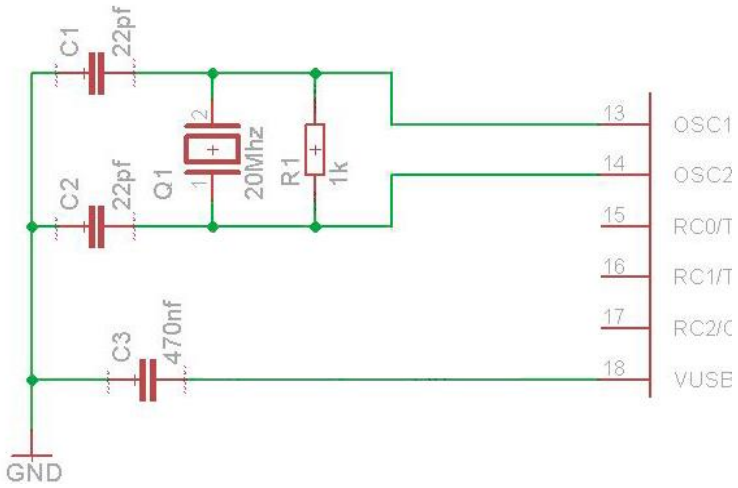

Figura 2.4: Circuito de la conexión del reloj externo.

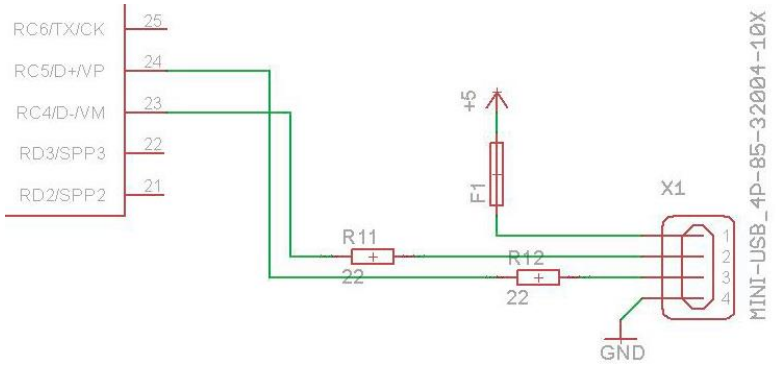

Figura 2.5: Circuito de la conexión de la interfaz USB.

Para esta parte del circuito se utilizó:

- 1 Conector mini-USB hembra clase B.
- 2 Resistencias de 20 Ohms a ½ watts.
- 1 Oscilador XT de 20 MHz.

- 2 Capacitores cerámicos de 22 pF.
- 2 Leds de 1.3 v.
- 2 Resistencias de 330 Ohms.

Circuito de la Etapa de Potencia: El circuito integrado que ocupa nuestro driver es un L298 (ver figura 2.6) encargado de proporcionar la potencia suficiente para mover el motor PAP. Además contiene diodos para evitar rebotes de corriente y proteger al microcontrolador, soporta pasos simples, completos y de medio paso. Este circuito está integrado por:

- 1 L298.
- 8 diodos 1N4004.

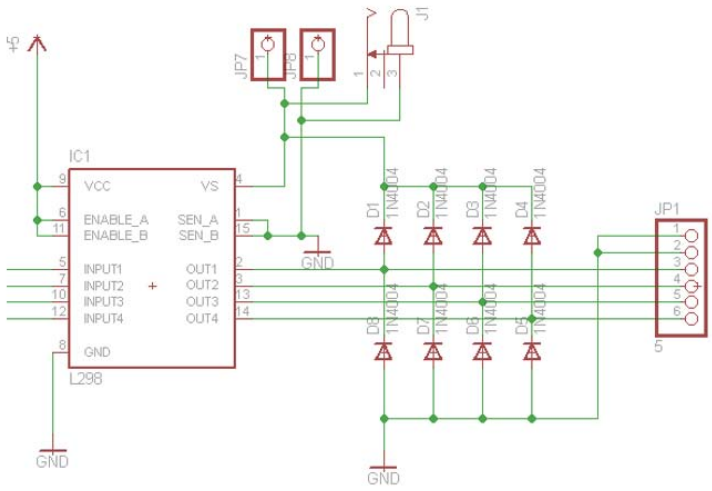

Figura 2.6: Circuito de la etapa de potencia del motor PAP.

Sensores: Se le agregaron sensores al dispositivo para determinar la posición de la plataforma, se decidió ocupar fotodiodos y fototransistores por su fácil manipulación y para nuestra aplicación es suficiente. Cabe mencionar que los emisores (led infrarrojo) deben ser montados en el carro de la impresora y los receptores en el riel de la misma, para poder controlar la posición del carro de impresora.

- 2 Leds Infrarrojo de 5 mm IR333C
- 2 Fototransistores de 5mm PT331C
- 2 resistencias de 330 Ohms de ½ watts
- 2 resistencias de 10 kOhms de ½ watts
- 2 micro-switch.

El circuito donde se encuentran los sensores de infrarrojo y micro-switch se muestra en la Fig. 2.7.

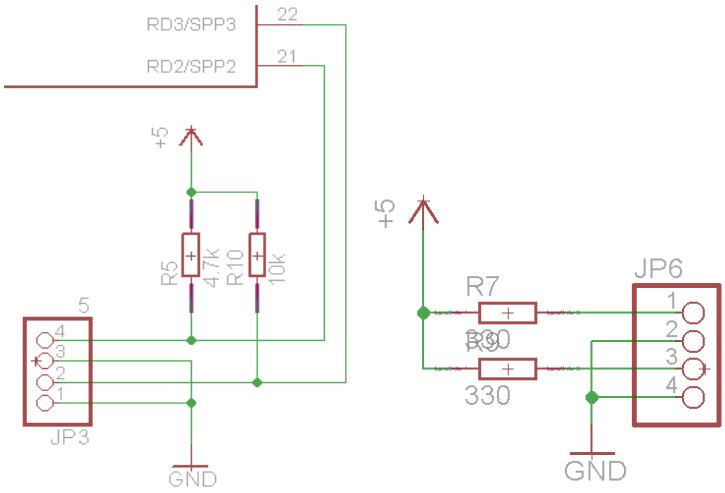

Figura 2.7: Circuito de los sensores que detectaran la posición de la placa.

2.2 Programación

Generamos un sistema de control en el cual el usuario va ingresar órdenes o datos directo en el ordenador, el cual va a servir de intermediario para comunicarnos con nuestro sistema como se muestra en la figura 2.8.

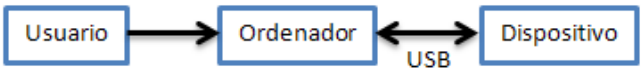

Figura 2.8: Esquema general del Dispositivo.

Para ello usamos una interfaz USB de clase HID para comunicar el dispositivo con el ordenador en el cual tenemos que generar un programa de control (HID) y una aplicación (API) para que comunique el ordenador con el dispositivo.

2.2.1 Generación del programa HID para el microcontrolador

El programa HID es la base principal de nuestro sistema de control, el cual estará contenido en el microcontrolador PIC18F4550 para recibir instrucciones directas del ordenador por medio de la interfaz de USB, además de realizar las funciones básicas de nuestro sistema para el control del desplazamiento de la muestra. En el siguiente diagrama de flujo (Figura 2.9) se puede observar el proceso básico del programa.

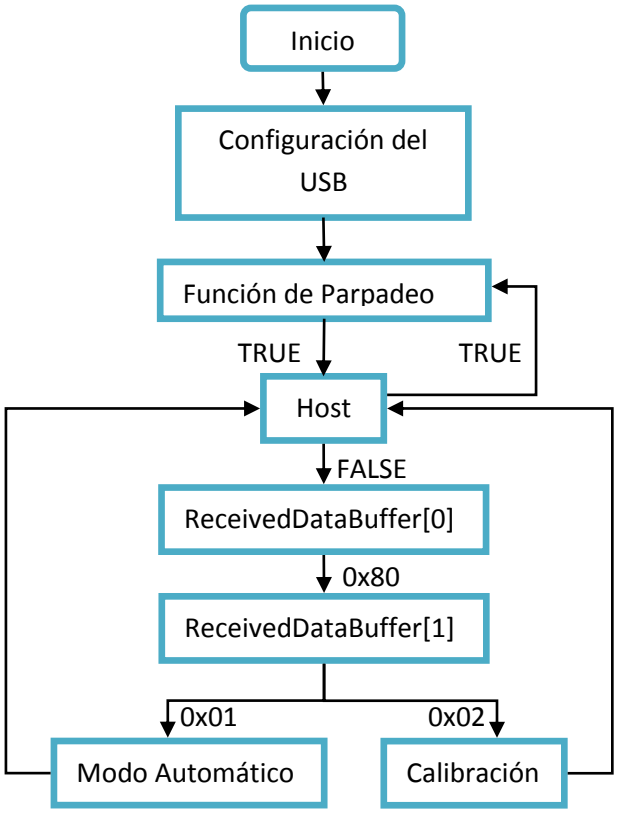

Figura 2.9: Diagrama de la estructura del programa HID.

Para el desarrollo de nuestra programación utilizamos el lenguaje C con el programa MPLAB IDE 8.92 utilizando como compilador MPLAB C18 Compiler 3.42, además se ocupó el demo USB Device - HID - Simple Custom Demo - C18 - PICDEM FSUSB de Microchip Solutions v2010-10-19 como plantilla para el desarrollo del programa.

Configuración del proyecto en MPLAB C18 Compiler

La configuración de las librerías en MPLAB es crucial para que nuestro programa funcione correctamente. Antes de eso es necesario instalar el compilador C18 para MPLAB y la paquetería de Microchip Solutions v2010-10-19 la cual se puede encontrar de manera gratuita en la página oficial del fabricante del microcontrolador. [11]

A continuación se detallan los pasos necesarios para configurar el programa:

Primero debemos de abrir el proyecto del ejemplo USB Device - HID - Simple Custom Demo - C18 - PICDEM FSUSB el cual se encuentra en la dirección **C:\Microchip Solutions v2010-10-19\USB Device - HID - Custom Demos\Generic HID – Firmware**

Ya que MPLAB esté abierto, nos vamos a la barra de menú en **Proyetc ->Build Options -> Proyect** en la pestaña **Directories** cambiaremos la opción por **Intermediary Directory** como se muestra en la figura 2.10. Aquí agregaremos la dirección donde se encuentra

nuestro proyecto que en este caso será**: C:\Microchip Solutions v2010-10-19\USB Device - HID - Custom Demos\Generic HID – Firmware**.

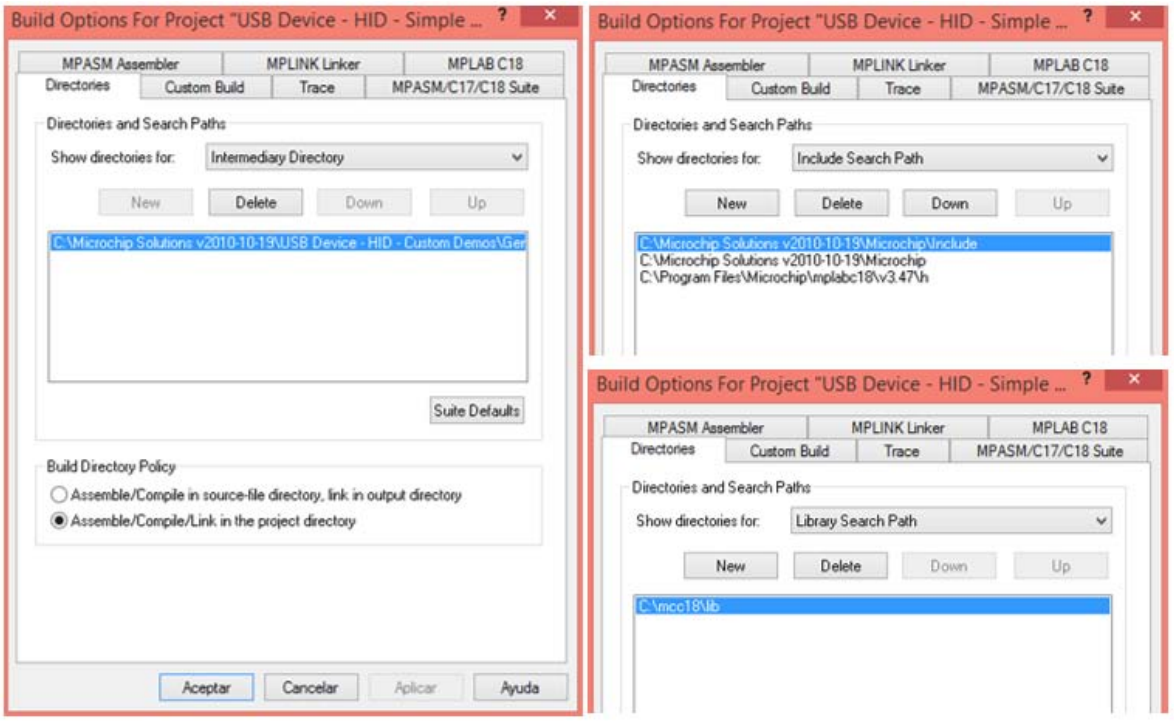

Figura 2.10 : Configuraciones de la ventana "Build Options for Proyect".

Después cambiaremos otra vez la opción de **Show directories for** por **Include Search Patch,** donde incluiremos las siguientes direcciones: **C:\Microchip Solutions v2010-10-19\Microchip\Include**, **C:\Microchip Solutions v2010-10-19\Microchip** y **C:\Program Files\Microchip\mplabc18\v3.47\h**.

Otra vez cambiaremos la opción de **Show directories for** por **Library Search Patch** donde agregaremos la dirección **C:\mcc18\lib**. Ahora solo queda darle en la opción de Aceptar.

Ahora verificamos si nuestras herramientas de MPLAB están bien configuradas debido a que este proyecto es del 2010 y han hecho varias modificaciones y varias versiones de las herramientas de microchip, y las direcciones de estas herramientas suelen cambiar dependiendo la versión. Para ello seleccionamos en **Proyect -> Select Language ToolSuite…** donde verificaremos que el compilador C18 este activado, y la dirección de las herramientas sea la correcta en caso de que no lo sea aparecerá con una "x" roja del lado izquierdo o si la dirección está incompleta no aparecerá el número de la versión. Como se muestra en la siguiente figura 2.11 así deberán de aparecer nuestras configuraciones.

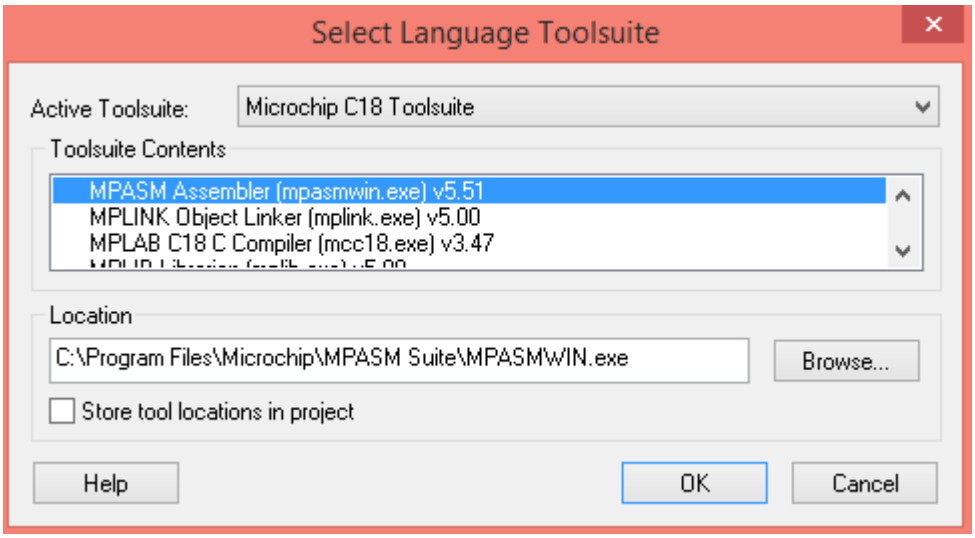

Figura 2.11: Configuracion de las herramientas de compilacion.

En nuestro caso, las herramientas utilizadas contienen las siguientes direcciones (ver Fig. 2.12):

Toolsuite	Direccion
MPASM Assember v5.51	C:\Program Files\Microchip\MPASM Suite\MPASMWIN.exe
MPLINK Object Linker v5.00	C:\Program Files\Microchip\mplabc18\v3.47\bin\mplink.exe
MPLAB C18 v3.47	C:\Program Files\Microchip\mplabc18\v3.47\bin\mcc18.exe
MPLIB Librarian v5.00	C:\Program Files\Microchip\mplabc18\v3.47\bin\mplib.exe

Figura 2.12: Tabla de las direcciones de herramientas de MPLAB.

Ahora ya podemos compilar nuestro proyecto para verificar que esta bien configurado.

Archivos del Proyecto de MPLAB C18 Compiler

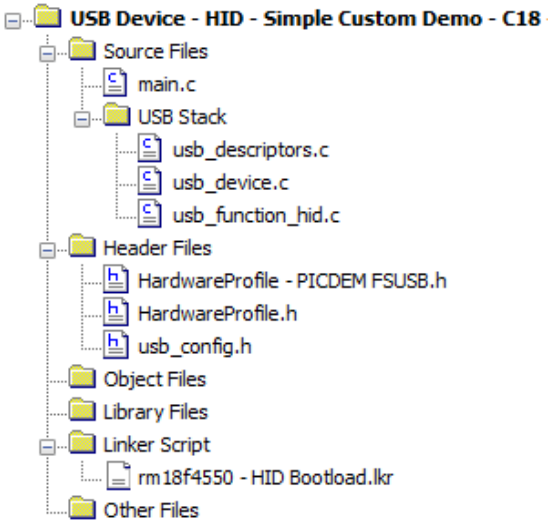

Figura 2.13: Estructura del proyecto en MPLAB.

El proyecto está compuesto por varios archivos como se muestra en la figura 2.13, los cuales configuran o ejecutan cierta parte de la programación. En este trabajo solo mostraran los archivos que hemos modificado pero los demás se pueden consultar en la página de microchip.

usb_descriptors.c: En este archivo configuramos los descriptores de nuestra interfaz USB. Como se puede ver en el apéndice A, se definió clase HID, con la versión 1.11 con subclase "costum", con un VID de 0x0461 y un PID 0x000A, el cual contiene solo una configuración, con dos end point de 64 bytes y reportes de 8 bits.

HardwareProfile –PICDEM FSUSB.h: En esta parte del programa definimos las entradas y salidas de nuestro sistema. En el apéndice B se puede ver estas configuraciones

HardwareProfile.h: En este archivo definimos el microcontrolador Motorola PIC-18F4550 así como sus funciones, este código puede verse en el apéndice C.

main.c: El archivo main.c es la base del programa en donde se controlan los puertos, y el modulo USB, básicamente es el que realiza todas las acciones de control para hacer funcionar todo el sistema.

Inicio del programa HID

Básicamente nuestro programa se centró en el archivo main.c donde agregamos todas las instrucciones para la aplicación de nuestro proyecto como se verá a continuación.

Incluir Bibliotecas: En esta primera parte del programa se incluyen los recursos que van a ocupar nuestros programas como archivos de apoyo, en nuestro caso agregamos:

```
#include "./USB/usb.h"
#include "HardwareProfile.h"
#include "./USB/usb_function_hid.h"
#include <delays.h>
```

Fuses configuraciones: Aquí se activan o desactivan los módulos que se van a usar de nuestro micro-controlador según sea el caso. Estos módulos se encargan de hacer cosas específicas como ayudar en la protección del hardware, software o en la comunicación entre otras funciones.

Unos de los fuses más importantes son los encargados de manipular la frecuencia de oscilación de nuestro microcontrolador y el USB como se muestra abajo.

```
#pragma config PLLDIV   = 5
#pragma config CPUDIV   = OSC1_PLL2
#pragma config USBDIV   = 2
#pragma config FOSC     = HSPLL_HS
```

Donde podemos ver que la frecuencia de entrada que tenemos es la del cristal de 20 MHz en la cual el *prescaler* divide por 5 para obtener 4 MHz en la entrada del PLL el cual produce 96 MHz de salida, después esta frecuencia es divida entre 2 para alimentar al USB con una frecuencia de 48 MHz, por el otro lado la frecuencia del PLL alimenta al *postcaler* para determinar la frecuencia de trabajo, que en nuestro caso es entre 2 obteniendo una frecuencia igual al USB.

Definición de variables y funciones: Ahora definimos las variables a usar, las cuales nos servirán para manipular, hacer operaciones y guardar datos de nuestro programa.

```
unsigned char ReceivedDataBuffer[64]; //end point de entrada
unsigned char ToSendDataBuffer[64]; // end point de salida
void moveright(); //funcion de motor PAP giro a la derecha
void moveleft(); // funcion de motor PAP giro a la izquierda
int tiempo=255; // variable para los retardos
USB_HANDLE USBOutHandle = 0;
USB_HANDLE USBInHandle = 0;
BOOL blinkStatusValid = TRUE;
```

Una de las variables más importantes son las que contienen los end point de entrada y salida, las cuales están definidas por ReceivedDataBuffer y ToSendDataBuffer con un tamaño de 64 bytes, que deben de coincidir con el buffer de la API para el buen funcionamiento de la interfaz y evitar la pérdida de datos, ya que estos son los encargados de comunicarse con el Host del ordenador.

Inicialización del USB: Es necesario activar la interfaz USB después de haber configurado todos los parámetros, desde las configuraciones básicas del programa, hasta las configuraciones del USB y sus interrupciones para poder usarla. El USB se activa con la función USBDeviceInit().

Declaración de Funciones: En esta parte del programa especificamos las acciones a realizar de nuestras funciones, las cuales son las siguientes:

void BlinkUSBStatus(void): En donde indicamos que se generará una rutina de parpadeo, la cual nos indica si el dispositivo USB está siendo enumerado por el host.

void retardo(): Esta función será la encargada de definir el ancho de pulso de las señales enviadas a nuestro motor PAP, que básicamente consiste en un retardo que depende de la variable tiempo, la cual es ingresada por el usuario mediante la interfaz USB.

```
{
for (i=1;i<=1;i++)//ReceivedDataBuffer[2]
{
Delay10KTCYx(tiempo);
i++;}}
```

void moveright(): Encargada de mover un motor de pasos hacia la derecha, la cual contiene medio pasos para aumentar al doble la precisión de nuestro motor, incluyendo un ciclo condicional while ((sw1&&sw2)==1)el cual nos indica que mientras que el dato 0 y 1 de nuestro puerto D tenga un valor de uno, se seguirá repitiendo el ciclo indefinidamente, para no quedar encerrados en este ciclo al final de la rutina de los pasos revisamos el buffer al final de la secuencia de pulsos para asesorarnos que nuestro dato no cambio con la ayuda de la instrucción if(!HIDRxHandleBusy(USBOutHandle)).

```
void moveright()
{
while ((sw1&&sw2)==1) { // Rutina de Medio paso
tiempo=ReceivedDataBuffer[3];
 PORTB= 0b10011001;
retardo();
PORTB= 0b10001000;
retardo();
PORTB= 0b11001100;
retardo();
PORTB= 0b01000100;
retardo();
PORTB= 0b01100110;
retardo();
PORTB= 0b00100010;
retardo();
PORTB= 0b00110011;
retardo();
PORTB= 0b00010001;
retardo();
if(!HIDRxHandleBusy(USBOutHandle)){
 switch(ReceivedDataBuffer[0]){
}
}
USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
}
}
```

void moveleft(): Encargada de mover el motor hacia la izquierda, estructurada igual que la función void moveright(), con la diferencia de la secuencia de pasos que son contrarios para mover el motor hacia el sentido contrario.

```
void moveleft()
{
while ((sw3&&sw4)==1){
  tiempo=ReceivedDataBuffer[3];
PORTB= 0b00010001;
retardo();
PORTB= 0b00110011;
retardo();
```

```
PORTB= 0b00100010;
retardo();
PORTB= 0b01100110;
retardo();
PORTB= 0b01000100;
retardo();
PORTB= 0b11001100;
retardo();
PORTB= 0b10001000;
retardo();
PORTB= 0b10011001;
retardo();
if(!HIDRxHandleBusy(USBOutHandle)){
 switch(ReceivedDataBuffer[0]){
}
}
USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
}
}
```

void ProcessIO(void): Esta función incluye la estructura principal de nuestro programa ya que aquí es donde se envían y reciben los datos en el Host, indicando que hacer en cada caso.

Rutina de Parpadeo Después de iniciar la función void ProcessIO(void) iniciamos la rutina de parpadeo para poder corroborar la buena comunicación de nuestro dispositivo con el ordenador. Y confirmar que el dispositivo HID se haya enumerado con el host del ordenador.

```
if(blinkStatusValid)  {
   BlinkUSBStatus();
  }
```

Después inicializamos las interrupciones con las que trabajara el USB y salimos de la rutina de la función BlinkUSBStatus(), después de haber confirmado la buena comunicación de la interfaz.

```
if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1)) return;
  blinkStatusValid = FALSE;
  sLED_1_Off();
  sLED_2_On();
```

Revision de Host: Para revisar lo que contiene el Host mandamos a llamar a la instrucción `if(!HIDRxHandleBusy(USBOutHandle))`
Después realizamos una toma de decisiones con la declaración switch-case. Donde condicionamos a nuestro programa con el primer elemento del end point ReceivedDataBuffer[0], para poder indicar si nuestro reporte es de entrada o de salida.

Recepción de datos: Leemos los datos contenidos en el host, y si el dato ReceivedDataBuffer[0] tiene un valor numérico de 0x80, nuestro dispositivo recibirá el reporte. Este número también indica el número de reporte.

```
if(!HIDRxHandleBusy(USBOutHandle)){// Checa si hay un dato recibido del Host
switch(ReceivedDataBuffer[0]){// Condicionamos el primer dato de nuestro reporte para verificar si es un dato de entrada
case 0x80:  // Señal de entrada
```

En la siguiente parte del programa tomaremos el dato ReceivedDataBuffer[1], para condicionar nuestro programa con if-else y decidir qué acción queremos realizar con los datos recibidos.
Si nuestro segundo elemento del end point ReceivedDataBuffer[1] tiene el valor 0x01 la acción que realizaremos será realizar todo el proceso de manera automática.

```
if(ReceivedDataBuffer[1]==0x01){ // Modo Automatico
 moveright(); //Mueve hacia Arriba
 retardo(); //Retardo de un segundo
 moveleft(); //Mueve hacia abajo
   }
```

Si nuestro segundo elemento del end point ReceivedDataBuffer[1] tiene el valor 0x02 la acción que realizaremos será la calibración de nuestro dispositivo.

```
else if(ReceivedDataBuffer[1]==0x02){ //Modo de reinicio o Calibracion
 if((sw1||sw2==0)&&(sw3||sw4==0)){
PORTB= 0b00000000;//Mantener motor ahi
   }
else if((sw1||sw2==0)&&(sw3||sw4==1)){
moveleft(); mover hacia arriba
   }
else if((sw1||sw2==1)&&(sw3||sw4==0)){
PORTB= 0b11111111;  //Mantener el motor ahi
   }
else if((sw1||sw2==1)&&(sw3||sw4==1)){
moveleft();//mueve el motor hacia arriba
   }
 PORTB=0b11111111;
   }
```

Envío de datos: En caso de que el dato ReceivedDataBuffer[0] tenga un valor numérico de 0x81, automáticamente enviará los datos del reporte al host de nuestro ordenador para que después sea leído. Debemos mencionar que en nuestra aplicación no enviamos reportes de datos, pero se menciona en este proyecto como información básica para la comunicación USB.

```
case 0x81:  //Get push button state
      ToSendDataBuffer[0] = 0x81;
```

ToSendDataBuffer[1] = (sw1 == 1)? 0x01: 0x00;

2.2.2 Generación de la aplicación API

Nuestra aplicación se apoyó en la librería HIDAPI, la cual tiene toda la información adecuada para interactuar con un dispositivo HID además de que no es necesario conocer a groso modo toda la librería para su uso. Otra de nuestras grandes ventajas es que la podemos ocupar en Windows, Linux y Mac OS X, pero para cada sistema operativo hay que crear un archivo del sistema para poder configurarla. Esta aplicación fue desarrollada en C++ de Visual Studio Express 2010. En el diagrama (Figura 2.14) se muestra la estrutura de nuestra aplicación.

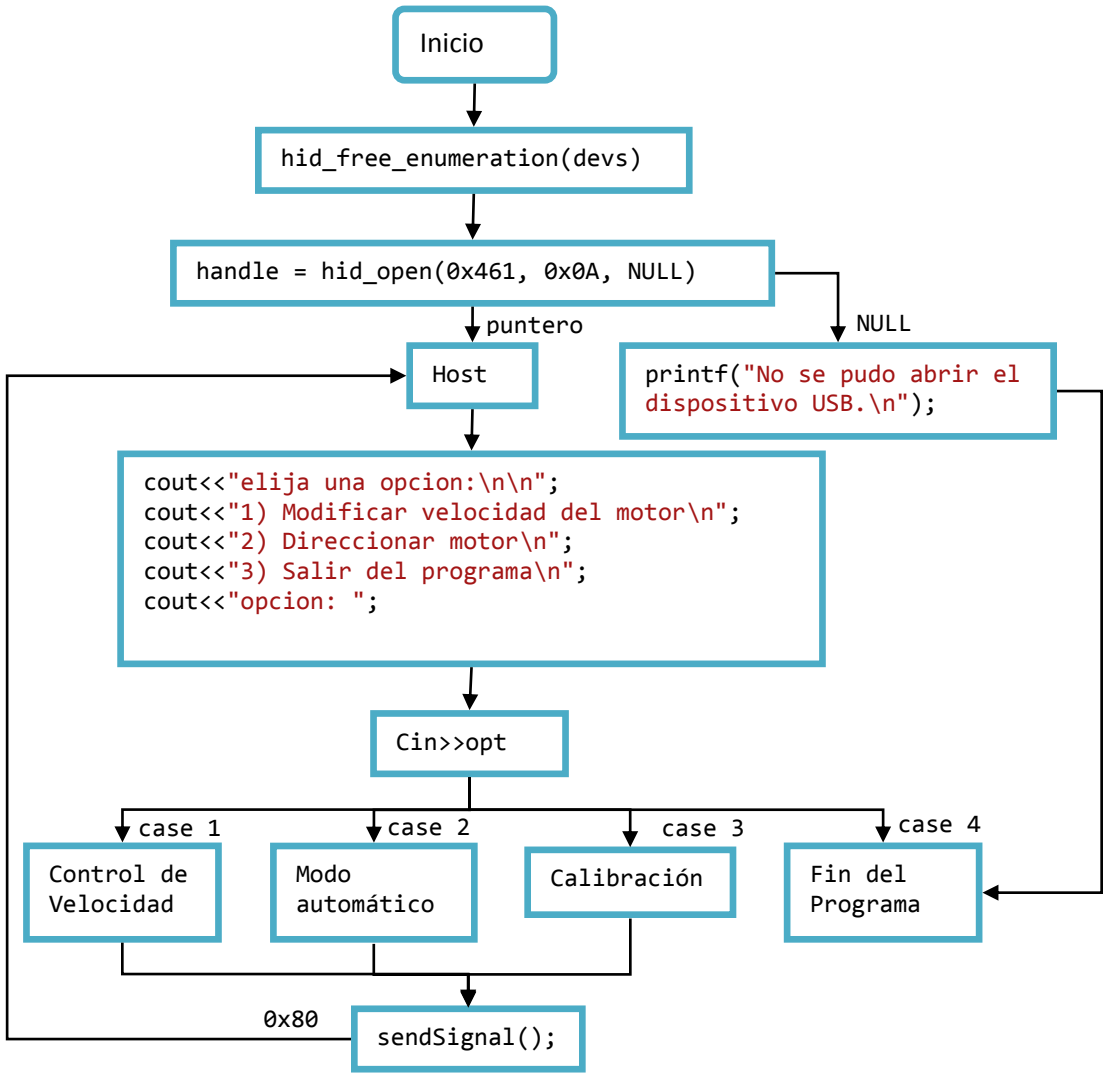

Figura 2.14: Diagrama conceptual de la aplicación API.

Incluir bibliotecas: En nuestro API agregamos la librería "hidapi.h" que es la encargada de comunicarse con el dispositivo de USB. La librería iostream es para las funciones básicas de programación.

```
#include <iostream>
#include "hidapi.h"
using namespace std;
```

Definición de variables: Definimos las variables que vamos a usar para el buffer, y las de velocidades y retardos de nuestro programa.

```
unsigned char inBuf[65], outBuf[65]; // variables del buffer
unsigned char vel1[8];//variables de velocidades
#define MAX_STR 255
wchar_t wstr[MAX_STR];
hid_device *handle;
int vel,x;
float vel2;
```

Declaración de funciones: Declaramos solo dos funciones que son para facilitar la escritura al enviar o leer datos. La función getState () se usa en la lectura de datos y la sendsignal() para el envío de datos.

```
int getState(){
  inBuf[0] = 0;    // primer byte siempre = 0 para dispositivos que solo aceptan un informe
  inBuf[1] = 0x81;  // segundo byte = Repor number (Read)
  hid_write(handle, inBuf, 65);
  // Read requested state
  return hid_read(handle, inBuf, 65);
}

/** Send an Output report to write the state (cmd 0x80)*/
int sendSignal(){
  outBuf[0] = 0;    // primer byte siempre = 0
  outBuf[1] = 0x80; // segundo byte = Repor number (write)
  return hid_write(handle, outBuf, 65);
}
```

Enumeración de dispositivos de clase HID USB: Nuestro programa busca todos los dispositivos conectados por USB de la clase HID y mandará a imprimir en pantalla la cantidad de dispositivos HID encontrados. En caso de que no encuentre ningún dispositivo mandara a pantalla cero dispositivos encontrados.

```
  devs = hid_enumerate(0x0, 0x0);
  cur_dev = devs;
  int numDevsFound = 0;
  while (cur_dev) {
    numDevsFound++;
    cur_dev = cur_dev->next;
```

```
}
hid_free_enumeration(devs);

printf("Dispositivos HID encontrados: %i\n", numDevsFound);

if(!numDevsFound){
  printf("\n");
  system("pause");
  return -1; }
```

Abrir dispositivo usando el VIP y PID: Después abrimos nuestro dispositivo HID para poder establecer la comunicación USB, lo identificamos ya que a la API le proporcionamos el VIP y PID información esencial del dispositivo. En caso de que no coincidan los datos con el dispositivo, mandamos un mensaje de advertencia de que no se encuentra el dispositivo seleccionado.

```
handle=hid_open(0x461, 0x000A, NULL);//Modificar el VID y PID Microchip (0x4d8,0x0x3f)
 if(!handle){
   printf("No se pudo abrir el dispositivo USB.\n");
   system("pause");
   return -1;
 }
```

Opciones de Manipulación: En esta parte condicionamos la variable opt con un switch-case en la cual se pueden seleccionar apretando las teclas 1 a 4 para seleccionar cualquiera de las acciones que queremos que realice nuestro programa, ya sea calibración, ajuste de velocidad o modo automático.

```
int opt;
 do{
   cout<<"elija una opcion:\n\n";
   cout<<"1) Control de velocidad\n";
   cout<<"2) Modo Automatico\n";
   cout<<"3) Calibracion\n";
   cout<<"4) Salir del programa\n";
   cout<<"opcion: ";
   cin>>opt;
```

A continuación se describe cómo se definieron las variables del sistema.

1. Modificar velocidad: En esta parte el usuario agrega la velocidad deseada, la cual debe estar dentro del intervalo de 400 μm a 10,000 μm y se almacenará en un vector de 8 bytes. Estas velocidades están calculadas considerando que el motor está conectado de manera directa a la polea sin sistemas de engranes, como se muestra en el diagrama físico.

```
case 1: //READ
            cout<<"\nEscribe la velocidad requerida (Desde 400um a 10 000um)";
                  cin>>vel1;
```

Para poder enviar la velocidad al microcontrolador es necesario realizar ciertos cálculos. Primero hay que transformar la velocidad en números, ya que lo que el usuario envía con el teclado del ordenador son caracteres. Para ello revisamos el código ASCII el cual nos ayudara a traducir nuestra velocidad. Como se vió en la declaración de variables del programa, la variable vel1 la definimos como un vector de 8 bytes, ocasionando que la información enviada por el usuario se guarde en este vector, de tal manera que cada cifra de la velocidad o carácter enviado por el usuario este contenido en un byte diferente del vector vel1.
Esto nos permitirá aplicar la transformación, a cada uno de las cifras para después solo juntarlas y nos dará la velocidad real enviada por el usuario en números enteros.

```
vel= 10000*(vel1[0]-48)+1000*(vel1[1]-48)+100*(vel1[2]-48)+10*(vel1[3]-48)+vel1[4]-48;
```

Es necesario saber que nuestro programa HID recibirá el dato para poder modificar la velocidad del motor, el detalle es que el microcontrolador espera el valor de la variable tiempo la cual es de tipo int y solo acepta un valor de 8 bits (0 a 255), la cual modifica directamente a los retardos, incluidos en las rutinas que contienen la serie de pulsos para mover a nuestro motor. Para calcular la variable tiempo, usamos las siguientes líneas de programación:

```
vel2 = 94270/vel;  //representa la variable que modifica nuestro retardo, sin redondeo
x=floor(vel2+0.5); //variable que modifica nuestro retardo redondeada
outBuf[4]=x; //guardamos x en el buffer de salida
res = sendSignal(); //Envio de datos
system("CLS");
if(res>0)
cout<<"**** Se ha cambiado la velocidad satisfactoriamente ****\n\n"<<x;
else
cout<<"ERROR! No se pudo modificar la velocidad \n";
break;
```

Donde la primera expresión, contiene vel2 la cual contendrá el valor de nuestra variable tiempo y se calcula dividiendo 122522 entre la velocidad ingresada por el usuario. La constante 122522 se deduce de la ecuación de la velocidad lineal:

$$v = \frac{d}{t}, \qquad (2.1)$$

Donde d es la distancia, y t es el tiempo. Para nuestro arreglo, tomamos como distancia (d) el perímetro de engranaje del motor el cual tiene un diámetro de 13 mm. Es decir, la distancia recorrida en una vuelta será de 40.84 mm.

$$d = \pi * D = 2\pi(13\ mm) = 40.84mm = 40840\ \mu m$$

Y t lo tomamos como el tiempo que tarda en dar una vuelta el motor PAP, para su cálculo hemos considerado que el motor tiene una resolución o ángulo de paso de 1.8° y va estar operando con medio paso es decir se va mover al doble de los pasos calculados.

$$(No.\,de\ Pasos) = \frac{2(360°)}{Angulo\ de\ paso} = \frac{720°}{1.8°} = 400\ pasos$$

Donde tenemos que t será igual al número de pasos realizados en una vuelta, multiplicado por el tiempo que tarde un retardo en ejecutarse:

$$t = 400 * Retardo\ , \qquad (2.2)$$

donde t es el tiempo que tarda en dar una vuelta el motor y Retardo es el tiempo que dura un pulso.

Un retardo se calcula con la ecuación 2.3:

$$Retardo = 10kTCY * i, \qquad (2.3)$$

donde 10k es un múltiplo de 10,000 (ver en la biblioteca delays.h) y TCY es el tiempo en que se ejecuta una instrucción del microcontrolador y se calcula dividiendo 4 entre la frecuencia del oscilador o de trabajo (Fosc) e i es un dato que va en el intervalo de 0 a 255.

$$TCY = \frac{4}{Fosc} = \frac{4}{48MHz} = 8.33x10^{-8}s$$

Nos damos cuenta que la variable i es la misma que nosotros definimos como tiempo en el programa HID, entonces esta variable es la que queremos calcular. Para ello despejamos *i* de la ecuación 2.3, y sustituimos retardo por t en función de distancia y velocidad lineal, para los cálculos en programación.

$$i = \frac{d}{400*10kTCY*v}\ , \qquad (2.4)$$

De donde obtenemos la relación entre la variable i y la velocidad lineal:

$$i = \frac{40840um}{(400)(10\ 000)(8.33x10^{-8}s)v} = \frac{122522\ um/s}{v}, \qquad (2.5)$$

2. Direccionar Motor: Solo se envía un dato con valor 1 al seleccionar la opción al host para mandar a la rutina de modo automático como se vio en el programa del microcontrolador.

```
case 2:  //TOGGLE (implica un write)
      // El puerto de salida comienza en el byte 2
```

```
outBuf[2] = 0x01;//Configuracion Automatica

res = sendSignal();

system("CLS");
if(res>0)
  cout<<"**** Se ha activado el modo Autoomatico ****\n\n";
else
  cout<<"ERROR! No se pudo conectar con el dispositivo.\n";
break;
```

3. Calibración: Se manda un dato al Host con valor de 2, el cual le va indicar al dispositivo efectuar la rutina de calibración del dispositivo como se vio en el programa del microcontrolador.

```
case 3:  //TOGGLE (implica un write)
      // El puerto de salida comienza en el byte 2
      outBuf[2] = 0x02;//Configuracion de calibracion

      res = sendSignal();

      system("CLS");
      if(res>0)
        cout<<"**** Se ha activado el modo de Calibracion ****\n\n";
      else
        cout<<"ERROR! No se ha podido comunicar con el dispositivo.\n";
      break;
```

4. Salir del Programa: Al acabar de configurar salimos del programa, esto no afectará en nada a lo que está ejecutando en nuestro dispositivo.

```
default:
      cout<<"opcion no valida\n";
   }//end of SWITCH
  }while(opt !=4);
  system("pause");
return 0;
```

2.3 Simulación y generación del circuito impreso

Antes de diseñar nuestro circuito impreso, decidimos simularlo en el programa Isis de Proteus para comprobar su buen funcionamiento tanto en la interfaz como los puertos de entrada y salida, el cual cuenta con módulos virtuales para la simulación de la interfaz USB, y de todos nuestros componentes. El diagrama correspondiente se muestra en la Figura 2.15.

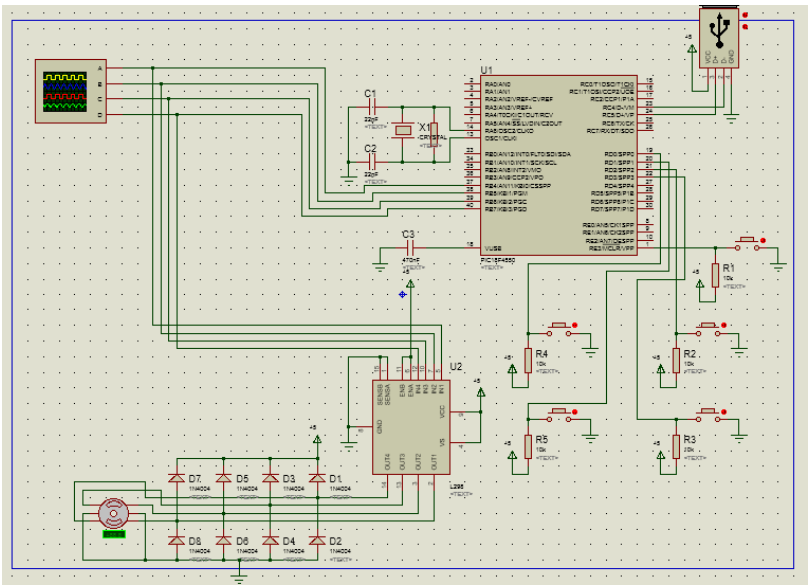

Figura 2.15: Diagrama del circuito simulado en Proteus.

El diagrama del circuito de este proyecto o esquemático fue creado en Eagle 7.1.0 en su versión gratuita, debido a que contiene todos los componentes a usar, facilitando el trabajo de diseño. En la figura 2.16 se muestra el esquemático de nuestro circuito.

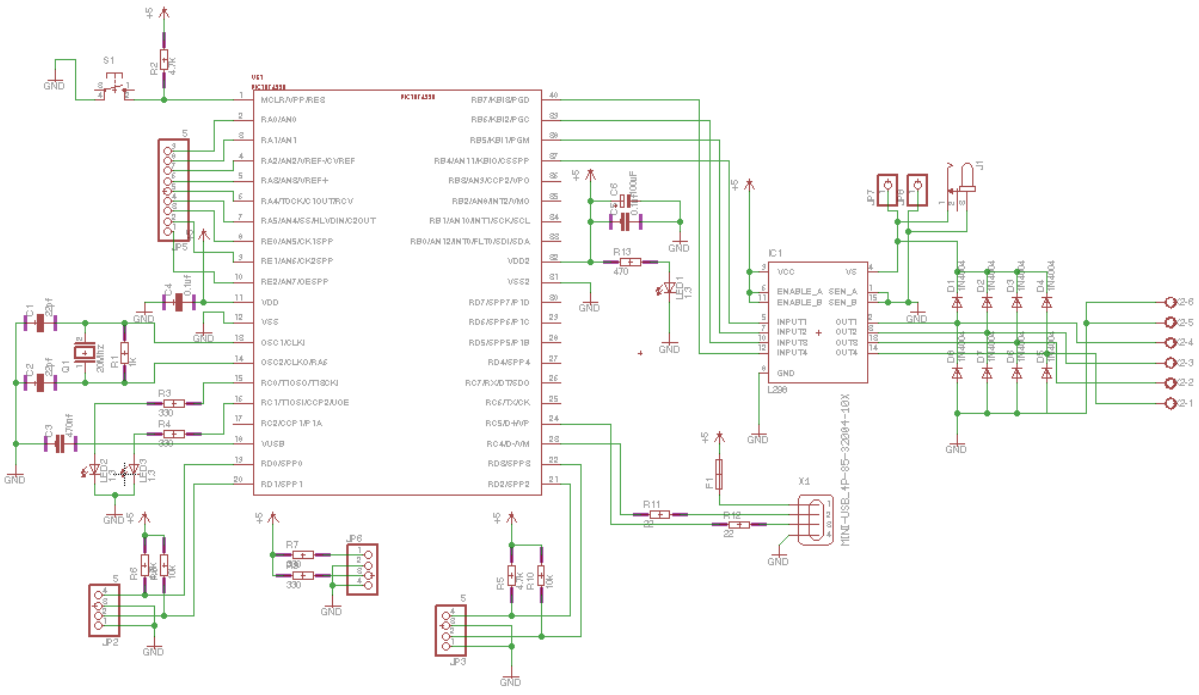

Figura 2.16: Diagrama del circuito eléctrico

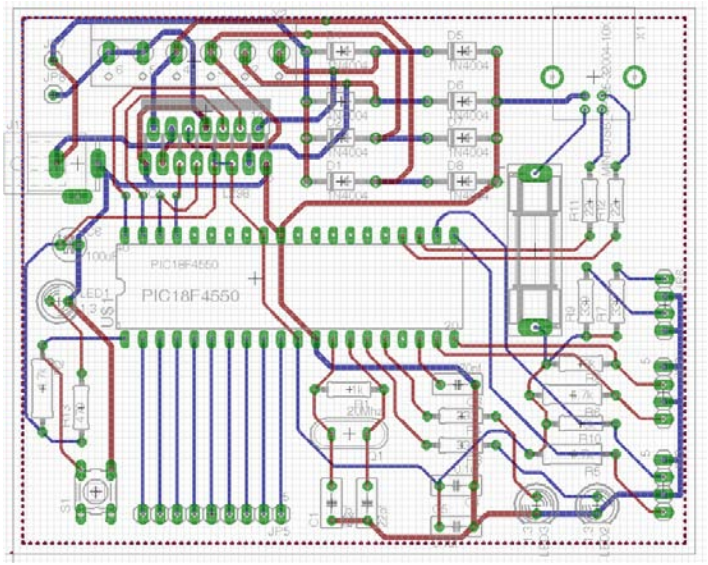

Figura 2.17 Diseño de PCB en Eagle

Después de haber generado el esquema, se diseñó la placa de circuito impreso o Printed Circuit Board (PCB) de 8x10 cm tomando en cuenta las reglas básicas de diseño de circuitos impresos [11], como podemos ver en la Figura 2.17.

A nuestro diseño le agregamos planos de tierra para mejorar la respuesta de las señales emitidas por el microcontrolador, además de evitar el ruido de las mismas (Figura 2.18 y Figura 2.19).

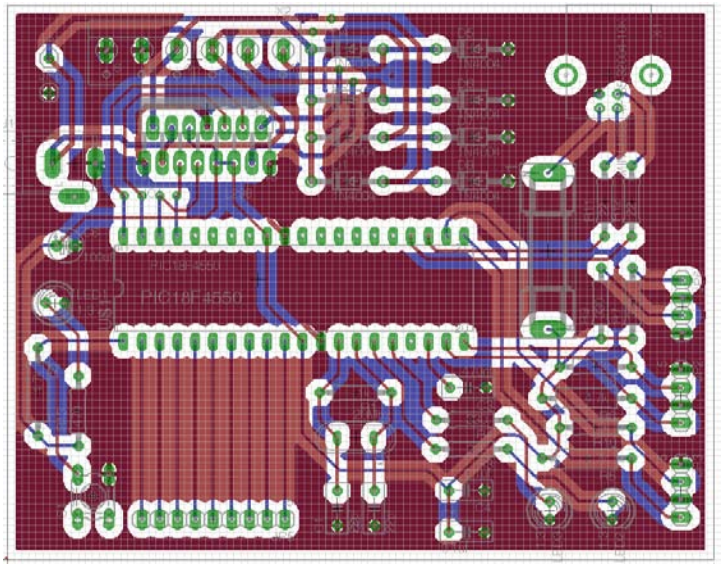

Figura 2.18: Diseño de PCB con Planos de Tierra en Eagle

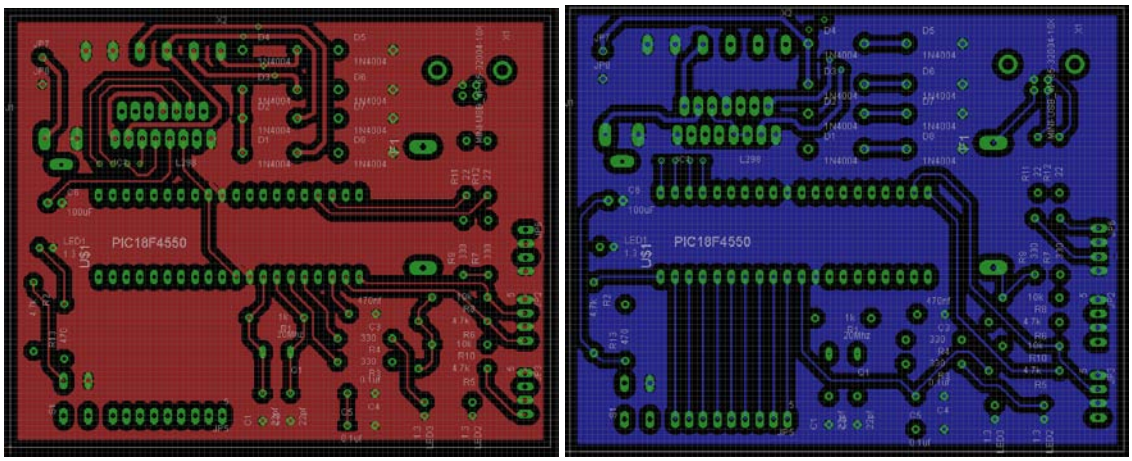

Figura 2.19: Capas superior e inferior (TOP y BOTTON) con planos de Tierra.

En el laboratorio de Electrónica del CCADET imprimimos la PCB del proyecto, donde ocupan el software de PCad versión 2006 para el manejo del equipo PROTOMAT S62 (Figura 2.20), que es una máquina de CNC encargada de la impresión de las PCBs. Desafortunamente PCad no recibe soporte técnico, además la biblioteca de archivos electrónicos es muy limitada.

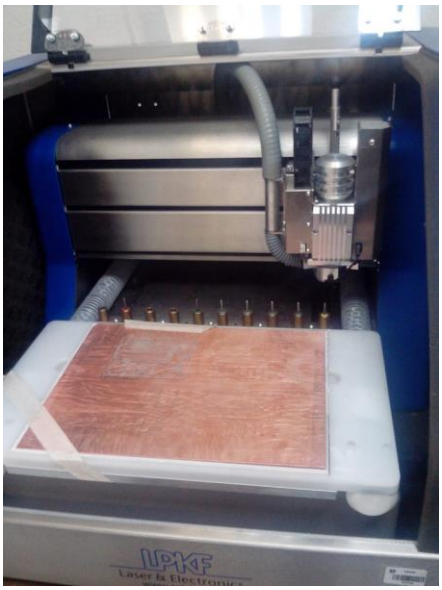

Figura 2.20: Equipo PROTOMAT S62

Tuvimos que generar los Gerber compatibles de tres de nuestras capas en Eagle y después los visualizamos en circuitCAM para verificar que se generaron de manera correcta los archivos. Los archivos generados fueron cuatro:

.GBL: Donde se encuentra la información de los Pads y pistas de la capa Bottom generada en Eagle(Figura 2.21).

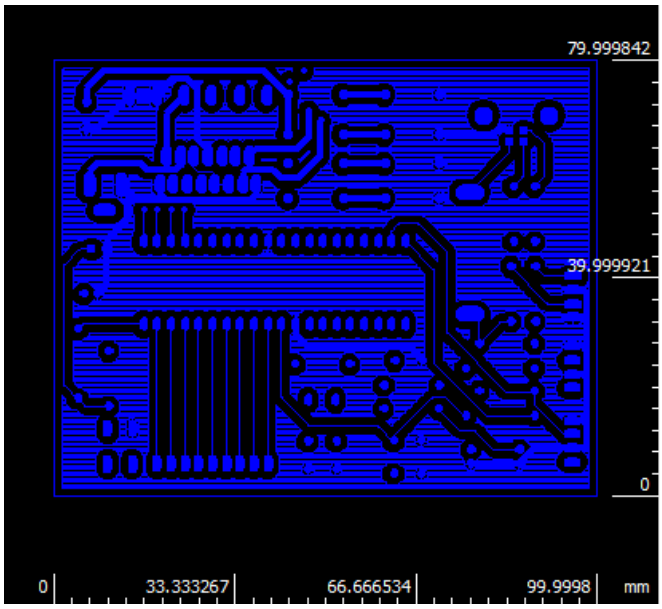

Figura 2.21: Capa botton visualizado en circuit CAM

.GTL: se encuentra la información de los Pads y pistas de la capa Top generada en Eagle. (Figura 2.22)

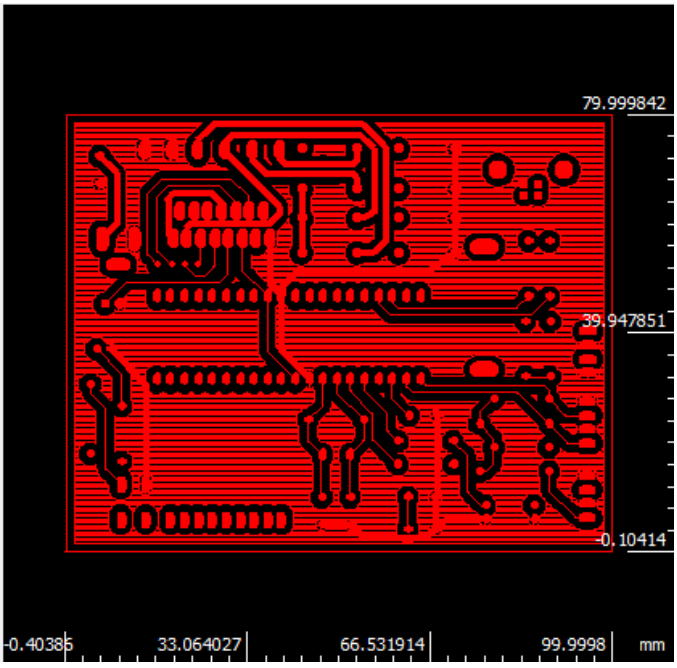

Figura 2.22 Capa top visualizado en Circuit CAM

.drl: Se encuentra la información de las herramientas de perforación

.drd y .dri: Donde se encuentra la ubicación y tamaño de los barrenos, donde se pondrán los componentes electrónicos (Figura 2.23).

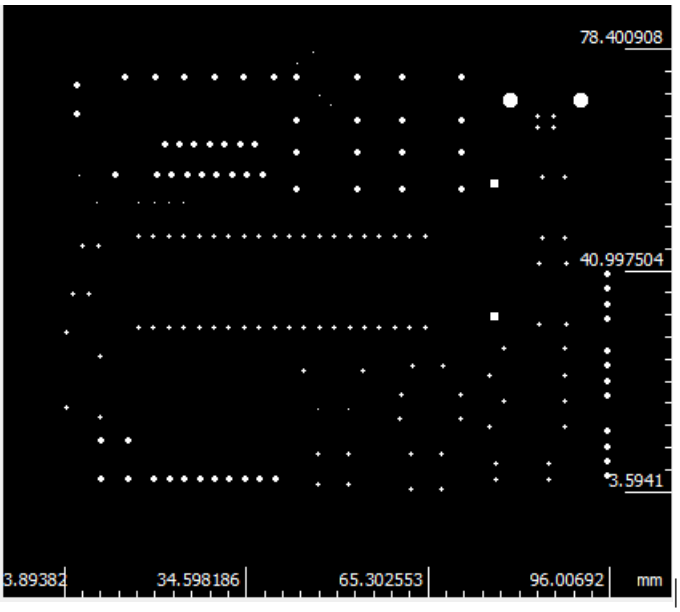

Figura 2.23: Capa de barrenos visualizada en Circuit CAM

Después de haber impreso nuestra PCB, corregimos fallos de fabricación y soldamos todos sus componentes en la misma. Después de haber obtenido el sistema electrónico, probamos los sensores, los micro-switchs y caracterizamos el motor Sanyo Denky 103-550-0149.

2.4 Caracterización de velocidades

Velocidades angulares

Se caracterizaron las velocidades angulares del motor PAP Sanyo Denky 103-550-0149, con carga, y sin carga (los resultados se muestran en el siguiente capítulo en la Tabla 3.1). Para las medidas con carga se utilizó la masa máxima a diferentes velocidades. La velocidad angular y la carga máxima son las variables que más nos importan para nuestra aplicación, ya que la velocidad lineal del dispositivo dependerá directamente de la velocidad angular del motor PAP, mientras la carga máxima nos indicará cuánta masa podrá soportar nuestro motor antes de perder el paso o la secuencia de pulsos. Cabe mencionar que todos los experimentos de las velocidades tanto angulares como lineales se repitieron 5 veces.

Para el cálculo de las velocidades angulares teóricas (W) se utilizó la siguiente ecuación de la velocidad angular.

$$W = \frac{2\pi}{t} \qquad (2.5)$$

Donde W es la velocidad angular calculada del motor PAP, 2π significa una vuelta del motor en radianes y t es el tiempo calculado para una vuelta del motor.

En donde t es igual a al tiempo de la ecuación 2.3. Sustituyendo obtenemos la ecuación 2.6. La cual se usó para calcular todas las velocidades teóricas en términos de i.

$$W = \frac{2\pi}{400*10kTCY*i} \qquad (2.6)$$

Para las velocidades angulares experimentales (W) sin carga y con carga (Wexp y Wcp) simplemente medimos cuanto tiempo tardaba en dar una vuelta, el engrane del motor con un cronómetro, variando la cantidad de pulsos introducidos por unidad de segundo, por medio de la variable i, la única diferencia fue que en las mediciones de Wcp le agregamos una determinada carga al motor. En donde W experimental del motor PAP se midió usando la ecuación de velocidad angular (ecuación 2.5), donde 2π significa una vuelta o un ángulo de 360° en radianes y t es el tiempo medido en una vuelta del engrane del motor.

En los experimentos de Wcp calculamos la carga máxima (Figura 2.24), es decir cuánta masa soportaba el motor antes de perder la secuencia de pasos a distintas velocidades, para verificar que nuestra velocidad no varía con respecto a la masa soportada. Para ello se ocupó el engrane de motor como polea donde amarramos un hilo el cual contendría un vaso de unicel de 5gr de masa. En él se depositaron canicas de una masa de 5gr a 10gr donde agregamos solo una cierta cantidad pesadas con una bascula de resolución de 2.5 gr antes de que el motor pierda la secuencias de pasos para tomar la medida, este proceso se hizo a diferentes velocidades debido a que la fuerza del motor o torque varía en cada una de ellas, es decir el motor no va soportar lo mismo a su velocidad máxima que ha su velocidad mínima.

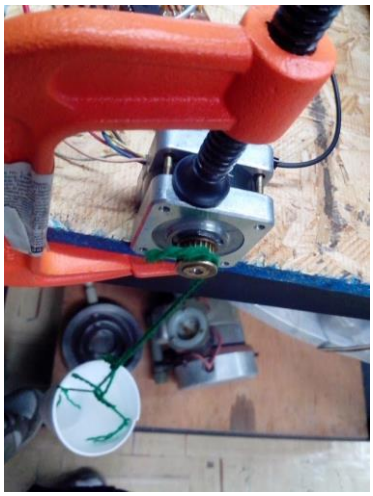

Figura 2.24: Montaje del experimento para las mediciones de velocidades con carga

Caracterización de velocidades lineales

La caracterización de las velocidades lineales es de suma importancia ya que nosotros verificamos que nuestros valores introducidos concuerdan con los valores medidos, para

ello tuvimos que montar el sistema físico (Figura 2.25). Es suma importancia decir que estos cálculos solo sirve para el sistema físico ocupado en este trabajo, ya que si se cambia o agregan engranajes, tornillos sin fin u otro componente mecánico, afectara directamente a la velocidad, incrementándola o disminuyéndola, según sea el caso.

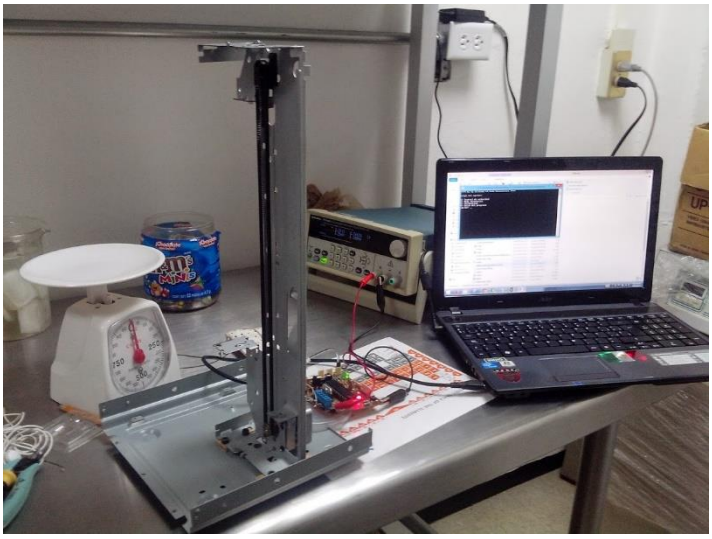

Figura 2.25: Montaje del sistema físico para la medición de velocidades

Para las velocidades lineales (V_t) teóricas fue necesario tomar las propiedades de nuestro sistema físico, donde podemos tomar como la distancia el perímetro interno del engrane, y el tiempo se puede calcular, ya que nosotros conocemos la cantidad de pulsos introducidos en una vuelta, como se vio en la ecuación 2.4 podemos despejar y obtener la ecuación 2.7.

$$v = \frac{d}{400*10kTCY*i}. \qquad (2.7)$$

Donde nos damos cuenta, que siempre la velocidad dependerá de la variable i la cual controlamos con el ordenador. Para medir las velocidades experimentales (Vexp) fue necesario montar el sistema físico del dispositivo, y medir cuanto tiempo tardaba en desplazarse hacia abajo el carro de la impresora en una longitud de 5 cm.

CAPITULO 3

RESULTADOS Y DISCUSIÓN

En este trabajo se obtuvo una PCB de un sistema de control de velocidades para un motor unipolar o bipolar PAP (figura 3.1). Esta posee las siguientes características:

- Conexión USB 2.0 de clase HID compatible con Linux, Windows y Mac.
- Entrada de alimentación externa de 3 a 20 Volts.
- Control de motor PAP de medio paso a 1 A de corriente eléctrica máxima.
- Botón de Reset del sistema.
- Tamaño de PCB de 8x10 centímetros.

En donde se le agregó planos de tierra para evitar cualquier ruido en la transferencia de datos, además podemos dividir nuestra PCB en dos regiones, la primera que incluye el microcontrolador, capacitores, oscilador, leds, etc., la cual tiene un tamaño de pista 0.4 mm para manejar 500mA de corriente máxima, y la segunda región la cual contiene el controlador para los motores PAP con un tamaño de pista de 0.8 mm para aguantar corrientes máximas de 1 A. Este circuito requiere de una conexión USB, y de una fuente de alimentación externa de 1 A, para su funcionamiento, cuenta con un fusible de 500 mA, para evitar quemar el dispositivo, o descomponer el ordenador en un corto circuito.

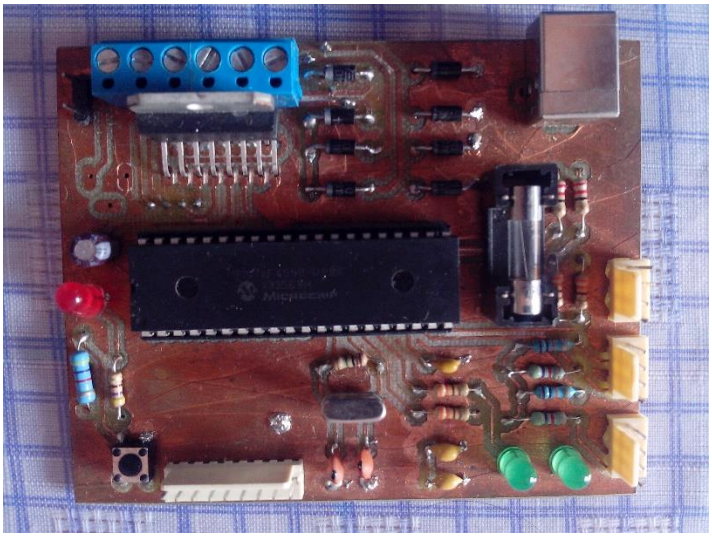

Figura 3.1: Circuito impreso para un sistema de control de velocidades.

Además se obtuvo una aplicación para comunicación del dispositivo con el ordenador compatible con Windows XP, 7 y 8.1 adaptable a Linux y Mac, que controla tres funciones de nuestro proceso: automático, calibración y variación de velocidad en un rango de 400-10,000 μm/s.

De la caracterización del motor PAP Sanyo Denky 103-550-0149 se obtuvieron los promedios de 5 repeticiones de experimentos de velocidades angulares sin y con carga (Wexp y Wcp), la masa máxima de la carga (M) soportada a distintas velocidades del motor PAP, y de las velocidades lineales (Vexp) del sistema físico. Los resultados de se muestran en la Tabla 3.1, que también muestra la variable i con la cual controlamos la velocidad, de acuerdo a los cálculos realizados en la programación del sistema.

Tabla 3.1: Tabla de velocidades angulares y lineales en función del parámetro i. Velocidades angulares sin carga: calculada (Wt) y experimental (Wexp), velocidad angular experimental con caga (Wcp), masa de la carga (M), velocidades lineales: calculada (Vt) y experimental (Vexp). Los valores de las velocidades experimentales corresponden al promedio de 5 repeticiones.

i	pulsos/s	Wt (rad/s)	W exp (rad/s)	Wcp (rad/s)	M (gr)	Vt(µm/s)	Vexp(µm/s)
255	4.7	0.0739	0.0745	0.0736	125.0	480.5	478.9
250	4.8	0.0754	0.0760	0.0752	125.0	490.1	490.2
245	4.9	0.0769	0.0775	0.0759	122.5	500.1	495.0
240	5.0	0.0785	0.0793	0.0638	122.5	510.5	508.1
235	5.1	0.0802	0.0805	0.0799	121.3	521.4	519.8
230	5.2	0.0820	0.0826	0.0820	123.8	532.7	527.4
225	5.3	0.0838	0.0843	0.0842	122.5	544.5	544.7
220	5.5	0.0857	0.0865	0.0792	123.8	556.9	555.6
215	5.6	0.0877	0.0884	0.0882	123.8	569.9	572.1
210	5.7	0.0898	0.0904	0.0908	125.0	583.4	586.9
205	5.9	0.0919	0.0930	0.0930	125.0	597.7	596.7
200	6.0	0.0942	0.0952	0.0952	126.3	612.6	615.8
195	6.2	0.0967	0.0974	0.0973	126.3	628.3	628.2
190	6.3	0.0992	0.1004	0.1001	125.0	644.9	642.7
185	6.5	0.1019	0.1028	0.1051	125.0	662.3	666.7
180	6.7	0.1047	0.1056	0.1069	121.3	680.7	688.7
175	6.9	0.1077	0.1089	0.1083	121.3	700.1	706.2
170	7.1	0.1109	0.1120	0.1106	123.8	720.7	720.5
165	7.3	0.1142	0.1151	0.1164	123.8	742.6	750.8
160	7.5	0.1178	0.1170	0.1195	125.0	765.8	769.2
155	7.7	0.1216	0.1192	0.1247	120.0	790.5	788.7
150	8.0	0.1257	0.1257	0.1293	120.0	816.8	819.7
145	8.3	0.1300	0.1293	0.1343	118.8	845.0	847.5
140	8.6	0.1346	0.1317	0.1372	120.0	875.2	877.3
135	8.9	0.1396	0.1384	0.1415	121.3	907.6	919.3
130	9.2	0.1450	0.1422	0.1468	123.8	942.5	943.4
125	9.6	0.1508	0.1500	0.1518	127.5	980.2	980.5
120	10.0	0.1571	0.1563	0.1579	131.3	1021.0	1020.6

115	10.4	0.1639	0.1628	0.1671	130.0	1065.4	1077.7
110	10.9	0.1714	0.1746	0.1795	127.5	1113.8	1116.2
105	11.4	0.1795	0.1817	0.1816	126.3	1166.9	1168.3
100	12.0	0.1885	0.1928	0.1928	126.3	1225.2	1231.7
95	12.6	0.1984	0.2034	0.2054	126.3	1289.7	1295.5
90	13.3	0.2094	0.2152	0.2123	125.0	1361.4	1366.4
85	14.1	0.2218	0.2236	0.2261	125.0	1441.4	1445.4
80	15.0	0.2356	0.2345	0.2381	123.8	1531.5	1506.2
75	16.0	0.2513	0.2524	0.2555	122.5	1633.6	1645.9
70	17.1	0.2693	0.2698	0.2809	122.5	1750.3	1773.4
65	18.5	0.2900	0.2910	0.2965	121.3	1885.0	1908.8
60	20.0	0.3142	0.3243	0.3208	121.3	2042.0	2066.7
55	21.8	0.3427	0.3573	0.3491	117.5	2227.7	2253.0
50	24.0	0.3770	0.3858	0.3881	117.5	2450.4	2500.0
45	26.7	0.4189	0.4249	0.4488	112.5	2722.7	2777.8
40	30.0	0.4712	0.4695	0.4914	113.8	3063.1	3125.0
35	34.3	0.5386	0.5331	0.5826	110.0	3500.6	3571.4
30	40.0	0.6283	0.6226	0.6562	116.3	4084.1	4166.7
25	48.0	0.7540	0.7592	0.7679	111.3	4900.9	5000.0
20	60.0	0.9425	1.0173	1.0173	108.8	6126.1	6250.0
15	80.0	1.2566	1.5080	1.4451	111.3	8168.2	8333.3
10	120.0	1.8850	2.3038	2.3038	110.0	12252.2	12500.0
5	240.0	3.7699	3.4558	3.7699	81.3	24504.5	25000.0

De los valores de esta tabla sacamos varias gráficas para ver la relación que se tiene entre las variables experimentales y teóricas, como se puede apreciar más adelante.

En la figura 3.2 se aprecia la gráfica que relaciona la velocidad angular con respecto a la variable i, con un ajuste de manera exponencial esto se debe a la forma de la ecuación 2.6 donde la velocidad angular depende de la variable i, y se encuentra en el denominador ocasionando que en cada incremento de i disminuya cada vez más hasta llegar a su límite 255. También se puede apreciar las barras de error, las cuales corresponden a la desviación estándar del promedio de los datos. De la figura 3.2 nos damos cuenta que las medidas varían mucho entre mayor es la velocidad con una desviación máxima de 1rad/s, en nuestro caso no tiene mucha relevancia ya que estaremos manejando velocidades bajas y la desviación estándar de estas velocidades es mucho más baja (0.0007 rad/s como mínima). Uno de los factores que causó estos intervalos de error fue la forma como se tomaron las medidas, ya que para velocidades altas no es tan viable medir solo con un cronómetro, para ellos sería necesario ocupar un tacómetro.

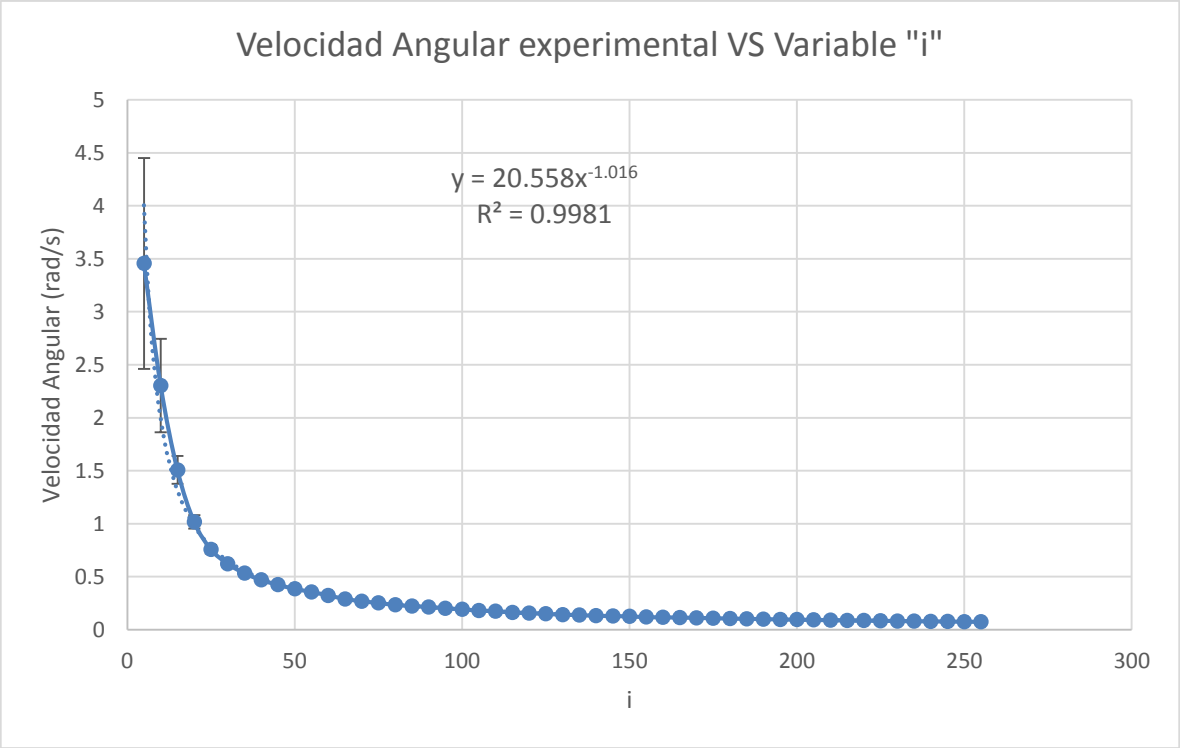

Figura 3.2: Gráfico de la relación velocidad angular experimental y la variable i.

En la figura 3.3 comparamos todas nuestras velocidades angulares respecto a los pulsos introducidos al motor PAP, donde vemos que las velocidades experimentales con la teórica son muy similares principalmente después de meter más 48 pulsos/s, dándonos una gran confiabilidad en nuestro sistema a velocidades bajas. También se puede observar que la velocidad angular no es afectada por la masa de la carga, pero se pudo observar en la experimentación al momento de tomar las medidas de la velocidades angulares con carga cómo el motor perdía el paso al exceder la masa de la carga máxima, ocasionando que el motor no fuera capaz de dar ni una vuelta. La desviaciones estándar de las velocidades angulares con peso van desde 0.0004 rad/s a 1.4 rad/s, donde la mayor desviación estándar se encuentra en las velocidades altas y la desviación estándar inferior esta en las velocidades bajas como se puede ver en las barras de error. En la figura 3.4 se puede observar que son las mismas medidas que en la figura 3.3 pero con respecto a la variable i donde podemos visualizar de mejor manera la similitud entre las velocidades teóricas y experimentales, por lo que podemos decir que el motor a paso que estamos ocupando es muy confiable.

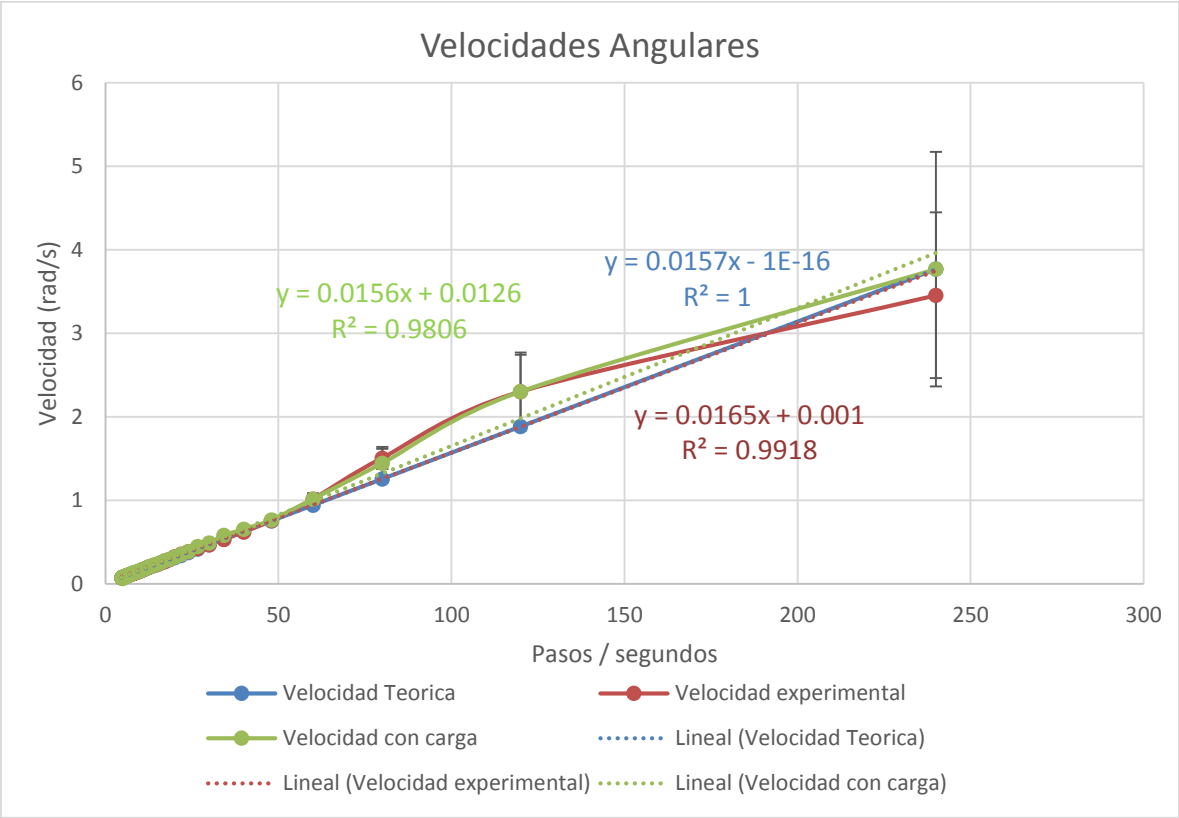

Figura 3.3: Gráfico de velocidades angulares y pasos entre segundos.

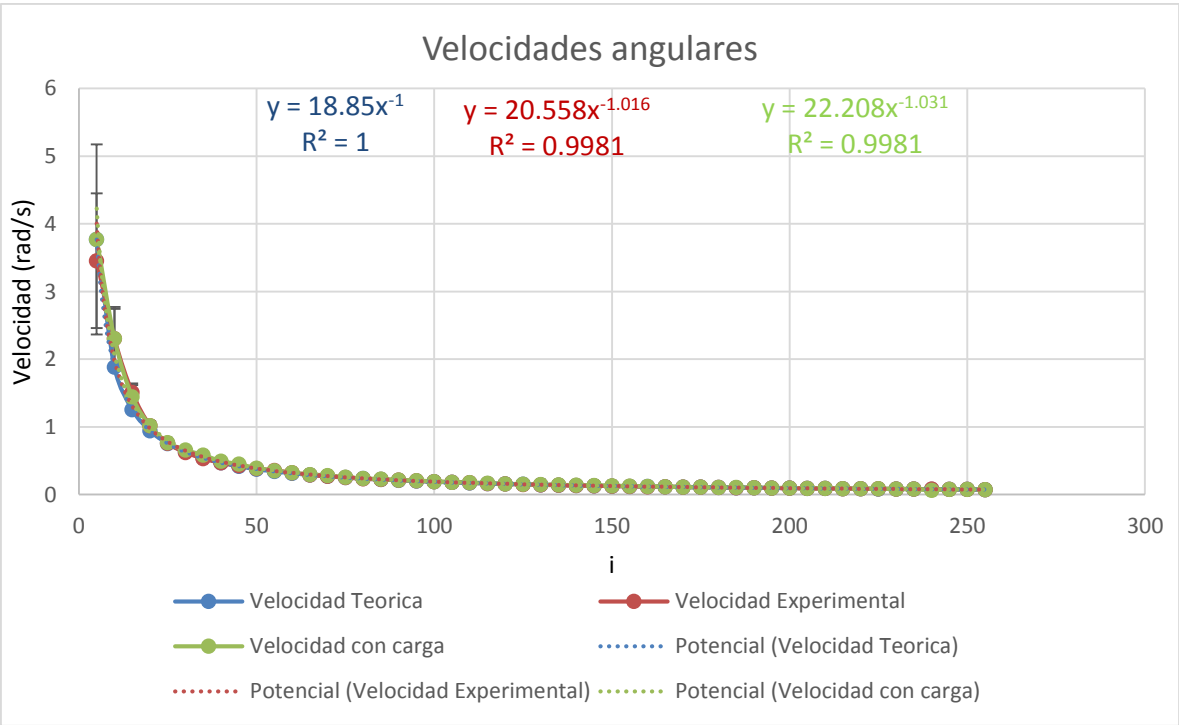

Figura 3.4: Gráfico de velocidades angulares con la variable "i"

Regresando al experimento de las velocidades con carga, también graficamos la relación que hay entre masa máxima de la carga y el numero de pasos entre segundo introducidos al motor (Figura 3.5). Donde las medidas tomadas de la masa no son tan variables como se puede observar en las barras de error con una desviación estándar promedio de 2.4 gr. También se puede observar que la masa máxima disminuye entre mayor velocidad hay, esto se debe a que el motor pierde fuerza cada vez que va incrementando la velocidad.

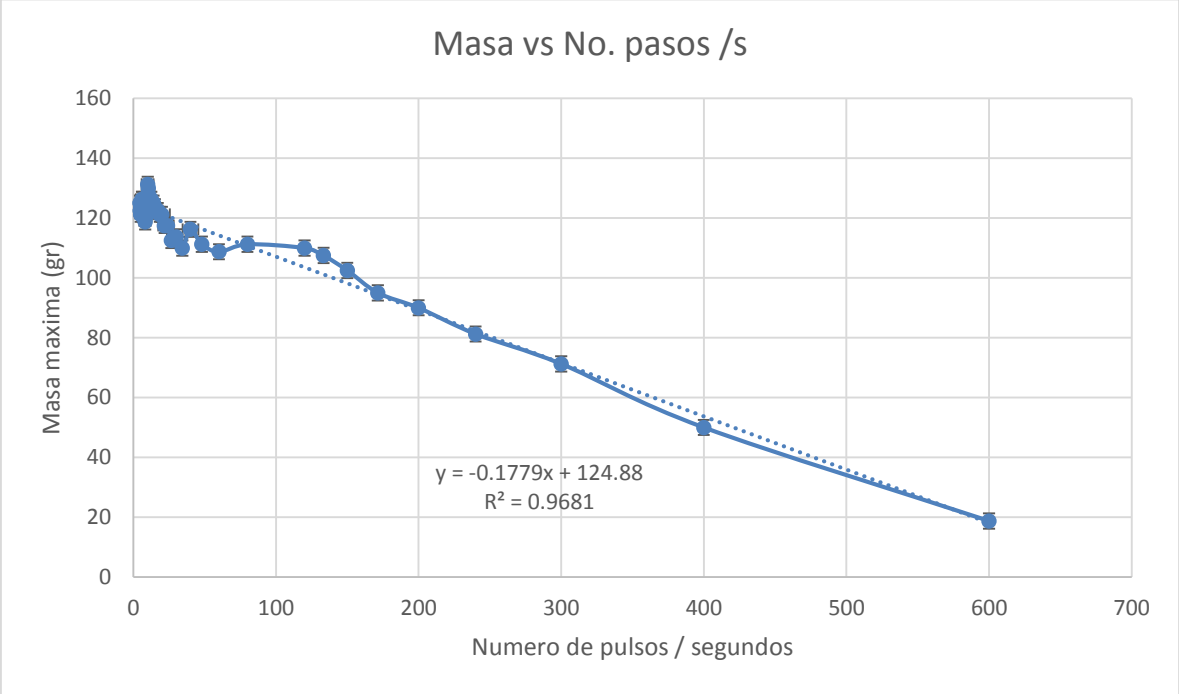

Figura 3.5: Relación entre masa y número de pulsos por segundo.

En la Figura 3.6 podemos observar las velocidades lineales teóricas y experimentales con respecto a la variable i y con respecto a la cantidad de pulsos entre segundo introducidos al motor a pasos en la Figura 3.7. Estas velocidades fueron tomadas en nuestro dispositivo ya montado. Podemos observar que las barras de error prácticamente no se notan con una desviación estándar promedio de 8.9 μm/s, es decir las mediciones de las velocidades no variaron mucho. Esto se debe a que los errores de medición disminuyeron, ya que fue más sencillo tomar el tiempo en que tardaba en recorrer el carro de impresora a 5 cm que medir el tiempo de una vuelta del motor PAP ya que el intervalo de tiempo se amplió. Con esto podemos decir que nuestro sistema es muy confiable, para obtener las velocidades deseadas, ya que es mínima la diferencia que hay entre las velocidades teóricas y experimentales con un error no mayor al 2%.

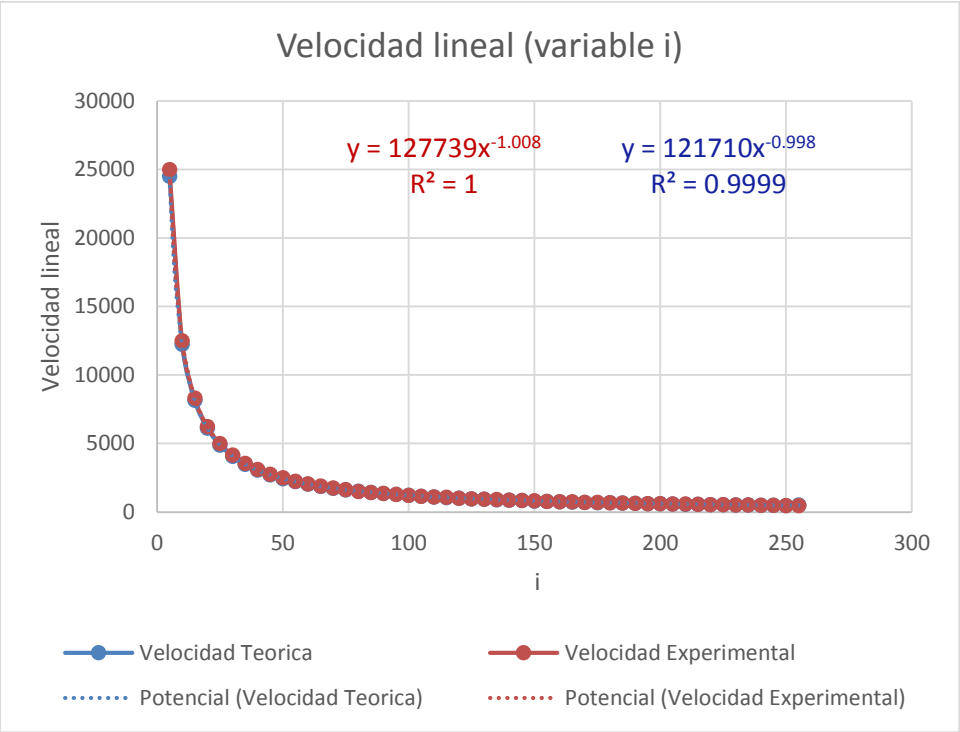

Figura 3.6: Grafica de Velocidades lineales con la variable i.

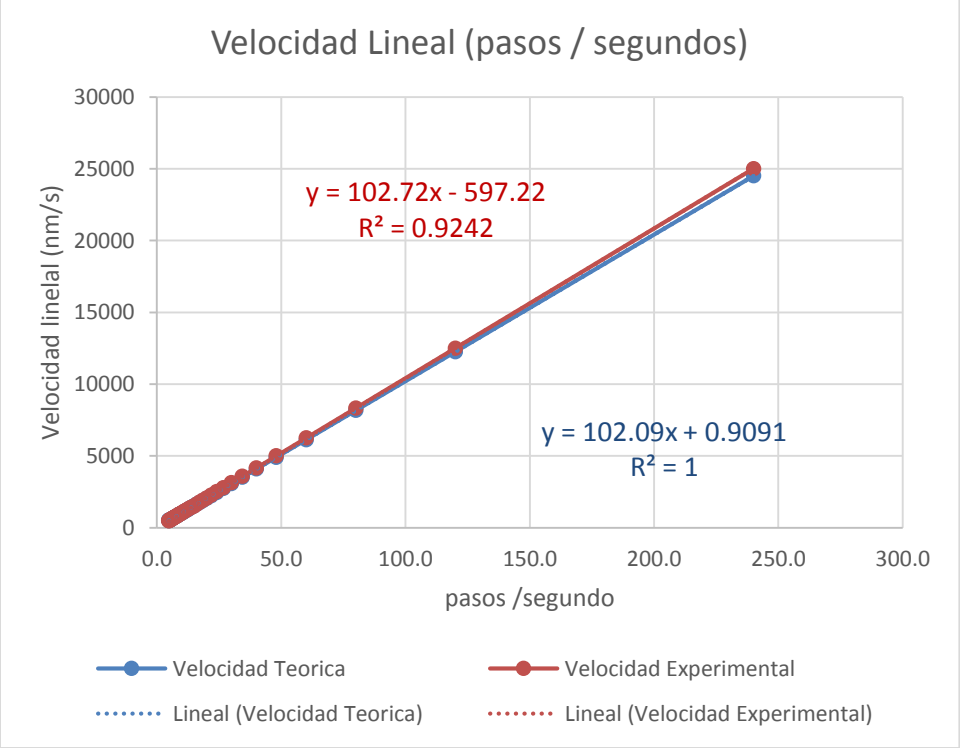

Figura 3.7: Grafica de velocidades lineales con pasos/ segundo

CONCLUSIONES Y PERSPECTIVAS

En este trabajo se obtuvo un sistema de control de velocidades en donde la parte física comprende la adaptación de un carro y riel de impresora mientras que la parte electrónica se basa en el control de un motor PAP unipolar. El sistema posee control de posición y cambio de velocidad entre 400 μm/s a 10 000 μm/s, controlado por el usuario desde el ordenador, con la ayuda de una API desarrollada en Visual C++ que comunica el *Host* del ordenador con la interface USB 2.0 de clase HID de nuestro dispositivo. Se compararon las velocidades angular y lineal del dispositivo ya montado con y sin carga, con los resultados esperados teóricamente, encontrando un error menor al 2 %. También se encontró el valor de carga máxima que soporta el dispositivo para diferentes velocidades. El sistema diseñado y puesto en marcha está listo para ser utilizado en diversas aplicaciones, incluyendo la estructuración de superficies mediante recubrimiento por inmersión.

Para la construcción de este sistema fue necesario superar diversas dificultades. Una de las más influyentes fue el control de la interfaz USB, que requirió de una investigación detallada del complejo protocolo USB para hacer más fácil y estándar la comunicación entre dispositivos (computadora,....). Otro obstáculo para el desarrollo de este proyecto fue que utilizamos un motor semi-usado del cual se carece de muchos datos técnicos, que facilitarían en un futuro el diseño mecánico del sistema. Aun así se obtuvo control en el sistema de velocidades que cubre nuestras necesidades iniciales, en dónde se ha podido lograr el rango de velocidades lineales en un carro de impresora que va desde los 400um/s a los 10 000 um/s sin tantas vibraciones y efectos de inercia. Sin embargo las vibraciones en las ultimas velocidades bajas se notan un poco, por ello no se opto en bajar aun mas los rangos de velocidad; esto se debe a que lo que le introducimos al motor son pulsos eléctricos ocasionando un cambio brusco de voltaje entre pulso y pulso de 3.5 a 0 volts.

Este trabajo será de gran ayuda, para futuros proyectos, ya ha que sus características lo hacen ideal para diversas aplicaciones, además del conocimiento adquirido es muy importante para adaptar el sistema a posibles aplicaciones futuras.

Perspectivas

Se espera continuar con este proyecto mejorando las capacidades del EEMGRI con ayuda del conocimiento a adquirir en la maestría de ingeniería eléctrica. Aunque ya se tiene todo el sistema electrónico del dispositivo, falta mejorar su sistema mecánico y probarlo en diversas aplicaciones. Para su uso en la técnica de recubrimiento por inmersión, en primera requerimos implementarlo y probarlo para ver en qué velocidades es más eficiente, y que materiales podrán ser usados para nuestro dispositivo. Una de las mejoras que deseamos es disminuir las velocidades lineales hasta un mínimo de micrómetros por segundo , lo que incrementara las aplicaciones del EEMGRI. La idea es que éste sea capaz de crear monocapas de poliestireno para futuras aplicaciones.

También se tiene pensado agregar micro-pasos al controlador del motor PAP, para poder aumentar la resolución del motor y poder usarlo con velocidades más bajas en donde el movimiento no es tan continuo. También se espera desarrollar su interfaz gráfica en QT para un mejor control y visualización para el usuario. Además una de las ventajas de usar QT es que es multiplataforma y la interfaz funcionaría para los principales sistemas operativos.

BIBLIOGRAFIA

[1] GIULIANI ANDRES Maximiliano. "Formación de estructuras coloidales mediante mecanismos interfaciales." Programa de Doctorado Inestabilidades en Sistemas Disipativos Universidad de Navarra, Pamplona, España 2008.

[2] NUERMAIMAITI Ajiguli. "Studies on the Self-organization of Colloidal Nanoparticles at Interfaces" Department of Physics and Astronomy, Uppsala University. Tesis de doctorado.

[3] VILLAVICENCIO Jose, Valera Benjamín, Mata Esther. "Sistema Electrónico para actualización de un Dip Coater" CCADET-UNAM, México, 2013.

[4] KUO Benjamin. "Sistemas de control automático" Séptima Edición, Editorial Pearson. México, 1996.

[5] CONTI, Fransisco. "Motores paso a paso" Editorial Alsina. Buenos Aires, 2011.

[6] ACARNLEY, Paul. "Stepping Motors a guide to theory and practice" Cuarta Edicion, The Institution on Engineering and Technology, Reino Unido, 2007.

[7] Microchip Technology Incorporated. (2006) "PIC18F2455/2550/4455/4550 Data Sheet."[Archivo PDF] Tomado de: www.microchip.com

[8] AXELSON Janet. "USB Complete Fourth Edition" Lakeview Research LLC,5310 Chinook Ln., Madison WI 53704, USA, 2009.

[9] USB Implementers Forum.(2001)" Device Class Definition for Human Interface Devices (HID)." [Archivo PDF] Tomado de: www.usb.org

[10] Signal 11 Sotware (2010)." HIP API for Linux, Mac Os X, and Windows." (Fecha de consulta: 21 de abril de 2015) URL: http://www.signal11.us/oss/hidapi/

[11]Microchip Technology Inc. Bibliotecas de Microchip para aplicaciones. (Fecha de consulta: 21 de abril de 2015) Tomado de: http://www.microchip.com/pagehandler/en-us/devtools/mla/archives.html

[12] PAREJA, Miguel. "Diseño y desarrollo de circuitos impresos con KiCad" Ed. Grupo RC, Madrid, España, 2010.

APENDICE A
usb_descriptors.c

```
/********************************************************************
 FileName:        usb_descriptors.c
 Dependencies:    See INCLUDES section
 Processor:             PIC18 or PIC24 USB Microcontrollers
 Hardware:              The code is natively intended to be used on the following
                        hardware platforms: PICDEM™ FS USB Demo Board,
                        PIC18F87J50 FS USB Plug-In Module, or
                        Explorer 16 + PIC24 USB PIM.  The firmware may be
                        modified for use on other USB platforms by editing the
                        HardwareProfile.h file.
 Complier:        Microchip C18 (for PIC18) or C30 (for PIC24)
 Company:               Microchip Technology, Inc.

* Descriptor specific type definitions are defined in:
 * usb_device.h
 * Configuration options are defined in:
 * usb_config.h
 ********************************************************************/
#ifndef __USB_DESCRIPTORS_C
#define __USB_DESCRIPTORS_C
/** INCLUDES *******************************************************/
#include "./USB/usb.h"
#include "./USB/usb_function_hid.h"
/** CONSTANTS ******************************************************/
#if defined(__18CXX)
#pragma romdata
#endif
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
   0x12,    // Size of this descriptor in bytes
   USB_DESCRIPTOR_DEVICE,              // DEVICE descriptor type
   0x0200,              // USB Spec Release Number in BCD format
   0x00,               // Class Code
   0x00,               // Subclass code
   0x00,               // Protocol code
   USB_EP0_BUFF_SIZE,         // Max packet size for EP0, see usb_config.h
   0x0461,              // Vendor ID
   0x000A,               // Product ID: Custom HID demo
   0x0002,              // Device release number in BCD format
   0x01,               // Manufacturer string index
   0x02,               // Product string index
   0x00,               // Device serial number string index
   0x01                // Number of possible configurations
};
/* Configuration 1 Descriptor */
```

```
ROM BYTE configDescriptor1[]={
   /* Configuration Descriptor */
   0x09,//sizeof(USB_CFG_DSC),    // Size of this descriptor in bytes
   USB_DESCRIPTOR_CONFIGURATION,               // CONFIGURATION descriptor type
   0x29,0x00,           // Total length of data for this cfg
   1,                   // Number of interfaces in this cfg
   1,                   // Index value of this configuration
   0,                   // Configuration string index
   _DEFAULT | _SELF,              // Attributes, see usb_device.h
   50,                   // Max power consumption (2X mA)
   /* Interface Descriptor */
   0x09,//sizeof(USB_INTF_DSC),   // Size of this descriptor in bytes
   USB_DESCRIPTOR_INTERFACE,              // INTERFACE descriptor type
   0,                   // Interface Number
   0,                   // Alternate Setting Number
   2,                   // Number of endpoints in this intf
   HID_INTF,              // Class code
   0,     // Subclass code
   0,     // Protocol code
   0,                   // Interface string index

   /* HID Class-Specific Descriptor */
   0x09,//sizeof(USB_HID_DSC)+3,    // Size of this descriptor in bytes
   DSC_HID,               // HID descriptor type
   0x11,0x01,               // HID Spec Release Number in BCD format (1.11)
   0x00,                // Country Code (0x00 for Not supported)
   HID_NUM_OF_DSC,         // Number of class descriptors, see usbcfg.h
   DSC_RPT,               // Report descriptor type
   HID_RPT01_SIZE,0x00,//sizeof(hid_rpt01),      // Size of the report descriptor
   /* Endpoint Descriptor */
   0x07,/*sizeof(USB_EP_DSC)*/
   USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
   HID_EP | _EP_IN,                //EndpointAddress
   _INTERRUPT,                    //Attributes
   0x40,0x00,                //size
   0x01,                    //Interval
   /* Endpoint Descriptor */
   0x07,/*sizeof(USB_EP_DSC)*/
   USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
   HID_EP | _EP_OUT,                //EndpointAddress
   _INTERRUPT,                    //Attributes
   0x40,0x00,                //size
   0x01                    //Interval
};
//Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
}};
//Manufacturer string descriptor
```

```
ROM struct{BYTE bLength;BYTE bDscType;WORD string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'C','C','A','D','E','T',' ','L','i','c ',
'T','e','c','h','n','o','l','o','g','y',' ','U','N','A','M'
}};
//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[22];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'D','i','p',' ','C','o','a','t','i','n','g',
'D','e','v','i','c','e',' ','U','N','A','M'
}};
//Class specific descriptor - HID
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
{
   0x06, 0x00, 0xFF,      // Usage Page = 0xFF00 (Vendor Defined Page 1)
   0x09, 0x01,            // Usage (Vendor Usage 1)
   0xA1, 0x01,            // Collection (Application)
   0x19, 0x01,            //     Usage Minimum
   0x29, 0x40,            //     Usage Maximum           //64 input usages total (0x01 to 0x40)
   0x15, 0x01,            //     Logical Minimum (data bytes in the report may have minimum value = 0x00)
   0x25, 0x40,                        //     Logical Maximum (data bytes in the report may have maximum
value = 0x00FF = unsigned 255)
   0x75, 0x08,            //     Report Size: 8-bit field size
   0x95, 0x40,            //     Report Count: Make sixty-four 8-bit fields (the next time the parser hits an
"Input", "Output", or "Feature" item)
   0x81, 0x00,            //     Input (Data, Array, Abs): Instantiates input packet fields based on the above report
size, count, logical min/max, and usage.
   0x19, 0x01,            //     Usage Minimum
   0x29, 0x40,            //     Usage Maximum           //64 output usages total (0x01 to 0x40)
   0x91, 0x00,            //     Output (Data, Array, Abs): Instantiates output packet fields.  Uses same report
size and count as "Input" fields, since nothing new/different was specified to the parser since the "Input" item.
   0xC0}                  // End Collection
};
//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
   (ROM BYTE *ROM)&configDescriptor1
};
//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
   (ROM BYTE *ROM)&sd000,
   (ROM BYTE *ROM)&sd001,
   (ROM BYTE *ROM)&sd002
};
/** EOF usb_descriptors.c ***************************************************/
#endif
```

APENDICE B
HardwareProfile- PICDEM FSUSB.h

```
/********************************************************************
 File Description:

 Change History:
  Rev   Date         Description
  1.0   11/19/2004   Initial release
  2.1   02/26/2007   Updated for simplicity and to use common
                     coding style
  2.3   09/15/2008   Broke out each hardware platform into its own
                     "HardwareProfile - xxx.h" file
********************************************************************/

#ifndef HARDWARE_PROFILE_PICDEM_FSUSB_H
#define HARDWARE_PROFILE_PICDEM_FSUSB_H

    /********************************************************************/
    /******** USB stack hardware selection options *********************/
    /********************************************************************/
    //This section is the set of definitions required by the MCHPFSUSB
    //  framework.  These definitions tell the firmware what mode it is
    //  running in, and where it can find the results to some information
    //  that the stack needs.
    //These definitions are required by every application developed with
    //  this revision of the MCHPFSUSB framework.  Please review each
    //  option carefully and determine which options are desired/required
    //  for your application.

    //The PICDEM FS USB Demo Board platform supports the USE_SELF_POWER_SENSE_IO
    //and USE_USB_BUS_SENSE_IO features.  Uncomment the below line(s) if
    //it is desireable to use one or both of the features.
    //#define USE_SELF_POWER_SENSE_IO
    #define tris_self_power     TRISAbits.TRISA2    // Input
    #if defined(USE_SELF_POWER_SENSE_IO)
    #define self_power          PORTAbits.RA2
    #else
    #define self_power          1
    #endif

    //#define USE_USB_BUS_SENSE_IO
    #define tris_usb_bus_sense  TRISAbits.TRISA1    // Input
    #if defined(USE_USB_BUS_SENSE_IO)
    #define USB_BUS_SENSE       PORTAbits.RA1
    #else
    #define USB_BUS_SENSE       1
    #endif
```

```
//Uncomment the following line to make the output HEX of this
//  project work with the MCHPUSB Bootloader
//#define PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER

//Uncomment the following line to make the output HEX of this
//  project work with the HID Bootloader
#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER

/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/******** Application specific definitions ************************/
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/

/** Board definition ***********************************************/
//These defintions will tell the main() function which board is
//  currently selected.  This will allow the application to add
//  the correct configuration bits as wells use the correct
//  initialization functions for the board.  These defitions are only
//  required in the stack provided demos.  They are not required in
//  final application design.
#define DEMO_BOARD PICDEM_FS_USB
#define PICDEM_FS_USB
#define CLOCK_FREQ 48000000


/** LED ************************************************************/
#define mInitAllLEDs()      LATB = 0x00; TRISB = 0x00; LATC &= 0xF0; TRISC &= 0xF0;// PORTB =
0b00111111;

#define mLED_1          LATBbits.LATB0
#define mLED_2          LATBbits.LATB1
#define mLED_3          LATBbits.LATB2
#define mLED_4          LATBbits.LATB3
#define mLED_5          LATBbits.LATB4
#define mLED_6          LATBbits.LATB5

#define sLED_1          LATCbits.LATC0
#define sLED_2          LATCbits.LATC1

#define mGetLED_1()       mLED_1
#define mGetLED_2()       mLED_2
#define mGetLED_3()       mLED_3
#define mGetLED_4()       mLED_4
#define mGetLED_5()       mLED_5
#define mGetLED_6()       mLED_6
```

```
#define sGetLED_1()       sLED_1
#define sGetLED_2()       sLED_2

#define mLED_1_On()       mLED_1 = 1;
#define mLED_2_On()       mLED_2 = 1;
#define mLED_3_On()       mLED_3 = 1;
#define mLED_4_On()       mLED_4 = 1;
#define mLED_5_On()       mLED_5 = 1;
#define mLED_6_On()       mLED_6 = 1;

#define sLED_1_On()       sLED_1 = 1;
#define sLED_2_On()       sLED_2 = 1;

#define mLED_1_Off()      mLED_1 = 0;
#define mLED_2_Off()      mLED_2 = 0;
#define mLED_3_Off()      mLED_3 = 0;
#define mLED_4_Off()      mLED_4 = 0;
#define mLED_5_Off()      mLED_5 = 0;
#define mLED_6_Off()      mLED_6 = 0;

#define sLED_1_Off()      sLED_1 = 0;
#define sLED_2_Off()      sLED_2 = 0;

#define mLED_1_Toggle()   mLED_1 = !mLED_1;
#define mLED_2_Toggle()   mLED_2 = !mLED_2;
#define mLED_3_Toggle()   mLED_3 = !mLED_3;
#define mLED_4_Toggle()   mLED_4 = !mLED_4;
#define mLED_5_Toggle()   mLED_5 = !mLED_5;
#define mLED_6_Toggle()   mLED_6 = !mLED_6;

#define sLED_1_Toggle()   sLED_1 = !sLED_1;
#define sLED_2_Toggle()   sLED_2 = !sLED_2;

/** SWITCH *********************************************************/
#define mInitAllSwitches()  TRISDbits.TRISD0=1;TRISDbits.TRISD1=1;
#define mInitSwitch2()     TRISDbits.TRISD0=1;
#define mInitSwitch3()     TRISDbits.TRISD1=1;
#define sw1             PORTDbits.RD0
#define sw2             PORTDbits.RD1
#define sw3             PORTDbits.RD2
#define sw4             PORTDbits.RD3
#define sw5             PORTDbits.RD4
#define sw6             PORTDbits.RD5

/** POT ************************************************************/
//#define mInitPOT()
{TRISAbits.TRISA0=1;ADCON0=0x01;ADCON2=0x3C;ADCON2bits.ADFM = 1;}
/** USB external transceiver interface (optional) ******************/
```

```
    /*#define tris_usb_vpo        TRISBbits.TRISB3    // Output
    #define tris_usb_vmo        TRISBbits.TRISB2    // Output
    #define tris_usb_rcv        TRISAbits.TRISA4    // Input
    #define tris_usb_vp         TRISCbits.TRISC5    // Input
    #define tris_usb_vm         TRISCbits.TRISC4    // Input
    #define tris_usb_oe         TRISCbits.TRISC1    // Output

    #define tris_usb_suspnd     TRISAbits.TRISA3    // Output*/

    /** I/O pin definitions ********************************************/
    #define INPUT_PIN 1
    #define OUTPUT_PIN 0
#endif  //HARDWARE_PROFILE_PICDEM_FSUSB_H
```

APENDICE C
Hardware Profile.h

```
/********************************************************************
 FileName:      HardwareProfile.h
 Dependencies:  See INCLUDES section
 Processor:     PIC18, PIC24, or PIC32 USB Microcontrollers
 Hardware:      The code is natively intended to be used on the
                following hardware platforms:
                  PICDEM™ FS USB Demo Board
                  PIC18F46J50 FS USB Plug-In Module
                  PIC18F87J50 FS USB Plug-In Module
                  Explorer 16 + PIC24 or PIC32 USB PIMs
                  PIC24F Starter Kit
                  Low Pin Count USB Development Kit
                The firmware may be modified for use on other USB
                  platforms by editing this file (HardwareProfile.h)
 Compiler:          Microchip C18 (for PIC18), C30 (for PIC24),
                or C32 (for PIC32)
 Company:       Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PIC® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PIC Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

 ********************************************************************
 File Description:

 Change History:
  Rev   Date         Description
  1.0   11/19/2004   Initial release
  2.1   02/26/2007   Updated for simplicity and to use common
                     coding style
```

```
  2.3   09/15/2008   Broke out each hardware platform into its own
                     "HardwareProfile - xxx.h" file
********************************************************************/

#ifndef HARDWARE_PROFILE_H
#define HARDWARE_PROFILE_H

//#define DEMO_BOARD USER_DEFINED_BOARD

#if !defined(DEMO_BOARD)
   #if defined(__18CXX)
      #if defined(__18F4550)
         #include "HardwareProfile - PICDEM FSUSB.h"
      #endif
   #endif
#endif

#if !defined(DEMO_BOARD)
   #error "Demo board not defined.  Either define DEMO_BOARD for a custom board or select the correct
processor for the demo board."
#endif

#endif  //HARDWARE_PROFILE_H
```