ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA

DESARROLLO DE UN SISTEMA EMBEBIDO CON APLICACIONES GRÁFICAS Y CAPACIDAD USB ON-THE-GO PARA MANEJO DE ARCHIVOS EN FLASH DRIVE UTILIZANDO LA TARJETA STARTER KIT PIC24F

LUIS FELIPE URRESTA CUEVA

SANGOLQUÍ-ECUADOR

2010

CERTIFICACIÓN

Certificamos que el presente proyecto de grado, **"Desarrollo de un sistema embebido con aplicaciones gráficas y capacidad USB On-The-Go para manejo de archivos en flash drive utilizando la tarjeta Starter Kit PIC24F"**, fue desarrollado en su totalidad por el señor Luis Felipe Urresta Cueva, bajo nuestra dirección.

Atentamente,

Ing. Byron Navas	Ing. Luis Orozco
Director	Codirector

AGRADECIMIENTOS

Mi agradecimiento a los amores de mi vida Jesús y María, por su inefable amor que me impulsa a perseverar con alegría cada día.

A mis padres Luis Felipe y Luisa, por su acertada educación que me inculcó valores cristianos, el valor del trabajo duro y la lucha por conseguir los sueños.

A mis abuelas Elba y Luisa, por su ejemplo de tenacidad y perseverancia para salir de las más duras situaciones que la vida puede plantear, luchando por el bienestar de sus hijos.

A mis hermanas Margie y Dianasol, quienes se han preocupado por darme amor y guía en mis años de vida.

A toda mi familia, por su apoyo y los calurosos lazos de amor fraterno que nos han mantenido unidos todos estos años.

Al Ing. Byron Navas y al Ing. Luis Orozco por compartir sus conocimientos y el tiempo que han dedicado a este proyecto.

A Mónica Zea y mis hermanos de la comunidad 41 y de la Obra en geneal, por su amor y amistad incondicionales que me han mostrado cuanto Dios me ama.

A Milton Paredes, María Fernanda Barreto, el Padre Juan y demás personas que han ayudado a mí encuentro con Dios y María para ser una mejor persona.

DEDICATORIA

A Jesús y María, los amores de mi vida…

PRÓLOGO

La empresa Microchip inaugura el tercer milenio lanzando al mercado revolucionarios dispositivos que doblan en capacidad y en ancho de bus a todos sus antecesores, se les da a conocer como los dsPIC de 16 bits; que además incluyen, un módulo de procesamiento digital de señales (dsp). Por otro lado, la interfaz USB empezó a sobresalir sobre sus competidoras por su gran desempeño, facilidad de uso y escases de errores [*2*]. Para el año 2004, estos dos grandes avances, exitosos en sus respectivos campos, fueron combinados para dar origen a una nueva gama de microcontroladores con conectividad USB On-The-Go (portable), se los bautizó como PIC24.

Por otro lado, la tecnología cada vez se adentra más en el entorno diario, muchos artefactos que nos rodean ahora vienen dotados de inteligencia, tal es el caso de los electrodomésticos, los automóviles, los ascensores, etc. El hombre se comunica con estos dispositivos electrónicos a través de teclados y botones y los artefactos le responden con pequeños focos o sonidos que el hombre debe interpretar para comprenderlos. Pero el hombre nunca está satisfecho con lo que ha creado y desea mejorarlo, por eso desea que el dispositivo no solo emita un sonido o encienda una luz, desea que el artefacto le responda más explícitamente, le indique más coloridamente, que el artefacto sea más “amigable”. Para este fin, el hombre ha agregado una nueva característica a los sistemas embebidos, una interfaz gráfica. Es consecuencia, la tendencia actual de los sistemas embebidos poseen una pantalla en donde el dispositivo puede proveer de toda la retroalimentación que el usuario necesite. El avance no quedo en ese punto, ya que una vez adentrados en diseños que sean llamativos y eficientes a la vez, los diseñadores pensaron en integrar el espacio físico que ocupa la pantalla y el teclado creando un solo periférico. Estos son las conocidas touchscreen, popularizadas en el famoso iPhone. Todas estas modernas características están incluidas en el desarrollo de este proyecto, ya que el Starter kit PIC24 tiene integrados elementos de hardware que utilizan esta tecnología.

ÍNDICE

CAPÍTULO 1
INTRODUCCIÓN

CAPÍTULO 2
MARCO TEÓRICO

CAPÍTULO 3
DESARROLLO DEL FIRMWARE

CAPÍTULO 4
TUTORIALES

CAPÍTULO 5
CONCLUSIONES Y RECOMENDACIONES

CAPÍTULO I

INTRODUCCIÓN

1.1 DESCRIPCIÓN DE LOS SISTEMAS EMBEBIDOS

Un sistema embebido es un computador diseñado para una o algunas tareas específicas. Está compuesto por un microprocesador, que juntamente con algunos elementos conocidos como periféricos y circuitos internos en el chip, crean un pequeño módulo de control, el cual puede ser embebido en otro dispositivo eléctrico o mecánico para dotarlo de "inteligencia" a bajo costo [*1*].

Ejemplos de sistemas embebidos podemos encontrarlos en casi todo nuestro alrededor. En telecomunicaciones, los teléfonos celulares, routers, network bridges, etc. Los electrodomésticos tales como microondas, refrigeradoras, lavadoras, secadoras, entre otros, vienen dotados de inteligencia a bajo costo para mejorar eficiencia, amigabilidad y dar características extras al usuario. Además de estos, existen asistentes personales digitales (PDAs), reproductores MP3, cámaras digitales, receptores GPS, alarmas de autos y mas, que contienen pequeños controladores que posibilitan una interfaz con el usuario.

Considérese la alarma de una casa, su función es constantemente evaluar las señales de los sensores y sonar la alarma sí alguno de ellos se activa. La alarma tiene un pequeño programa, o firmware, estructurado en un bucle infinito; este bucle, revisa el estado de los sensores, si estos cambian de estado, el software activará una interrupción en la cual se activa una salida conectada a una sirena. El programa puede tener otras funciones, tales como prueba del sistema, establecer nueva contraseña, etc.

Sí se pusiera una computadora con parlantes y se conectaran los sensores a su puerto paralelo, el sistema funcionaria de igual manera, pero la relación costo-beneficio caería en gran desventaja. En especial porque la alarma puede funcionar con baterías largo tiempo incluso si la electricidad se corta.

1.2 DIFERENCIAS ENTRE UN SISTEMA EMBEBIDO Y UN PC

La diferencia principal radica en que el sistema embebido está diseñado para una o varias tareas específicas y una PC está diseñada para correr varios tipos de aplicaciones y para conectar cualquier dispositivo externo. Un sistema embebido tiene un solo programa grabado, por esto, solo requiere suficiente poder de cálculo, memoria y capacidad para esa tarea, lo que lo hace más barato y confiable. Una computadora posee un procesador para propósitos generales, lo cual resulta en un mayor precio; además, posee más periféricos como discos, tarjeta de audio, tarjeta de video, puertos de conexión a internet, etc. Un sistema embebido posee un microprocesador de bajo costo y circuitos periféricos en el mismo chip.

Frecuentemente, un sistema embebido es invisible; esto es, debido a que es solo un sub-módulo de otra pieza y hace solo una función muy pequeña de toda la máquina o dispositivo. En algunos casos, el controlador adiciona inteligencia a bajo costo solo para los sub-sistemas críticos del aparato. Las soluciones que presenta pueden ir desde lo más simple hasta lo más complejo, por ejemplo, desde un reproductor de audio hasta soluciones para instrumentos quirúrgicos de alta precisión.

El éxito de los sistemas embebidos ha llevado a los desarrolladores a explorar nuevas fronteras para emularlos más a un PC. La interfaz gráfica se ha popularizado a tal punto que día a día encontramos mas dispositivos que cuentan con esta opción; a medida que se ha difundido, se ha vuelto una característica de facto para los diseñadores en sus aplicaciones. Debido a que los sistemas embebidos se usan en lugares de poco espacio físico, la inclusión de un teclado es un reto considerable [*2*].

En 1971, en la universidad de Kentucky, el Doctor Sam Hurst desarrollo la tecnología “touch sensor”, y esta desemboco en la creación de las pantallas conocidas como touchscreens, conocidas por el popular teléfono iPhone. De esta manera el mismo espacio es usado por la pantalla y por el teclado; además el valor agregado es un dispositivo más llamativo y más amigable para el usuario [*3*].

1.3 LA INTERFAZ USB ON-THE-GO

“Consumidores individuales y empresariales necesitan conectar sus dispositivos móviles unos con otros, y con varios periféricos. Esto es confirmado por las docenas de métodos de conectividad usados por diferentes fabricantes de productos móviles incluyendo varios tipos de cables, conectores, slots, dongles y siete tipos diferentes de tecnologías de tarjetas de memoria” [*4*]. Este hecho ha creado un caos de conectividad; la estandarización para la conectividad entre sistemas móviles es inminente.

Universal Serial Bus (USB), es el nombre que fue dado a una interfaz creada desde cero para integrar las soluciones a todas las necesidades que la conectividad entre dispositivos presenta, tales como protocolos, velocidad de transmisión, conectividad física, etc. Su invención, se estima que fue en el año 1995 y aunque el crédito de haberla inventado la debaten varias empresas, frecuentemente se le atribuye al equipo de M-Systems. El grupo de desarrolladores que la creó, concedió una lista de deseos a todo usuario, esta lista incluye:

- **Fácil de usar**, se auto configura sin molestar al usuario, tampoco pregunta detalles de setup.
- **Rápida**, evita los cuellos de botella para que no se de una comunicación lenta.
- **Confiable**, raramente se producen errores; además, los errores que si ocurren tienen corrección automática.
- **Flexible,** así varios tipos de periféricos pueden usarla.
- **Barata,** para que sea accesible a los fabricantes que van a incluir esta interfaz en sus productos [*4*].

Además, al haber sido diseñada también para computadores portátiles, consume poca energía. Todas estas características son exactamente las mismas que requieren los dispositivos móviles; entonces, los fabricantes de la industria de telecomunicaciones colaboraron para desarrollar una tecnología estándar usando una versión de USB adaptada para aplicaciones móviles. El resultado fue USB OTG (On-The-Go) [*5*].

USB OTG es un suplemento que aumenta las capacidades de la especificación USB 2.0 para que sea posible utilizarla en dispositivos móviles. Entre las capacidades aumentadas puede contarse, el ahorro de energía, simplificación de comandos, etc. USB tradicionalmente ha funcionado con una topología anfitrión-periférico, donde el anfitrión es un PC y el periférico es un dispositivo relativamente torpe. Partiendo de esto, los desarrolladores tenían que incluir nuevas características para que USB pueda mejorar los estándares para dispositivos móviles, están características eran:

- Un nuevo estándar para cables y conectores USB de fábrica, que sean más pequeños.
- Adicionar la capacidad de ser anfitrión a dispositivos que tradicionalmente han sido solamente periféricos, para habilitar conexiones punto-punto.
- La habilidad para ser anfitrión o periférico (dispositivos de doble rol), y dinámicamente cambiar de uno a otro.
- Requerimientos de energía aun más bajos, para facilitar la inclusión de USB en dispositivo que funcionan con baterías.

USB OTG fue presentado al mundo el 18 de diciembre del año 2001 por el USB Implementers Forum. La aceptación de los usuarios, la abundancia de dispositivos USB y PCs con capacidad USB (más de 1.4 billones), son las razones por las que USB OTG tiene tanto éxito donde otras tecnologías de conectividad han fracasado.

1.4 LA ARQUITECTURA HARVARD MODIFICADA

La Harvard Mark I fue una computadora basada en relés. Fue un diseño innovador ya que separaba el almacenamiento de las instrucciones en una cinta ponchada de 24 bits

de ancho y el almacenamiento de datos en contadores electro-mecánicos de 23 bits de ancho. Tenía una capacidad limitada de almacenamiento, la cual estaba completamente contenida dentro del CPU. Este primer modelo, demostró que el separar físicamente el espacio de memoria para datos y el espacio para instrucciones es una ventaja de la arquitectura Harvard a la Von Neuman. Sin embargo, la diferencia principal es que la arquitectura Harvard posee un bus para datos y otro para control, mientras que Von Neuman utiliza el mismo bus para ambas cosas. Por ejemplo, en Von Neuman el procesador puede estar o leyendo/escribiendo datos o leyendo una instrucción, pero no ambas cosas al mismo tiempo ya que posee el mismo bus para ambas acciones. En la arquitectura Harvard, el procesador puede leer una instrucción y hacer un acceso a la memoria de datos al mismo tiempo porque posee buses diferentes para cada acción [*6*].

La separación física de las memorias de datos y de instrucciones, es ciertas veces considerada como la característica que distingue a las computadoras Harvard modernas. La verdadera distinción de una máquina Harvard a una Von Neumann, es que la memoria para datos y para instrucciones; ocupan diferentes espacios de dirección. En otras palabras, una dirección de memoria no identifica únicamente una locación de almacenamiento (como lo hace en la arquitectura Von Neumann); también, se necesita saber el espacio de memoria (instrucciones o datos) para el que se aplica la dirección.

Dos hechos provocan que la arquitectura Harvard tenga una desventaja. El primero, es que el mecanismo de ejecución debe separadamente cargar una instrucción y un dato. El segundo, una arquitectura Harvard moderna usualmente usa una tecnología de solo lectura para la memoria de instrucciones y de lectura/escritura para la memoria de datos. Entonces, el procesador puede empezar la ejecución de un programa pregrabado al momento de energizar el chip pero en este punto la memoria de datos esta en un estado o dirección desconocida; entonces, el problema radica en que no es posible proveer ningún tipo de dato predefinido al programa pregrabado, ya sea un valor (p.ej.: pi = 3.14159). Consecuentemente, la ventaja de la ejecución de un programa pregrabado se anula. La solución es dotar de un “pathway” de hardware e instrucciones en lenguaje de máquina, para que el contenido de la memoria de instrucciones pueda ser leído como si fueran datos. Así, inicialmente los valores de los datos pueden ser copiados de la memoria de

datos a la memoria de instrucciones cuando el firmware se carga. Es precisamente este "pathway" el que hace a una arquitectura Harvard, una arquitectura Harvard modificada.

Todo microcontrolador de la familia PIC24, la cual será discutida en la sección 1.5, posee una arquitectura Harvard modificada de 16 bits. El núcleo del CPU del PIC24 tiene muchas nuevas y mejoradas características, tales como:

- Paths de datos de 16 bits y de dirección de 24 bits, con la habilidad de mover información entre espacios de datos y de memoria.
- Direccionamiento lineal de más de 8 Mbytes (espacio para programa) y 64 Kbytes (espacio para datos).
- Un registro de trabajo en un arreglo de 16 elementos con soporte de built-in software stack.
- Un multiplicador de hardware de 17x17 con soporte para operaciones matemáticas de enteros.
- Soporte de hardware para divisiones de 32 para 16-bits.
- Un set de instrucciones que soporta múltiples modos de direccionamiento, y esta optimizado para lenguajes de alto nivel tal como 'C'.

1.5 LA FAMILIA PIC24F

A comienzos de esta década, en el año 2001, la empresa Microchip, presentó sus nuevos dispositivos conocidos como dsPIC; éste, es un microprocesador (PIC) que incluye un núcleo de procesamiento de señales (dsp) dentro del mismo chip, de ahí su nombre dsPIC. Para el año 2004, los dsPIC se volvieron de uso masivo y para este mismo año Microchip saca a la venta los microcontroladores de 16 bits, que esta vez no incluyen el núcleo de procesamiento de señales, y los bautiza como la nueva familia de PICs, la familia PIC24. Dentro de esta familia, se han clasificado dos "sub familias", la familia PIC24F y la PIC24H, cuya diferencia principal radica en que la familia H presenta una capacidad de procesamiento mucho mayor (casi 3 veces más) que la familia F.

Los microcontroladores PIC24 combinan un desempeño de control muy alto con nuevas características, además, poseen también todos los periféricos de la familia de 8 bits PIC18F, pero mejorados. Finalmente, pueden llegar a ejecutar 40 millones de instrucciones por segundo (MIPS). Adicionalmente, están diseñados para ser programados en Lenguaje C cuyo compilador se llama C30. Por otro lado, también poseen un juego de instrucciones assembler el cual es completo e incluso más amplio que el de las otras familias. De la familia PIC24H solo se mencionarán pocos detalles; quizás el más importante, es el que esta familia está diseñada para aplicaciones que demanden máxima capacidad de procesamiento, ya que logran hasta 40 MIPS. Adicionalmente posee 8 canales de acceso directo a memoria (DMA). En el resto de características en ambas familias son muy similares.

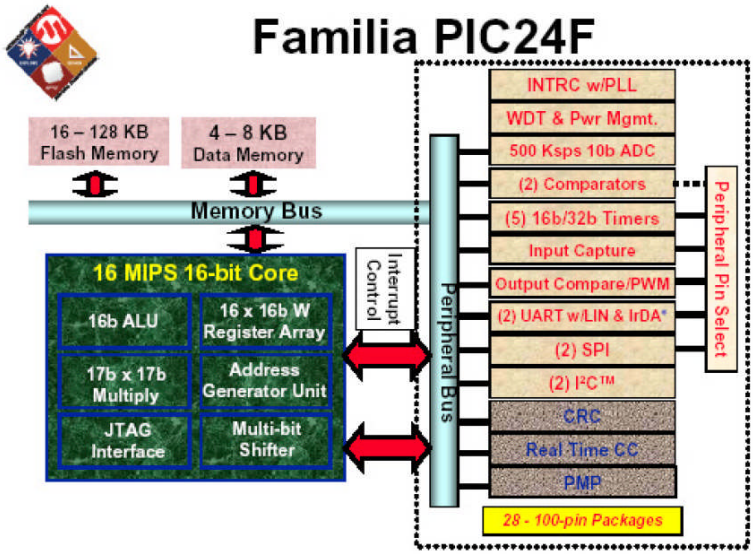


Figura. 1.1. Diagrama de la familia PIC24F [*7*]

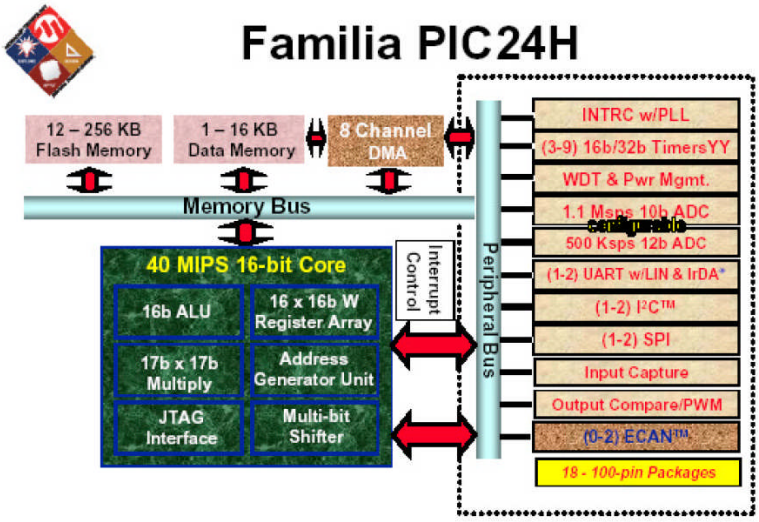


Figura. 1.2. Diagrama de la familia PIC24H [*7*]

Ambos buses, tanto de memoria como de periféricos, son de 16 bits. Las diferencias más notables están descritas en ambos gráficos.

La familia PIC24F tiene una larga lista de características que destacan a esta familia, entre las cuales se cuenta:

- Arquitectura Harvard modificada
- 16 MIPS de operación a 32 MGHz
- Oscilador interno de 8 MHz de operación con divisor programable por software y posibilidad para usar un PLL (Phase Locked Loop) interno.
- El PLL interno, puede multiplicar hasta por cuatro la frecuencia del oscilador principal.
- Arquitectura optimizada para lenguaje C con 76 instrucciones y modos de direccionamiento flexibles.
- Hardware de multiplicación de 17 x 17 bits con soporte para multiplicación con enteros.
- Direccionamiento lineal de memoria de programa hasta de 12 Mbytes.
- Modos de gerenciamiento de consumo seleccionables [*7*].

1.6 NOVEDADES INCLUIDAS EN LOS PIC24F

La familia PIC24 no solamente trae un bus expandido de 16 bits, un rendimiento elevado y memoria expandida; además, trae nuevos módulos como el CTMU, RTCC y el Peripheral Pin Select. La integración de estas nuevas características hace de estos microcontroladores perfectos para las más variadas aplicaciones desde las más simples hasta las más complejas.

Puede citarse como la caracterisitca más poderosa a Peripheral Pin Select (PPS), que es la función de seleccionar el pin periférico que se desea usar como salida. El PPS permite a los periféricos ser manipulados sobre un conjunto de pines de E/S que el mismo configure de entre un conjunto de 28 pines. El usuario puede independientemente manipular las señales de entrada y/o salida de cualquiera de los muchos periféricos digitales a cualquiera de los pines de E/S [*8*].

La interface CTMU es un extra a todas las otras ventajas análogas. En la familia PIC24F se incluye un módulo totalmente nuevo de interfaz: el CTMU. Este módulo provee un método conveniente para precisión en medición de tiempo y generación de pulsos, y puede servir como interfaz para sensores capacitivos. Un mejorado puerto paralelo esclavo, además de una alta configurabilidad. Uno de los puertos de E/S de propósito general puede ser reconfigurado para una mejorada comunicación de datos paralela. En este modo, el puerto puede ser configurado para operaciones como maestro o esclavo, y soporta transferencias de 8 y 16 bits con hasta 16 líneas de dirección externas en modo maestro.

El nuevo módulo de hora y fecha, Real-Time Clock/Calendar, implementa un reloj y calendario totalmente equipado con las características como detección de año bisiesto y función de alarmas en hardware. De esta manera, se liberan recursos de los timers y espacio de programa de memoria para el programa de la aplicación central [*8*].

1.7 LA FAMILIA PIC24FJ256GB110

La familia PIC24FJ256GB110 combina un set expandido de periféricos y un rendimiento computacional mejorado con una nueva opción de conectividad: UBS On-The-Go. Esta familia, además provee una nueva plataforma para aplicaciones USB de alto desempeño, las cuales pueden requerir más que una plataforma de 8-bits. Las características en ahorro de energía incluyen un sistema de cambio de osciladores durante la ejecución llamado On-The-Fly Clock Switching. Este sistema permite realizar el cambio de oscilador del microprocesador al Timer1 al oscilador de bajo consumo RC, permitiendo al usuario incorporar ideas de ahorro de energía en sus diseños mediante software. El ahorro de energía basado en instrucciones puede suspender toda operación, o selectivamente apagar el núcleo del microprocesador mientras deja los periféricos activos, eso con una sola instrucción de software [*8*].

Con la familia PIC24FJ256GB110, la empresa Microchip introduce al mercado la funcionalidad USB On-The-Go en un solo chip. Este nuevo módulo, provee la funcionalidad en chip como DEVICE, y también provee la funcionalidad USB HOST independiente. Al implementar HNP (Host Negociation Protocol), este módulo puede cambiar dinámicamente entre función de anfitrión y dispositivo, permitiendo al usuario un amplio y versatil rango de aplicaciones USB sobre una plataforma de microcontrolador. Adicionalmente a la función anfitrión USB, la familia de chips PIC24FJ256GB110 provee una solución completa USB en un solo chip, incluyendo un receptor/transmisor, regulador de voltaje, y generador de voltaje para energizar al bus durante las operaciones de anfitrión [*7*].

CARACTERÍSTICAS	PIC24FJ256GB106
Frequencia de operación	DC – 32 MHz
Memoria de programa (bytes)	256K
Memoria de programa (instrucciones)	87552
Memoria de datos (bytes)	16384
Puertos E/S	Puertos B, C, D, E, F, G
Pins remapeables	29 (28 E/S, 1 sola entrada)
Interrupción de notificación de cambio en la entrada	49
Comunicaciones Seriales: UART	4(1)
SPI (3 hilos/4 hilos)	3(1)
I^2C	3
Comunicación paralela: (PMP/PSP)	Si
JTAG Programación/ Boundary Scan	Si
Conversor ADC de 10 bits (Canales de Entrada)	16
Interfaz CTMU	Si
Resets (y delays)	POR, BOR, RESET, Instrucción,

Tabla. 1.1. Características del PIC24FJ256GB106

CAPÍTULO 2

FUNDAMENTO TEÓRICO

2.1 EL STARTER KIT PIC24F

El Starter Kit PIC24F es un producto de la empresa Microchip que fue lanzado al mercado a inicios del 2007, fue introducido por la prensa como “una nueva herramienta que contiene todo lo necesario para empezar a explorar las altas prestaciones y versatilidad de los PIC24F” (Figura 2.1) [7]. Es un kit de muy bajo costo (60 USD). Incluye una tarjeta que posee el microcontrolador PIC24FJ256GB106 y todos los elementos de hardware necesarios para el desarrollo de aplicaciones. Además, incluye un CD con la versión más reciente, al momento de ser empacado, del programa MPLAB IDE y un cable USB con terminales A y mini-B. Posee su propio programador o “In-Circuit-Debugger”.

Figura. 2.1. Imagen publicitaria y número de producto del Starter Kit PIC24F

La tarjeta Starter Kit es una herramienta diseñada como una solución todo-en-uno para depurar y desarrollar aplicaciones. Es una tarjeta electrónica de unos 12x5 cm, su circuitería, capacitores, resistencias, chips y demás están descubiertos, pero a pesar de esto es una tarjeta robusta y resistente al medio ambiente (Figura 2.2). Sin embargo, se debe tener cuidado con las cargas electrostáticas al trabajar con ella y es recomendable su uso en una estación libre de estática. El único hardware extra que necesita, es un cable USB para tomar alimentación y comunicarse con MPLAB IDE.

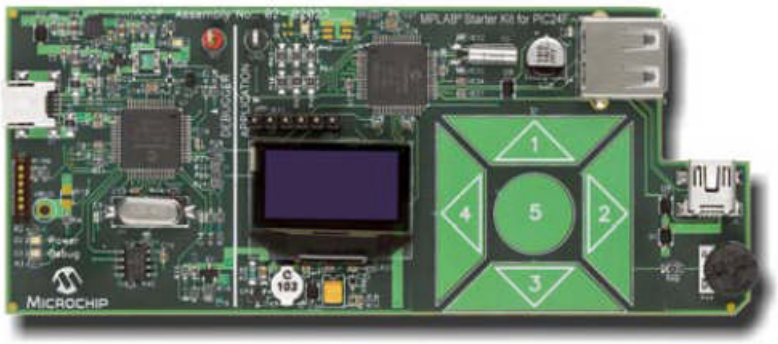


Figura. 2.2. Tarjeta Starter Kit PIC24F

Empezar a usar la tarjeta además de económico, es simple. Una vez que se dispone del producto, lo único que se debe hacer, es insertar el CD que viene en el kit, instalar el software MPLAB IDE, conectar el cable USB entre la PC y el kit ¡y listo!. Entre los requerimientos necesarios para utilizar el Starter kit están:

- PC con CD-ROM drive
- Un puerto USB disponible, o un USB hub
- Microsoft Windows 2000 SP4, Windows XP SP2, p Windows Vista (32-bit)
- También se requiere una USB Flash Drive, que no está incluida en el kit

La capacidad USB OTG del PIC24FJ256GB106 le permite trabajar en dos modalidades: como host (anfitrión) o como device (dispositivo). Debido a esta cualidad, el Starter Kit fue dotado con todo el hardware necesario para que el PIC trabaje en cualquiera de sus dos modalidades. El modo de anfitrión es el modo predeterminado. En

este modo, el kit funciona independientemente de un PC excepto por el cable USB que le brinda alimentación, este cable va de un puerto USB del PC al conector USB mini B al lado izquierdo de la tarjeta [*9*].

Como muestra la Figura 2.3, la tarjeta está claramente divida por una línea blanca vertical. Esta línea, separa en la izquierda a un depurador o “debugger” y en su derecha a todos los componentes de la aplicación. El debugger o depurador**,** es mejor conocido por su nombre completo como In-Circuit-Debugger (ICD).

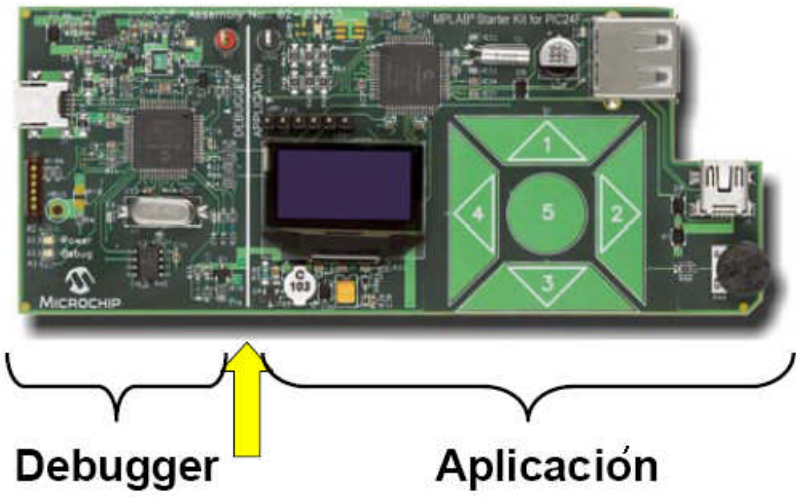


Figura. 2.3. Tarjeta PIC24F Debugger y Aplicación

De aquí en adelante, se describirá en la sección 2.1.1 todo lo referente a un ICD, y en las secciones siguientes se describirán los componentes de hardware y software de la aplicación.

2.1.1 EL IN-CIRCUIT-DEBUGGER.

El ICD es una herramienta que presenta la ventaja de analizar el sistema durante la ejecución; es decir, el ICD permite al usuario “ver” dentro del microcontrolador para detectar exactamente que efecto, tanto en hardware como en software, tienen las instrucciones en tiempo real y si estas producen algún error. Además de servir para depurar, el ICD sirve también como programador, siendo así una solución todo en uno [*10*].

El In-Circuit-Debugger es una pieza de hardware que se conecta entre el PC y el microcontrolador para depurar el firmware de la aplicación en tiempo real de manera rápida y fácil (Figura 2.4) [*3*]. El programa, en este caso MPLAB, puede crear puntos de rompimiento o breakpoints, correr el código, ejecutarlo paso a paso, examinar las variables, registros y demás. Un ICD ocupa cierto espacio de memoria y algunos pines de E/S del microcontrolador durante las operaciones de depuración.

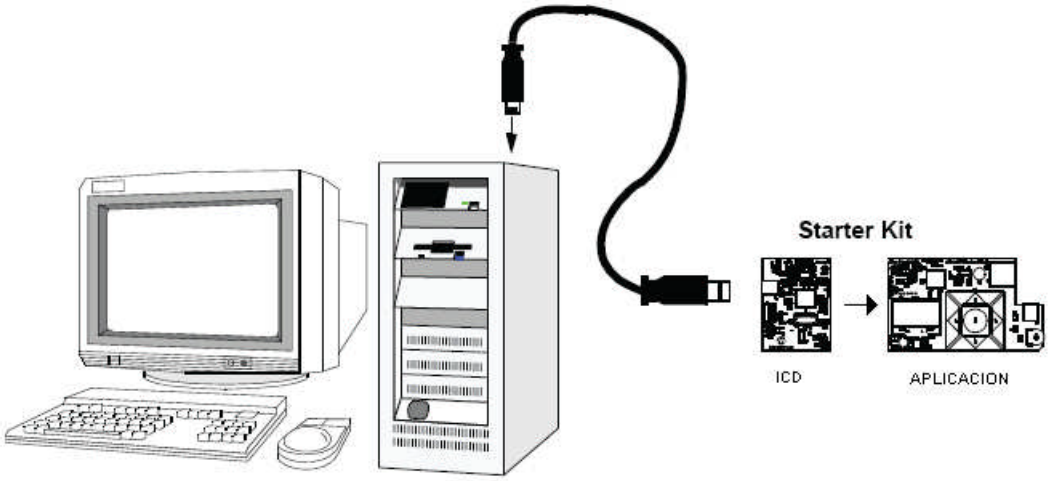


Figura. 2.4. Configuración del Starter Kit e identificación del ICD

Sí quisiéramos arreglar el motor de un auto, abrimos el capot, prendemos el motor y le conectamos equipo de diagnóstico y monitoreo, luego esperamos a ver en cuando se produce el error. Cambiamos de marcha, aceleramos, frenamos y hacemos todas las acciones posibles hasta que encontramos la falla. Análogamente, el microcontrolador es el motor, el equipo de diagnostico y monitoreo es el ICD, y las acciones de frenado, aceleración y mas serían los breakpoints, correr paso a paso ejecutadas desde MPLAB. Sí no se tuviera un ICD, se tendría que suponer cual es la posible falla, arreglarla y encender de nuevo el auto para ver si se corrigió, sí no se corrige, apagar el auto y empezar el procedimiento de nuevo [*10*].

2.1.1.1 Definición. La base de un ICD es un sistema de monitoreo de firmware. Pero para ser más que eso, es necesario agregar al procesador cristales azules conocidos como "silicon features[1]". John Day, un analista de Microhip Technology, redacta una

[1] **Silicon feature se refiere a componentes de hardware de alta integración**

definición técnica de un ICD, esta es: "*Herramientas de desarrollo con silicon features especiales para sustentar la depuración de código y tener comunicación serial entre el anfitrión (PC) y el microcontrolador se describen como debuggers"* [*11*].

Esta característica, junto con el protocolo In-Circuit Serial Programming (ICSP), o programación serial en circuito, ofrece depuración y carga de un programa flash in-circuit desde la interfaz gráfica del usuario del MPLAB IDE. El diseñador puede desarrollar y depurar el código fuente viendo las variables, ejecutando paso a paso, y estableciendo breakpoints o corriendo a toda velocidad. Con estas opciones se puede realizar una revisión completa de la ejecución en tiempo real del hardware y el firmware [*12*].

2.1.1.2 Componentes del ICD. El ICD consiste en tres componentes básicos: el módulo ICD y el ICD header.

Cuando la orden llega de MPLAB, el módulo ICD programa y ejecuta comandos de depuración al microcontrolador usando el protocolo ICSP, el cual esta comunicado mediante un cable de cinco hilos. Un jack modular puede ser diseñado en la circuitería de la tarjeta, como es el caso del Starter Kit, para soportar directamente la conexión al módulo ICD, o el header ICD puede ser usado para conectarse en un socket DIP [*13*].

El header ICD contiene un microcontrolador y un jack modular para conectarse al módulo ICD. El header ICD puede ser conectado en cualquiera de las tarjetas demo o en el hardware hecho por el usuario. En el caso del kit, el header ICD está integrado en la misma tarjeta [*13*]. Generalmente un ICD también ofrece LEDs, DIP switches, potenciómetros análogos y un área para desarrollo. El Starter Kit no es la excepción, ya que posee LEDs para mostrar un correcto funcionamiento, alimentación, etc [*13*].

2.1.1.3 El ICD del Starter Kit PIC24F. En su configuración por defecto, el Starter Kit funciona como una tarjeta que se alimenta del bus USB. El voltaje que el kit toma es de 5 Voltios, pero viene sin regularse. El primer elemento que encuentra el voltaje entrante es un regulador lineal de voltaje llamado "low-dropout (LOD) linear regulator MC1727" [*8*]

que regula el voltaje a 3.3 voltios. Para indicar que el ICD y el resto de la tarjeta están alimentados correctamente se verifica el LED verde (LED2) de la tarjeta.

El lado izquierdo de la tarjeta, el ICD, está controlado por el PIC18F6750 que funciona a 48 MHz. Este PIC posee un motor USB que provee la interfaz de comunicación entre el Starter Kit y el PC. Este microcontrolador administra la depuración o programación del PIC24FJ256GB106 controlando las señales MCLR, PGC1/EMUC1 y OGD1/EMUD1. La alimentación del PIC24F se cambia entre "on" y "off" mediante un transistor PNP (Q1) configurado como switch. El PIC18F67J50 también provee sincronización o "clocking". Una memoria EEPROM serial 25LC010A es usada para almacenar el número serial del Starter Kit y la información necesaria para controlar la depuración [*9*].

2.1.1.4 Diagrama de bloques del ICD. El siguiente cuadro se puede observar el diagrama de bloques del depurador/programador del Starter Kit. En la Figura 2.5 [*14*] es necesario prestar atención a que el ICD posee un cristal de 12 MHz el cual es diferente del cristal que se usa para la aplicación y las iníciales ICSP marcan para In-Circuit-Serial-Programmer, el cual es el programador del PIC24F:

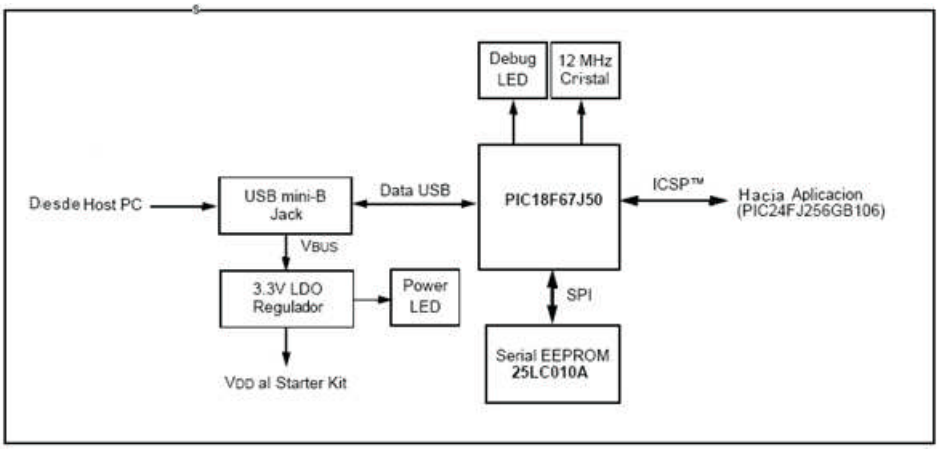


Figura. 2.5. Diagrama de bloques del ICD del Starter Kit PIC24F

Como expresa el diagrama de bloques de la Figura 2.5, el conector USB mini B, transmite tanto los datos del bus al PIC18F67J50, como la alimentación al regulador de voltaje. Al configurar la tarjeta para trabajar en modo device, y si no se desea usar la

alimentación del USB, se puede cambiar la configuración para utilizar una alimentación externa, que a su vez también se entregará al regulador. Una explicación más detallada de este tema se da en el capítulo cuarto, en el tutorial del Starter Kit PIC24F. [*14*]

2.1.1.5 El ICD vs emuladores y simuladores. Además del ICD, existen otras herramientas de desarrollo tales como los In-Circuit Emulators o In-Circuit Simulators. Es necesario comprender cual herramienta se debe usar para cada diseño y porque el ICD fue escogido para la tarjeta Starter Kit PIC24F.

Parámetros	Burn and learn	Software simulation	In-circuit simulation	In-circuit emulator	In-circuit debugger
Ejecución en Real-time	Si	No	No	Si	Si
No pérdida de pines de E/S del MCU al depurar	Si	Si/A veces	Si/A veces	Si	No
Ver y modificar RAM y periféricos	No	Si	Si/Limitada	Si	Si
Necesidad de E/S para depuración	Ninguna	Ninguna	Puerto serial, algunos pines E/S	Ninguna	Dos o menos E/S
Ejecución paso a paso	Ninguna	Si	Si	Si	Si
Breakpoints por hardware	Ninguna	Si/Ilimitada	Una/Ilimitada	Una /Ilimitada	Si/ Una

Tabla. 2.1. ICD vs emuladores y simuladores

Como se aprecia en la tabla 2.1, el ICD es sin duda la mejor opción ya que presenta menor complejidad al conectarse con la tarjeta, no perdida de pines, ejecución en real-time, etc.

2.1.2 EL MICROCONTROLADOR

En el corazón de la aplicación del StarterKit se encuentra el PIC24FJ256GB106, miembro de la familia PIC24FJ256GB110. Es un chip de 16 bits con alto rendimiento de procesamiento, modernas cualidades, un set extendido de periféricos, y con la novedosa capacidad de conectividad USB On-The-Go; en la cual puede trabajar en modo “Host” o modo “Device”. En el StarterKit, el PIC está en su presentación de 64 pines como se muestra en la Figura 2.6 [*14*].

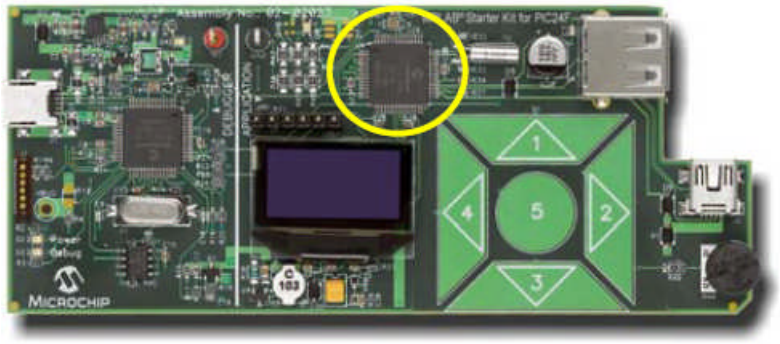


Figura. 2.6. El PIC24FJ256GB106 en el Starter Kit PIC24F

Las grandes bondades de los microcontroladores de 16 bits ya han sido descritas a lo largo de este documento en los numerales 1.4, 1.5, 1.6 y 1.7; sin embargo, es necesario una descripción más amplia, especialmente de los módulos que sea aprovechan en esta tarjeta, la cual será redactada en las siguientes secciones.

2.1.3 EL TECLADO CAPACITIVO

Cinco botones capacitivos proveen al usuario de un teclado para ingresar datos y manejar la aplicación tal y como se ve en la Figura 2.7 [*14*] cada uno posee un número[2]. Los botones se conectan al módulo Charge-Time-Measure-Unit que será explicado con mayor detenimiento en la siguiente sección.

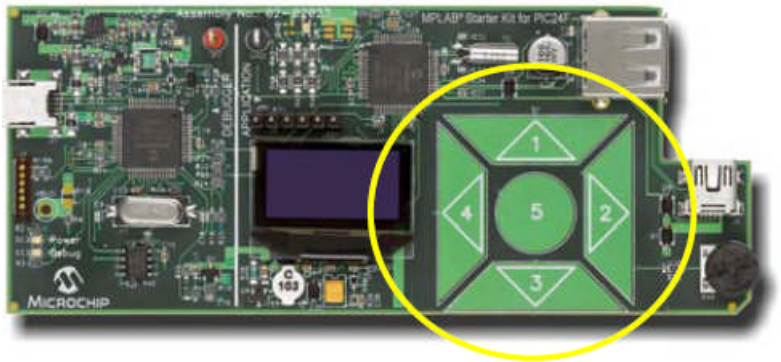


Figura. 2.7. El teclado capacitivo en el Starter Kit PIC24F

[2] **Es importante que se preste atención a que número corresponde a cada botón: 1 arriba, 2 derecha, 3 abajo, 4 izquierda y 5 centro/seleccionar.**

2.1.3.1 Definición. Es un periférico de entrada para el usuario que aprovecha la capacitancia propia de la yema de los dedos de un ser humano para funcionar. Sus botones no generan ninguna señal eléctrica al ser presionados, únicamente son parte de un circuito conectado a un sistema que censa, si una capacitancia externa es inducida. Este sistema, puede ser un oscilador RC que trabaja juntamente con comparadores, o puede ser el módulo CTMU del microcontrolador. A continuación se explicaran ambos sistemas. Los teclados capacitivos, aunque parezca lo contrario, son de gran robustez, calidad y durabilidad, pero también son costosos [*15*].

2.1.3.2 La tecnología Touch-sensing. La tecnología touch-sensing esta crecientemente siendo adoptada por los diseñadores de sistemas embebidos para mejorar la apariencia y la durabilidad de las interfaces en las aplicaciones, dispositivos electrónicos, aparatos electrónicos en medicina, automóviles y muchas otras aplicaciones en el mercado. Es por esto, que el Starter Kit aprovecha la necesidad de un teclado para introducir esta tecnología [*15*].

Los microcontroladores que han dominado esta tecnología han sido los PIC16F887, PIC16F616 Y PIC16F690. Muchas técnicas para censamiento capacitivo son usadas actualmente en la industria. Muchas de ellas son basadas en medir la frecuencia o ciclo de trabajo el cual se altera al introducir una capacitancia adicional del dedo de una persona. Otros métodos usan una carga balanceando o elevando y disminuyendo las mediciones del tiempo. Esta solución mide la frecuencia usando un oscilador free-running RC [*15*].

Mientras que touch-sensing ha estado rondando los laboratorios de electrónica por más de 50 años, por fin se está volviendo popular. Muchas revistas en línea la describen como la tecnología “en boga”. Un ejemplo clásico de un switch capacitivo, es la lámpara touch. La lámpara touch es un simple switch capacitivo que enciende/apaga una bombilla o funciona también como dimer [*15*].

Los nuevos MCU permiten un control más integrado de la touch-technology. Estos proveen la habilidad para ejecutar censado capacitivo, toma de decisiones, acciones en respuesta y otras tareas pertinentes para el sistema también [*15*].

2.1.3.3 Funcionamiento. Un sensor capacitivo se activa al detectar una capacitancia externa inducida en el sensor por la presencia del dedo de una persona. Estos sensores, no detectan la deformación del teclado; es decir, no funcionan de una manera mecánica o por deformación como los sensores piezoelectricos, no se activan por ejercer presión sobre ellos, los teclados que funcionan de esta manera son los teclados resistivos.

Como se menciona anteriormente, los sensores capacitivos se activan al detectar un aumento en su capacitancia, producido por el acercamiento de un dedo; por esta razón, no se activan si se los presiona con un lápiz o un puntero de plástico, metal o de cualquier otro material. Para ilustrar de mejor manera el funcionamiento de estos sensores refiérase a la Figura 2.8 [*19*].

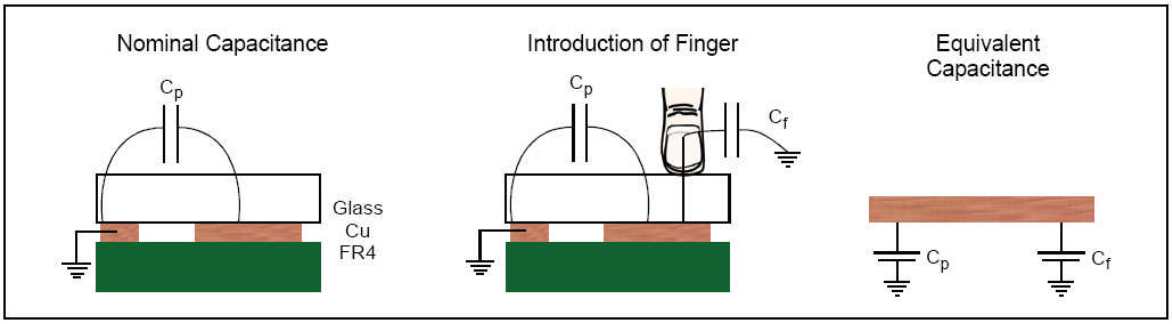


Figura. 2.8. Funcionamiento de un teclado capacitivo

La Figura 2.8 nos indica que el detectar que un botón ha sido presionado, se debe a la capacitancia introducida por el circuito tierra-dedo-vidrio, el cual es paralelo a la capacitancia parasita del circuito natural a tierra. Las capacitancias en paralelo se suman; entonces, cuando un dedo se acerca al teclado se incrementa la capacitancia total del circuito. Este cambio, en manera porcentual, se expresa en la ecuación 2.1:

$$\Delta C\% = \left(\left(C_p + C_f\right) - C_p\right)/C_p = C_p/C_f$$

Ecuación 2.1

Este cambio establece el criterio que se necesita para detectar el que un dedo introducirá una capacitancia adicional causando un cambio en la constante de tiempo RC del oscilador. Dicho criterio es "*Incrementar la constante RC hará decrecer la frecuencia del oscilador y este cambio es detectable por el microcontrolador*" [*15*].

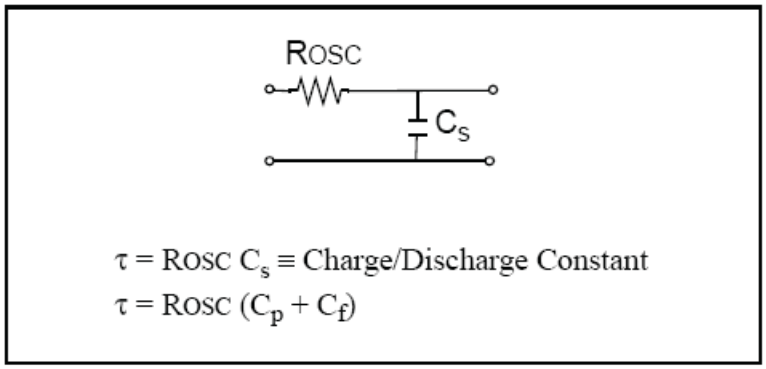


Figura. 2.9. Circuito oscilador RC

La Figura 2.9. [*19*] muestra el circuitoque simula el circuito que se producirá para estudiarlo y obtener las ecuacion necesarias. Debido a que el valor de C_f es muy pequeño, es preferible que el valor de C_p sea pequeño. Esto es de preferencia por que si C_p es de un valor bajo, resultará en un gran cambio porcentual en la capacitancia y la frecuencia. Para que este concepto sea más facilmente comprendido, refierase a la ecuacion 2.1 donde se expresa el porcentaje en el cambio de capacitancia con respecto a los valores de C_py C_f.

El presionar de un dedo puede variar en cualquier valor entre un rango de 5 a 15 pF, esto como un lineamiento. La capacitancia de una dedo no debe ser asumida como una constante o un valor absoluto. El como hacer la capacitancia C_p de un valor pequeño y otros factores seran descritos posteriormente [*15*].

La ecuación de la capacitancia queda descrita como:

$$C = \frac{\varepsilon_0 \varepsilon_r A}{d}$$

Ecuación 2.2

donde:

ε_0 *es Permitividad del espacio libre*

ε_r *es Constante dieléctrica relativa*

A *es Área de la placa*

d *es Distancia entre las placas capacitivas*

C *es Capacitancia*

Ahora, con un concepto mas claro de que es aquello que hay que detectar, se necesita un oscilador cuya frecuencia sea dependiente de nuestra placa que sensa la capacitancia, C_s. Este diseño usa un oscilador de relajación para crear una frecuencia dependiente del valor de la capacitancia. El valor de la resistencia del oscilador RC es un parámetro de diseño cuyo objetivo es poner la frecuencia de oscilación en un rango de 100 a 400 KHz. La frecuencia exacta no es importante, pero tener una frecuencia alta produce mas cuentas en el proceso de medición, consecuentemente una mejor resolución se tiene a una frecuencia alta.

Para detectar que un botón ha sido presionado, primero se debe configurar propiamente el sistema. Luego, los pasos clave son los siguientes:

1. Generar una señal de oscilación a través del capacitor del sensor.
2. Contar los flancos ascendentes usando T1CKI.
3. Al final de un período de medición fijo, obtener la lectura (frequencia en cuentas).
4. Determinar si la frecuencia actual es menor en valor al valor normal promedio cuando no esta presionado el teclado.

2.1.3.4 El Oscilador RC. El oscilador de relajación, es un oscilador RC free-running que usa dos comparadores con un latch SR para cambiar la dirección del voltaje del capacitor del sensor en carga o descarga. El oscilador cargará o descargará el capacitor a un ritmo determinado por la constante de tiempo RC y se cargará entre los límites superior e inferior por la entradas positivas a los comparadores. El tiempo requerido para cargar de un límite inferior a un límite superior y descargarlo de vuelta al límite inferior es el período del oscilador [*15*].

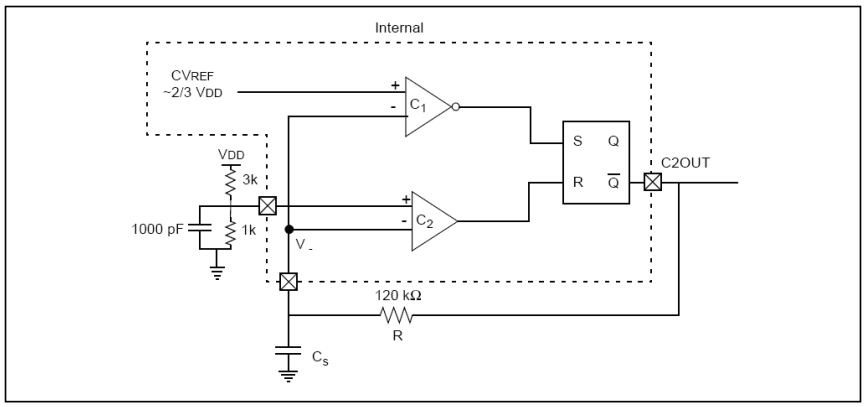


Figura. 2.10. Circuito RC y comparadores para determinar cambios en capacitancia [*15*]

El circuito oscilador es mostrado en la Figura 2.10 [*19*]. Las entradas positivas de los comparadores son los límites superior e inferior de carga. C1+ es interno, pero C2+ debe ser alimentado externamente para configurar el límite inferior. El capacitor de 1000pF está en ese lugar para rechazar el ruido de alta frecuencia que provee de la fuente de voltaje y asegura un límite inferior estable. El voltaje V-, se cargará y descargará entre los límites y es gobernado por señales de nivel lógico en las salidas de los dos

comparadores C2OUT, dicha salida esta configurada para $\bar{Q}$ para obtener un comportamiento adecuado de carga y descarga. La resistencia feedback R, forma la red RC con el sensor en la placa nombrada como C_s, este proceso se muestra en la Figura 2.11 [*19*] y las diferentes regiones de carga y descarga se muestran en la Figura 2.12 [*19*].

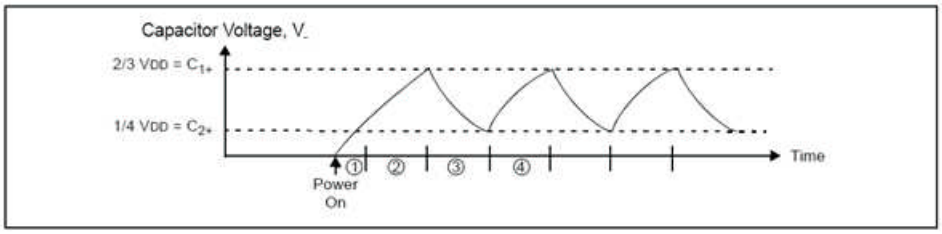


Figura. 2.11. Ciclo de carga y descarga del capacitor

La tabla 2.2 explica claramente las regiones del ciclo de carga y descarga del capacitor. Esto servirá para la comprensión de los ciclos del CTMU.

Time	Set Bit	Reset Bit	C2OUT Result	Action
1	C1+ > V_ S = 0	C2+ > V_ R = 1	$\Rightarrow \overline{Q} = 1$	"Begin Charging"
2	C1+ > V_ S = 0	C2+ < V_ R = 0	$\Rightarrow \overline{Q} = \overline{Q}_{n-1}$	"Last State (Charging)"
2 ⇒ 3	C1+ < V_ S = 1	C2+ < V_ R = 0	$\Rightarrow \overline{Q} = 0$	"Begin Discharging"
3	C1+ > V_ S = 0	C2+ > V_ R = 0	$\Rightarrow \overline{Q} = \overline{Q}_{n-1}$	"Last State (Discharging)"
3 ⇒ 4	C1+ > V_ S = 0	C2+ > V_ R = 1	$\Rightarrow \overline{Q} = 1$	"Begin Charging"
4 = 2	Charge cycle begins to repeat as region two.			

Tabla. 2.2. Regiones del ciclo de carga y descarga del capacitor

2.1.3.5 El teclado capacitivo en el Starter Kit PIC24F. En el Starter Kit PIC24F, no existe un oscilador RC con comparadores como el descrito anteriormente, pero era necesaria una explicación sobre este tema para que este documento sea una guía teórica más completa sobre la tecnología touch-sensing. En el Starter kit PIC24F, se utiliza una alternativa al oscilador RC, es el módulo CTMU. Este módulo cumple exactamente con la misma tarea de censar variación en la capacitancia, pero lo hace sin la necesidad de un

circuito RC ni comparadores. El módulo solo y de manera independiente puede cumplir con este trabajo. Ahora, es necesario recalcar que el CTMU sirve para varias y versátiles aplicaciones, un teclado touch es solo una de ellas. Puede trabajar con cualquier sensor o dispositivo que base su funcionamiento en capacitancia o en señales analógicas.

Con el objetivo de controlar las aplicaciones diseñadas por los programadores, el Starter Kit PIC24F usa el módulo Charge Time Measure Unit (Unidad de Medición de Tiempo de Carga) para implementar un teclado capacitivo touch-sensing, sensible al tacto, sobre el Starter Kit.

El touch pad de la siguiente Figura es usado para seleccionar las opciones de los menús y que escoger la opción deseada por el usuario. Cada pantalla que contenga un menú podrá usar el teclado y cuando se seleccione una opción con las flechas de navegación (botones 1, 2, 3, 4) las opciones serán subrayadas y se seleccionan con el botón central, el botón 5. Todos los botones están mostrados en la Figura 2.12 [*3*].

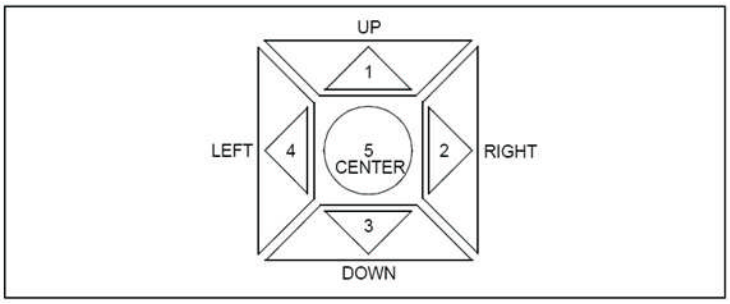


Figura. 2.12. Teclado capacitivo del Starter Kit PIC24F

2.1.4 LA PANTALLA OLED.

Para dotar de una retroalimentación gráfica al usuario el kit tiene una pantalla LED orgánica monocromática de 128x64 pixeles, la cual está conectada a su respectivo drive y este a su vez al puerto paralelo del PIC. [*14*]

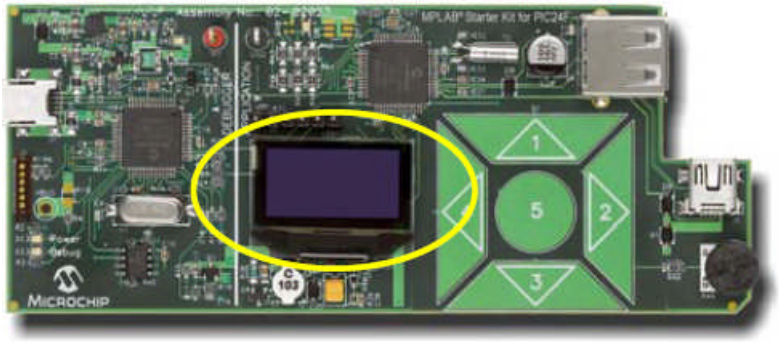


Figura. 2.13. La pantalla OLED en el Starter kit PIC24F

2.1.4.1 La tecnología OLED. LED, no solo son las siglas para Light Emiting Diode (diodo emisor de luz); también, son las siglas para identificar a la tecnología más moderna en iluminación. Su funcionamiento es el mismo de un diodo normal, solo que al usar distintos materiales es posible obtener luz. La luz obtenida puede ser de diferentes colores, puede emitir hasta más del doble lúmenes que las lámparas incandescentes, y por si fuera poco, consumen hasta 10 veces menos electricidad. Al utilizar componentes orgánicos en los diodos, la tecnología se denomina como *Organic* LED [*16*].

Para explicar en qué consiste un OLED, deberíamos centrarnos primero en el mundo de la electrónica molecular, o moletrónica. Ésta es la rama de la ciencia que estudia el uso de moléculas orgánicas en la electrónica [*16*].

El diodo orgánico se compone de tres partes, dos de ellas, son las que forman parte del diodo en sí: el ánodo y el cátodo. La tercera es la capa de material orgánico, que es donde se produce la luz. El sustrato es la parte que soporta el OLED y está compuesto de un material plástico transparente, cristal, o algún tipo de metal.

La producción de luz la hacen de una forma muy parecida a los LEDs convencionales. El proceso se llama Electrofosforescencia [*17*].

La **Electrofosforescencia**, es el fenómeno producido cuando la corriente eléctrica transporta electrones desde el cátodo hasta el ánodo, haciendo que éstos traviesen los dos polímeros: la capa conductiva y la emisora. Como los electrones se colocan en los huecos de un átomo que “busca” un electrón, en ese movimiento sueltan su energía extra en

forma de fotones, y son estos fotones la luz que emiten los OLED, tal como muestra la Figura 2.14 [*17*].

Figura. 2.14. Esquema de un foco OLED

2.1.5 LED RGB

Para no dejar de lado los colores, el kit ha sido complementado con un LED RGB (Figura 2.15) [*14*] que gracias a sus componentes azul, rojo y verde puede producir casi cualquier color con gran precisión. La generación de los colores esta contralada por modulaciones PWM que se producen desde el PIC.

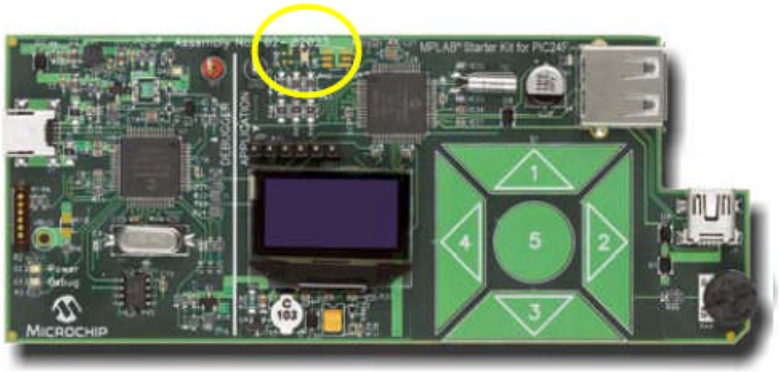


Figura. 2.15. El LED RGB en el Starter kit PIC24F

El LED RGB o Red, Green and Blue varia sus componentes de sus tres colores para producir así cualquier color posible entre 16 millones. El microcontrolador utiliza la modulación PWM para controlar este componente de Hardware. El LED usado en el Starter Kit es de pequeño tamaño, no más de 5 milímetros, como se muestra en la Figura 2.16 [*14*].

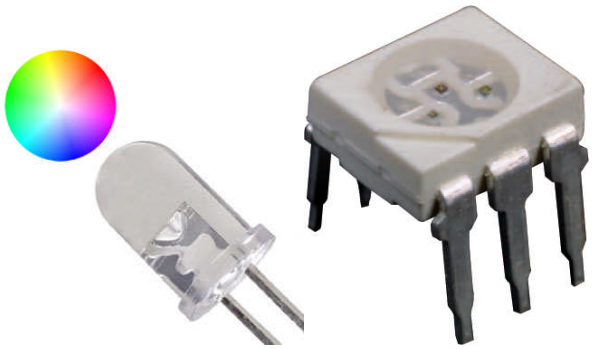


Figura. 2.16. Presentación del LED RGB

2.1.6. POTENCIÓMETRO

Adicionalmente, un potenciómetro fue agregado para que el usuario lo use como entrada de datos o manejo de las aplicaciones. Es un potenciómetro de 10 KΩ, Figura 2.17 [*14*].

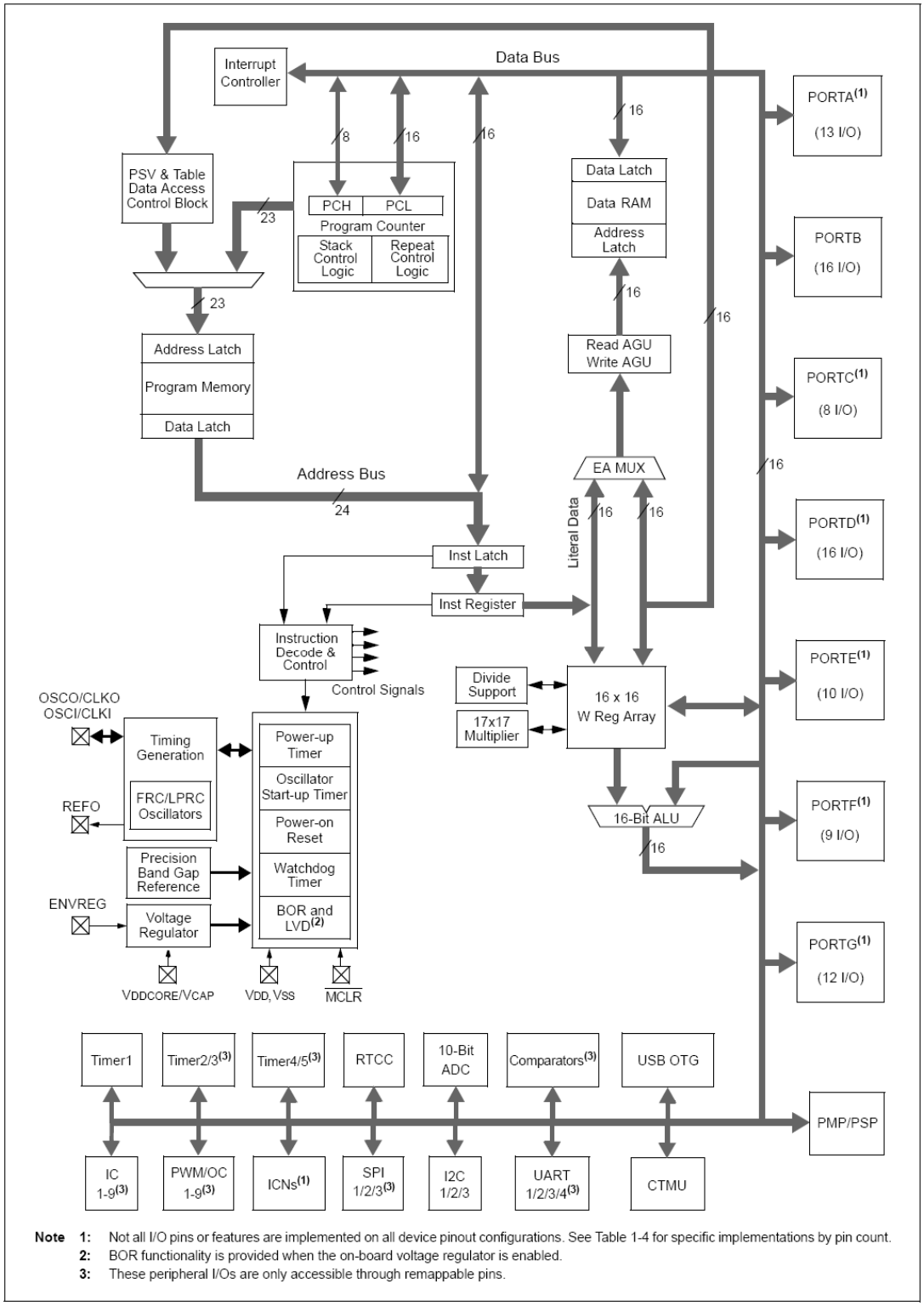
Figura. 2.17. El potenciómetro en el Starter kit PIC24F

2.2. EL PIC24FJ256GB106

Repetidamente a lo largo del documento se ha hablado sobre la arquitectura de 16 bits Harvard modificada, la capacidad de procesamiento, el set expandido de periféricos, la interfaz USB OTG, etc. Ahora, es tiempo de analizar más de cerca a este poderoso microcontrolador. Se empezará por una descripción del diagrama de bloques, en el cual se podrá constatar todas sus características expuestas en las secciones anteriores. Es

necesario que la Figura 2.18 sea vista con detenimiento, se analice los buses de datos y de direcciones, las conexiones de periféricos, las conexiones de su ALU, etc.

2.2.1 DIAGRAMA DE BLOQUES



Note 1: Not all I/O pins or features are implemented on all device pinout configurations. See Table 1-4 for specific implementations by pin count.
2: BOR functionality is provided when the on-board voltage regulator is enabled.
3: These peripheral I/Os are only accessible through remappable pins.

Figura. 2.18. Diagrama de bloques de la familia PIC24FJ256GB110 [8]

2.2.2. EL CPU

2.2.2.1. Descripción. El CPU de la familia PIC24F tiene una arquitectura Harvard modificada con una palabra de instrucciones de 24 bits de ancho con una longitud variable del campo opcode. El contador del programa (PC) es de 23 bits de ancho. Una sola instrucción de un ciclo con mecanismo prefetch es usado para ayudar a mantener un buen throughput y provee una ejecución predecible. Todas las instrucciones se ejecutan en un solo ciclo, con excepción de instrucciones que cambian el flujo del programa y la instrucción double-word (MOV.D) [*8*].

Los dispositivos PIC24F tienen 16 registros de trabajo de 16 bits cada uno en el modelo de programador. Cada uno de los registros de trabajo puede actuar como registro para datos, o dirección de offset. El decimosexto registro de trabajo (W15) opera como un puntero de pila de software para interrupciones y llamadas [*8*].

La arquitectura de conjunto de instrucciones, ISA por Instruction Set Architecture, ha sido significativamente ampliada mucho más alla de aquella de los PIC18, pero mantiene un nivel aceptable de compatibilidad. Todas las intrucciones de los PIC18 y sus modos de direccionamiento son respaldados en los PIC24, o en caso de no serlos, existen macros simples. Muchos de las ampliaciones en la ISA han sido dirigidos en función de las necesidades de eficiencia del compilador [*8*].

Para la mayoría de instrucciones, el núcleo es capaz de ejecutar una lectura de memoria de datos (o de programa), una lectura de un registro de trabajo (datos), una escritura en memoria de datos y una lectura de instrucción de programa, todo esto por cada ciclo de instrucción. Como resultado, una instrucción de hasta tres parámetros puede ser ejecutada en un solo ciclo, permitiendo operaciones trinarias (eso es A + B = C) [*8*].

Un multiplicador de alta velocidad de 17 bits por 17 bits ha sido incluido para incrementar significativamente la capacidad y el throughout del núcleo aritmético. El multiplicador soporta los modos con signo, sin signo y mezclado de los operandos,

también soporta multiplicación de enteros de 16 bits por 16 bits o 8 bits por 8 bits. Toda la operación de multiplicacion se ejecuta en un solo ciclo. Los modos que soporta el multiplicador son:

1. Valores con signo de 16 bits x 16 bits
2. Valores sin signo de 16 bits x 16 bits
3. Valor con signo de 16 bits x valor sin signo de 5 bits
4. Valor sin signo de 16 bits x valor sin signo de 16 bits
5. Valor sin signo de 16 bits x valor sin signo de 5 bits
6. Valor sin signo de 16 bits x valor con signo de 16 bits
7. Valor sin signo de 8 bits x valor sin signo de 8 bits

La ALU de 16 bits ha sido modificada para incrementar la capacidad para operar divisiones de enteros con hardware que le asiste con un algoritmo de división iterante non-restoring. Este opera en conjunto con el mecanismo de loop REPEAT y una selección de instrucciones iterativas de división para soportar 32 bits (o 16 bits) divididos por enteros con o sin signo de 16 bits. Todas las operaciones de división requieren 19 ciclos para completarse, pero es posible interrumpirlas en cualquier ciclo [*8*].

La familia PIC24F tiene un esquema de vectores con hasta 8 fuentes para traps sin máscara y hasta 118 fuentes de interrupciones. Cada fuente de interrupción puede ser asignada con una de siete niveles de prioridad. A continuación el diagrama de bloques del CPU [*8*].

2.2.2.2. Diagrama de bloques del CPU. En la Figura 2.19 [8], se presenta el diagrama de bloques del CPU, sus interrupciones, registros, buses y otros.

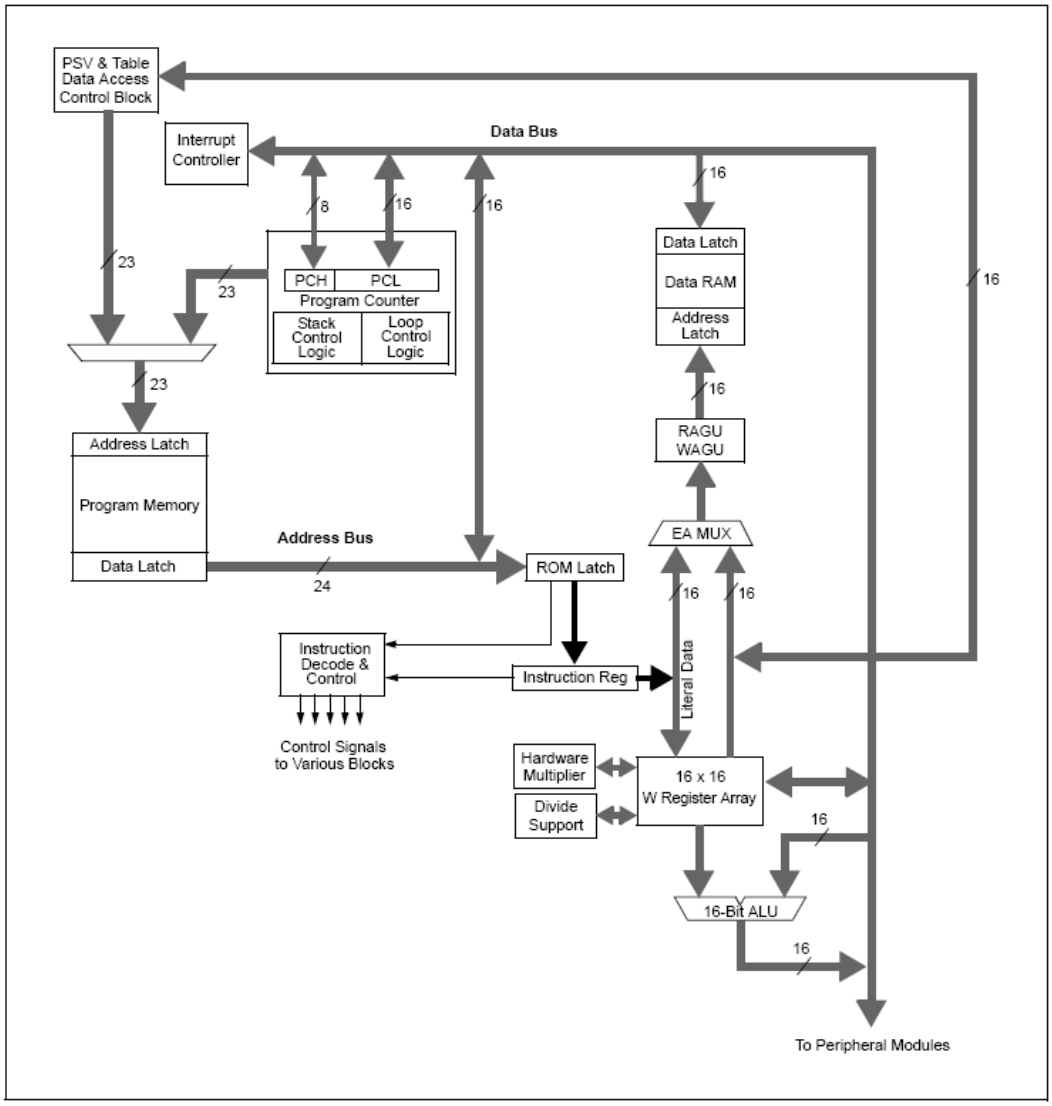


Figura. 2.19. Diagrama de bloques del CPU de la familia PIC24FJ256GB110

2.2.2.3. Organización de la memoria del programa. Como un dispositivo de arquitectura Harvard, los PIC24F separan el espacio físico y los buses de la memoria de datos de la memoria del programa. Esta arquitectura también permite el acceso directo a la memoria del programa desde el espacio de datos durante la ejecución del código. El espacio es direccionable por un vector de 24 bits derivado del contador del programa (PC) durante la ejecución del programa o de la tabla de operaciones [*8*].

El acceso del usuario a la memoria del programa es restringido para la mitad inferior del rango de direcciones (000000h hasta 7FFFFFh). La única excepción es para permitir el acceso a los bits de configuración y las secciones de ID del dispositivo.

El mapa de la memoria para el PIC24FJ256GB106 está mostrado en la Figura 2.20 [8]:

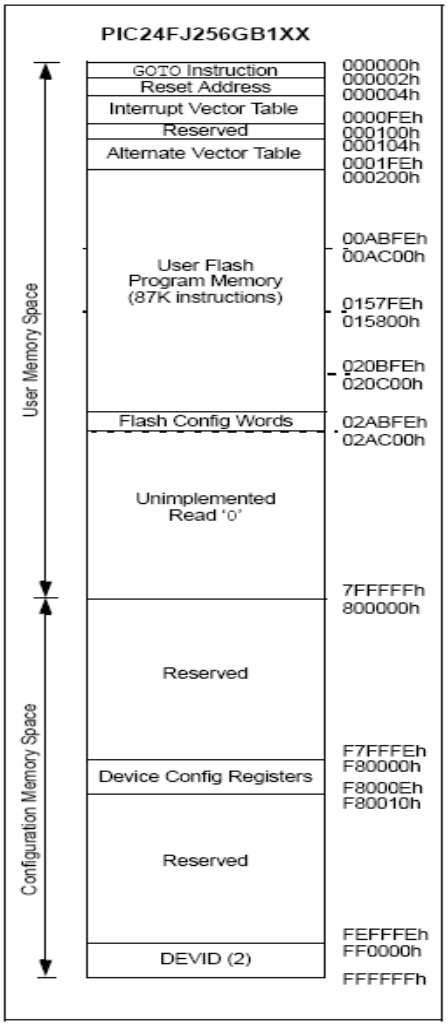


Figura. 2.20. Mapa de memoria de la familia PIC24FJ256GB110

2.2.2.4. Mapa de registros. En la Figura 2.21 [8] se representa el mapa de registros de la familia de PIC24F. Se debe prestar atención especial a los registros de trabajo.

Register(s) Name	Description
W0 through W15	Working Register Array
PC	23-Bit Program Counter
SR	ALU STATUS Register
SPLIM	Stack Pointer Limit Value Register
TBLPAG	Table Memory Page Address Register
PSVPAG	Program Space Visibility Page Address Register
RCOUNT	Repeat Loop Counter Register
CORCON	CPU Control Register

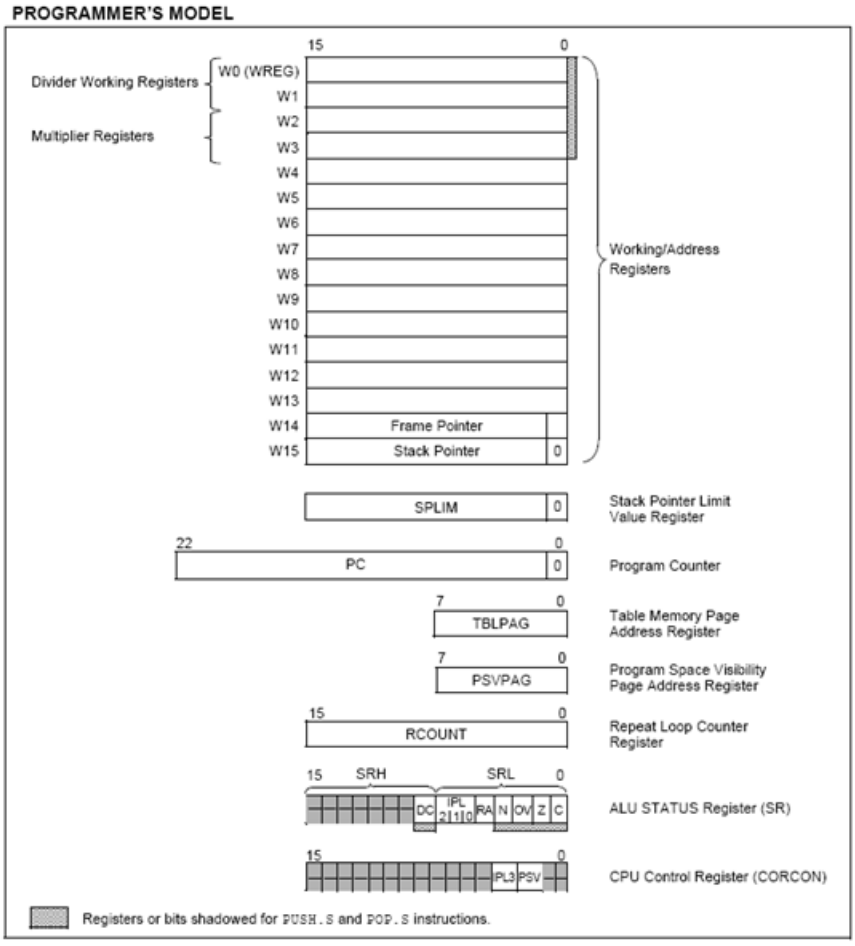


Figura. 2.21. Mapa de registros de la familia PIC24FJ256GB110

2.2.3. CAPACIDAD USB ON-THE-GO EN MODO HOST

En la Figura 2.22 [*14*] conector receptor USB tipo A se introdujo en la tarjeta para explotar la utilidad del PIC mencionada anteriormente USB OTG en su función en modo Host.

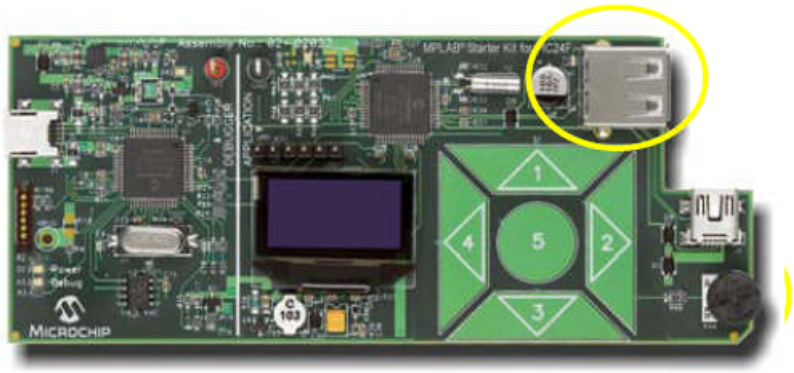


Figura 2.22. El puerto USB tipo A receptor en la tarjeta Starter kit PIC24F

El microcontrolador usado en la tarjeta contiene un motor de interfaz serial conocido como SIE (Serial Interface Engine), el cual es compatible con una interfaz USB On-The-Go de full-speed (12 Mbps) o de low-speed (1.5Mbps). La capacidad USB OTG permite a la tarjeta cambiar dinámicamente entre host y device utilizando el protocolo diseñado para dispositivos OTG, el Host Negociation Protocol (HNP).

Para operar en modo host, se necesitan realizar algunas tareas. En este modo, el software del dispositivo se encarga de la configuración y de invocar a los “USB transfers”.

Las actividades (en orden) que se deben ejecutar para poner en marcha el modo HOST son:

1. Habilitar el modo host configurando el registro U1CON. Esto habilita los bits de control de USB OTG para dicho modo.
2. Habilitar las resistencias pull-down de los terminales D- y D+ mediante la configuración de los bits 4 y 5 (DPPULDWN y DMPULDWN) del registro U1OTGCON. Del mismo modo, deshabilitar las resistencias pull-up

encerando los registros DPPULUP y DMPULUP, que se encuentran en los bits 7 y 6 del registro U1OTGCON.

3. En este punto los paquetes SOF (Start Of Frame) comienzan a ser generados con el contador cargado en 12000. Se elimina el ruido en el USB limpiando el bit SOFEN, el cual es el bit 0 en el registro U1CON, para deshabilitar la generación de los paquetes SOF.
4. Habilitar las interrupciones que se puedan producir al agregar un dispositivo. Esto se logra habilitando el bit ATTACHIE, el sexto en el registro U1IE.
5. Esperar para que el dispositivo agregado genere la interrupción y se presente en el sexto bit, U1/R = 1. Luego que esto ocurra, se espera 100 ms hasta que la alimentación se estabilice.
6. Revisar el estado de los bits JSTATE y SE0 en el registro U1CON. Si el estado de JSTATE es 0, el dispositivo que se agregó es de baja velocidad. Si es de baja velocidad, se configuran los bits LSPDEN y LSPD del registro U1ADDR y U1EP0 para habilitar la operación de baja velocidad o low-speed.
7. Reiniciar el dispositivo USB configurando el bit RESET del registro U1CON por al menos durante 50 ms enviando la señal de reset, luego de este tiempo, terminar la señal de reset limpiando el bit RESET.

2.2.4. CAPACIDAD USB ON-THE-GO EN MODO DEVICE

El conector receptor USB tipo mini B también fue añadido para la funcionalidad USB OTG, pero este es para cuando la tarjeta funcione en modo Device. Este proyecto no explotará este modo, tal como se aprecia en la Figura 2.23 [*14*].

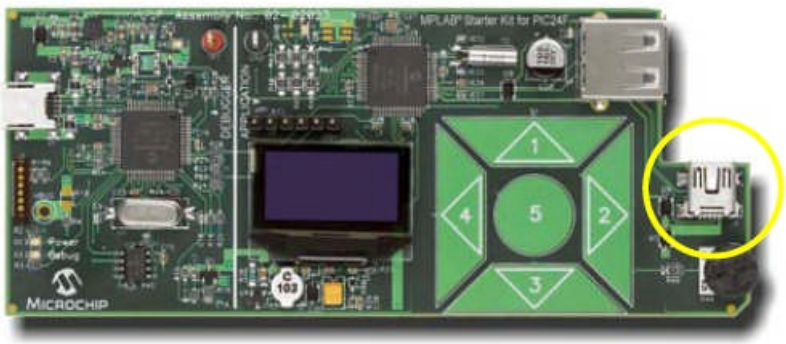


Figura. 2.23. El puerto USB mini B receptor en la tarjeta Starter kit PIC24F

Ahora, se describe como se desarrolla la tarea para comportarse como un dispositivo USB, es decir, el modo DEVICE. Los pasos que se siguen son los siguientes:

1. Reiniciar el buffer puntero Ping-Pong; luego borrar el buffer Ping-Pong con el bit de reinicio el PPBRST en el registro U1CON.
2. Deshabilitar todas las interrupciones (U1EIE y U1IE = 00h).
3. Limpiar cualquier bandera de interrupciones escribiendo FFh al registro U1IR y al U1EIR.
4. Verificar que el V_{BUS} este activado.
5. Habilitar el módulo USB configurando el bit USBEN en el registro U1CON.
6. Configurar el bit OTGEN en el registro U1OTGCON para empezar las operaciones USB.
7. Habilitar el buffer endpoint zero para recibir el primer paquete de setup configurando los bits EPRXEN y EPHSHK = 1 en el registro U1EP0.
8. Energizar el módulo USB configurando el bit USBPWR del registro U1PWRC.
9. Habilitar las resistencias pull-up para señalar cuando se adjunte el Starter Kit configurando el bit DPPULUP del registro U1OTGCON [*18*].

2.2.5. EL MÓDULO CTMU

El módulo CTMU, o unidad de medición de tiempo de carga, es un módulo analógico flexible que provee una medición muy exacta de tiempo diferencial entre fuentes que emiten pulsos como las fuentes asincrónicas [*8*]. Sus características incluyen:

- Cuatro fuentes de trigger de activación por flanco
- Control de polaridad para cada fuente de activación por flanco
- Control de secuencia de flanco
- Control de respuesta ante flancos
- Resolución de medición de tiempo de 1 nano segundo
- Fuentes muy exactas de corriente, adecuadas para medición de capacitancia

Junto con otros módulos del mismo PIC24F, el CTMU puede ser usado para mediciones exactas de tiempo, capacitancia, cambios relativos en capacitancia, o generar salidas de pulsos que sean independientes del reloj del sistema. CTMU es ideal para trabajar con sensores basados en capacitancia [*8*].

Se controla a través de dos registros, el CTMUCON y el CTMUICON. El primero, trabaja como enable; además, controla la generacion de flancos, la selección de la fuente, su polaridad y su secuencia. El segundo registro, tiene control sobre la selección y el equilibrio de la fuente de corriente [*19*].

2.2.5.1 Medición de capacitancia. CTMU trabaja usando una fuente de corriente fija para cargar un circuito. El tipo de circuito depende del tipo de medición que se quiera hacer. En caso de una medición de carga, la corriente es fija y la cantidad de tiempo que la corriente es aplicada al circuito; es fija también. La cantidad de voltaje que lee el ADC es entonces la medición de la capacitancia del circuito.

La operación de CTMU está basada en la ecuación 2.3:

$$C = I \Big/ \frac{dV}{dT}$$

Ecuación 2.3

En palabras mas simples, la cantidad de carga medida en culombios en un circuito esta definida como corriente en amperios (*I*) multiplicada por la cantidad de tiempo en segundos que la corriente fluye (*t*). Carga es también definida como la capacitancia en faradios (C) multiplicada por el voltaje del circuito (*V*). Se resuelve y sigue la ecuación 2.4:

$$I \cdot t = C \cdot V$$

Ecuación 2.4

CTMU provee una constante conocida que es la fuente de corriente. El ADC es usado para medir el (*V*) en la ecuación, dejando dos variables desconocidas: capacitancia (*C*) y el tiempo (*t*). La ecuación 2.4 es usada para calcular el tiempo por la relación: [*19*]

$$t = (C \cdot V)/I$$

Ecuación 2.5

utiliza un valor de capacitancia fijo y conocido:

$$C = (I \cdot t)/V$$

Ecuación 2.6

utiliza un periodo de tiempo fijo en el que la corriente de la fuente es aplicada al circuito [*19*].

Ahora, en la Figura 2.24 se muestra el diagrama de bloques del CTMU para comprender mejor su funcionamiento. Se debe prestar atención a que la fuente de corriente es el “corazón” del módulo.

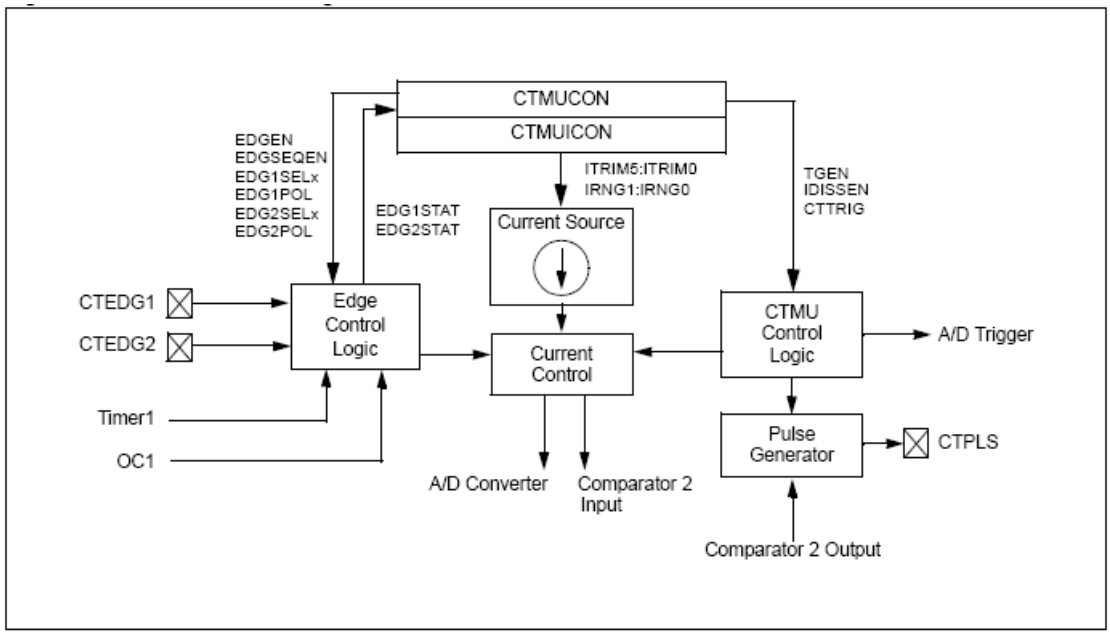


Figura. 2.24. Diagrama de bloques del CTMU [*19*]

En la Figura 2.25, se muestra las conexiones externas usadas para mediciones de capacitancia, y como el módulo CTMU y los módulos A/D están relacionados en esta aplicación. En este ejemplo tomado del datasheet de la familia PIC24FJ256GB110, se muestra el flanco que viene desde el Timer1 y esta es la configuración que se utiliza en el Starter Kit PIC24F:

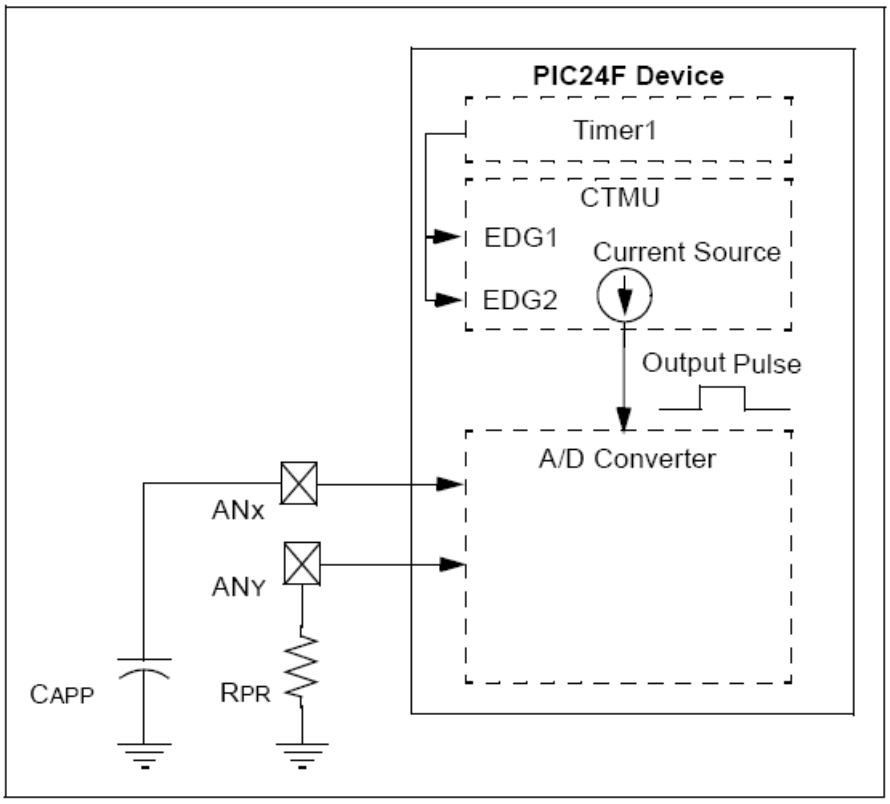


Figura. 2.25. Esquema para medición de capacitancia usando CTMU [*19*]

2.2.6. EL MÓDULO RTCC.

El módulo para manejo de fecha y hora, Real Time and Clock Calendar, fue diseñado por el porcentaje tan elevado de aplicaciones que requieren utilizar la fecha y la hora del tiempo real [*19*]. Ya que es una característica casi de facto, casi siempre se utiliza el Timer1 para programarla; claramente esto ocasiona un gasto de recursos del timer, instrucciones y ciclos que constantemente están ocupado valiosos recursos. Podría alguien imaginarse que debido a la gran capacidad de procesamiento de la familia PIC24F (16 MIPS), que el programar el Timer1 para una aplicación de reloj y calendario no haría gran diferencia, pero la verdad es que empobrecería el rendimiento notoriamente. Al liberar de esta responsabilidad al Timer1 se obtiene un sistema más robusto y una programación de mayor calidad.

El módulo RTCC está organizado en tres categorías de registros. Estas son: de control, de valor y de valor de alarma (Tabla 2.3). Solo se analizarán los registros de valor y de alarma ya que los registros de control casi no son necesarios de manipular. [*8*]

RTCPTR <1:0>	RTCC Value Register Window	
	RTCVAL<15:8>	RTCVAL<7:0>
00	MINUTES	SECONDS
01	WEEKDAY	HOURS
10	MONTH	DAY
11	—	YEAR

ALRMPTR <1:0>	Alarm Value Register Window	
	ALRMVAL<15:8>	ALRMVAL<7:0>
00	ALRMMIN	ALRMSEC
01	ALRMWD	ALRMHR
10	ALRMMNTH	ALRMDAY
11	—	—

Tabla. 2.3. Registros de RTCC [*8 ()*]

A continuación, se muestra el diagrama de bloques de este módulo, no con el fin de estudio profundo pero con el fin de tener una idea clara de su función.

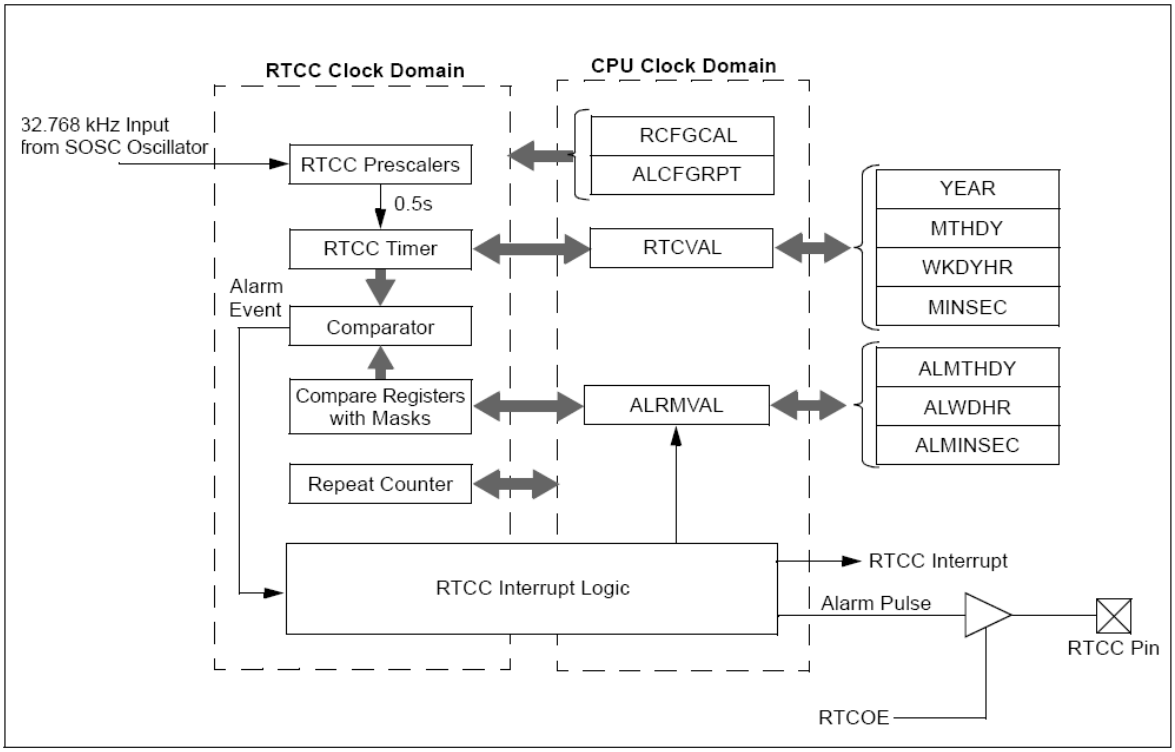


Figura. 2.26. Diagrama de bloques del módulo RTCC [*8*]

Este diagrama de bloques, presenta como la frecuencia ingresada del oscilador de 32 KHz alimenta al RTCC clock Domain donde están los prescalers, al timer y estos a su vez definen a los registros de año, mes, día, minuto, segundo a través del CPU Clock Domain. La lógica de interrupción del módulo RTCC funciona con un comparador que le sirve de entrada comparando el registro ALRMVAL con el valor actual de los registros de fecha y hora del RTCC para dar un pulso al pin RTCC siempre y cuando el registro de habilitación, el RTCOE, se lo permita.

2.3 EL SOFTWARE MPLAB IDE V8.20

El software MPLAB IDE es un programa diseñado para trabajar bajo el sistema operativo Windows®, para desarrollar aplicaciones para microcontroladores Microchip.

¿Qué es un IDE? Las siglas IDE significan Integrated Development Environment, lo que en español se traduce como ambiente de desarrollo integrado. Esta aplicación de software tiene como fin brindar facilidades de comprensión para programadores al desarrollar software. Los IDE son desarrollados para maximizar la productividad de programación. Esta pieza de software se compone de:

- **Project manager**: es el administrador de proyecto, provee integración y comunicación entre el IDE y las herramientas de lenguaje.
- **Editor**: es un editor de texto con todas las características para programadores que también sirve como ventana para el debugger.
- **Assembler/Linker y herramientas de lenguaje**: el assembler puede ser usando independientemente para trabajar con un solo archivo, o puede ser usado junto con el linker para un proyecto desde archivos fuente separados, librerías y objetos recompilados. El linker es el responsable para posicionar el código compilado en las áreas de la memoria del microcontrolador.
- **Debugger**: El debugger permite la creación de breakpoints, single stepping, ventanas watch para ver registros y todas las características necesarias para la depuración de código.
- **Motores de ejecución**: Hay simuladores de software en el MPLAB IDE para todos los microcontroladores PIC. Estos simuladores usan la PC para simular las instrucciones y algunas funciones de periféricos.

En la Figura 2.27 se pueden apreciar las ventadas de editor, ventana más grande, la ventana de Output, donde se encuentran las pestanas de Build, Starter Kit Debugger, Find in Files, etc. Y la ventana Project manager, la ventana de la izquierda donde se pueden agregar o eliminar archivos del proyecto.

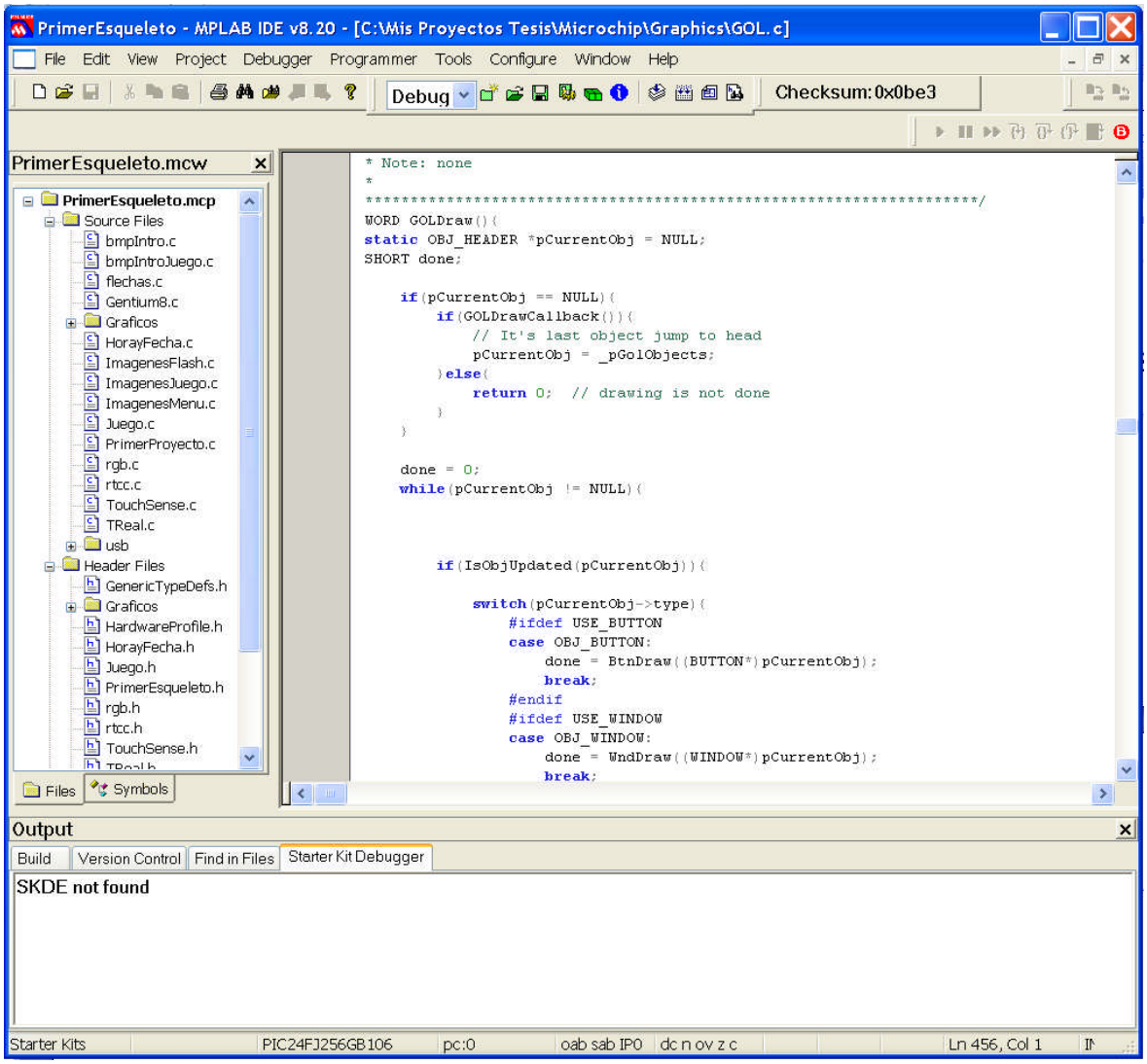


Figura. 2.27. Ventana de MPLAB IDE v8.20

En resumen, el MPLAB IDE es una "envoltura" que coordina todas las herramientas en una ventana de interfaz para el usuario, usualmente automáticamente. Por ejemplo, una vez que el código es escrito puede ser convertido en instrucciones ejecutables y descargado al microcontrolador para ver como funciona. En este proceso muchas herramientas son requeridas: un editor para escribir el código, un project manager para organizar archivos y configuraciones, un compilador o assembler para convertir el código fuente en lenguaje de maquina y algún tipo de hardware o software que simule la operación del microcontrolador [*20*].

2.3.1. EL CICLO DE DESARROLLO

El proceso para escribir una aplicación generalmente es como el que se describe en la Figura 2.28 [*20*]. Raramente todos los pasos, desde el diseño hasta la implementación, se pueden hacer sin un solo error y a la primera vez; frecuentemente el código es escrito, luego probado y finalmente modificado en orden de producir una aplicación que funcione correctamente. El IDE permite esto perfectamente, ya que el diseñador puede avanzar en los pasos de este ciclo sin distraerse, intercambiando entre las ventanas de las diferentes herramientas [*20*]. Todas las funciones del ciclo están integradas en una sola pantalla y también se pueden acomodar todas las herramientas de la preferencia en la pantalla.

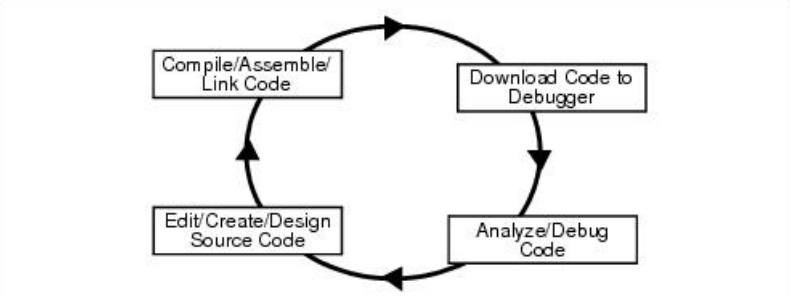


Figura. 2.28. Ciclo de desarrollo

2.3.2. EL PROJECT MANAGER

Uno de los componentes que más atención merece, es el project manager. Esta poderosa herramienta organiza los archivos a ser editados y otros archivos asociados para que puedan ser enviados a las herramientas de lenguaje para compilación o ensamblaje (assembly) y finalmente sean enviados al “linker”. El “linker” tiene la tarea de poner en su lugar los fragmentos de código del assembler, compilador y librerías en las áreas de memoria que les corresponden en el controlador embebido y asegurar que los módulos funcionen entre sí (que estén enlazados). Toda esta operación, que va desde el assembly y la compilación a través del proceso de enlace (linking) es llamado “buid”. Desde el project manager de MPLAB IDE, las propiedades de las herramientas de lenguaje pueden ser invocadas en manera diferente para cada archivo si se desea. El proceso de build integra a todas las herramientas de operaciones [*21*].

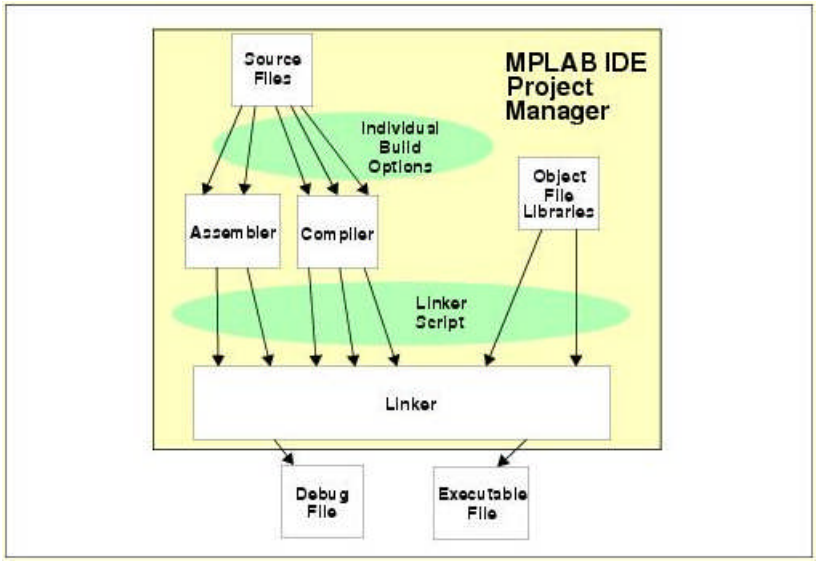


Figura. 2.29. El project manager de MPLAB IDE

La Figura 2.29 [*21*] muestra como trabaja el MPLAB IDE utilizando el project manager para enlazar los archivos de código y los objetos de la librería a través del linker para crear los archivos ejecutables y de Debug, tal como se describe en el párrafo anterior.

2.3.3. HERRAMIENTAS DE LENGUAJE

Las herramientas de lenguaje mejor conocidas como "lenguaje tools" son programas que se definen como cross-assemblers y cross-compilers [*22*].

Ahora, ¿Qué son los cross-assemblers y cross-compilers?. La mayoría de personas están familiarizadas con herramientas de programación que corren en una PC tales como Visual Basic y compiladores de lenguaje C, estas "lenguaje tools" están diseñadas y construidas para el set de instrucciones del PC, el cual es completamente diferente al de un microcontrolador. Por este motivo era necesario crear una herramienta de lenguaje o lenguaje tool que haga posible el trabajar con lenguajes familiares para el programador, pero que traduzcan el código al usado por los microcontroladores. A estos programas se les llama "cross-assembler" o "cross-compiler" y su idea está representada en la Figura 2.30 [*22*].

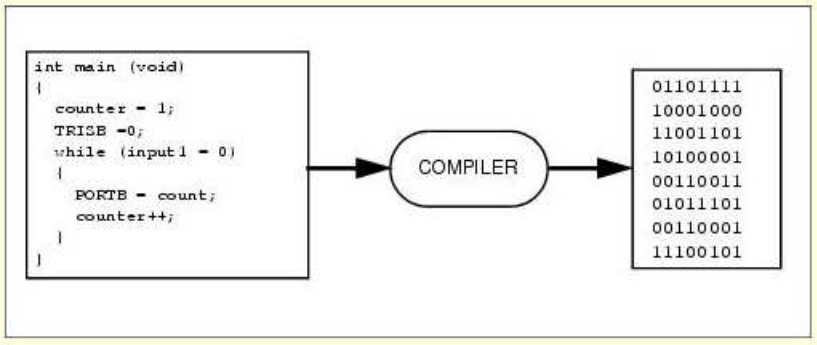


Figura. 2.30. El compilador convierte código fuente en lenguaje de máquina

Las herramientas en cuestión también producen un archivo debug, que el MPLAB usa para correlacionar las instrucciones de la máquina y las localidades de memoria con el código fuente [*22*]. Gracias a este archivo y la capacidad de compilar de manera cruzada, es que el software MPLAB tiene la capacidad de editar breakpoints, ver el contenido de las variables y además permite la ejecución paso a paso a través del código, para observar como se ejecuta la aplicación. Según el documento MPLAB Quick Start Guide, las herramientas de lenguaje también difieren de los compiladores que corren y ejecutan código para PC, porque estos deben estar muy conscientes del espacio. Mientras menor sea el código producido, ¡mejor será el código!; porque esto permite que se ocupe un microcontrolador de menor capacidad de memoria y esto reduce el costo. El tamaño de un programa para PC típicamente se extiende a algunos Mbytes cuando son moderadamente complejos. El tamaño de un programa simple para un sistema embebido, será de algunos Kbytes o menos. Un sistema embebido mediano necesitará 32K o 64K de código para una solución compleja. Algunos sistemas embebidos utilizan almacenamiento en orden de megabytes para tablas más amplias, mensajes de texto de usuario o data loggin [*22*].

CAPITULO 3

DESARROLLO DEL SOFTWARE EMBEBIDO

3.1. LINEAMIENTOS PARA DESARROLLO DEL SOFTWARE EMBEBIDO

Este tema que es indispensable y base para el tema que se pretende elaborar. Para tratar de manera más organizada y eficiente, se usará una guía práctica descrita por la página oficial de la comunidad de desarrolladores de firmware.

- **CONOCER LOS REQUERIMIENTOS**

Para empezar a hacer una arquitectura de un firmware embebido, se debe tener requerimientos claros, que definan el "que" va hacer el sistema. Debe notarse que cada requerimiento debe ser un "que va a hacer" y no un "como se va hacer". Cada requerimiento debe tener dos características: la no ambigüedad y la posibilidad de comprobar su funcionamiento. Si no es ambiguo, el requerimiento no necesita de más explicación, es conciso. Debe ser posible verificar que el requerimiento se cumpla mediante una o varias pruebas [*23*].

Los requerimientos de la aplicación de este proyecto, cumpliendo estas características, son:

1. Disponer de una interfaz gráfica amigable para el usuario
2. Elaborar cuatro aplicaciones para demostrar las siguientes características[3]:
    a. El poder procesamiento
    b. La capacidad de trabajar con la interfaz gráfica (la pantalla OLED)
    c. La capacidad USB OTG en modo Host

[3] **Se pueden incluir en una sola aplicación dos de estas características**

   d. El LED RGB
   e. El módulo CTMU
   f. El módulo RTCC

3. Debe tener un menú principal para escoger cualquiera de las aplicaciones
4. Cada aplicación debe tener su propio menú
   a. Cada aplicación debe poder volver al menú principal mediante un botón del teclado
5. La aplicación gráfica, debe hacer al usuario interactuar a través del teclado touch y del potenciómetro
6. La aplicación LED RGB debe mostrar la capacidad del LED de producir casi cualquier color
7. La aplicación de USB OTG modo host, debe tener la capacidad de trabajar con los archivos de una flash drive 2.0 común y corriente
8. La aplicación del módulo CTMU debe ser transparente para el usuario
9. La aplicación del módulo RTCC debe tener una opción para igualar la fecha y la hora [*23*]

Los requerimientos presentados cumplen con las características aconsejadas: Incluyen la frase "La aplicación debe…" que hace alusión a lo "que debe hacer" el producto. No revela el "como se debe hacer", no son ambiguos y sobre todo son fáciles de probar su buen funcionamiento. Michael Barr, autor del artículo *Firmware architecture in five easy steps* al cual se esta haciendo referencia, declara "…muy frecuentemente una base pobre de requerimientos es lo que inhibe la arquitectura de su desempeño…" [*23*].

- **SEPARAR LA ARQUITECTURA DEL DISEÑO**

La arquitectura no entra en detalles específicos de la solución, al contrario; es la estructura sobre la cual se construirá la solución.

La arquitectura debe ser desarrollada cuidadosamente y a través de un proceso que procure garantizar los requerimientos de la aplicación. En una analogía, un arquitecto

primero describe la estructura de un edificio, hace un modelo a escala, dibuja las dimensiones exteriores y el número de pisos, pero el número de cuartos por piso y el uso específico de estos corresponde al diseño y no a la arquitectura. [*23*]

La mejor forma de diseñar una arquitectura, es con un diagrama de bloques. La arquitectura del sistema identificará el flujo de datos y las conexiones entre subsistemas. Recomendaciones para una arquitectura según el artículo que se ha citado repetidamente son: no agregar detalles confusos y escribir pocos nombres de componentes.

El diseño de un sistema es la capa media del "cómo se debe hacer"; incluye las funciones o los nombres de las variables. En otras palabras, el diseño de firmware identifica las responsabilidades de cada tarea dentro del subsistema específico o drivers y los detalles de las interfaces entre subsistemas. El documento de diseño también identifica tareas, funciones, métodos, parámetros y nombres de variables que deben concordar con todos los niveles implementados. Refiriéndose al ejemplo anterior, el diseño es el momento en el que se ubican y dan nombres a los cuartos poniéndoles tareas especificas. [*23*]

- **MANEJO DEL TIEMPO**

Muchas aplicaciones mencionarán valores de tiempo específicos para una u otra acción dentro de sus requerimientos. En idioma inglés, se las conoce como "timelines" o "deadlines". La arquitectura debe ser diseñada para que este requerimiento se cumpla fácilmente dentro del tiempo establecido, garantizando los recursos del procesador necesarios para este fin.

A las aplicaciones se las clasifica, respecto al tiempo real, en: non-real-time, soft-real-time, y hard-real-time. Las soft deadlines son usualmente las más difíciles de definir, probar e implementar. Por ejemplo, en una aplicación de audio/video sería imposible perder un segundo en la unión del audio con el video. La manera más fácil de tratar a una soft deadline, es como una hard deadline que siempre debe ser cumplida.

En la siguiente sección se hablará de cómo hacer una partición en el diseño en hardware y en software; por ahora es necesario adelantar una corta idea sobre este tema. Cuando se identifiquen las deadlines, se las debe "empujar" del software hacia el hardware. Solo cuando esto no es posible, se debe usar un ISR (Interrupt Service Routine), y únicamente cuando un ISR no sea útil una tarea high-priority debe ser usada. [*23*]

Mantener alejada la funcionalidad de real-time del bulk[4] de software es valioso por dos razones importantes. La primera por que simplifica el diseño y la implementación del llamado non-real-time software. La segunda es porque en una arquitectura que separa los timelines del bulk de software, el código puede ser agregado sin que el firmware pierda robustez [*23*].

- **DISEÑAR PARA PRUEBAS**

Las pruebas son columnas sólidas para un buen sistema embebido. Deben realizarse a diferentes niveles, estos son:

- **Pruebas del sistema:** verifican el producto como un todo, cumpla o exceda todos los requerimientos.
- **Pruebas de integración:** verifican que los subsistemas identificados en los diagramas de arquitectura interactúen como se esperaba y que produzcan salidas razonables.
- **Pruebas por unidades:** verifican que cada componente individual de software identificado en cada nivel funcione como se espera [*23*].

- **PLANEAR PARA FUTUROS CAMBIOS**

Planear para hacer cambios en un futuro, comprende el pensar que cambios específicos podrían hacerse y que sean fáciles de implementar. Generalmente las características o componentes que se desean agregar a una aplicación, se agrupan en

[4] La palabra Bulk se refiere a una gran cantidad de algo que esta sin empacar. En este caso se refiere a una cantidad de código.

"paquetes". Además, cuando se tiene un modelo, el más básico o estándar de un producto o aplicación se le llama el paquete "foundation". Por ejemplo, un modelo de un producto puede ser el paquete foundation + paquete A, donde paquete A incluye una característica de modo de ahorro de energía.

Usando este criterio, se pueden crear varias aplicaciones a partir de paquetes libres de bugs[5] de manera rápida. "*El mayor reto en planificar para hacer cambios está en encontrar el balance acertado entre paquetes muy pequeños o muy grandes. Como muchos otros detalles de la arquitectura de firmware, alcanzar este balance es más un arte que una ciencia*" [*23*].

3.1.1. PARTICIÓN DEL HARDWARE/SOFTWARE

Para empezar con este tema, es necesario entender primero cual es el papel del hardware y cual es el del software dentro de un sistema embebido. Para esto, se tomarán dos referencias bibliográficas:

1. El sitio web www.embbeded.com en uno de sus artículos de "technical insights" afirma "*Si se mira de cerca del papel que ha desempeñado el software en la historia de los sistemas embebidos, uno se puede dar cuenta que ha contribuido poco a la funcionalidad de estos*" [*24*].
2. La segunda es el libro Real Time Embedded System Desing de donde se toma la tabla 3.1 [*25*] que nos muestra:

TABLE 1 Traditional hardware/software values

Hardware values	Software values
Integration	Utility and usability
Differentiation	Control
Economics	Communication
Practicality	Flexibility

Tabla. 3.1. Valores que aportan el HW/SW a un sistema. www.embbeded.com

[5] **Bug es un término usado en ingeniería de software para denominar a fallas que aparentemente no tienen una causa lógica.**

El texto además añade: "*Luego de 30 años de evolución tecnológica en semiconductores, el hardware genera los beneficios de integración y define también el costo del producto y determina cuan práctico puede ser."*[*25*]

Se conoce que un sistema embebido depende tanto del software como del hardware. Como definir que parte de las especificaciones y requisitos del sistema serán resueltos por uno y por el otro se llama: decisión de partición.

Considérese el concepto de algoritmo como "un conjunto de pasos que implementan un diseño". Traduciendo este concepto a diseño de sistemas embebidos, un algoritmo es: "*una combinación de componentes de hardware y software que trabajan juntos para dar una solución en un sistema integrado*" [*25*]. El algoritmo de un sistema, puede tener tres variaciones de acuerdo a su partición hardware/software, estas son: un algoritmo puramente de software, puramente hardware o una combinación de ambos.

Por ejemplo, cuando la PC estaba dando sus primeros pasos con los procesadores 8086, 80286 y 80386, estos no tenían una unidad de procesamiento de punto flotante (FPU). Los procesadores requerían de un FPU externo para ejecutar directamente cálculos de este tipo que estén en el código. Sí no se podía contar con una FPU externa, era necesario hacer un algoritmo de software, usando una rutina para emular el comportamiento de un FPU externo. Tras programar este algoritmo, el código se volvía mucho más pesado y por ende más lento [*25*].

Por otro lado, en los días en que el hyper terminal y el internet se popularizaban, algunas PC requerían tarjetas de módems que se conectaban al desaparecido puerto ISA, y contenían la circuitería necesaria para modulación/demodulación. También, existía el Winmodem que se conectaba en un slot PCI y usaba el CPU de la PC para directamente manejar las funciones de modem. Finalmente, los fanáticos de juegos de PC saben cuan importante es una tarjeta de video high-performance para la velocidad del juego [*25*]. Es importante para la decisión de partición, el procesador que se use y como se lo implemente dentro del diseño general. Se puede escoger entre varios cientos de microprocesadores,

microcontroladores y ASIC[6] y dependiendo del procesador se escoge instrumentos de programación, compilación, IDE, etc.

- **LA DECISION DE PARTICIÓN**

Diseñar el hardware para un sistema embebido es más que seleccionar el procesador correcto, conectarlo a unos cuantos periféricos y soldar todo en una placa. Decidir como particionar el diseño en una funcionalidad de SW y de HW[7], es la llave para la creación de un sistema embebido.

"La decisión de partición tiene un impacto significativo en el costo final del proyecto, tiempo de desarrollo y riesgo" [*25*].

La complejidad del problema de partición es desconcertante. Hoy en día, muchos sistemas son tan complejos que requieren de herramientas asistidas por computadora. Sin embargo, Charles H. Small se refiere al proceso de partición de SW/HW como: "*En la práctica, el análisis de los intercambios entre SW y HW para una partición es más bien un procesos informal hecho con lápiz y muchas hojas de papel"* [*26*].

Idealmente, la decisión de partición debe hacerse después de que haya un entendimiento de todas las alternativas que estén al alcance para resolver el problema. Mientras más se pueda tardar la toma de la decisión, más información va a tenerse sobre que partes del algoritmo necesitan ser en HW y cuales pueden ser desarrolladas en SW.

Añadir HW significa más velocidad, pero más costos y más puntos de posibles fallas.[8] También significa que el proceso de diseño se vuelve más riesgoso porque rediseñar un elemento de HW es considerado más serio que encontrar un error en SW y reconfigurar el código.

[6] ASIC, o application-specific IC, son circuitos integrados diseñados específicamente para una aplicación.
[7] HW es hardware, SW es software
[8] Esto tampoco es un concepto de "blanco o negro" porque más funciones en software requieren más espacios en memoria ROM, velocidad de chip, incluso puede llevar a volver a escoger un procesador más costoso y retrasar el trabajo.

El problema fundamental, es que no se puede depurar el sistema hasta que el hardware este disponible y listo para trabajar con él. Además, si se retarda la decisión de partición demasiado, el equipo de desarrollo de software se deja en espera para que el equipo de hardware termine la tarjeta.

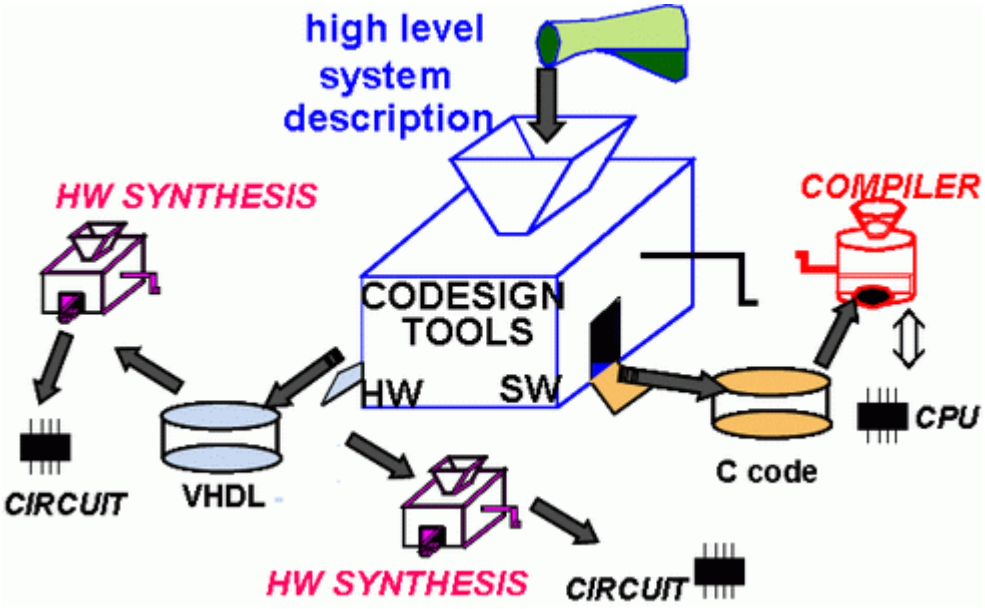


Figura. 3.1 Co-diseño del SW/HW

La Figura 3.1 [*25*], explica las herramientas y procesos que se utilizan en el codiseño de hardware y software, luego de hacer una descripción de alto nivel del sistema. Entre estas herramientas están: VHDL para el hardware y código C y el compilador para el software. El resultado de ambos procesos es un CPU con firmware listo para ser usado y un circuito que ejecuta todas las tareas encargadas al hardware.

- **FUSIONANDO EL HARDWARE CON EL SOFTWARE**

En la actualidad, la tecnología de diseño para HW y SW se está fusionando (Figura 3.1). Vale la pena preguntarse si las técnicas tradicionales para diseño son todavía válidas [*26*]. Para desarrollar un diseño rápidamente, es necesario no perder el tiempo corrigiendo errores, así que es mejor hacer un plan que contemple cada posible fallo desde un principio.

"Hardware/software co-verification" es un proceso en el que se ata mas estrechamente la integración de hardware y software. En este proceso, el hardware se representa por código VHDL o Verilog[9] y se vuelve una plataforma virtual para hardware en software.

En la mayoría de ocasiones, es necesario contar con el hardware para tener una idea de cómo va a responder al software, pero en ausencia de hardware real, se puede escribir funciones en código que representen el comportamiento virtual del hardware. Los equipos de software generalmente pasan un tiempo mínimo en el desarrollo de este código que luego no puede ser rehusado. La prueba real y exhaustiva de la interfaz software-hardware no empieza hasta que el hardware real este disponible, lo cual es una perdida de tiempo. "Mientras más avanzado este el proyecto, si una falla es encontrada (sin distinguir entre software o hardware) más costoso es arreglarla" [*26*].

3.1.2. ARQUITECTURA Y DISEÑO DEL FIRMWARE PARA LA APLICACIÓN

El ser humano toma la información que llega a su cerebro a través de sus sentidos, principalmente por la vista. Esta "entrada" de información le permite desenvolverse en un ambiente, saber por donde dirigirse y encontrar cualquier cosa que desee. Una PC ofrece casi toda su información aprovechando este sentido y apenas la refuerza con sonidos. Principalmente es la retroalimentación gráfica la que le permite a la persona utilizar una PC incluso de manera intuitiva. Explotando este hecho, la PC se ha vuelto cada vez más y más amigable. Recuérdese el primer sistema operativo MS-DOS, todo en blanco y negro, con comandos de ejecución que debían ser ingresados manualmente, casi nadie (que no fuera un profesional de sistemas) sabía utilizar una PC. Cuando el sistema operativo, Windows 3.1, se lanzó en forma masiva, revolucionó al mundo con un sistema que presentaba gráficamente una oficina virtual (escritorio, PC, tacho de basura, carpetas de documentos, etc.), solo ahí, muchas mas personas fueron capaces de comprender como trabajar en un ordenador y se interesaron en su uso.

[9] VHDL code, es lenguaje para programar elementos de HW en SW, viene de lenguaje de descripción de hardware VHSIC (Very High Speed Integrated Circuit)

Verilog es otro lenguaje de descripción de hardware (HDL) usado para modelar sistemas electrónicos, soporta prueba e implementación de circuito analógicos, digitales y de señal mista a diferentes niveles de abstracción.

En esta época, en la que los sistemas embebidos tratan de cada vez más emular a la PC, la retroalimentación gráfica es una de "las últimas fronteras". Debido a esto, la arquitectura de construcción de los sistemas embebidos debe redefinirse con una estructura que permita trabajar conjuntamente la interfaz gráfica con la aplicación. Este reto, esta siendo superado gracias al uso de herramientas de lenguaje conocidas como API. Antes de seguir adelante, las API serán descritas brevemente.

- **APPLICATION PROGRAMMING INTERFACE**

Una Application Programming Interface, es una interfase implementada por programas de software que permite que estos interactúen entre sí. Esto es igual a cuando, por software se implementa una interface de usuario para que un humano use una aplicación, solo que las API son para que el software interactué con software.

Las API se implementan por aplicaciones, librerías y sistemas operativos para definir como otro software puede hacer requests o calls a servicios para su uso. Una API determina el vocabulario y convención de las llamadas que el programador debería emplear para usar estos servicios. Pueden incluir especificaciones para: rutinas, estructuras de datos, clases de objetos y protocolos; todos usados para comunicación entre el consumidor y aquel que implemente la API [*27*].

Una definición más técnica, se tomará de la revista virtual PCmagazine: "*API es un lenguaje y formato de mensajes usados por una aplicación para comunicarse con el sistema operativo u otro programa de control, tal como sistemas para manejo de bases de datos (DBMS) o protocolos de comunicaciones" [27].* Las APIs son implementadas escribiendo funciones de llamada en el programa, las cuales proveen el enlace hacia la subrutina o función requerida para la ejecución. Así, una API implica que algún módulo del programa esté disponible en la aplicación para desarrollar la operación o que debe estar ligado al programa existente para ejecutar su tarea" [*28*].

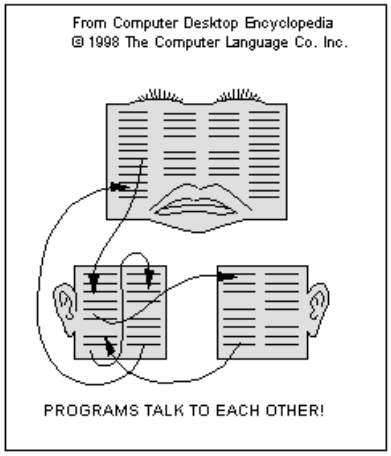


Figura. 3.2. ¡Los programas hablan entre sí!

La Figura 3.2 [*28*] representa es una idea abstracta de cómo se comunica el código programado entre sí, para pasar entre ellos datos, instrucciones, etc. Puede decirse que los programas se comunican a través de mensajes y el canal por donde viaja el mensaje son punteros, funciones, etc.

- **LA LIBRERÍA DE GRÁFICOS DE MICROCHIP**

La Librería de gráficos "Graphics Library v1.65" de la compañía Microchip®, es la herramienta base que se utilizó para el diseño del firmware. *Es la base de la arquitectura para la aplicación.* Como se ha mencionado en este capítulo, el hecho que la tarjeta posea una interfaz gráfica, hace que la librería sea primordial en el diseño.

Al empezar este proyecto, se contaba con un programa demo sobre todas las características de la tarjeta[10], pero no existe un tutorial específico para desarrollar un proyecto nuevo en el Starter Kit PIC24F como tal. Existen otros equipos y tarjetas para iniciarse en el aprendizaje de las interfaces gráficas para PICs los cuales son más promocionados; entre estos están el Explorer 16 y la Graphics PICtail™ Version 2. También existen documentos como la Application Note 1136 (AN1136), y el software de la Graphics Library v1.65 que incluye todas las Application Notes sobre gráficos, la utilidad Bitmap and Font converter, diagramas esquemáticos y el Help de la librería.

[10] **Disponible en www.microchip.com/graphics**

Luego de una consulta extensa a los participantes de los foros[11] (Figura 3.3) de la página principal de Microchip sobre gráficos para dispositivos PIC24F, las respuestas indicaban que el mejor inicio sería lograr imprimir algo en la pantalla. A primera impresión, el código del programa demo es indescifrable pero con paciencia y análisis se puede encontrar las pautas ya que está bien comentado y cada función es explicada con claridad.

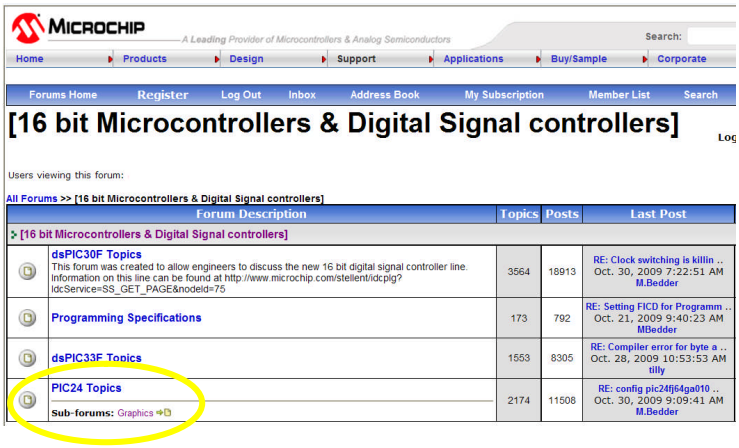


Figura. 3.3. Página de los foros de Microchip sobre PIC24F, sub foro de gráficos [*11*]

Como se mencionó antes, el primer paso es poder presentar algo en pantalla. Para esto es necesario entender la estructura básica del sistema de la librería y justamente ese es el tema que se tratará a continuación.

La introducción del documento AN1136, de Microchip, da una breve orientación sobre el uso y propósito de la librería, dice: "La librería de gráficos Microchip fue creada para cubrir un amplio rango de dispositivos controladores de displays. Diseñada para usarse con microcontroladores PIC, ofrece una Interfaz de Programación para Aplicación (API) que se encarga de dibujar objetos "primitivos" (línea, punto, círculo, etc.) y también de objetos avanzados del tipo widget[12]. La librería también facilita la integración de dispositivos de entrada a través de una interfaz de mensajes" [*29*]. Las aplicaciones creadas usando la librería también encontrarán un proceso simple para cambiar entre dispositivos de HW sí la necesidad se presenta, es decir puede trasportarse de un hardware a otro. La

[11] **www.microchip.com/forums**

[12] ***Widget*, también conocido como artilugio o control, es un componente gráfico con el cual el usuario interactúa, como por ejemplo, un check box, un slider, caja de texto, botón. TEXTO TOMADO DE AN1136**

arquitectura diseñada por capas hace que esto sea posible. La estructura de la librería se muestra a continuación.

- **ESTRUCTURA DE LA API DE GRÁFICOS**

La aplicación de este proyecto en su totalidad esta diseñada para interactuar con el usuario, no es una aplicación stand-alone[13]. Por este motivo, está diseñada siempre pensando en generar una retroalimentación gráfica para cada estado, dar menús para las opciones, usar el teclado para selección de opciones, etc. La herramienta primordial es la API que permite la creación de un entorno gráfico, entrada del usuario y programar procesos de acuerdo a cada comando. Por estas razones, la estructura de la aplicación es la misma de la librería [*2*].

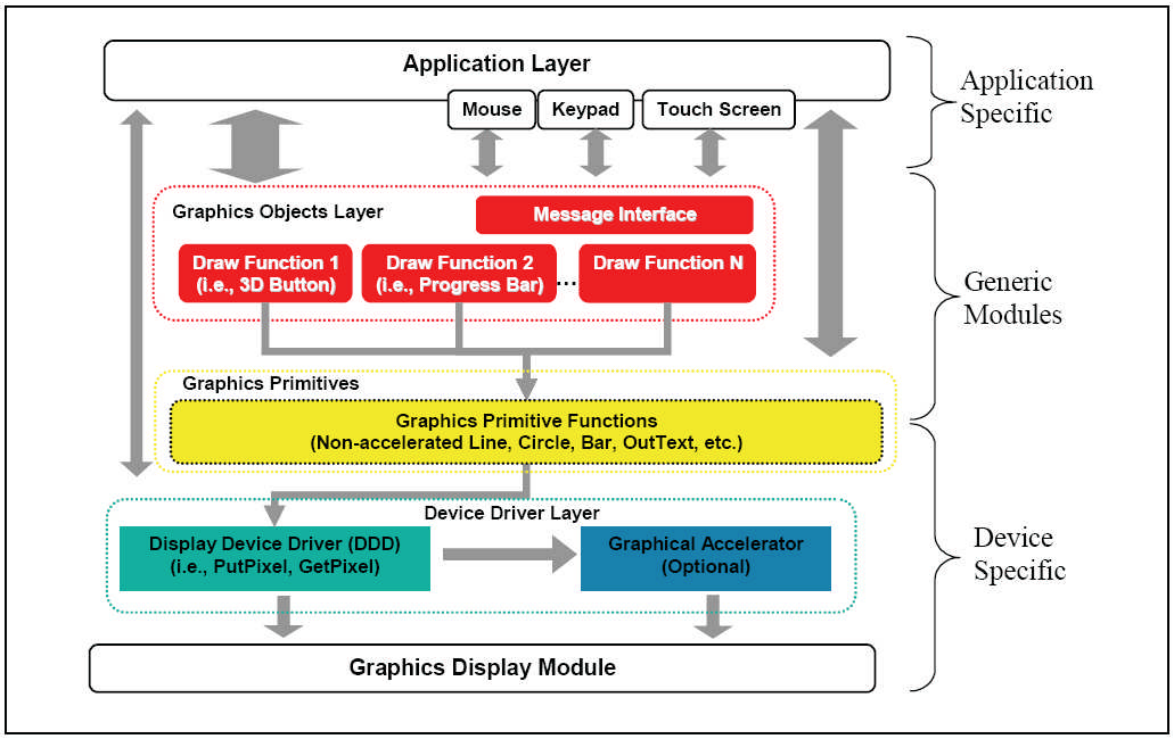


Figura. 3.4. Sistema típico usando Microchip Graphics Library

La Figura 3.4 [*2*] muestra que la estructura es una arquitectura diseñada por capas, cada capa tiene funciones que desempeñan las actividades que les corresponden. Para que las funciones se comuniquen entre sí, existe un sistema de mensajes entre ellas. En la capa

[13] **Aplicación stand-alone es aquella que no necesita re acción del usuario**

más baja (la de comunicación con el driver), las funciones son cajas negras, no necesarias de ser comprendidas por el desarrollador.

La capa de aplicación es el programa hecho por el usuario. La capa GOL (Graphics Object Layer) es la encargada de dibujar los widgets tales como: botones, sliders, check box, etc. A los widgets, dentro de la librería se les refiere como objetos u objetos gráficos. Como se menciono antes, existe un sistema de mensajes entre funciones, la capa GOL recibe mensajes de la capa de aplicación [*2*]. La siguiente capa es la Graphics Primitive Layer, o capa de gráficos primitiva, esta es la encargada de dibujar la parte más primitiva o básica de los objetos gráficos, como líneas, barras, círculos, rectángulos, etc. La siguiente capa, la DDD o Display Device Driver, es la capa que depende de la pantalla que se esta usando. Esta capa es la comunicación entre el firmware y el driver de la pantalla que se use. En caso del Starter Kit, corresponde a una pantalla OLED de 128x64 pixeles y el driver usado es el SH1101A que está conectado al puerto paralelo. Debe mencionarse también, que esta librería contiene una gran cantidad de drivers pre-programados y una gran facilidad para migrar entre ellos; si las necesidades de hardware lo demandan. La API permite a la aplicación diseñada tener acceso a cualquier capa de la librería. Los procesos de dibujo y de mensajes entre capas son manejados internamente y pueden ser transparentes a la aplicación [*2*].

Como siguiente paso, se analizará las funciones que se usan de cada capa de la librería. Lo que se refiere a archivos .c y .h en los que se encuentran estas funciones será descrito en los tutoriales en el capítulo 4. Se empezará por describir las funciones por cada capa.

3.1.2.1. Capa de aplicación. La capa de aplicación es el programa hecho por el diseñador que interactúa con la capa GOL. Utiliza de esta, los objetos que requiere haciendo un llamado a sus funciones de creación de los objetos.

Dentro de la capa de aplicación, estan todas las pantallas diseñadas para el firmware (menú de inicio, menú de juegos, resultados del juego, etc.) estas, serán explicadas más

adelante y en detalle, por ahora, se tomará como ejemplo la pantalla del menú de inicio para demostrar el proceso de su creación.

Por ejemplo, la pantalla de menú inicio, deberá contar con cuatro botones para acceder a las cuatro diferentes aplicaciones programadas, un título y una línea de intstrucciones. Un primer esquema sería el representado por la Figura 3.5:

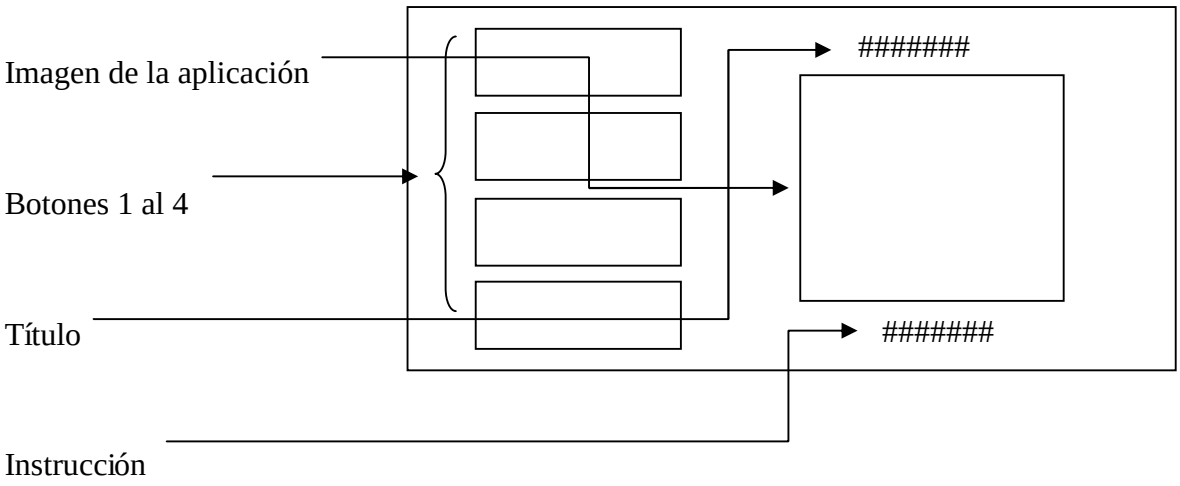


Figura. 3.5. Ejemplo del diseño de la capa de aplicación: menu inicio

Recuérdese que por el momento solo se está explicando la CAPA DE APLICACIÓN, por lo que no se detallará en que parte del programa se encuentran definidas estas funciones. Ahora, para crear todos estos objetos se llamará desde la CAPA DE APLICACIÓN a las funciones de la capa GOL enviándoles el mensaje que contenga la informacion que éstas necesiten. En este caso la informacion del mensaje será: parámetros de tamaño, coordenadas y otros dependiendo del objeto.

- **FUNCIONES DE LA CAPA DE APLICACIÓN USADAS EN EL FIRMWARE**

Las tablas 3.2, 3.3, 3.4, 3.5 y 3.6 resumen las funciones que se usan para programar la interfaz gráfica de la aplicación. Estas funciones son programadas y declaradas en la capa GOL, en el archivo GOL.c y GOL.h.

Función	Parámetros	Observaciones
`BUTTON * BtnCreate( WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT radius, WORD state, void * pBitmap, XCHAR * pText, GOL_SCHEME * pScheme );`	ID: Identificación única definida por el usuario para la instancia del objeto	No retorna nada. No tiene efectos secundarios
	Left: Posición del extremo izquierdo del objeto[14]	
	Top: Posición tope máxima del objeto	
	Right: Posición del extremo derecho del objeto	
	Bottom: Posición fondo maxima del objeto	
	Radio: Radio de las esquinas del objeto	
	Estado: Configura el estado inicial del objeto	
	pBitmap: Puntero al bitmap usado en la carilla del objeto, la dimensión del bitmap no debe ser mayor a la dimensión del botón	
	pText: Puntero al texto del botón	
	pScheme: Puntero al estilo usado	
Overview: Esta función crea un objeto (BUTTON) y lo pone al fin de la lista de objetos[15].		

Tabla. 3.2. Función BtnCreate

Función	Parámetros	Observaciones
`PICTURE * PictCreate( WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, char scale, void * pBitmap, GOL_SCHEME * pScheme );`	ID: Identificación única definida por el usuario para la instancia del objeto	No retorna nada. No tiene efectos secundarios
	Left: Posición del extremo izquierdo del objeto	
	Top: Posición tope máxima del objeto	
	Right: Posición del extremo derecho del objeto	
	Bottom: Posición fondo maxima del objeto	
	Estado: Configura el estado inicial del objeto	
	Escala: Configura el valor de escala para dibujar el bitmap	
	pBitmap: Puntero al bitmap usado en la carilla del objeto, la dimensión del bitmap no debe ser mayor a la dimensión del botón	
	pScheme: Puntero al estilo usado	
Overview: Esta función inserta una imagen bitmap en la pantalla. Los pixeles no deben superar el tamaño de la pantalla y para el caso del Starter Kit PIC24F deben ser monocromáticos. Crea un objeto (PICTURE) al final de la lista de objetos.		

Tabla. 3.3. Función PictCreate

[14] El valor de las variables left, top, right y bottom deben ser dados en pixels.
[15] La lista de objetos, es una lista que contiene todos los objetos que deben ser graficados.

Función	Parámetros	Observaciones
`STATICTEXT * StCreate( WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR * pText, GOL_SCHEME * pScheme );`	ID: Identificación única definida por el usuario para la instancia del objeto	
	Left: Posición del extremo izquierdo del objeto (En pixeles)	No retorna nada.
	Top: Posición tope máxima del objeto	
	Right: Posición del extremo derecho del objeto	No tiene efectos secundarios
	Bottom: Posición fondo maxima del objeto	
	Estado: Configura el estado inicial del objeto	
	pText: Puntero al texto del botón	
	pScheme: Puntero al estilo usado	
Overview: Crea un objeto (STATICTEXT) y lo coloca al final de la lista de objetos. Los margenes deben ser seleccionados para que entre el texto en alto y largo.		

Tabla. 3.4. Función StCreate

Función	Parámetros	Observaciones
`void GOLFree();`	Ninguno	Todos los objetos en la lista de objetos activa serán borrados.
Overview: Esta función libera toda la memoria usada por objetos en la lista activa e inicializa el puntero _pGolObjects a NULL para empezar una lista nueva y vacia. Esta función debe ser llamada solo dentro de la función GODrawCallBack(), que a su vez es llamada por las funciones GOLDraw() y GOLMsg()[16]. Este requerimiento asegura que la configuración mas primitiva de dibujo no se altera por la máquina de estados de los objetos.		

Tabla. 3.5. Función GolFree

Macro	Parámetros	Observaciones
`#define SetColor(color) _color = color`	Color	Ninguna
Overview: Esta macro define el color para dibujar. Para el caso de la pantalla monocromática si se escoje el color WHITE para dibujar, el fondo de la pantalla automáticamente sera negro y viceversa.		

Tabla. 3.6. Macro SetColor

[16] **Estas funciones están explicadas con claridad en la sección "Flujo de la librería de gráficos"**

A continuación, se presentan las funciones descritas implementadas en el código. Este segmento de código es tomado de la función MostrarMenuPrincipal() declarada en el archivo principal PrimerProyecto.c[17]:

```
/*************************************************
Mostrar la Pantalla de Menu Principal
*************************************************/
void MostrarMenuPrincipal( void)
{
GOLFree();
SetColor(BLACK );
ClearDevice();

       PictCreate(     ID_ENCABEZADO_BMP,
                       112,0,127,14,
                       PICT_DRAW,
                       1,
                       &iconoEspe2,
                       NULL );

       StCreate(       ID_ENCABEZADO_TEXTO,
                       70, 0, 110, GetTextHeight( (void *)&FONTDEFAULT )-1,
                       ST_DRAW | ST_RIGHT_ALIGN,
                       "MENU",
                       NULL );

// Menu

       PictCreate(     ID_BMP_MENU,
                       70,20,102,52,
                       PICT_DRAW,
                       1,
                       &arrow4DOWN1_4bpp_32x32,
                       NULL );

       StCreate(       ID_STR_MENU,
                       46, 10, 100, 19,
                       ST_DRAW|ST_RIGHT_ALIGN,
                       "Presione",
                       pScheme);

       StCreate(       ID_STR_MENU_DOS,
                       60, 52, 127, 63,
                       ST_DRAW|ST_RIGHT_ALIGN,
                       "para empezar",
                       pScheme);

// Botones


       BtnCreate(      ID_BTN_UNO,

                       MargenEscIzq,MargenEscSup,MargenEscDer,MargenEscInf,2,
                       BTN_HIDE|BTN_TEXTRIGHT,
                       NULL,
                       "JUEGO",
                       NULL );

       BtnCreate(      ID_BTN_DOS,
                       MargenEscIzq,  MargenEscSup  +  15,MargenEscDer,MargenEscInf  +  14,2,
                       BTN_HIDE|BTN_TEXTRIGHT,
                       NULL,
                       "USB",
                       NULL );

       BtnCreate(      ID_BTN_TRES,
                       MargenEscIzq,MargenEscSup   +   30,MargenEscDer,MargenEscInf   +   29,2,
                       BTN_HIDE|BTN_TEXTRIGHT,
```

[17] **Se detallará los archivos .c y .h y su contenido en el capítulo 4**

```
                    NULL,
                    "RGB",
                    NULL );

}
```

Antes de continuar al estudio de la siguiente capa, algunos conceptos deben ser definidos.

- **LISTA DE OBJETOS ACTIVA**

Para facilitar el trabajo, la librería de gráficos enlaza a todos los objetos creados por el usuario en una lista. Esta lista puede ser manipulada en cualquier momento por mensajería o por el proceso para dibujar. Los objetos que se crean se añaden automáticamente al final de la lista. Solo un objeto se puede añadir a la vez. Se pueden crear varias listas de objetos. Para varias listas, la aplicación es responsable del manejo de cambiar entre listas. Este esquema permite tratar a cada lista como una página de display. Solo la lista activa será desplegada en pantalla [*2*].

- **PROCESO DE DIBUJO**

Para crear los objetos, la aplicación debe llamar al administrador del proceso de dibujo, la función GOLDraw(). Esta función, analiza la lista activa y redibuja los objetos si sus estados lo demandan. Cuando los objetos s terminan de graficar, el estado de los objetos se borra automáticamente. El primer objeto creado será dibujado (estilo FIFO). Luego de que todos los objetos de la lista hayan sido creados, la función GOLDraw() llama a la función GOLDrawCallback(). En esta función es posible dibujar lo que el usuario desee.

3.1.2.2. Capa gol (Graphics Object Layer). Esta capa cumple varias funciones, entre estas están:

- Programación de los objetos (Button.c, StcText.c, etc)
- Estado de los objetos
- Manejo de los objetos
- Estilo de los esquemas
- Mensajería GOL

- **FUNCIONES CALLBACK**

¿Qué son y para qué sirven? Una función callback es una sección de código ejecutable que es pasada como argumento a otra sección de código. Permite que una capa de software de nivel inferior pueda llamar a una subrutina (o función) definida en una capa superior. La Figura 3.6 [*30*] presenta una explicación simple pero muy acertada de su funcionamiento:

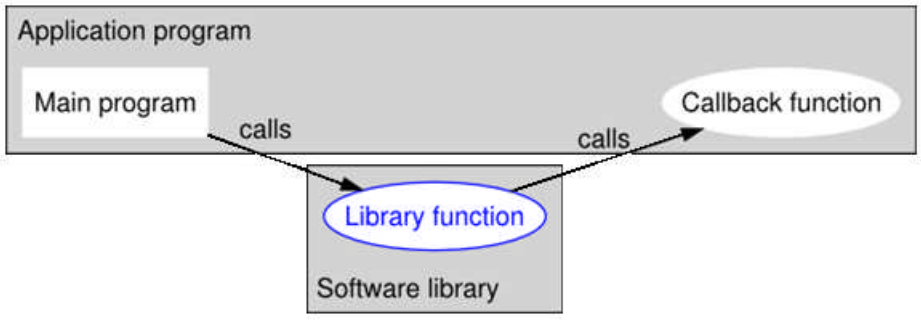


Figura. 3.6. Llamada a una función Callback

En la mayoría de casos, las funciones de la capa superior hacen un llamado a las funciones definidas en el código inferior pasándoles un puntero conocido como ***handle*** a la función. Mientras la función de bajo nivel se ejecuta, esta puede llamar a la función callback (cualquier número de veces) para realizar una tarea. Por el momento este es el único escenario que es necesario comprender acerca de las funciones callback [*30*].

De entre todas las funcionalidades de la capa GOL, dos son las más importantes y cada una tiene una función con su respectiva callback. Las tablas 3.7 hasta 3.10 presentan las

funciones que son tan necesarias de conocer antes de proseguir con el desarrollo de este proyecto.

- Manejo de objetos: `WORD GOLDraw ();`

  o `WORD GOLDrawCallback ();`

- Mensajería GOL: `void GOLMsg(GOL_MSG *pMsg);`

  o
```
WORD GOLMsgCallback(  WORD objMsg,
                      OBJ_HEADER* pObj,
                      GOL_MSG* pMsg,
                      );
```

Función	Parámetros	Observaciones
`WORD GOLDrawCallback();`	Ninguno	Ninguna
Overview: Esta función ***DEBE SER IMPLEMENTADA POR EL USUARIO OBLIGATORIAMENTE***. Esta función es llamada dentro de la función GOLDraw() cuando se termina de dibujar los objetos de la lista activa. Aquí, es donde el usuario debe insertar las funciones que grafiquen la pantalla (p.ej.: MostrarMenuPrincipal()). El color para dibujar, tipo de línea, posición del cursor, región de clipping y el tipo de letra no serán cambiadas por GOL sí esta función retorna un 0. Para pasar el control para dibujar a GOL, esta función debe retornar un valor que no sea cero. Sí la mensajería GOL no esta usando la lista actual, es seguro modificar la lista aquí.		
Return: 1 sí la función GOLDraw() tendrá control para dibujar la lista activa. 0 si el usuario desea tener el control para dibujar el mismo.		

Tabla. 3.7. Función GOLDrawCallback

Función	Parámetros	Observaciones
`WORD GOLDraw();`	Ninguno	Ninguna
Overview: Esta función circula a través de la lista de objetos activa y redibuja los objetos que necesitan ser redibujados. Un redibujado parcial o total es hecho dependiendo del estado de los objetos. La función GOLDrawCallback() es llamada por GOLDraw() cuando el proceso de dibujar los objetos en la lista activa se ha completado.		
Return: un 1 si los objetos de la lista activa se han terminado de dibujar.		

Tabla. 3.8. Función GOLDraw

Función	Parámetros	Observaciones
`WORD GOLMsgCallback( WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg );`	Ninguno	Ninguna
Overview: *EL USUARIO DEBE IMPLEMENTAR ESTA FUNCION.* GOLMsg() llama a esta función cuando un mensaje valido para un objeto en la lista activa es recibido. **La acción del usuario para el mensaje debe ser implementada aquí**. Sí esta función retorna un valor diferente de cero, el mensaje para el objeto sera procesado por default. Sí cero es retornado, GOL no hará ninguna acción.		
Return: 1 si la función GOLDraw() tendrá control para dibujar la lista activa. 0 si el usuario desea tener el control para dibujar el mismo.		

Tabla. 3.9. Función GOLMsgCallback

Función	Parámetros	Observaciones
`void GOLMsg(GOL_MSG *pMsg);`	GOL_MSG *pMsg: Puntero al mensaje GOL del usuario	Ninguna
Overview: Esta función recibe un mensaje GOL del usuario y da vueltas a través de la lista activa de objetos para identificar que objetos son afectados por el mensaje. Para los objetos que son afectados, el mensaje es traducido y la función GOLMsgCallback() es llamada. **En la función callback, el usuario tiene la habilidad para implementar una acción para dicho mensaje**. Si la función callback retorna un valor que no es cero, por default la función OBJMsgDefault() es llamada para procesar el mensaje para el objeto. Si un cero es retornado, la función OBJMsgDefault() no es llamada[18]. Esta función debe ser llamada cuando se termine el proceso de dibujo. Puede ser llamada cuando GOLDraw() retorne un valor diferente de cero o dentro de la función GOLDrawCallback.		

Tabla. 3.10. Función GOLMsg

[18] **Este tema, al parecer confuso será tratado con mayor detenimiento en las siguientes secciones donde se explica el flujo de mensajes entre las capas.**

3.1.2.3. Capa graphics primitive. La capa "primitiva" es una capa independiente que contiene funciones **básicas** para dibujar[19]. Dichas funciones básicas, pueden ser llamadas desde la capa de aplicación directamente y su proceso de dibujo no requiere de la función GOLDraw(), solo se ejecutan inmediatamente. Las funciones de esta capa son:

- Funciones de Set Up
- Funciones de texto
- Funciones de línea
- Funciones de rectángulo
- Funciones de círculo
- Cursor gráfico
- Funciones de Bitmap
- Memoria externa

Solo se detallarán las funciones usadas en la aplicación[20]:

Función	Parámetros	Observaciones
`void ClearDevice();`	Ninguno	Ninguna
Overview: Esta función limpia la pantalla con el color establecido y configura la posición del cursor gráfico a (0,0).		

Tabla. 3.11. Función ClearDevice

Función	Parámetros	Observaciones
`void InitGraph();`	Ninguno	Ninguna
Overview: Esta función inicializa el controlar de la pantalla, configura las propiedades de línea en SOLID_LINE, configura la pantalla entera en BLACK, configura el color para dibujar en WHITE, configura el cursor en la esquina superior izquierda (0,0 en pixeles). **Esta función debe ser llamada antes de usar la Graphics Primitive Layer.**		

Tabla. 3.12. Función InitGraph

[19] **Estas funciones pueden ser implementadas en la capa DDD en caso de que la pantalla soporte aceleración por hardware de la función**

[20] **Para referencias de todas las funciones descargar Microchip Graphics Library v1.65 Help**

Funciones de texto, entre estas funciones se encuentran:

- SetFont
- OutChar
- OutText
- OutTextXY
- GetTextHeight
- GetTextWidth

La función SetFont no fue usada porque no solo se utiliza un tipo de letra, un solo archivo de “font image” fue agregado al código. No hay más fuentes para escoger.

La única función usada fue:

Función	Parámetros	Observaciones
`SHORT GetTextHeight( void* font);`	void* Font: Es el puntero a la imagen de la letra	Ninguna
Overview: Esta función retorna la altura del tipo de letra que se esta usando. Todos los caracteres dados en la tabla del tipo de letra tienen una altura constante.		

Tabla. 3.13. Función GetTextHeight

Funciones de línea:

Función	Parámetros	Observaciones
`void Line( SHORT x1, SHORT y1, SHORT x2, SHORT y2 );`	X1: Coordenada x del punto de inicio Y1: Coordenada y del punto de inicio X2: Coordenada x del punto de final Y2: Coordenada y del punto de final	Ninguna
Overview: Esta función dibuja una línea con el tipo de línea definido desde el punto de inicio hasta el punto final		

Tabla. 3.14. Función Line

Funciones de rectángulo, círculo y cursor no fueron usadas.

Función de Bitmap:

Dentro de todas las funciones de Bitmap solo fue necesario el siguiente macro:

Estructura/Tipo de dato definido por el diseñador	Campos

`typedef struct {`
 	
`TYPE_MEMORY type;`
`FLASH_BYTE* address;`
 	
`} BITMAP_FLASH;`
	TYPE_MEMORY type : debe ser FLASH
	
	FLASH_BYTE* address: dirección de la imagen bitmap

Overview: Es la estructura para un Bitmap almacenado en memoria FLASH	

Tabla. 3.15. Macro BITMAP_FLASH

Se usa esta macro ya que la imagen esta guardada como archivo .c en la memoria flash. La imagen es traducida gracias a la utilidad Bitmap and Font converter de Microchip, se la puede descargar de www.microchip.com/graphics.

3.1.2.4. Capa display device driver. La capa Display Device Driver, ó capa del driver de la pantalla, es la capa que constituye la selección del archivo del driver del display basada en la configuración especificada en el archivo GraphicsConfig.h, implementado en la capa de aplicación. Esta capa es la más transparente al desarrollador, por eso no volverá a ser tratada, pero para que no surjan mayores dudas en el futuro, se hará una explicación general de su funcionamiento y de los archivos .c y .h donde reside y los que son necesarios para que una aplicación funcione. Más adelante se hará referencia a esta sección.

Esta capa se organiza como muestra la Figura 3.7:

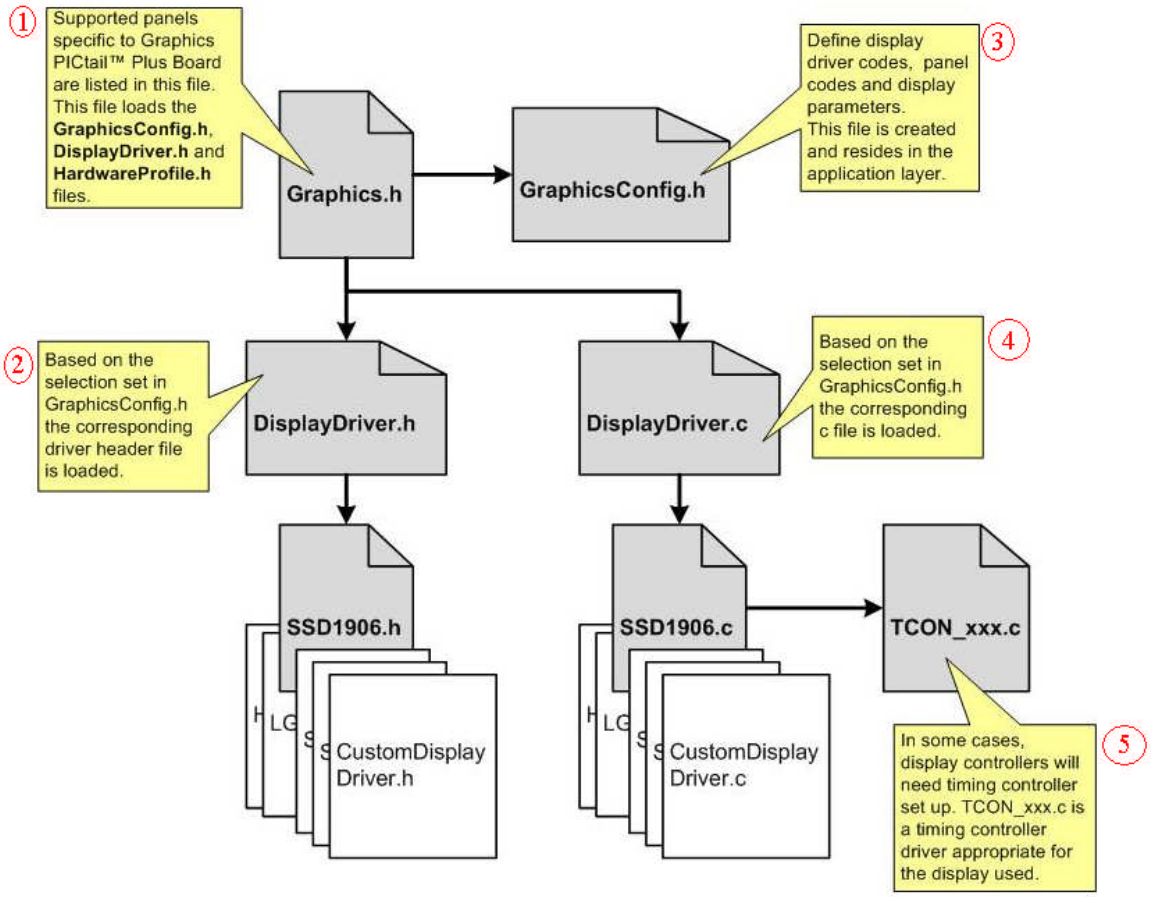


Figura. 3.7. Organización de la capa Display Device Driver

1. Los archivos para trabajar con las diferentes tarjetas para gráficos PICtail™. Este archivo carga los archivos: GraphicsConfig.h, DisplayDriver.h y HardwareProfile.h.
2. Basado en la selección del archivo GraphicsConfig.h, el archivo header del driver correspondiente es cargado.
3. Define los códigos del driver, códigos de paneles y parámetros del display. Este archivo es creado y reside en la capa de aplicación.
4. Basado en la selección hecha en GraphicsConfig.h el archivo .c correspondiente es cargado.
5. En algunos casos, los controles del display necesitan control de tiempo. TCON_xxx.c es un controlador de tiempo apropiado dependiendo del display. Para el caso del Starter Kit PIC24F los archivos serán SH1101A.c y SH1101A.h [*29*].

Funciones de nivel primitivo (primitive):

Función	Parámetros	Observaciones
`WORD GetPixel( SHORT x, SHORT y );`	SHORT x: Coordinada del punto x SHORT y: Coordinada del punto y	Ninguna
Overview: Retorna el color del pixel de la coordenada x,y.		

Tabla. 3.16. Función GetPixel

Función	Parámetros	Observaciones
`WORD PutPixel( SHORT x, SHORT y );`	SHORT x: Coordinada del punto x SHORT y: Coordinada del punto y	Ninguna
Overview: Pone un pixel y la coordenada x,y del color establecido.		

Tabla. 3.17. Función PutPixel

Función	Parámetros	Observaciones
`void DelayMs( WORD time );`	WORD time: Retardo en milisegundos	Ninguna
Overview: Retarda la ejecucion en tiempo especificada en milisegundos. El retardo correcto se da solo para microcontroladores de 16 MIPS.		

Tabla. 3.18. Función GetPixel

3.1.3. FLUJO Y MENSAJERIA DE LA APLICACIÓN

El diseño del firmware para la aplicación esta basado en todos los lineamientos que se han especificado hasta el momento. Los requerimientos ya fueron planteados en la sección 3.1. Todas las funciones que se tomarán del recurso API Microchip Graphics Library han sido ya explicadas y detalladas. Las capas de la arquitectura también han sido ya descritas y las funciones necesarias enumeradas.

Ahora, teniendo ya el conocimiento de para que sirve y como funciona cada capa, el siguiente paso es diseñar el flujo que tendrá el firmware. En otras palabras, como van a interactuar las capas, que sistema de mensajes van a usar, como será estructurada la función main, como será la máquina de estados y que funciones son necesarias programar para desarrollar las aplicaciones. Estos temas serán desarrollados en las siguientes secciones.

3.1.3.1. Flujo de la librería de gráficos. El flujo del programa se basa en el flujo de la librería de gráficos, ya que como se menciona en la sección 3.1 es una aplicación que prioriza la interacción con el usuario.

La creación, manejo y destrucción de los objetos esta previsto en la librería. Todo esto es posible ya que la estructura maneja un proceso de mensajería que será detallado en la siguiente sección. Los mensajes recibidos son procesados y basándose en el contenido del mensaje, los objetos afectados son alterados. La librería entonces redibuja automáticamente el objeto dependiendo de su estado [*2*].

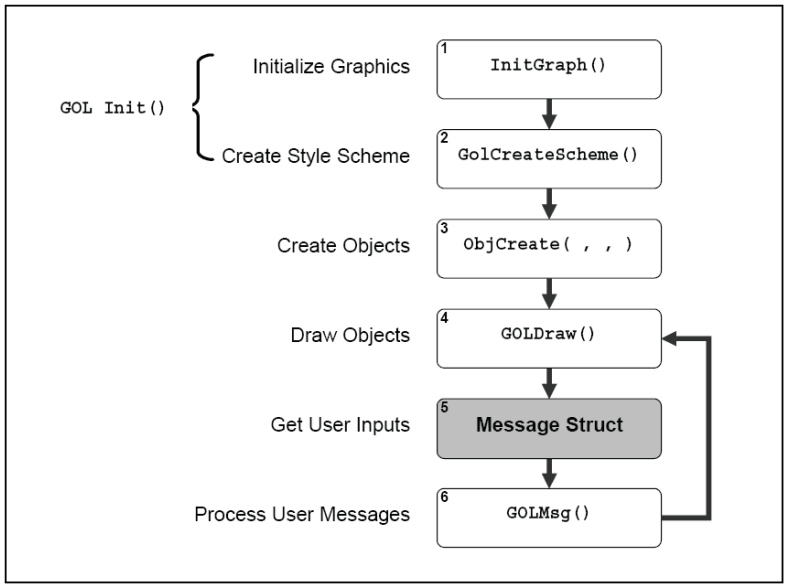


Figura. 3.8. Flujo de la librería de gráficos [*2*]

En la Figura 3.8 se muestra el flujo para usar la librería. Asumiendo que el módulo de display (pantalla OLED) y sus drivers ya han sido añadidos al proyecto correctamente, los siguientes pasos incluirán:

1. Iniciar la función InitGraph() para reiniciar el controlador del display, mover el cursor a (0,0) e inicializar el display a todo negro.
2. Crear el esquema. Este paso es opcional, ya que existe un esquema creado por defecto. Luego, la función GOLCreateScheme() es llamada para definir el cambios al estilo por defecto.
    a. En este caso las funciones InitGraph() y GOLCreateScheme() son desarrolladas por una sola función al llamar a GOL_Init().
3. La creación de objetos corresponde la capa de aplicación y se puede hacer en la función GOLDrawCallback.
4. La función GOLDraw(), la administradora de dibujo, se encarga de hacer los dibujos de la lista activa de objetos.
5. Obtener entradas del usuario (User inputs). Este paso será descrito en la siguiente sección al explicar la mensajería en la capa GOL.
6. El siguiente paso es procesar los mensajes del usuario, esto se hace en la función GOLMsg().
7. Luego que la acción del usuario ha sido procesada y traducida, es posible que los objetos hayan cambiado de estado o que una nueva lista de objetos haya

sido activada. En definitiva, una entrada del usuario es seguro que afectará a al menos un objeto de la lista o la apariencia de la pantalla, es por eso que se debe volver al paso 4 para que los objetos afectados sean redibujados. El regresar al paso 4 significa que se debe hacer todos estos pasos de una manera cíclica, traduciendo a lenguaje C: ¡dentro de un bucle!

3.1.3.2. Mensajería de la capa gol. Como se ha mencionado antes, la librería de gráficos presenta una plataforma para trabajar con distintos dispositivos de entrada como touchscreens, teclados, teclados touch pad (Starter Kit PIC24F), etc [*2*].

Para trabajar con periféricos de entrada, la librería proporciona una estructura de mensajería entre capas que le permite:

- Esperar que el usuario haga una acción (presione un botón, suelte un botón, mueva el dedo)
- Traducir el mensaje crudo (raw message) que llega del dispositivo
- Interpretar y ejecutar la acción programada en el firmware para la acción hecha por el usuario[21]

La ayuda de la librería de gráficos "Microchip Graphics Library Help" en la sección GOL Messages dice "*Para facilitar el procesamiento de las acciones del usuario en los objetos, un sistema de mensajería es usado. El usuario pasa mensajes desde los periféricos de entrada hacia la capa GOL usando un mensajes GOL*." Este mensaje es descrito en la tabla 3.19.

[21] **Este proceso, ligado al proceso de dibujar esta explicado en la Figura 3.8**

Estructura/Tipo de dato definido por el diseñador	Campos
typedef struct{ BYTE type; BYTE uiEvent; int param1; int param2; } **GOL_MSG**;	Type: define el tipo de dispositivo donde los mensajes fueron creados. Estos son los dispositivos implementados en la capa de interfaz de usuario (hardware). Algunos tipos de dispositivos pueden ser: TYPE_UNKNOWN TYPE_KEYBOARD TYPE_TOUCHSCREEN TYPE_MOUSE uiEvent: Identificación del evento del usuario o tipo de acccion hecha sobre el objeto. Los posibles eventos son: EVENT_INVALID EVENT_MOVE EVENT_PRESS EVENT_RELEASE param1: la definición de los parámetros 1 y 2 varía según el tipo de dispositivo. Por ejemplo, parámetro 1 y 2 son definidos como las coordenadas x,y para TYPE_TOUCHSCREEN. Para TYPE_KEYBOARD, param1 esta definido como la identificación del objeto y param2 como el código de carácter del teclado.
Overview: Es la estructura del mensaje que se obtiene cuando un usuario ejecuta una acción. La función GOLMsg() acepta esta estructura y procesa el mensaje para todos los objetos de la lista activa.	

Tabla. 3.19. Estructura del tipo de dato creado: GOL_MSG

El mecanismo de mensajería sigue el flujo mostrado en la Figura 3.9:

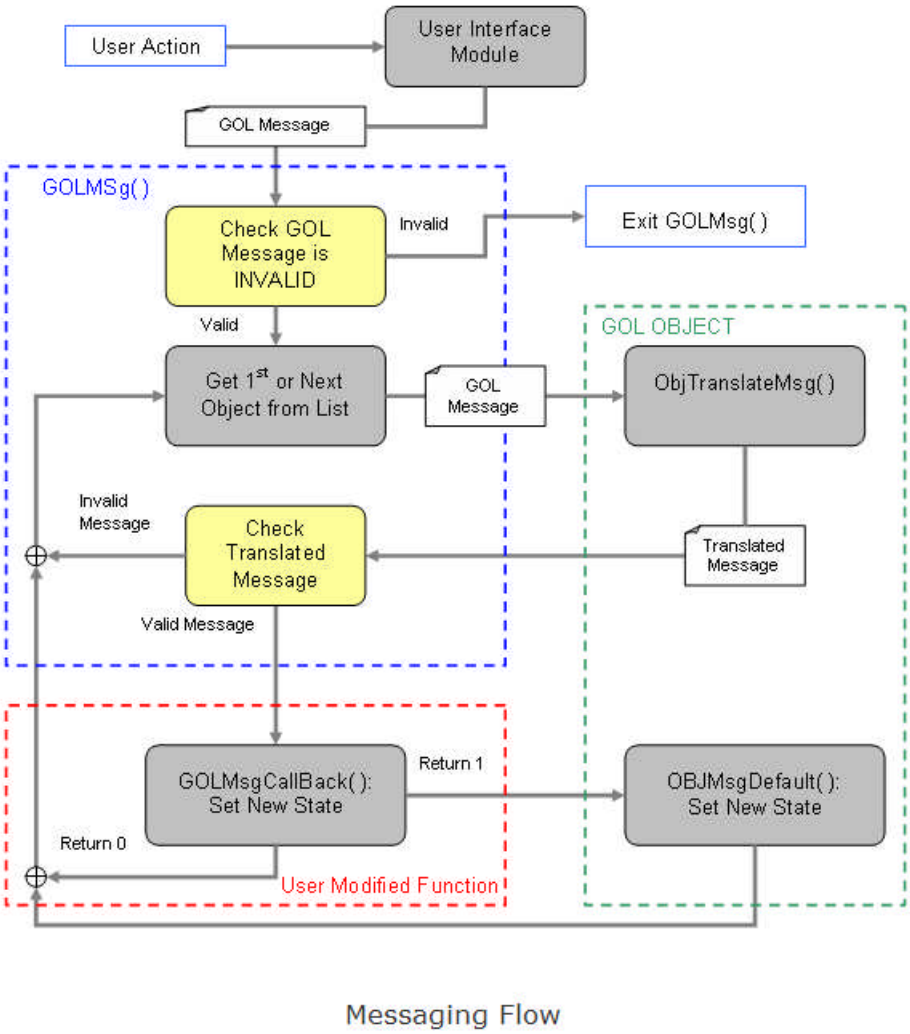


Figura. 3.9. Flujo de la mensajería

La Figura 3.9 [*2*], tomada de Microchip Graphics Library Help, muestra como funciona el diseño de la estructura de mensajería. El proceso es:

1. El módulo de interfase con el usuario (touchscreen, teclado) envía un mensaje GOL (con la estructura GOL_MSG).
2. Un bucle evalúa que objeto es afectado por el mensaje. Esto es hecho dentro de la función GOLMsg().
3. Los objetos afectados retornan el mensaje traducido basado en los parámetros del mensaje GOL.

4. El usuario puede cambiar la acción por default con la función callback. En caso de que la función callback retorne un valor distinto de cero, el mensaje será procesado de la manera default.
5. El objeto debe ser redibujado para reflejar el nuevo estado.

El mensaje traducido es un conjunto de acciones únicas para cada tipo de objeto. El documento Microchip Graphics Library provee más información sobre cada objeto.

3.2 DESARROLLO DEL FIRMWARE PARA EL STARTER KIT PIC24F

3.2.1. ARQUITECTURA GENERAL DE LA APLICACIÓN

En esta sección se detallará una estructura básica que seguirá la aplicación desde que la tarjeta se energice y llegue al menú de inicio. Recordando la analogía al desarrollo de un edificio, ahora se procederá a detallar en el plano cuantos pisos tendrá el edificio, cuantas habitaciones habrá en cada piso, donde estarán las salidas de emergencia, etc.

3.2.1.1. Flujo de inicio del programa. El flujo del programa será de acuerdo al flujo prediseñado de la API que se usa, la librería de gráficos. El flujo será diseñado para cuando la tarjeta es energizada, llegue al bucle principal, vaya a la aplicación llamada y siempre retorne al menú inicio.

Para detallar el flujo del programa se usarán varios gráficos que indicarán cada vez más detalles y especificaciones, empezando desde lo más general.

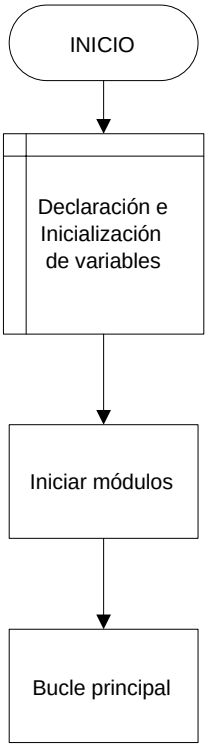


Figura. 3.10. Flujo del firmware para el Starter Kit PIC24F

La Figura 3.10 indica los bloques que se siguen luego de prender la tarjeta. Las variables que son inicializadas son de distintos tipos: #define, global y bits de configuración. Los módulos que se inicializan son los que se usan para las aplicaciones. Módulo RTCC, CTMU, USBUtilize y el módulo de gráficos GOLInit.

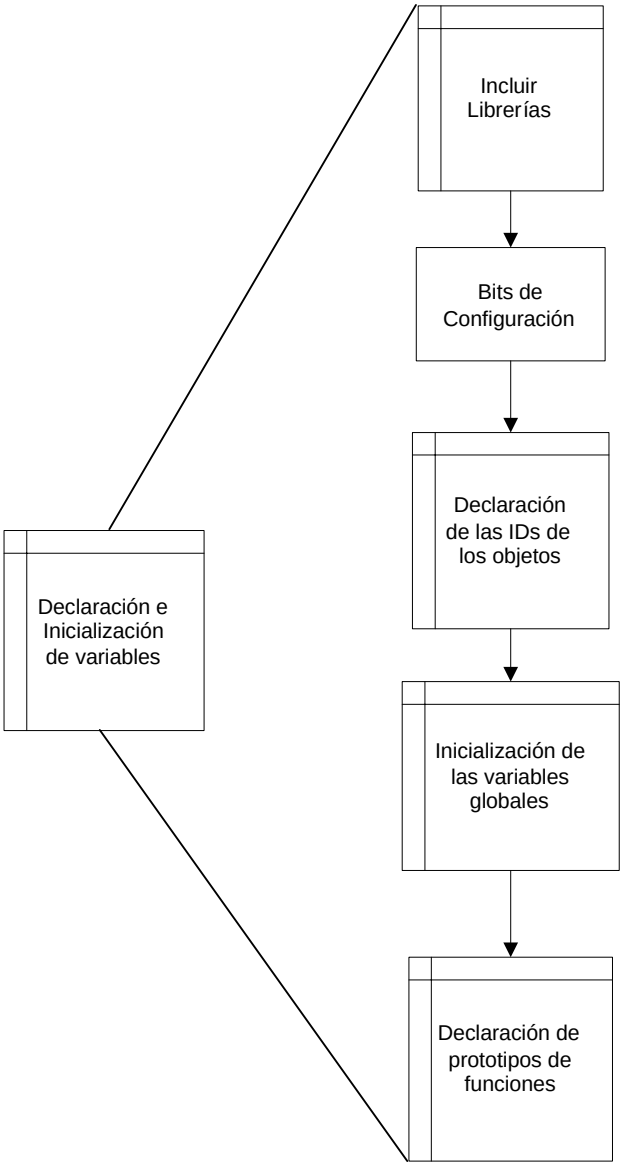


Figura. 3.11. Declaraciones e inicializaciones

Todos los bloques de la Figura 3.11 están en el archivo “main.c” donde se encuentra la función principal. Para el caso de esta aplicación a este archivo se le denominará PrimerProyecto.c. En este mismo archivo se incluye la función `int main (void)`, la cual determina a PrimerProyecto.c como el archivo main.c, por lo tanto, es el que primero que es leído por el compilador.

Luego de las declaraciones de la Figura 3.11, es el momento en que se debe declarar la función main, será de la siguiente manera: int **main** (void) {…}. La Figura3.12 indica el flujo inicial.

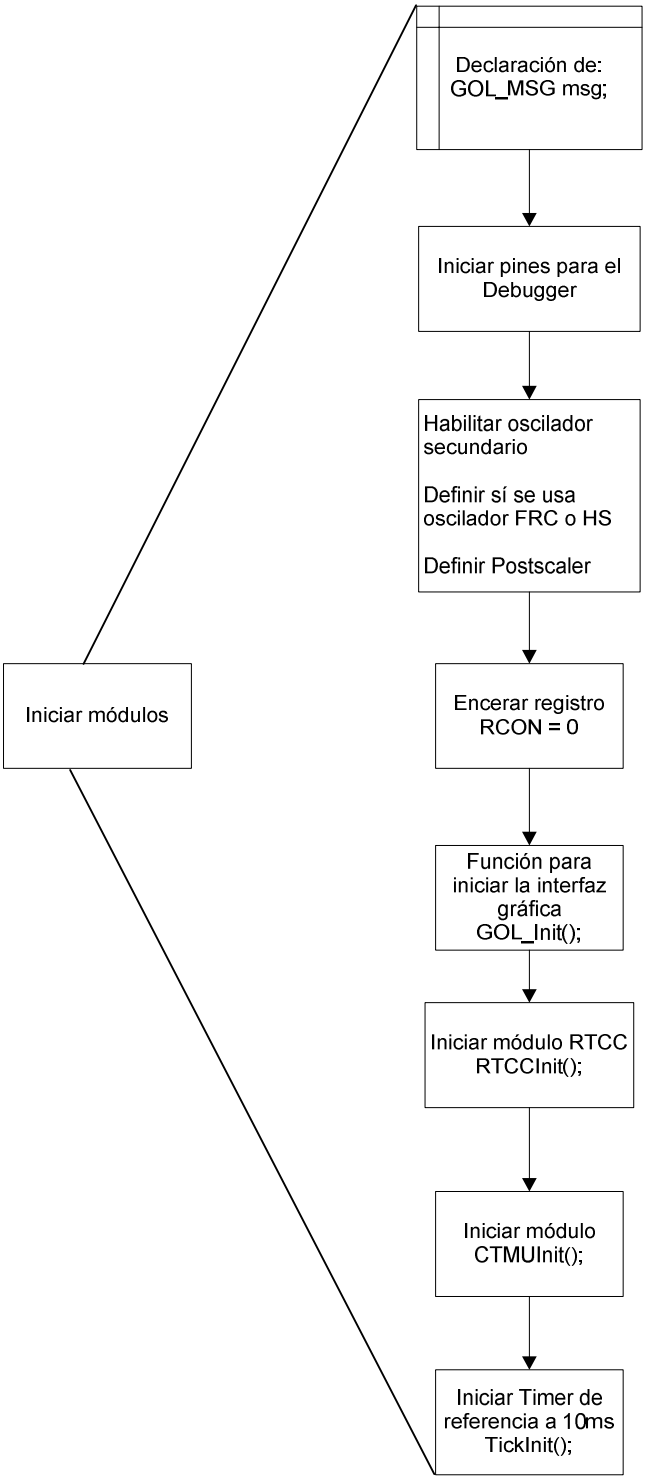


Figura. 3.12. Inicialización de módulos

3.2.1.2. El bucle principal. El bucle principal es un bucle infinito tipo *while*. Este bucle se tomo del modelo del documento AN1136. Las funciones que están en inglés son propias de la API, como por ejemplo, la función para tomar un mensaje crudo o "*raw message*" y la validación del evento que registre el Touchpad, además de las variables que usan estas funciones. Este bucle se basa en el flujo de la librería de gráficos de la Figura 3.8.

```
// Loop principal

while( 1)
{

    if (GOLDraw())                          // Dibujar pantalla o re-dibujar objetos afectados
    {
        TouchSenseButtonsMsg( &msg );       // Obeter un mensaje crudo del touch pad
        if ((msg.uiEvent != EVENT_INVALID) && ((tick - displayChangeTime) > MESSAGE_DEAD_TIME))     // Validar el mensaje
        {
            TraducirTouchpad( &msg );      // Traducir el mensaje crudo obtenido
            GOLMsg( &msg );                // Procesar el mensaje
        }

    }

}
```

Figura. 3.13. Loop principal

Como puede observarse en la Figura 3.13, la complejidad del bucle principal es mínima. Son apenas cinco líneas de código dentro del *while*. Pero si se analiza un poco más a fondo este bucle, se puede notar cuan efectivo es, cuanto tiempo de procesamiento ahorra y ayudará a comprender como elaborar la máquina de estados.

```
while( 1)
{
```

El bucle será infinito ya que siempre se cumple la condición: 1.

```
if (GOLDraw())
{
```

En la tabla 3.8, se describe que la función GOLDraw, retorna un "1" cuando el proceso de gráficas ha sido terminado y un "0" si todavía no lo es.

```
TouchSenseButtonsMsg( &msg );
```

Esta función obtiene un "*raw message*" y es pre-programada en el demo de la tarjeta. Se describe en la tabla 3.20.

Función	Parámetros	Observaciones
`void TouchSenseButtonsMsg( GOL_MSG* msg )`	GOL_MSG* msg: el nuevo mensaje del sistema	No tiene returns.
Overview: Lo que se busca con esta función es detectar un cambio en el estado del botón. Sí existe un cambio, se envía un mensaje. **Notesé que esto significa que no se enviarán mensajes repetitivos sí el touchpad se mantiene presionado por un período largo**.		

Tabla. 3.20. Función TouchSenseButtonsMsg

```
if ((msg.uiEvent != EVENT_INVALID) && ((tick - displayChangeTime) > MESSAGE_DEAD_TIME))
{
```

Este statement[22] es para la validación del mensaje *msg*. Recuérdese la tabla 3.19, donde se especifica la estructura de GOL_MSG, ahí se habla de los posibles eventos generados en el campo uiEvent, sí el evento es inválido se al fin del bucle.

MESSAGE_DEAD_TIME es un macro que esta definido en las declaraciones de variables así: `#define MESSAGE_DEAD_TIME (1000/MILLISECONDS_PER_TICK)` recordando que MILLISECONDS_PER_TICK es definido como 10 milisegundos.

DisplayChangeTime es una **variable global** del tipo DWORD que se usa en la máquina de estados para indicar que la función que grafica la pantalla ha terminado su proceso.

`TraducirTouchpad( &msg );` Esta función se encarga de la traducción del mensaje producido por el usuario. Se describe en la tabla 3.21.

[22] **Statement es termino usado en computación para referirse a una declaración**

Función	Parámetros	Observaciones
`void TraducirToucpad( GOL_MSG* msg )`	GOL_MSG* msg: el mensaje actual en el sistema	No tiene returns.
Overview: Esta función traduce el mensaje del teclado capacitivo en un mensaje apropiado para la pantalla, basandose en lo que esta desplegado en la pantalla en ese momento. Esta función trabaja con la máquina de estados llamando a una función específica por estado.		

Tabla. 3.21. Función Traducir Touchpad

`GOLMsg( &msg );` Se llama a la función GOLMsg enviando como parámetro el mensaje actual "msg" para que se procese la programación del usuario para tal mensaje.

Nótese que hay un cierre de función "}" luego del cierre de función del bucle principal, este corresponde al cierre de la función main. La función principal termina en este punto.

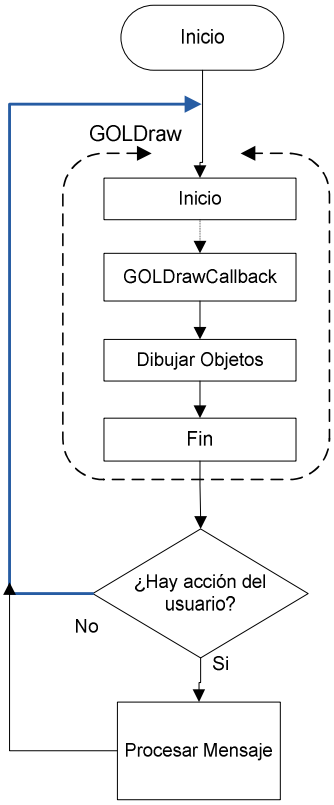
Las funciones mencionadas, además de las funciones callback están declaradas en el archivo main *PrimerProyecto.c*.

3.2.1.3. La máquina de estados. El proyecto actual presenta varias utilidades, algunas de las cuales tienen menús individuales. Poseen también distintas pantallas de introducción, instrucciones, etc. Es por eso que la mejor opción es organizar la máquina de estados de acuerdo a las pantallas que se presentan. Todas las funciones son fundamentales, pero dos son las más importantes: la función callback para dibujar y la función callback para procesar el mensaje. Ambas son llamadas desde sus funciones base[23] en el bucle principal.

Supóngase el siguiente caso. Se diseña un estado para el menú de inicio, se le nombra: MENU_INICIO. En este estado, en la pantalla se despliega el menú de inicio propuesto como ejemplo en la sección 3.1.2.1 en la Figura 3.5 y además se espera la acción

[23] **Funciones base de funciones callback son GOLDraw y GOLMsg.**

del usuario para escoger una de entre las cuatro opciones. En este estado, la función GOLDraw grafica los cuatro botones que llevan a cada una de las aplicaciones. Ahora, ¿Qué pasaría sí el usuario se toma un momento hasta seleccionar una opción? Como el estado se mantiene, la función GOLDraw volvería a empezar el proceso de graficar el menú. Es cierto que al terminar de imprimir la pantalla GOLDraw retornaría un "1" y este "1" es el que permite pasar el primer statement condicional del bucle principal. El problema radica en que mientras se esté en el estado MENU_PRINCIPAL[24] la función GOLDrawCallback seguirá haciendo el proceso de limpiar la pantalla, borrar la lista activa y crear los objetos del menú: los cuatro botones, bitmaps, título en instrucciones. *Por consecuente, la pantalla se graficará una y otra vez hasta que el usuario presione un botón y la máquina de estados vaya a otro estado,* tal como presenta la Figura 3.14.



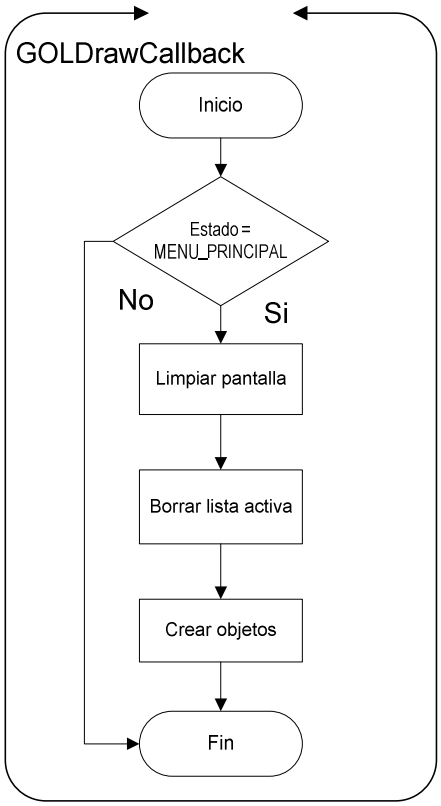


Figura. 3.14.A. Flujo del Firmware **Figura. 3.14.B. Flujo función GOLDrawCallback**

Como se puede apreciar en la Figura 3.14.A, el flujo se repite constantemente mientras no cambie la variable estadopantalla (flecha azul). La Figura 3.14.B indica lo que sucede cuando la función GOLDrawCallback es llamada y como se puede ver, la variable

[24] **La variable Estado se inicializa en MENU_INICIO**

estadopantalla no es modificada aquí tampoco. En consecuencia, el flujo se repite infinitamente, lo que implica que la pantalla se grafique repetidamente una y otra vez.

¿Cómo solucionar este problema? La mejor solución es separar este estado en dos, uno para graficar la pantalla y otro para esperar y procesar la acción del usuario, como muestra la Figura 3.15:

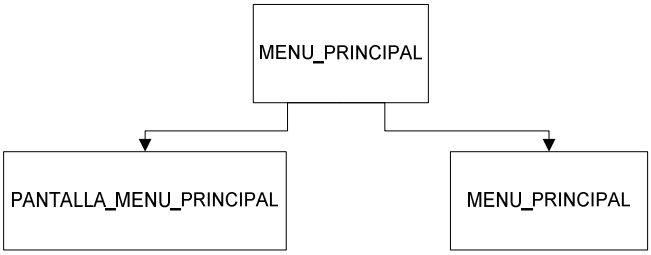


Figura. 3.15. División en dos estados

De esta manera el flujo quedaría así:

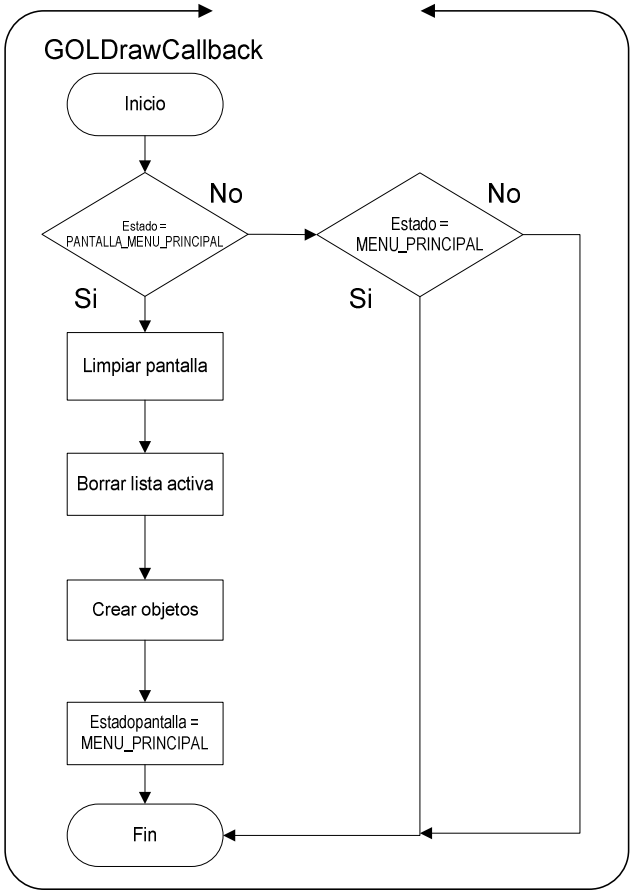


Figura. 3.16. Flujo luego de la división en dos estados

Al energizar la tarjeta se recorre por primera vez el flujo de la Figura 3.14.A. Como puede notarse en la Figura 3.16, que estudia el flujo de GOLDrawCallback, la última acción del flujo en el estado PANTALLA_MENU_PRINCIPAL, es igualar la variable estadopantalla a MENU_PRINCIPAL. Entonces, a partir de la segunda vez que se recorra el flujo ya no se entra al estado de PANTALLA_MENU_PRINCIPAL, sino que se va al estado MENU_PRINCIPAL. De esta forma no se grafica una y otra vez la pantalla. Se la grafica una vez y luego se espera que el usuario haga una acción para traducirla.

Ahora, ya que existe una idea de cómo ir creando estados de acuerdo a las necesidades del diseñador, se procede ahora a redactar como crear la máquina de estados.

El lenguaje C para microcontroladores permite la creación de un tipo de dato definido por el usuario. Esto es posible gracias a la estructura *typedef.* Typedef es una declaración. Su propósito es crear nuevos tipos de datos a partir de tipos de datos existentes. La declaración de esta variable crea nuevas locaciones de memoria. Ya que un typedef es una declaración, se puede mezclar con declaraciones de variables, aunque una práctica común es definir a los typedefs primero y luego las variables [*31*]. El tipo de dato creado se llamará ESTADOS_DE_LA_PANTALLA y constará con todos los estados necesarios. Ahora, hay que crear una variable de este tipo de dato, deberá ser global (extern). Se llamará “estadopantall**a**”. Toda esta declaración se hará en el archivo cabecera de *PrimerProyecto.c*, el archivo *PrimerEsqueleto.h.* El manejo de la máquina de estados se hará con una estructura switch case.

En esta sección no se ha pretendido numerar todos los estados; por el contrario, se ha tratado de no hacerlo, solo de indicar como se crean de acuerdo a la necesidad, siempre teniendo en cuenta el flujo del firmware. Los estados serán analizados de acuerdo a la aplicación que correspondan y serán explorados en la siguiente sección junto con las funciones que encierran.

3.1.2. DISEÑO DEL FIRMWARE

El firmware consta de cuatro aplicaciones diseñadas para explotar los módulos más recientes y poderosos de la familia PIC24F. Además, se diseño una aplicación que se encargue de inicializaciones, configuraciones y de presentar un menú en pantalla para seleccionar entre las otras cuatro aplicaciones. A esta última aplicación se le denominará aplicación “principal, a las demás como “secundarias”[25].

Las aplicaciones son construidas desde las tres funciones de la API que corresponden programar al diseñador. Es decir, se debe construir lo que se desea que aparezca en pantalla (GOLDraw), como se desea interpretar los mensajes del teclado (TraducirTouchpad) y que se desea que haga ese mensaje (GOLMsg). Cada aplicación puede ocupar varios estados de la máquina de estados. En cada estado (dependiendo de la función que lo haya llamado) se programan las funciones necesarias para cumplir con el objetivo de la aplicación, tal como es explicado en la Figura 3.17:

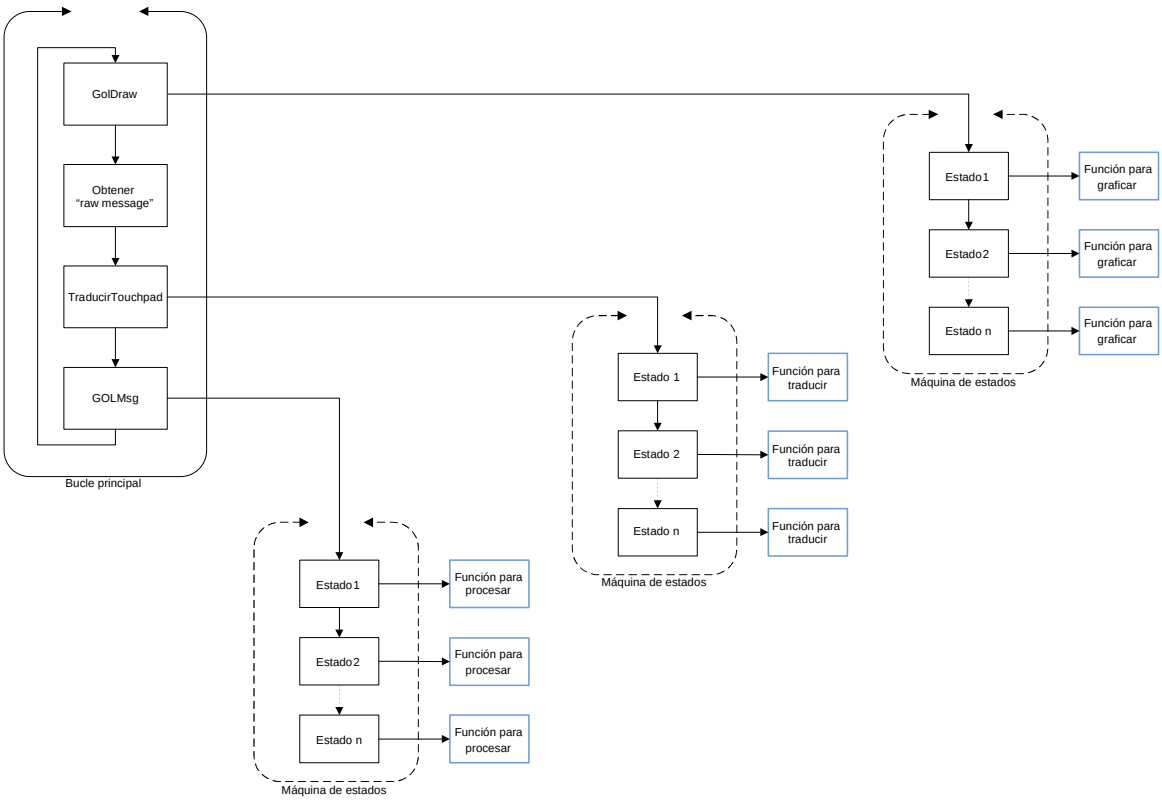


Figura. 3.17. Flujo del bucle principal en llamada a funciones diseñadas

[25] **Principal o secundaria no se refiere a una jerarquía o importancia de una mayor a la otra, ni que la principal administre los recursos destinados para las secundarias, de cierta forma es solo una denominación cronológica.**

Lo que se va a tratar en esta sección, es el diseño de las funciones para graficar, traducir y procesar. Ahora, para organizar su estudio se agruparán de acuerdo a la aplicación a la que pertenezcan. Por ejemplo, la aplicación secundaria “Gráficos” puede tener tres estados y cada estado diferentes funciones.

El estudio de esta sección entonces será el indicado en la Figura 3.18. En caso de que un estado no use una función en alguna de las tres funciones, no se le nombrará.

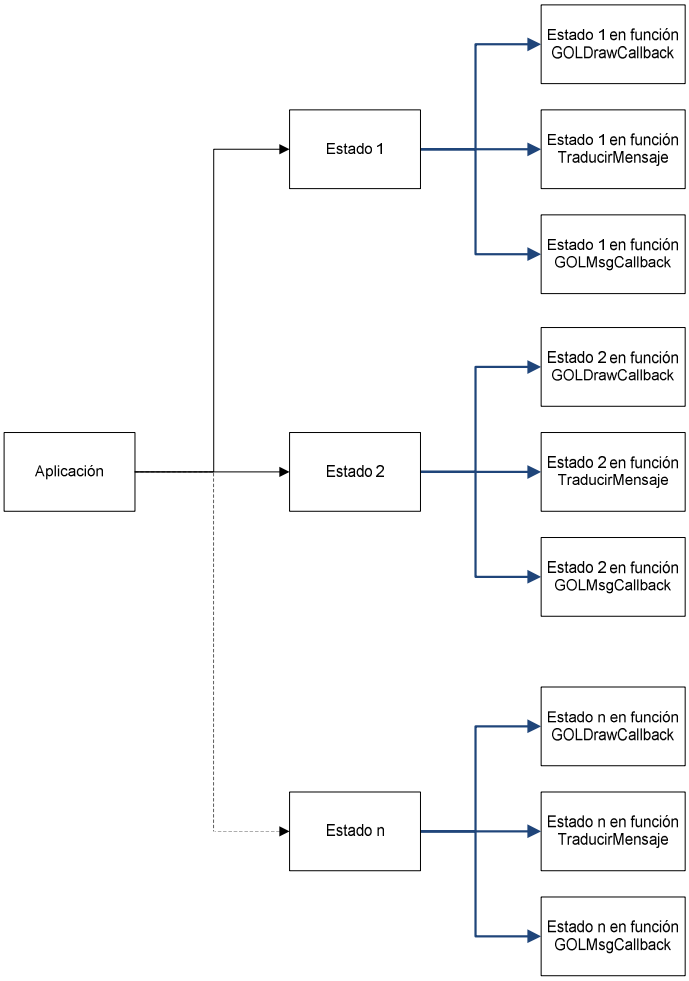


Figura. 3.18. Estudio de las funciones diseñadas para cada estado

3.1.2.1. Aplicación principal. En esta aplicación se integrará en el arranque del programa, la presentación del proyecto y presentación del menú de inicio. Se ocuparán tres estados dentro de la máquina de estados:

- PANTALLA_PRUEBA_TARJETA[26]
- PANTALLA_MENU_PRINCIPAL
- MENU_PRINCIPAL

PANTALLA_PRUEBA_TARJETA: Función GOLDrawCallback

El primer estado definido dentro de la estructura *typdef eum* debe ser igualado a 0.

```
/* ------------------------------------------
Tipos de datos creados
----------------------
El tipo de dato ESTADOS_DE_LA_PANTALLA fue crea-
do para que la variable estadopantalla pueda ma-
nejar una maquina de estados que controla toda
la aplicacion.
------------------------------------------ */
typedef enum
{
    PANTALLA_PRUEBA_TARJETA = 0             ,


} ESTADOS_DE_LA_PANTALLA;
```

Figura. 3.19. Declaración inicial de la máquina de estados

Este estado se encargará de hacer las operaciones necesarias para inicialización de la aplicación: comprobar que la pantalla funcione adecuadamente, presentar el título y autor del proyecto. Como todas estas operaciones son gráficas y no se requiere la acción del usuario, este estado solo ejecutará acciones en la función GOLDraw() y no en la función GOLMsg(). Ahora, en este estado se imprimirán pantallas para:

1. Pantalla de introducción
2. Pantalla de presentación del proyecto y autor
3. Pantalla del menú de inicio

[26] **No es necesario crear un estado PRUEBA_TARJETA porque no se espera ninguna acción del usuario.**

Antes de proceder a programar la creación de objetos, diseñar las pantallas y más, es necesario tener en cuenta que todo esto no puede ser desarrollado dentro de la misma función GOLDrawCallback. Sería una mejor alternativa, usar una función que haga todo lo necesario en una sección de código diferente a la función callback. Esta función se llamará **ProbarTarjeta()**.

La función **ProbarTarjeta()** deberá seguir el flujo indicado en la Figura 3.18. Esta función deberá trabajar con ayuda de un temporizador y de un contador para que las pantallas tengan una duración establecida; por ejemplo, para que la pantalla de introducción dure 2 segundos, estar en blanco 1 segundo, etc. Recuérdese que precisamente para estos casos fue diseñado el modulo RTCC.

Con la ayuda de la función **RTCCProcessEvents()**, su variable **_tmr_str[11]** (correspondiente a los segundos) y un contador de segundos transcurridos (declarado para uso exclusivo de la función), se puede manejar perfectamente la temporización para cada pantalla. El contador declarado es: `static int sgdstranscurridos = 0;`

Función	Parámetros	Observaciones
`void RTCCProcessEvents(void) { }`	Ninguno	RTCCInit() debe ser llamada antes
Overview: Esta función obtiene el tiempo y la fecha actual de RTCC y los traduce en strings. Su salida consiste en actualizar los strings de fecha y hora: _time_str, _date_str, _time, _time_chk. ***El segundo actual es guardado en la variable _time_str[11].***		

Tabla. 3.22. Función RTCCProcessEvents

La función ProbarTarjeta() tendrá el siguiente flujo:

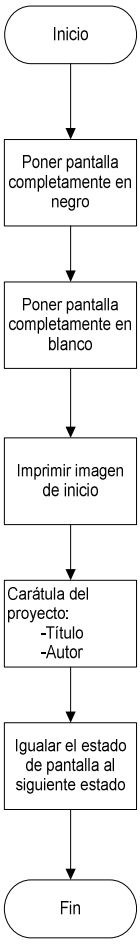


Figura. 3.20. Flujo de la función ProbarTarjeta()

Analícese que pasaría si añadimos esta función al flujo del programa:

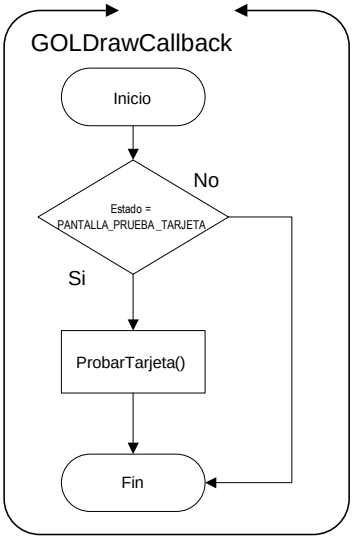


Figura. 3.21. Diagrama de flujo con la función ProbarTarjeta

Obsérvese la Figura 3.14.A junto con el flujo de la Figura 3.21, notará que se tiene el mismo problema anterior. Este problema, que fue expuesto en la sección 3.2.1.3 no puede

ser resuelto añadiendo un estado porque no se espera una acción del usuario. Se espera que se cumpla el tiempo configurado para que cada una de las 3 pantallas este impresa. Entonces, un sistema de control[27] para registrar los segundos que permanece cada pantalla debe ser implementado. Los estados que la pantalla debe adoptar para cada segundo están descritos en la Figura 3.22.

Pantalla:

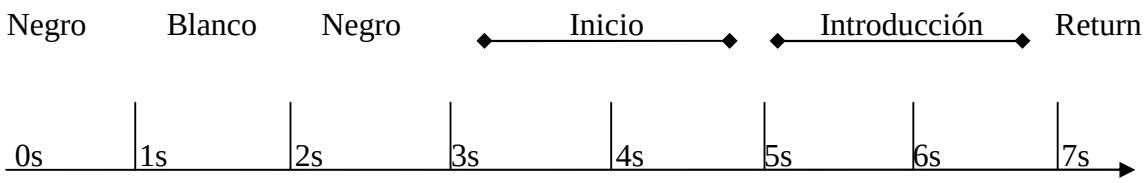


Figura. 3.22. Diagrama en el tiempo de la pantalla

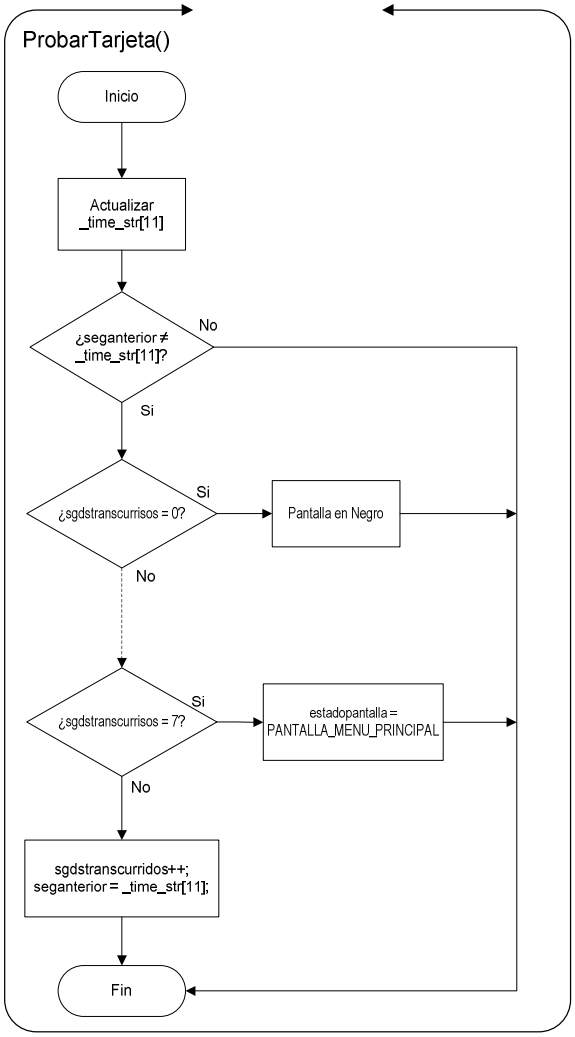


Figura. 3.23. Flujo con registro centinela para cambio de estado

[27] **A estos sistemas de control, algunos autores de lenguaje C los refieren como *registro centinela***

El flujo dentro de la función GOLDrawCallback no cambiará de cómo estaba en la Figura 3.21. En la función ProbarTarjeta (tabla 3.23) se programará (Figura 3.23) un registro centinela (sgdstranscurridos = 7), para cambiar la variable estadopantalla[28] al siguiente estado.

Función	Parámetros	Observaciones

`void ProbarTarjeta (void)`
`{`
`}`
	Ninguno	GOLInit() debe ser llamada antes

Overview: Esta función maneja las pantallas que se presentarán como introducción al proyecto y los tiempos que cada una permanezca. Usa la ayuda del módulo RTCC y contadores del tipo int para llevar cuenta de los segundos.		

Tabla. 3.23. Función ProbarTarjeta()

PANTALLA_PRUEBA_TARJETA: Función TraducirTouchpad

Recuérdese que la función TraducirTouchpad será llamada en caso de que alguna acción del usuario se ejecute. Pero para este estado, no se requieren acciones del usuario ya que se está probando el buen funcionamiento de la tarjeta. Sí el usuario presiona un botón del teclado mientras la función ProbarTarjeta está en ejecución, los mensajes serán catalogados como no validos (EVENT_INVALID) en el campo type del mensaje GOL_MSG, tal como se muestra en la Figura 3.24.

[28] Es posible acceder a la variable estadopantalla desde la función ProbarTarjeta() porque está declarada como tipo "extern"

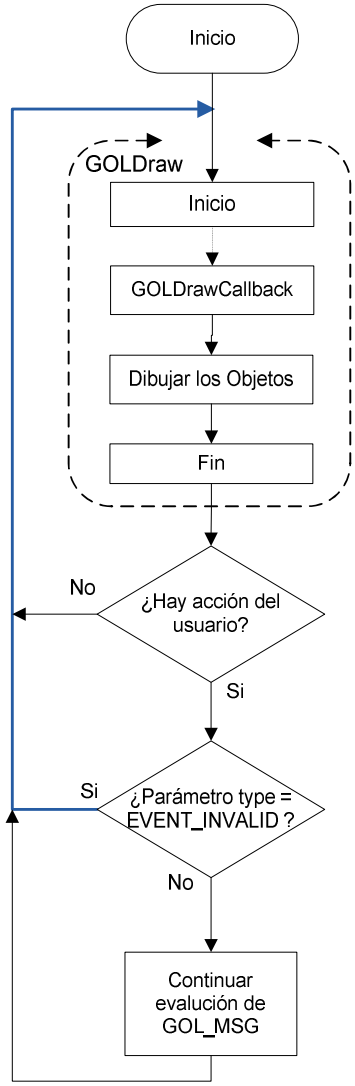


Figura. 3.24. Flujo cuando existe acción del usuario

El flujo que tomará la aplicación en este estado será el indicado por la flecha azul, es decir, cualquier acción que ejecute el usuario no pasará de la validación del mensaje.

PANTALLA_MENU_PRINCIPAL: Función GOLDrawCallback

El siguiente estado es PANTALLA_MENU_PRINCIPAL (Figura 3.25). El flujo es igual al de la Figura 3.21, solo que en vez de llamar a la función ProbarTarjeta(), llama a la función **MostrarMenuPrincipal()**. Esta función se encarga de la creación del menú principal, que lo componen los siguientes objetos:

- Botón tamaño normal “juego”
- Botón tamaño normal “usb”
- Botón tamaño normal “rtcc”
- Botón tamaño normal “rgb”
- Botón tamaño grande “JUEGO”
- Botón tamaño grande “USB”
- Botón tamaño grande “RTCC”
- Botón tamaño grande “RGB”
- Static text “MENU”
- Bitmap de flecha hacia abajo
- Static text “Presione”
- Static text “Para empezar”
- Botón escondido

Recuérdese, que los objetos se agregan a la lista de acuerdo al orden que fueron creados.

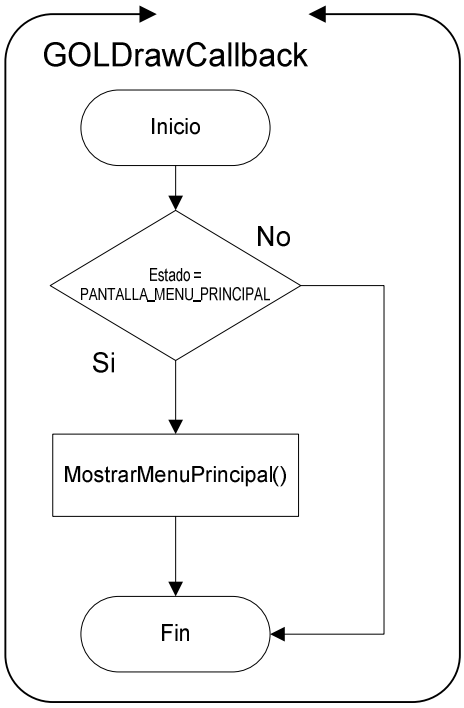


Figura. 3.25. Flujo del estado PANTALLA_MENU_PRINCIPAL

Función	Parámetros	Observaciones
`void MostrarMenuPrincipal (void) { }`	Ninguno	GOLInit() debe ser llamada antes
Overview: Esta función maneja la pantalla que se presentará menu principal en la que se podrá escoger entre las 4 aplicaciones desarrolladas, además presentará una imagen para cada opción.		

Tabla. 3.24. Función MostrarMenuPrincipal

MENU_PRINCIPAL: Función GOLDraw

El último estado de pantalla para la aplicación principal es el estado MENU_PRINCIPAL. Este estado espera la acción del usuario y cuando la recibe, llama a una función para traducir lo que dicha acción representa de acuerdo al menú principal. Además, el menú principal debe permanecer en un estado hasta que reciba la primera acción del usuario. Este estado indicará que la primera acción que debe realizar el usuario es presionar la flecha hacia abajo del teclado, de esta manera accederá al menú. Obsérvese la Figura 3.26 para una mejor comprensión:

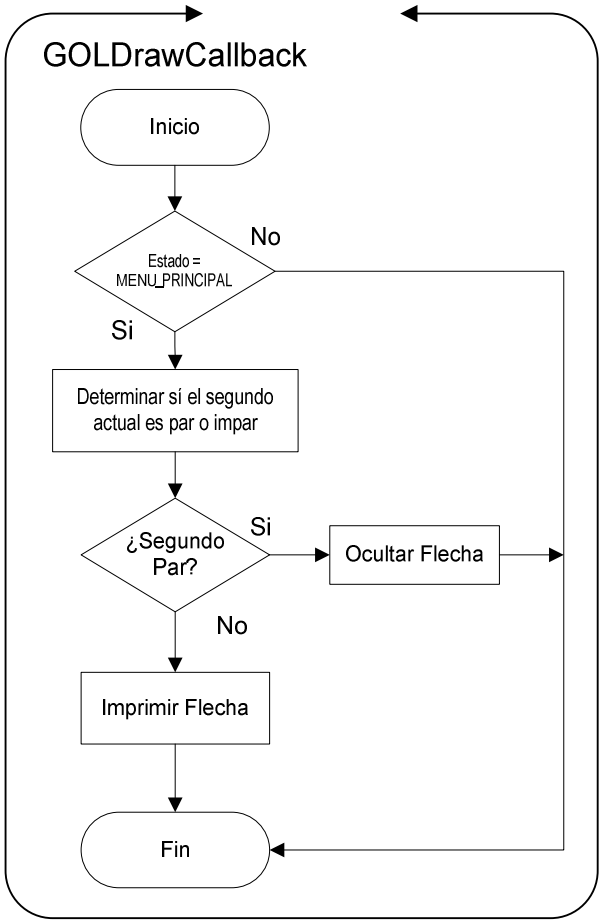


Figura. 3.26. Flujo de MENU_PRINCIPAL en GOLDraw()

La programación para este estado en la función GOLDrawCallback es bastante simple:

```
case MENU_PRINCIPAL:
    RTCCProcessEvents();
    segundos = _time_str[11];
    auxSegundos = segundos%2;
    if(auxSegundos)
    {
        SetState( (PICTURE *)GOLFindObject( ID_BMP_MENU ),  PICT_HIDE );
    }
    else
    {
        SetState( (PICTURE *)GOLFindObject( ID_BMP_MENU ),  PICT_DRAW );
    }

    break;
```

Se actualiza el segundo con la función RTCCProcessEvents, luego con una variable auxiliar se utiliza la operación residuo (%) para 2 y así se determina si el segundo actual es par o impar. Para un segundo par la flecha se dibuja, para un segundo impar la flecha se oculta. Para dibujar o esconder el botón se utiliza el macro:

Macro	Parámetros	Observaciones
`#define SetState( pObj, stateBits ) ((OBJ_HEADER*)pObj)->state |= (stateBits)`	pObj: puntero al objeto de interes statebits: Define que bits de estado se deben configurar. Depende de cada objeto	Ninguna
Overview: Esta macro define el estado para el objeto selecionado. Se debe verificar los posibles estados de cada objeto para usar esta función.		

Tabla. 3.25. Macro SetState

Función	Parámetros	Observaciones
`OBJ_HEADER* GOLFindObject( WORD ID );`	ID: La ID designada por el programador al momento de crear el objeto.	GOLInit() debe ser llamada antes
Overview: Esta función encuentra un objeto en la lista activa de objetos _pGolObjects, usando la ID dada.		

Tabla. 3.26. Función GOLFindObject

MENU_PRINCIPAL: Función TraducirTouchpad

Hasta este momento, la pantalla tiene impresos los cuatro botones normales, la instrucción "Presione…para empezar", el título "MENU" y la flecha hacia abajo:

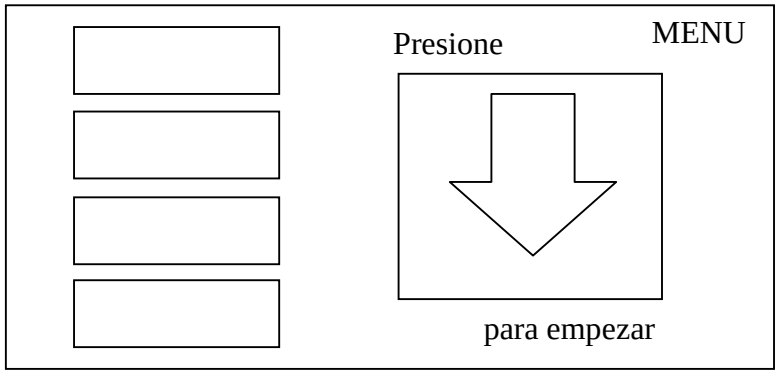


Figura. 3.27. Botones impresos en el menú principal (Botones normales)

Es recomendable estudiar los estados que puede tener el objeto botón[29] antes de continuar. Los objetos creados pero escondidos (no impresos) que tiene la pantalla son:

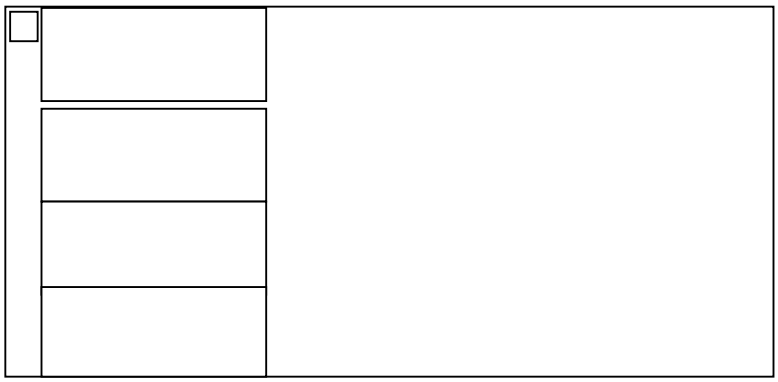


Figura. 3.28. Botones escondidos en el menú principal (Botones Grandes)

Estos botones "escondidos" se mantienen así hasta el momento en que la opción que representan es escogida. Cuando es seleccionada su opción, se muestra/imprime el botón grande y se esconde el botón normal. Un botón escondido es creado en la parte superior izquierda para ayudar a estado de espera inicial.

Los botones para navegar en el menú principal, al ser este un menú vertical, solo son el botón 1 y 3 (arriba y abajo) del teclado. Por tanto la primera acción "válida" que puede

[29] **Una memoria rápida: BTN_PRESSED (presionado/seleccionado), BTN_HIDE (escondido), BTN_DRAW (dibujado)**

hacer el usuario es navegar en el menú, usando el botón 1 (arriba) o el botón 3 (abajo). Se accede a la aplicación con el botón 5.

Del estado inicial (espera-por-acción) donde la flecha hacia abajo esta titilando, si se presiona el botón 3 se ubica el foco del menú en el botón “Juego”. Para indicar que este botón tiene el foco se muestra el botón grande y se esconde el pequeño, la pantalla queda entonces de esta manera:

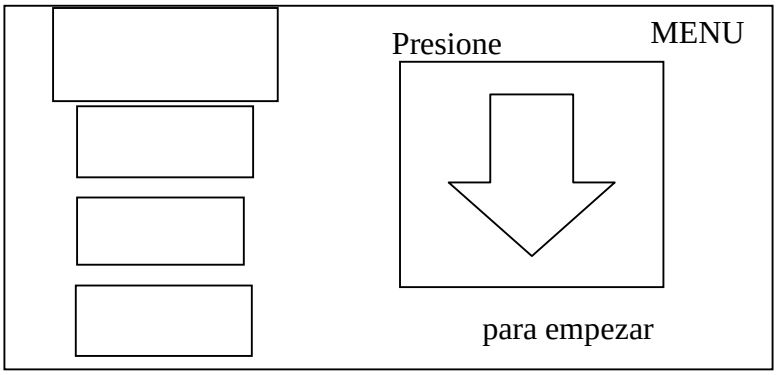


Figura. 3.29. Botón juego con foco

La Figura 3.30 muestra un gráfico de flujo de tiempo:

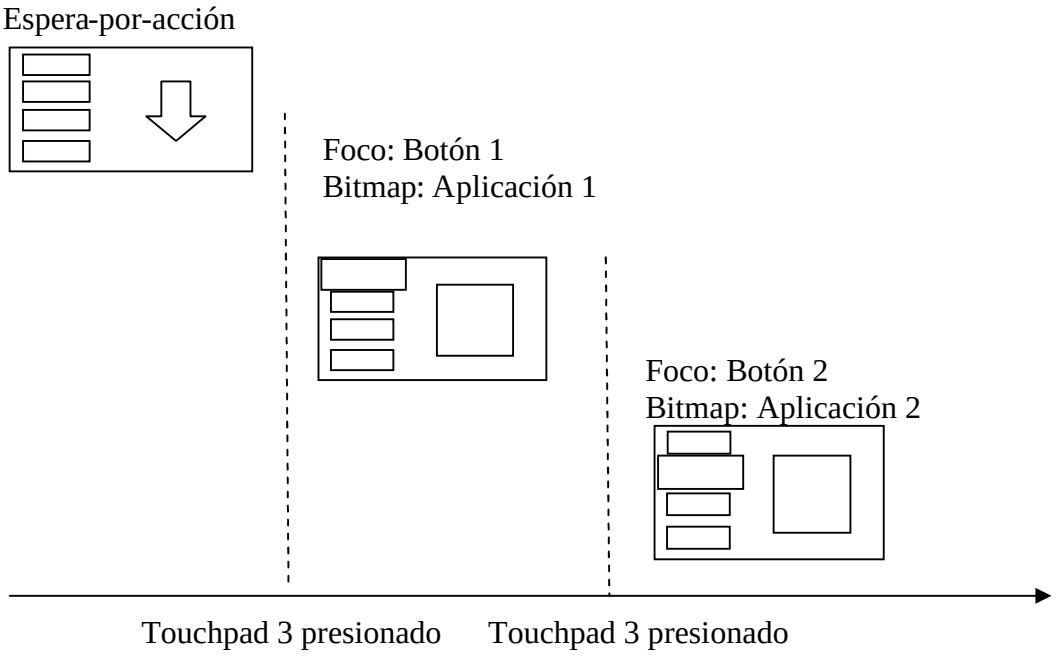


Figura. 3.30. Pantalla al presionar el Touchpad 3 dos veces

Antes de ejecutar la función se requiere verificar tres parámetros:

- Evento = EVENT_KEYSCAN
- Tipo = TYPE_KEYBOARD
- param1 = ID_TOUCH_PAD

Se debe determinar que botón fue accionado[30] para según eso saber a que acción desea ejecutar el usuario. Las posibles acciones pueden ser:

- SCAN_UP_PRESSED
- SCAN_DOWN_PRESSED
- SCAN_RIGHT_PRESSED
- SCAN_LEFT_PRESSED
- SCAN_UP_RELEASED
- SCAN_DOWN_RELEASED
- SCAN_RIGHT_RELEASED
- SCAN_LEFT_RELEASED
- SCAN_CR_PRESSED
- SCAN_CR_RELEASED

No se tomarán los casos _RELEASED, solo los casos _PRESSED ya que no son indispensables para el funcionamiento. Usando un switch case se evalúa el parámetro param2. En cada caso se debe seguir un flujo para determinar que botón ha sido presionado y que acción tomar de acuerdo a ese botón. Antes de proceder al diagrama de flujo de la función, hay que tener en cuenta que la pantalla del menú principal consta de 8 botones, 4 grandes (impresos cuando la opción es seleccionada) y cuatro normales (impresos cuando la opción no es seleccionada); debido a esto cuando se presione un botón de navegación se debe imprimir el botón grande de la nueva opción seleccionada, borrar el botón grande anterior e imprimir el botón normal de la opción anteriormente seleccionada. Teniendo en cuenta esto, el diagrama de flujo de la función es detallado en la Figura 3.27. El flujo con estas consideraciones es:

[30] **Por accionado entiéndase presionado o soltado**

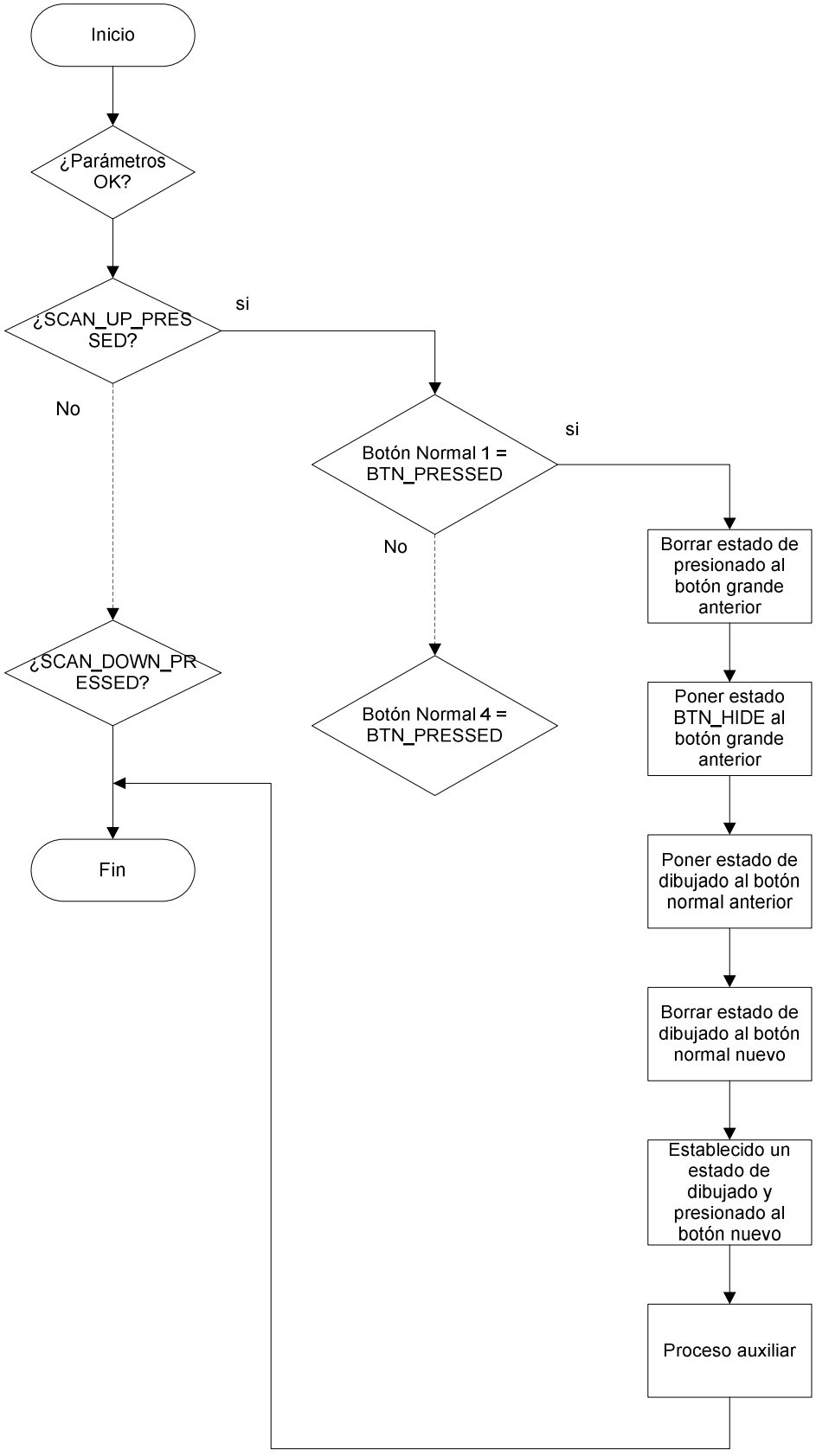


Figura. 3.31. Flujo de la función TraducirMensajeGenerico

La función TraducirTouchpad fue detallada en la tabla 3.21. Para el estado de pantalla MENU_PRINCIPAL, esta función llama a la función TraducirPantallaGenerica.

Función	Parámetros	Observaciones
`void TraducirPantallaGenerica( GOL_MSG* msg )`	GOL_MSG* msg: La información del mensaje actual	Ninguna
Overview: Esta función se encarga de la traducción del mensaje producido cuando se encuentra impresa la pantalla del menú principal para determinar que opción se ha escogido.		

Tabla. 3.27. Función TraducirPantallaGenerica

3.1.2.2. Aplicación secundaria: Gráficos. Esta aplicación persigue dos objetivos. El primero es demostrar la capacidad de la API para generar una interfaz gráfica poderosa, de alta calidad y muy completa. La segunda es demostrar la capacidad de procesamiento del PIC.

Los estados que corresponde a esta aplicación son:

- PANTALLA_JUEGO
- PANTALLA_MENU_JUEGO
- PANTALLA_INTRODUCCION_JUEGO
- PANTALLA_INICIAR_JUEGO
- PANTALLA_RECTA
- PANTALLA_JUGAR_RECTA
- PANTALLA_CURVA_IZQUIERDA
- PANTALLA_JUGAR_CURVA_IZQUIERDA
- PANTALLA_CURVA_DERECHA
- PANTALLA_JUGAR_CURVA_DERECHA
- PANTALLA_INSTRUCCIONES_JUEGO
- PANTALLA_INSTRUCCIONES

- PANTALLA_FIN_JUEGO
- PANTALLA_FIN_JUEGO_ESPERA
- PANTALLA_MENU_GRAFICOS
- MENU_GRAFICOS
- PANTALLA_TIEMPO_REAL
- TIEMPO_REAL

Todas las funciones contenidas en estos estados son programadas en el archivo juego.c y sus variables y más declaraciones en juego.h.

PANTALLA_JUEGO: Función GOLDrawCallback

Este estado solo posee una función llamada PantallaMenuJuego() y se encarga de limpiar la pantalla y crear los objetos para el menú del juego. El menú incluye:

- Botón para instrucciones
- Botón para jugar
- Imagen Bitmap

La función se limita a la creación de estos objetos. Luego de retornar de la función, la variable **estadopantalla** se iguala al siguiente estado PANTALLA_MENU_JUEGO.

PANTALLA_MENU_JUEGO: Función GOLMsgCallback

La función ProcesarMensajeMenuJuego() (tabla 3.28) se encarga de traducir la opción escogida por el usuario al seleccionar uno de los dos botones. Iguala **estadopantalla** de acuerdo al botón que se haya seleccionado a: PANTALLA_INSTRUCCIONES_JUEGO ó PANTALLA_INTRODUCCION_JUEGO.

Función	Parámetros	Observaciones
`WORD ProcesarMensajeMenuJuego( WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg )`	WORD translatedMsg: El mesaje traducido a nivel de control. OBJ_HEADER* pObj: Obejto para el que el mensaje se aplica GOL_MSG* pMsg: La informacion original del mensaje del sistema	Se debe llamar a PantallaMenuJuego primero. Esta función actualiza la variable **estadopantalla** para proceder hacia la selección del usuario
Overview: Esta función procesa el mensaje para el menú del juego, le permite escoger al usuario entre leer las instrucciones y jugar.		

Tabla. 3.28. Función ProcesarMensajeMenuJuego

PANTALLA_MENU_JUEGO: Función TraducirTouchpad

La función invocada en este estado por la función de traducción del Touchpad es **TraducirMensajeMenuJuego**. Esta función se encarga de la traducción del mensaje para el menú de juego que se encuentra impreso en pantalla, el cual presenta las opciones de "Instrucciones" y "Jugar". Esta función es muy parecida a la función TraducirPantallaGenerica ya que son de la misma naturaleza. La diferencia está los estados que iguala la variable estadopantalla al escoger las diferentes funciones.

Función	Parámetros	Observaciones
`void TraducirMensajeMenuJuego( GOL_MSG* msg )`	GOL_MSG* msg: La información del mensaje actual	Ninguna
Overview: Esta función se encarga de la traducción del mensaje producido cuando se encuentra impresa la pantalla del menú del juego para determinar si el usuario desea ver las instrucciones o jugar.		

Tabla. 3.29. Función TraducirMensajeMenuJuego

PANTALLA_INTRODUCCION_JUEGO[31]: Función GOLDrawCallback

Se llama a la función MostrarIntroduccionJuego, la cual se encarga de limpiar la pantalla, borrar la lista activa e imprimir una imagen creada por el diseñador del juego[32].

Ya que se está tratando el estado donde se muestra la introducción del juego, parece un buen momento para hacer una pequeña reseña sobre la aplicación secundaria: "juego". El juego fue creado para demostrar los dos objetivos: capacidad gráfica y poder de procesamiento. El tema que se escogió fue el de un juego de carreras de autos ya que los componentes de hardware cumplen ciertas ventajas, por ejemplo: la pantalla es rectangular pudiendo ofrecer una semejanza a la de un parabrisas y un mejor sentido panorámico y el potenciómetro se puede usar como volante ya que es parecido físicamente, haciéndolo girar se emula su uso. La inspiración se tomó del clásico juego de la consola Atari, el "Enduro". Más detalles se darán en los siguientes estados.

La imagen creada para la introducción trata de reflejar la temática del juego. El juego no es de una competencia con más autos, sino de tener la pericia para no chocar con los bordes de la carretera, es decir, es una competencia del conductor contra si mismo. Se diseño un nombre ficticio "Sunset Games" (Figura 3.32) para dar un entorno de juego más real. La imagen diseñada es:

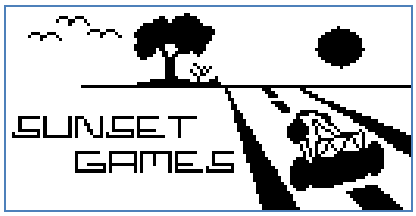


Figura. 3.32. Pantalla de Introducción del juego

La imagen se creó en el programa Paint, ya que es un mapa de bits monocromático.

Este estado solo se encarga de crear la imagen y enseguida igualar estadopantalla al siguiente estado.

[31] **PANTALLA_INTRODUCCION_JUEGO no lleva tilde en introducción por que en lenguaje C no se usan acentos.**
[32] **El juego y la imagen son creación inédita del autor de este proyecto de grado Luis Felipe Urresta Cueva**

Función	Parámetros	Observaciones
`void MostrarIntroduccionJuego( void)`	Ninguna	Ninguna
Overview: Esta función solamente se encarga de crear la imagen para la introducción.		

Tabla. 3. 30. Función MostrarIntroduccionJuego

PANTALLA_INICIAR_JUEGO: Función GOLDrawCallback

Cuando este estado cuando es llamado, la pantalla de introducción se encuentra impresa y se borrará instantáneamente si se ejecuta alguna acción, por eso, es preciso llamar a la función DelayMs (tabla 3.18) para permitir que la pantalla de introducción permanezca impresa lo suficiente para que pueda ser apreciada. El tiempo definido será 3500 milisegundos.

Luego de este período de espera, se llamará a la función CrearAutos() (tabla 3.31), encargada de la creación de los objetos para las imágenes Bitmap.

DESCRIPCIÓN DEL JUEGO

El procesador genera de manera aleatoria una de tres opciones:

- Una recta
- Curva a la derecha
- Curva a la izquierda

Y también genera aleatoriamente el tiempo que permanecerá en la opción escogida, de 1 a 4 segundos.

El conductor deberá girar el volante (potenciómetro) de acuerdo a la curva que esta en pantalla. Sí el tiempo de la curva o recta se vence y el jugador no tiene el auto en esa dirección, se restará una de sus vidas. Sí acierta, su puntaje se incrementa.

El jugador tiene 5 vidas. Cuando sus vidas lleguen a 0 ó cuando presione el botón 5 del Touchpad el juego terminará y se imprimirá la pantalla de fin del juego con el marcador del jugador.

Ahora, la función CrearAutos también se encarga de otras inicializaciones, como por ejemplo la inicialización de las variables creadas para manejar el juego en sí, estas variables son:

estadoDelAuto Es una variable que cambia de acuerdo a la acción del jugador, indica la dirección en la que el auto se encuentra, sus estados son:

- AUTO_YENDO_RECTO (inicializa en este estado)
- AUTO_CURVA_IZQUIERDA
- AUTO_CURVA_DERECHA

estadoAnteriorAuto Es una variable controlada por la generación aleatoria de la aplicación. Indica cual era el estado anterior del auto (mismos estados de la variable estadoDelAuto). Maneja los mismos estados que la variable estadoDelAuto.

estadoAnteriorPantalla Indica cual fue el último estado de la pantalla para controlar que no se repita la dirección. Es decir, que siempre se genere una dirección nueva para que no se ocasione el caso crítico de tener una misma curva impresa por largos periodos de tiempo, ya que esto haría que el juego se haga aburrido. Los posibles estados de la pantalla son:

- PANTALLA_JUGAR_RECTA (inicia en este estado)
- PANTALLA_JUGAR_CURVA_IZQUIERDA
- PANTALLA_JUGAR_CURVA_DERECHA

Dirección es la variable que maneja la función que genera números aleatoriamente. Puede tener 3 posibles valores:

- 0 = recta (inicia con este valor)
- 1 = izquierda
- 2 = derecha

Vidas es la variable que decrece con cada choque del jugador. Inicia en 5.

Puntaje es la variable que se incrementa cada vez que el jugador acierta. Inicia en 0.

Aparte de estas inicializaciones, esta función limpia la pantalla, borra la lista y crea tres objetos PICTURE con los bitmaps correspondientes al auto en las tres diferentes direcciones, como indica la Figura 3.33:

AutoCurvaIzquierda

AutoRecta

AutoCurvaDerecha

Figura. 3.33. Bitmaps de autos para el juego

Función	Parámetros	Observaciones
`void CrearAutos( void)`	Ninguna	Ninguna
Overview: Esta función solamente se encarga de crear los bitmaps para uso del juego y la inicialización de las variables también para uso del juego.		

Tabla. 3.31. Función CrearAutos

La siguiente tarea de este estado en la función callback de GOLDraw es iniciar el generador aleatorio de números. La función encargada de esta tarea es InitializeRandomNumberGenerator[33]. Esta función es solo una máscara para llamar a la función srand. Lo que sucede, es que antes de llamar a la función rand, que es la que se encarga propiamente de la generación, es necesario inicializar el motor de generación con llamando a la función srand. La función srand recibe un parámetro llamado semilla (RANDOM_SEED) para que la función rand pueda generar un número aleatorio. Una vez que se inicia, no es necesario volverla a llamar aun si la función rand se llama repetidamente. El valor semilla que será entregado será igual al segundo en el que se encuentre (_time_str[11]).

Función	Parámetros	Observaciones
`void InitializeRandomNumberGenerator ( void)`	Ninguna	Ninguna
Overview: Esta función inicializa el motor de generación de números aleatorios con la función srand. Tiene además un registro centinela para que solo se inicie una vez por ejecución del firmware.		

Tabla. 3.32. Función InitializeRandomNumberGenerator

La siguiente función que es llamada, es la que toma los valores de lectura del potenciómetro. El potenciómetro es conectado al primer canal de conversión ADC (revisar el diagrama esquemático de la tarjeta StarterKit PIC24F). Esta función se llama GraphReadPotentiometer (tabla 3.33). **Entrega el valor del potenciómetro en la variable potADC**.

[33] **Esta función esta en inglés porque es un modelo de uso generalizado para lenguaje C.**

Función	Parámetros	Observaciones
`void GraphReadPotentiometer( void )`	Ninguna	Ninguna
Overview: Esta función lee al potenciometro y almacena el valor leido en una variable global para que luego sea traducida como una acción del volante.		

Tabla. 3.33. Función GraphReadPotentiometer

La variable potADC y potADCAnterior son declaradas como **volatile WORD**. WORD es un tipo de variable compuesto por dos bytes y volatile es una característica que se puede dar una variable para indicar que su valor puede cambiar en cualquier momento de una manera brusca.

Ya que la función GraphReadPotentiometer ha sido llamada, se tiene la lectura del potenciómetro en su posición actual. Para tener una referencia de la dirección a donde se gira el potenciómetro, es necesario compararlo con su valor anterior. Por esta razón, se lee el valor del potenciómetro al iniciar la aplicación y se la iguala al valor anterior: potADCAnterior = potADC para cuando el jugador gire se tenga la referencia para calcular si se movió hacia la derecha o izquierda (Figura 3.34).

La carretera consta de dos partes: los bordes y la línea central entrecortada.

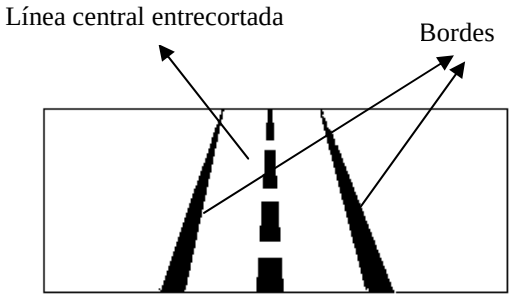


Figura. 3.34. Carretera en recta

Las líneas centrales entrecortadas para la recta, la curva a la derecha y la curva a la izquierda deben ser iniciadas con tres funciones individuales para cada dirección. Estas funciones proceden de la siguiente manera:

1. Limpian la pantalla y borran la lista de objetos activa
2. Ponen el fondo de la pantalla en negro
3. Dibujan la línea central entrecortada sin los cortes, la dibujan **sólida**
4. Guardan en un arreglo la información de un bloque de pixeles que contiene a la línea dibujada

Los pasos 3 y 4 serán entendidos de mejor manera en el momento que se analice la función AnimarJuego.

Al iniciar este estado, tres funciones son llamadas para preparar el Juego, estas son: IniciarCurvaIzquierda (Tabla 3.35), IniciarRecta (Tabla 3.36) e IniciarCurvaDerecha (Tabla 3.37).

Función	Parámetros	Observaciones
`void IniciarCurvaIzquierda ( void )`	Ninguna	Ninguna
Overview: Esta función crea la línea central de la carretera para una curva a la izquierda y guarda los datos del color de cada pixel en un arreglo. Además imprime en pantalla el número 3 para una cuenta regresiva a iniciar el juego.		

Tabla. 3.35. Función IniciarCurvaIzquierda

Función	Parámetros	Observaciones
`void IniciarRecta ( void )`	Ninguna	Ninguna
Overview: Esta función crea la línea central de la carretera para una recta y guarda los datos del color de cada pixel en un arreglo. Además imprime en pantalla el número 2 para una cuenta regresiva a iniciar el juego.		

Tabla. 3.36. Función IniciarRecta

Función	Parámetros	Observaciones
`void IniciarCurvaDerecha ( void )`	Ninguna	Ninguna
Overview: Esta función crea la línea central de la carretera para una curva a la derecha y guarda los datos del color de cada pixel en un arreglo. Además imprime en pantalla el número 1 para una cuenta regresiva a iniciar el juego.		

Tabla. 3.37. Función IniciarCurva Derecha

Luego de cada una de estas funciones se llama a la función DelayMs para dar un retardo de 600 ms, ya que se pretende que el jugador sepa en el momento exacto que va a empezar el juego para que no pierda una vida a causa de un inicio brusco. Por esa razón también, se inicia todas las variables en la recta para que el primer estado tanto del auto como de la carretera sea en una recta.

Los siguientes estados:

- PANTALLA_RECTA
- PANTALLA_JUGAR_RECTA
- PANTALLA_CURVA_IZQUIERDA
- PANTALLA_JUGAR_CURVA_IZQUIERDA
- PANTALLA_CURVA_DERECHA
- PANTALLA_JUGAR_CURVA_DERECHA

Son los estados en los que se desarrolla la aplicación en sí. Los estados PANTALLA_RECTA, PANTALLA_CURVA_IZQUIERDA y PANTALLA_CURVA_DERECHA se encargan de graficar la carretera y los autos al ser llamados por la función GOLDrawCallback y solo son llamados cuando hay un cambio de dirección.

Los estados PANTALLA_JUGAR_RECTA, PANTALLA_ JUGAR_CURVA_IZQUIERDA y PANTALLA_ JUGAR_CURVA_DERECHA son los encargados de crear la ilusión de movimiento del auto y traducir las órdenes que el usuario envía a través del potenciómetro mediante las funciones AnimarJuego y Conducir. Finalmente se genera una nueva dirección con la función GenerarNuevaDireccion.

Se tratará primero los tres estados primeramente mencionados, luego los estados _JUGAR.

PANTALLA_RECTA: Función GOLDrawCallback

En este estado se hace la llamada a la función MostrarPantallaRecta() (tabla 3.38).

Función	Parámetros	Observaciones
`void` **`MostrarPantallaRecta`** `( void )`	Ninguna	Ninguna
Overview: Esta función crea los bordes de la carretera en una recta, imprime las vidas y el puntaje.		

Tabla. 3.38. Función MostrarPantallaRecta

PANTALLA_CURVA_IZQUIERA: Función GOLDrawCallback

En este estado se hace la llamada a la función MostrarPantallaCurvaIzquierda.

Función	Parámetros	Observaciones
`void MostrarPantallaCurvaIzquierda ( void )`	Ninguna	Ninguna
Overview: Esta función crea los bordes de la carretera en una curva hacia la izquierda, imprime las vidas y el puntaje.		

Tabla. 3.39. Función MostrarPantallaCurvaIzquierda

PANTALLA_CURVA_DERECHA: Función GOLDrawCallback

En este estado se hace la llamada a la función MostrarPantallaCurvaDerecha.

Función	Parámetros	Observaciones
`void MostrarPantallaCurvaDerecha ( void )`	Ninguna	Ninguna
Overview: Esta función crea los bordes de la carretera en una curva hacia la derecha, imprime las vidas y el puntaje.		

Tabla. 3.40. Función MostrarPantallaRecta

Por esta ocasión se tratarán los tres estados como uno ya que los tres no difieren en su programación, todos llaman a las mismas funciones.

PANTALLA_JUGAR_RECTA, PANTALLA_JUGAR_CURVA_IZQUIERDA Y PANTALLA_JUGAR_CURVA_DERECHA: Función GOLDrawCallback

Lo primero que deben hacer estas tres funciones es llamar a la función RTCCProcessEvents para actualizar el tiempo, ya que se requerirá el uso de _time_str[11].

Luego se llamará la función **AnimarJuego**. Esta función es la que se encarga de animar la pantalla para que parezca que el auto se está moviendo. Deben analizarse que elementos podrían usarse para simular que el auto va hacia delante, podría usarse el auto, los bordes o la línea central. En caso de usar el auto tendría que simularse que las llantas se mueven, pero al ser tan pequeñas no se apreciaría con mucha facilidad. Sí se usaran los bordes seria un poco irreal el movimiento ya que cuando un pasajero va por la carretera en una recta o curva no hay una variación apreciable en los bordes porque son sólidos y del mismo color. Sí se usa la línea media se podría dar una noción de velocidad al hacer parecer que cada retazo de la línea se dirige hacia el auto. Entonces la simulación del movimiento se hará de la siguiente manera:

1. Se imprimen los bordes con las funciones PANTALLA_RECTA, PANTALLA_CURVA_IZQUIERDA o PANTALLA_CURVA_DERECHA dependiendo del caso.

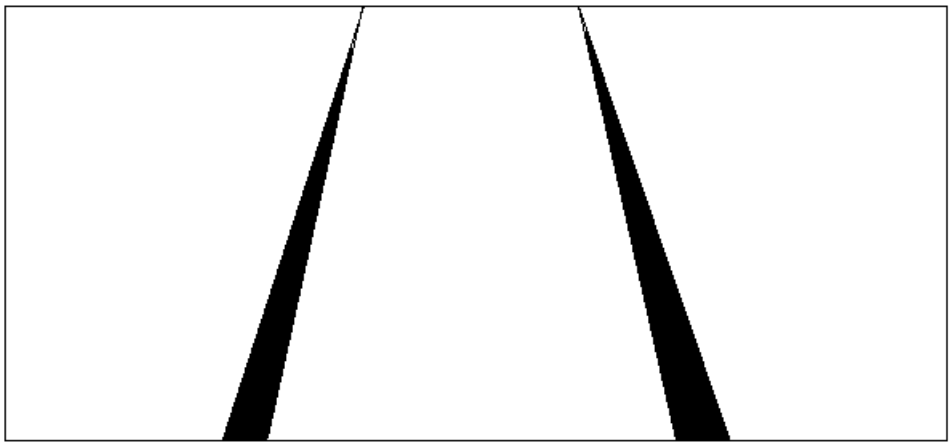


Figura. 3.35. Bordes de carretera en recta

2. Se imprimirá la línea central pero **sólida, no entre cortada.**

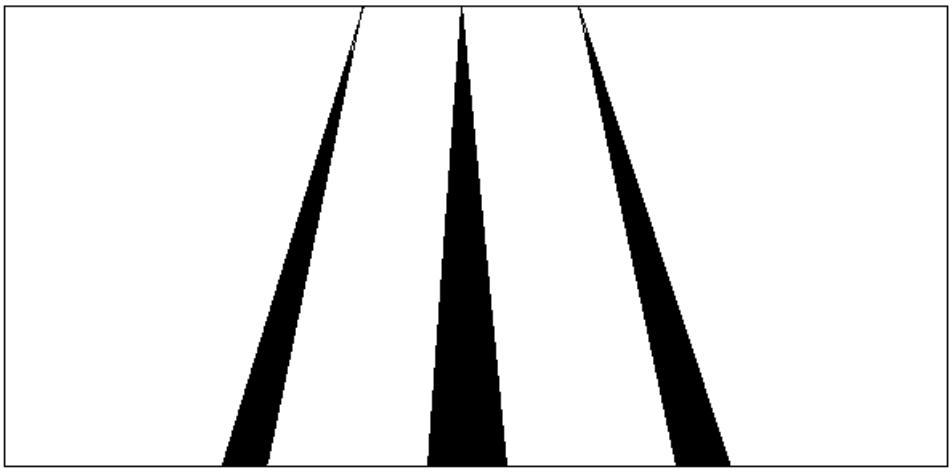


Figura. 3.36. Bordes y línea central solida

3. Luego, se crearán tres bloques de pixeles de color a lo largo de la línea media sólida que no corten los bordes.

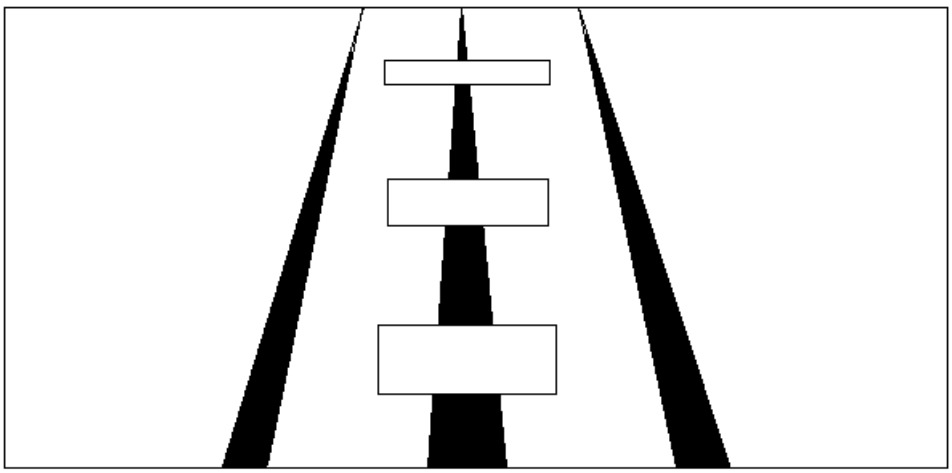


Figura. 3.37. Bordes, línea media y bloques

Los bloques de la Figura 3.37 están graficados con un borde negro que no se crea en la pantalla en la aplicación, solo se lo graficó para tener una noción de su dimensión. Cada uno de los tres bloques tiene una dimensión de altura diferente. El más alto esta más cerca y el más bajo más lejos para dar una noción de distancia al jugador, para que el retazo de la línea entrecortada de la parte de más arriba de la pantalla parezca estar más lejos que el retazo grueso de la línea media de la parte inferior de la pantalla. Sin los marcos la imagen queda como se aprecia en la Figura 3.38:

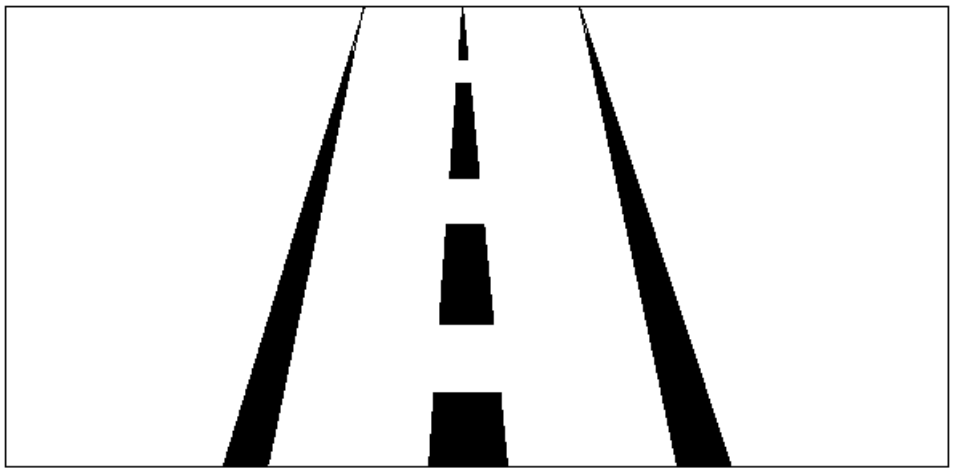


Figura. 3.38. Carretera con línea entrecortada media

De esta manera se cumple el tener una línea central entrecortada. Ahora, la pregunta que se tiene en frente es ¿Cómo generar movimiento? La solución de crear los tres bloques es la más óptima y creativa, ya que al crearlos como bloques se los puede mover.

Moviendo los tres bloques a la vez una misma distancia hacia la parte inferior de la pantalla se puede simular movimiento. Ahora, los bloques no son objetos como tal, son generados por dos estructuras *for* usados según el método de la burbuja para graficar un bloque de pixeles negros. Para “moverlos” (Figura 3.39) se deben ejecutar dos acciones:

1. Imprimiendo una nueva línea de pixeles negros luego de la ultima línea del bloque
2. Reimprimiendo la primera línea del bloque con el color original de cada pixel

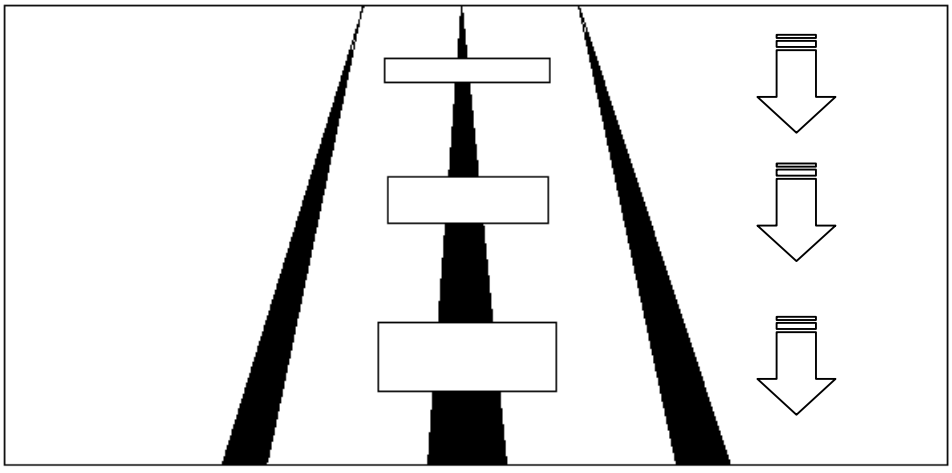


Figura. 3.39. Bloques de pixeles construidos para la animación

El siguiente problema que se afronta es ¿Cómo saber que color era la línea original, la línea antes de que se grafiquen los bloques de pixeles negros? Para esto se creo anticipadamente un respaldo en las funciones IniciarRecta() (tabla 3.35), IniciarCurvaIzquierda() (tabla 3.36) e IniciarCurvaDerecha() (tabla 3.37) se creó un arreglo de datos donde se guardó la información del color de cada pixel. El arreglo contiene los datos del color de los pixeles contenidos en el recuadro azul de la Figura 3.40

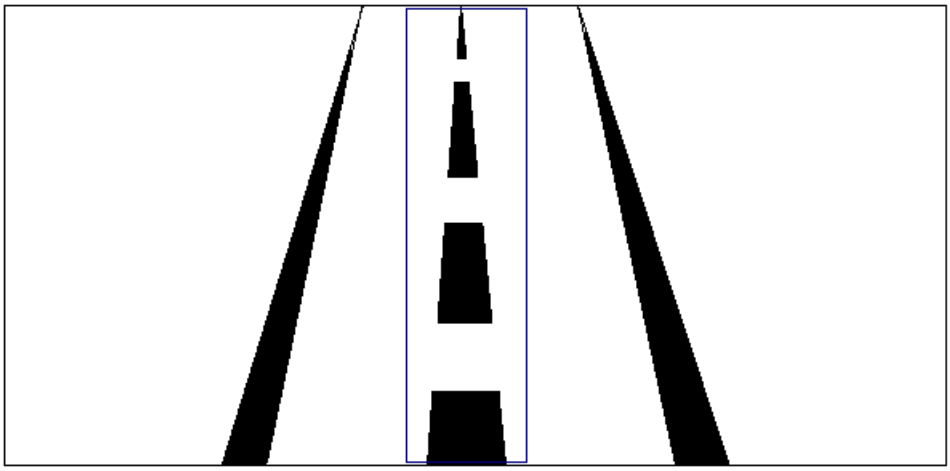


Figura. 3.40. Arreglo con información de color de pixeles

Con esta información, la resolución del paso 2 del movimiento de los bloques se reduce a buscar la información del color original de los pixeles en el arreglo, tomarla e imprimir un nuevo pixel con ese color.

La función AnimarJuego() solo es una administradora que determina la dirección que se requiere animar: recta, curva a la derecha o curva a la izquierda. Dependiendo de la variable “dirección” llama a la función que anime la pantalla para cada dirección:

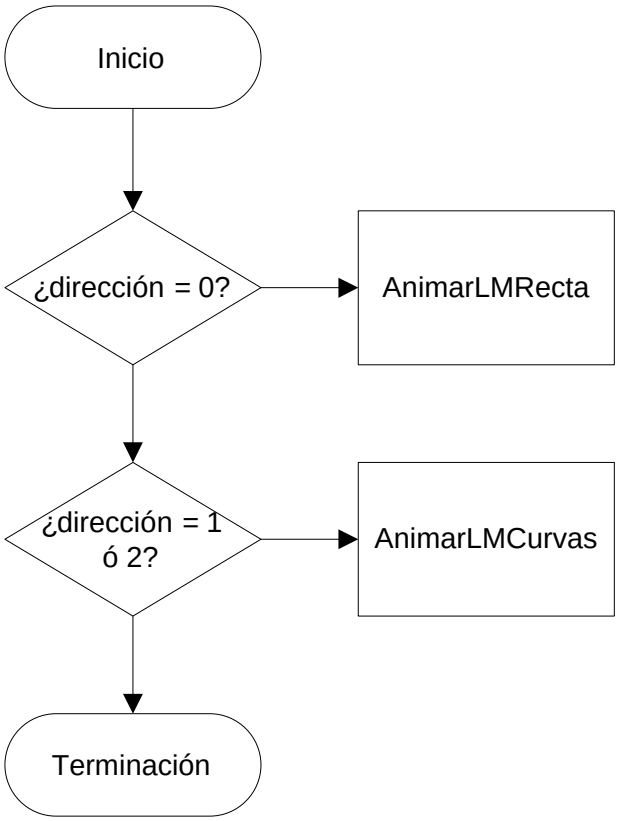


Figura. 3.41. Flujo función AnimarJuego

Las funciones AnimarLMRecta y AnimarLMCurvas son las encargadas de realizar la animación. El estudio se ha centrado en la dirección recta y seguirá siendo así, al final del estudio de estos diagramas de estado se hará mención a las diferencias con las funciones de las curvas. La función AnimarLMCurvas es la misma sin importar si la curva va hacia la derecha o hacia la izquierda, esto se explicará más adelante.

La función AnimarLMRecta ejecuta todos los pasos que se han estudiado hasta el momento para generar la línea central sólida, los bloques negros de pixeles y moverlos. Una observación que esta ya resuelta en la función, sin embargo no se ha hecho hasta el momento, es que una vez que la primera línea del tercer bloque llega al fin de la pantalla debe rehacerse al inicio. Con todas las soluciones y las observaciones integradas el diagrama de flujo será:

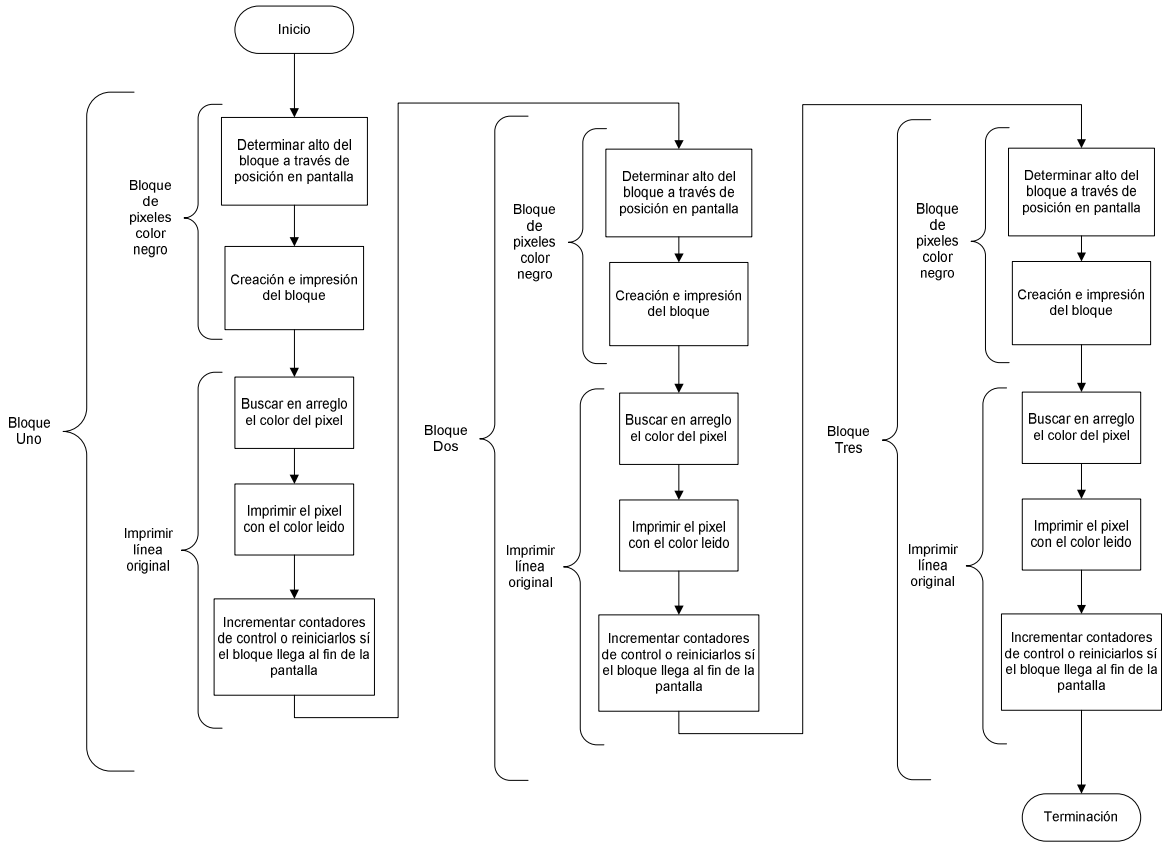


Figura. 3.42. Flujo de la función AnimarLMRecta

Antes de empezar a describir la función Conducir, es momento de hablar sobre las diferencias para la dirección 0 = recta y dirección 1 = curva izquierda o dirección 2 = curva derecha.

CONSTRUCCION DE LAS PANTALLAS

Recuérdese, que la pantalla OLED es de 128x64 pixeles, es decir la pantalla tiene una relación de 2x1 en su ancho por alto.

La **construcción de la perspectiva** para el juego se basa en los principios básicos del dibujo técnico.

- **El punto de vista normal (LH):** el nivel de los ojos del espectador está directamente relacionado con la línea del horizonte de la imagen. Cuando el espectador mira el plano, la *línea de horizonte* debe estar a la altura de sus ojos.
- **El punto de fuga (PF):** en la perspectiva "ideal" de un solo punto, la perspectiva lineal divide el espacio en un cubo, con el suelo cuadriculado (Figura 3.43), paredes y techo. Todas las paralelas (recesivas) que van disminuyendo (baldosas, parte superior de la mesa, extremos superiores e inferiores de la ventana) convergen hacia el punto de fuga central (PF) situado en la línea horizontal (LH)[34].

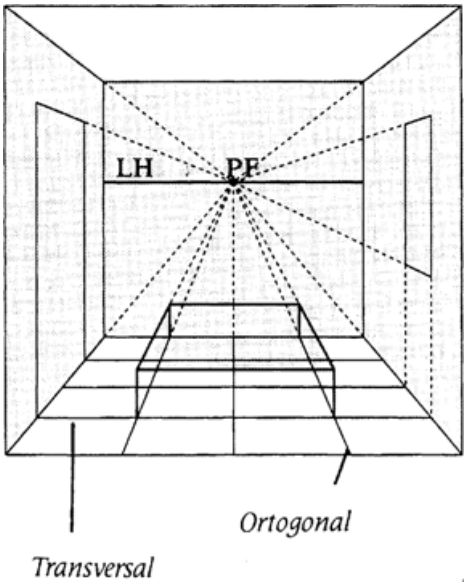


Figura. 3.43. Construcción de la perspectiva

La colocación de la línea horizontal será en parte superior de la pantalla, la línea que corresponde a x = 0. El punto de fuga será en la mitad de la LH en el punto x = 0, y = 64. En el punto de fuga se ubicará la línea media de la carretera, los bordes se ubicarán equidistantes de esta a una distancia de 1/8 del ancho total de la pantalla (16 pixeles). De esta manera el ancho de la carretera sobre la LH será de ¼ de la pantalla. En la parte inferior (x = 64) el ancho de la carretera corresponderá a ½ del ancho (64 pixeles). Es decir que la línea del borde tendrá un ángulo de:

[34] **http://www.ite.educacion.es/w3/eos/MaterialesEducativos/bachillerato/arte/arte/pintura/perspe-3.htm, La construcción de la perspectiva**

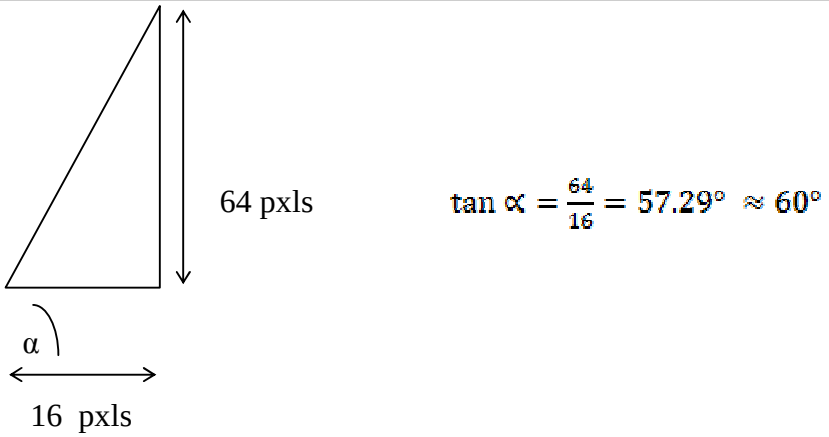
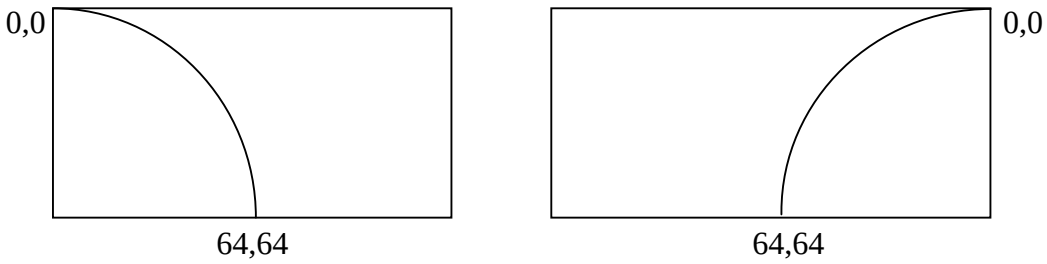


Figura. 3.44. Graficar la línea del borde en recta

Se cumple entonces con el ángulo de 60 grados sexagesimales recomendados para una perspectiva ortogonal.

Para las pantallas con curva a la derecha o izquierda, el punto de fuga se traslada sobre la línea horizontal a (0,0) para curva izquierda y (128,0) para curva a la derecha. Para graficar los bordes de manera curva no se uso la función Line, porque ésta solo sirve para graficar líneas rectas. Tampoco se usó la función Arc para graficar arcos porque no daban una imagen real de perspectiva. Así que se opto por usar la función PutPixel dentro de una función que genere una parábola usando bucles for.



$desde\ y = 0\ hasta\ y = 64$:

$$x = \sqrt{2 \times p \times y}$$

$donde\ p = foco\ de\ la\ parábola$

Figura. 3.45. Graficar la línea del borde en curva

El grueso de las funciones Line y PutPixel, es de un pixel por línea. Es por eso que se grafican más líneas aledañas a la línea original, pero dependiendo de en que posición vertical de la pantalla se encuentran. Si se encuentran en el primer 1/3 de la pantalla solo se agrega una línea a la izquierda y una a la derecha, en el segundo tercio se agrega una línea más a cada lado y en el último tercio otra más. Esta relación puede variar para dar una mejor perspectiva de acuerdo al caso.

El gráfico del auto también se hizo para dar la mejor perspectiva posible.

La función que se llama a continuación se llama Conducir (Tabla 3.41). Se encargará de traducir los mensajes del potenciómetro y expresarlos en movimientos del auto. El flujo que sigue esta función, primero evalúa sí el potenciómetro se movió a la derecha o a la izquierda, luego evalúa cual es el estado actual del auto con la variable estadoDelAuto. Esto lo hace para que si el auto se encuentra curvando a la izquierda y el potenciómetro se mueve hacia la derecha, el auto primero se imprima en el centro y luego a la derecha y no salte de un lado al otro.

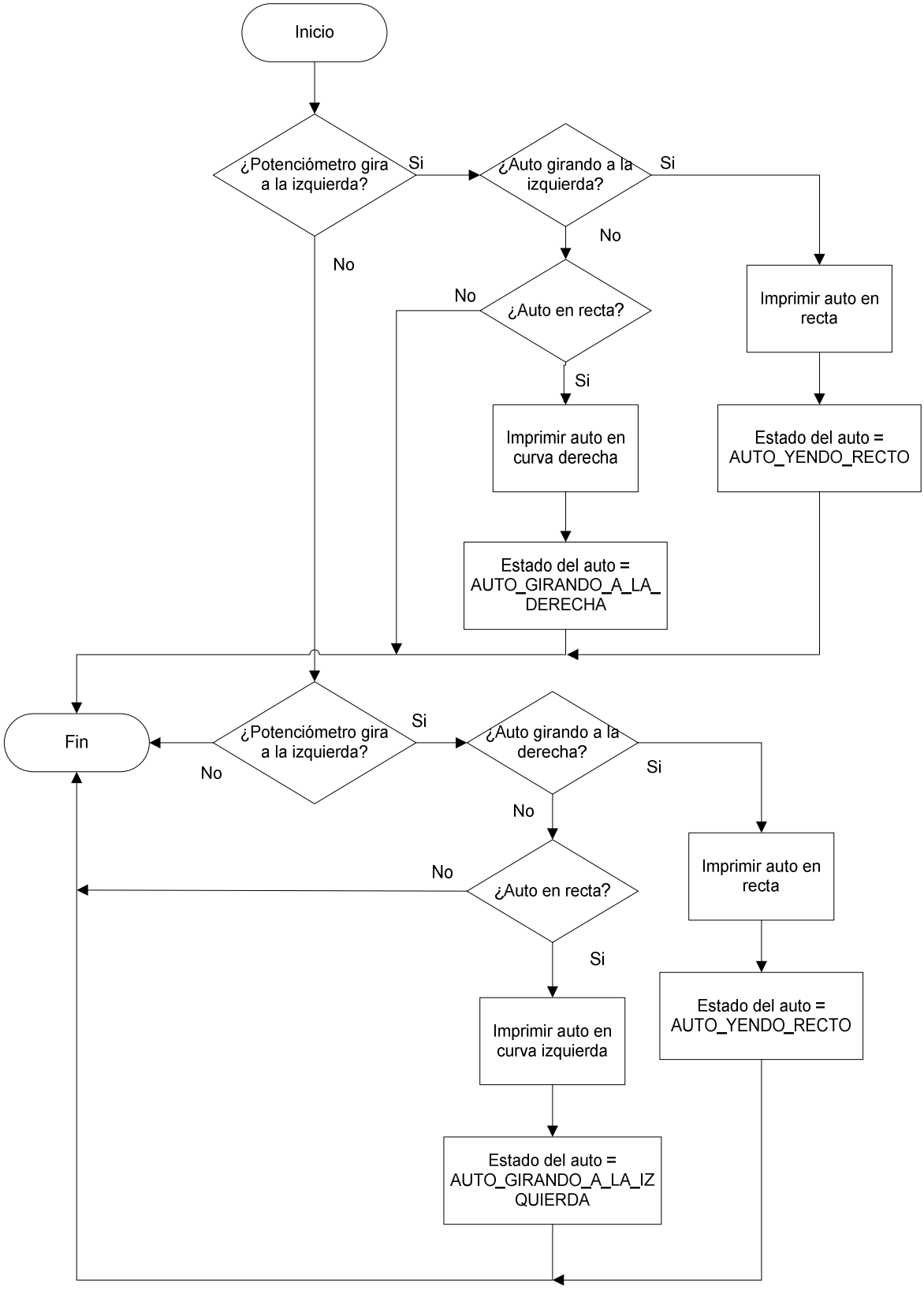


Figura. 3.46. Flujo función Conducir

Función	Parámetros	Observaciones
`void Conducir ( void )`	Ninguna	Ninguna
Overview: Esta función se encarga de la traducción de el giro del potenciómetro si se a la derecha o izquierda, de dibujar el auto según la nueva instrucción.		

Tabla. 3.41. Función Conducir

La siguiente y última función que se llama es la función GenerarNuevaDireccion. Esta función se encarga de las siguientes tareas:

- Genera aleatoriamente una nueva dirección
- Genera aleatoriamente el tiempo que se permanecerá en la nueva dirección
- Aumentar el puntaje del jugador
- Disminuir vidas las del jugador

La generación aleatoria de una nueva dirección se da cada vez que el tiempo definido para tal dirección se termine. Esto demanda un sistema de control que se explica de a continuación.

El tiempo que puede permanecer en una u otra dirección se ve limitado por la variable TiempoDeDireccion. Esta variable varía en un rango de 1 a 3 segundos y es generada por la función rand. El problema que se enfrenta ahora es ¿Cómo controlar que el tiempo se cumpla? Se pudiera intentar un contador, pero esto solo demandaría esfuerzo del procesador inútilmente. No se puede usar la función DelayMs porque el flujo del juego está pensado para un bucle infinito. La mejor solución es comparar la variable al tiempo controlado por la variable que se ha usado repetidamente, _time_str[11]. La siguiente inquietud es ¿Cómo verificar que el tiempo se cumpla? Por ejemplo, sí el tiempo de dirección es 3 segundos y el segundo entregado por _time_str[11] es igual a 31. La función esperará… 31…32… 33; es decir, 3 segundos. Pero si está en 32 esperará solo 2 segundos. Es por eso, que la variable TiempoDeDireccion no refleja exactamente los segundos que se

estará en este estado. Pero sí reflejará que el tiempo máximo que se estará un estado sea de 3 segundos en la mayoría de casos. En un caso único y crítico, _time_str[11] será igual a 58 donde se tendrá que esperar…58…59…0…1…2, es decir 5 segundos. La estadística nos enseña que las probabilidades de caer en este caso son 1/60 = 0.016667.

Cuando se cumpla la condición de que el segundo actual es múltiplo de la variable TiempoDeDireccion, se podrá acceder a la siguiente sección de código de la función GenerarNuevaDireccion encargada de aumentar el puntaje o restar una vida al jugador. La función GenerarNuevaDireccion es llamada por la función GOLDraw, eso significa que se repetirá infinitamente, aproximadamente cada 10 ms. Por esta razón es necesario que una vez que el segundo actual cambie, una bandera indique que la función ya se llamo para ese segundo, caso contrario se llamaría 100 veces a la función mientras dure el segundo actual. El diagrama de flujo queda indicado en la siguiente Figura.

PANTALLA_MENU_GRAFICOS: Función GOLDrawCallback

Aquí, se hace el llamado a la función MostrarPantallaMenuGraficos(), la cual se encarga de imprimir en pantalla dos botones para las opciones:

- Tiempo Real
- Juego

Función	Parámetros	Observaciones
`Void` **`MostrarPantallaMenuGraficos`** `( void )`	Ninguno	Ninguna
Overview: Esta función imprime en pantalla el menú de dos botones para el menú de gráficos que corresponden a la opción de Juego y de tiempo real		

Tabla. 3.42. Función MostrarPantallaMenuGraficos

MENU_GRAFICOS: Función GOLMsgCallback

En esta función se toma una acción de acuerdo al botón que el usuario haya escogido, la acción puede ser presentar la pantalla del menú del juego o la pantalla de tiempo real. Esto se hace mediante la función ProcesarMensajeMenuGraficos(translatedMsg, pObj, pMsg) mostrada en la tabla 3.43.

Función	Parámetros	Observaciones
`WORD` **`ProcesarMensajeMenuGraficos`** `( WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg )`	WORD translatedMsg: El mesaje traducido a nivel de control. OBJ_HEADER* pObj: Obejto para el que el mensaje se aplica GOL_MSG* pMsg: La informacion original del mensaje del sistema	Ninguna
Overview: Esta función procesa el mensaje del menú de gráficos entre sus dos opciones		

Tabla. 3.43. Función ProcesarMensajeMenuGraficos

MENU_GRAFICOS: Función TraducirTouchpad

En esta función se hace el llamado a TraducirMensajeMenuGraficos(pMsg) (tabla 3.43) para traducir el mensaje crudo que indica que botón ha sido escogido y presionado por el usuario.

Función	Parámetros	Observaciones
`WORD TraducirMensajeMenuGraficos(GOL_MSG pMsg )`	GOL_MSG *pMsg: puntero a mensaje actual	Ninguna
Overview: Esta función traduce el mensaje para el menú de gráficos.		

Tabla. 3.43. Función TraducirMensajeMenuGraficos

PANTALLA_TIEMPO_REAL: Función GOLDrawCallback

En esta función, dentro del estado PANTALLA_TIEMPO_REAL se ejecutan las siguientes acciones:

- Leer los datos de entrada del potenciómetro
- Almacenar el valor actual del potenciómetro
- Presentar la pantalla por primera vez si es el caso con la función MostrarPantallaTiempoReal
- Ir al estado TIEMPO_REAL

Función	Parámetros	Observaciones
`Void` MostrarPantallaTiempoReal (`void`)	Ninguno	Ninguna
Overview: Esta función imprime la primera pantalla de la aplicación de tiempo real tomando el valor inicial del potenciómetro tomado previamente con la función correspondiente revisada en la tabla 3.33.		

Tabla. 3.44. Función MostrarPantallaTiempoReal

TIEMPO_REAL: Función GOLDrawCallback

Actualizar la pantalla usando la función GraficarTiempoReal. Esta función se encarga de leer el valor actual del potenciómetro y llevarlo a una escala para saber a que valor corresponde en pixeles dentro de la pantalla.

Función	Parámetros	Observaciones
`Void GraficarTiempoReal ( void )`	Ninguno	Ninguna
Overview: Esta función actualiza la pantalla de acuerdo al valor del potenciómetro con una escala de valores para pixeles de la pantalla. 0kΩ -> pixel 64 y 10kΩ -> pixel 0.		

Tabla. 3.45. Función GraficarTiempoReal

3.1.2.3. Aplicación secundaria: USB. Esta aplicación descubre el uso de la librería USB MCHPFSUSB v2.4. Esta librería es descrita también como framework. Un framework es una estructura en software sobre la cual se programan aplicaciones, su diferencia con la librería de gráficos básicamente es que no genera una interfaz por capas a funciones más primitivas.

Ya que esta aplicación requiere una explicación más detallada y ciertas explicaciones teóricas, no se seguirá el mismo esquema que se uso para detallar la aplicación de gráficos. Se presentarán las ideas teóricas que se necesitan conforme a la presentación de la estructura del framework USB, luego se presentarán los diagramas de flujo y explicaciones de como funciona esta aplicación.

La página web oficial de microchip[35], la describe como "MCHPFSUB es un paquete de distribución que contiene una variedad de proyectos de firmware relacionados con USB para el PIC24F…" también la describe como "USB firmware framework para el PIC24F" que permite agregar fácil y rápidamente USB Device, Embbeded Host, Dual Role, o capacidad OTG a la aplicación del diseñador.

Esta librería, posee una herramienta de gran poder y extremadamente útil para el diseñador. Esta aplicación se llama **USBConfig.exe**.

Cada proyecto de firmware que requiera estas utilidades necesita un archivo de cabezera usb_config.h que define muchas macros que los stacks USB usan para saber como deben desarrollarse. En el caso de las aplicaciones USB embedded host existe también un archivo .c que necesita ser creado para describir la Targeted Peripheral List. La TPL es una lista de los dispositivos a los que se les da soporte. Este archivo .c también contiene varia información sobre que los stacks necesitan saber en orden para cargar y ejecutar los drives correctos para los dispositivos.

La herramienta USBConfig.exe es una interfaz fácil de usar para generar los archivos requeridos por los USB stacks.

[35] **www.microchip.com**

La herramienta USBConfiguration.exe es usada para configurar la funcionalidad USB Embedded host, y esa es la aplicación que se le dará en este proyecto. Además de esta funcionalidad, puede configurar:

- USB Configuration
- Embbeded Host Configuration
- OTG Configuration
- Targeted Peripheral List Configuration
- CDC Client Driver Configuration
- Generic Client Driver Configuration
- HID Client Driver Configuration
- Mass Storage Client Driver Configuration
- Printer Client Driver Configuration

Esta aplicación es implementada en una interfaz gráfica que administra todas estas aplicaciones en pestañas. En la parte inferior de la ventana presenta un botón “Generar” que genera los archivos usb_config.c y usb_config.h.

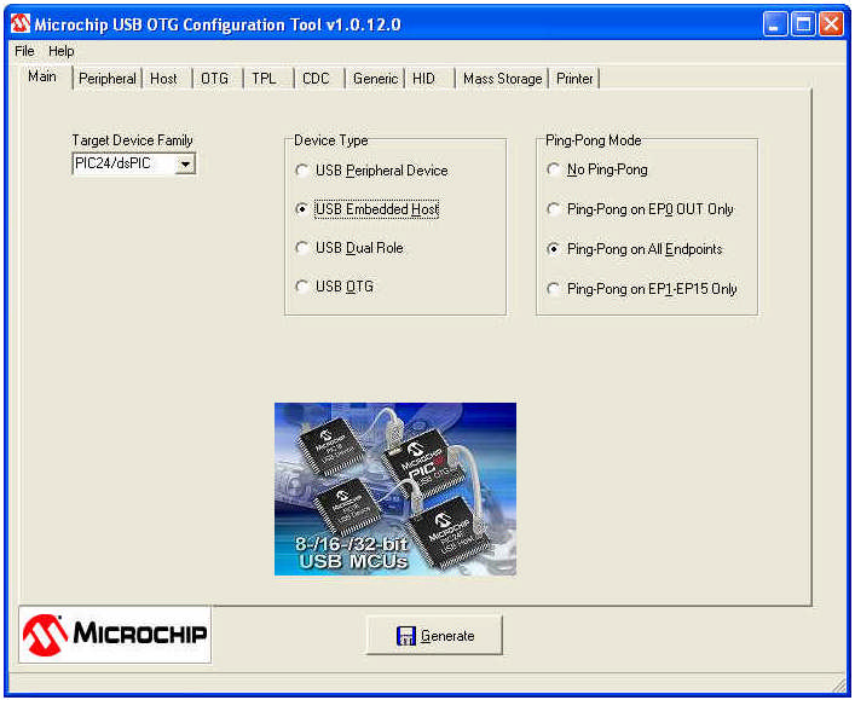


Figura. 3.47. Pantalla principal de USBConfig.exe

En esta sección se selecciona el tipo “USBEmbbeded” Host y el modo Ping-Pong “Ping-Pong on All Endpoints”, también se selecciona el dispositivo PIC24/dsPIC.

La siguiente pestaña que se debe configurar es la Host. El modo Host **debe** configurarse para la aplicación, no puede ser omitido.

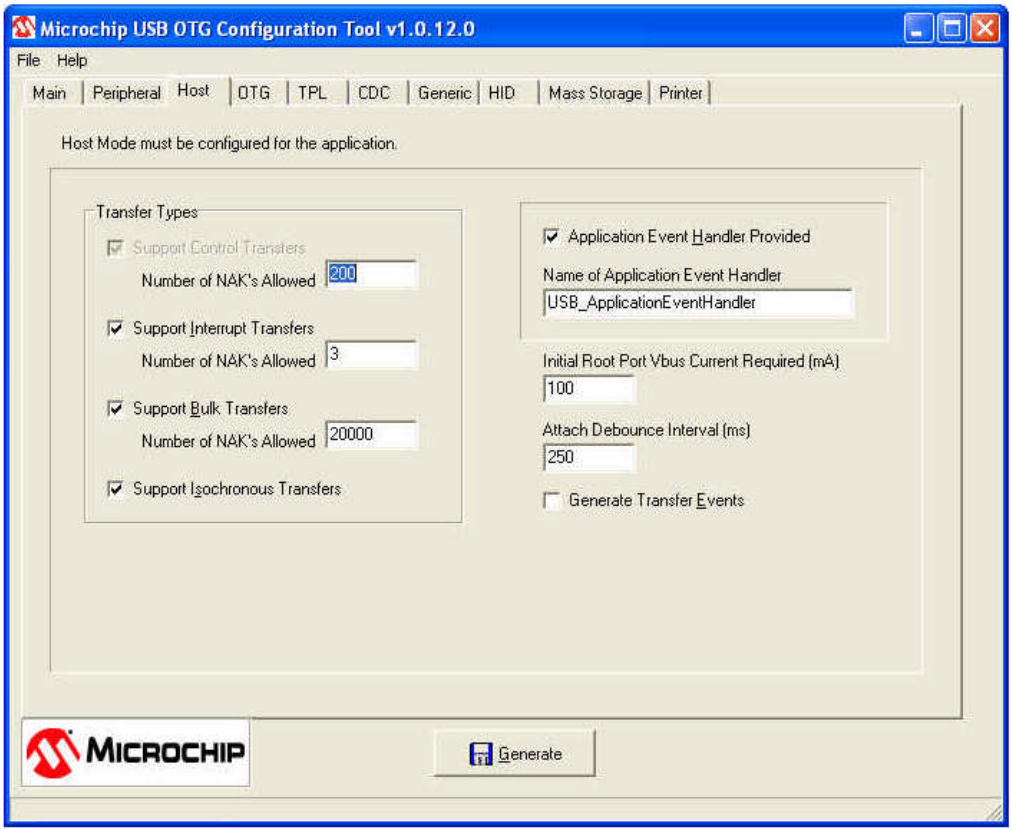


Figura. 3.48. Pestaña Host de USBConfig.exe[36]

La aplicación debe soportar ciertos dispositivos. Para este soporte, se debe habilitar los tipos de transferencia USB que manejarán estos dispositivos.

- **Control transfers:** siempre se debe soportar este tipo de transfer.
- **Interrupt trasnfers:** la interfaz clase HID y el driver del cliente genérico usa este tipo de transferencia. Este es un tipo de transfer común usado por dispositivos USB.

[36] **Los valores en este cuadro no son exactamente los valores que se uso en la aplicación**

- **Bulk Transfers:** la clase de almacenamiento masivo (mass storage) usa este tipo de transferencia. Es usado por dispositivos tales como Flash Drive USB.
- **Isochronous Transfers:** dispositivos multimedia generalmente usan estos tipos de transferencia.

El documento USB library Help da una referencia importante: *"**Sí usted no está seguro sí un tipo de transfer debe ser soportado, deje el soporte habilitado."***

Además, en esta pantalla se debe ingresar el número de NAKs[37] permitido para cada tipo de transfer antes que sea considerado como una falla. Control transfers usualmente tiene un bajo número de NAKs, porque dispositivos son requeridos para responder rápidamente al la mayoría de transfers de control. Interrupt transfers también usualmente tienen un bajo número de NAKs, ya que el dispositivo es frecuentemente sondeado en una base regular. (Note que una NAK en una interrupción IN, no resulta en un campo de transferencia. En vez de eso, el transfer es terminado satisfactoriamente con cero bytes recibidos, indicando que el periférico no tiene datos para el anfitrión. Bulk transfers, usualmente tienen un número muy alto de NAKs, ya que los dispositivos de almacenamiento pueden requerir un período relativamente largo de tiempo para ingresar al medio.

Si se desea que la aplicación reciba notificaciones en eventos de la stack USB, incluyendo requerimientos de energía, se debe revisar el **Application Event Handler Provided** e ingresar el nombre de la función en C, que implementa el handler de eventos.

El stack requiere que la alimentación sea aplicada a VBUS mediante el evento EVENT_VBUS_REQUEST_POWER. Ingrésese la corriente que la aplicación pueda soportar durante la enumeración en **Initial Vbus Current Required (mA)**.

Las especificaciones USB requieren que los dispositivos pueden estar listos para comunicaciones dentro de 100ms luego de ser colocada en el conector del bus.

[37] NAK se refiere a Not Acknowledge (no reconocido), se produce cuando un paquete no envía una confirmación de ser recibido

Las **USB Flash Drive** se ha popularizado, tomando el mercado de una manera que nunca antes un dispositivo de almacenamiento lo había hecho. Son baratas, rápidas y fáciles de usar. En la actualidad vienen en una variedad de figuras, tamaños y capacidad de almacenamiento. La mayoría de USB Flash Drives, con capacidad de hasta 2GB, utilizan el sistema de archivos FAT16[38] y un comando de interfaces llamado Small Computer System Interface (SCSI). Las Flash Drives con capacidad mayor a 2 GB usan el sistema de archivos FAT32, el cual no es soportado por los archivos de esta librería[39].

Ahora, es momento de entrar a describir rápidamente la arquitectura de la aplicación USB. No se entra en mayor detalle de las capas porque no es un tema que corresponda a este proyecto al haber sido delimitado así en la denuncia del mismo.

Estas capas en su mayoría son transparentes al programador y no es necesario su conocimiento para un correcto uso de la librería, esto tal como lo estipula el documento Application Note 1145A Using a USB Flash Drive on an Embbeded Host: "***Note que no es necesario estar familiarizado con la operación o configuración de la interfaz USB en orden de usar este framework…***[40]"

Entrando al estudio de la arquitectura de la aplicación USB, se tiene el cuadro presentado en el documento AN1145A que explica gráficamente las capas presentes en la aplicación, como se ven en la Figura 3.49:

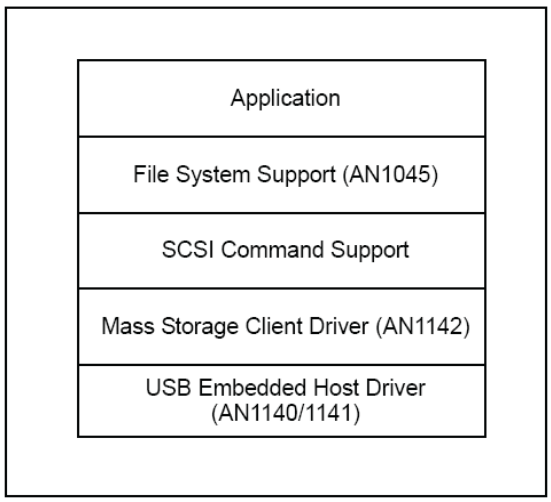


Figura. 3.49. Capas de la aplicación USB

[38] **FAT se refiere a File Allocation Table**
[39] **AN1145A, Nota sobre USB Flash Drives**
[40] **AN1145A, página 2, application architecture**

La capa **USB EMBEDDED HOST DRIVER** provee soporte genérico para anfitriones USB embebidos. La interfaz para esta capa es provisto automáticamente en el client driver de mass storage (almacenamiento masivo).

La capa **MASS STORAGE CLIENT DRIVER FOR USB EMBEDDED HOST** está descrita en el documento AN1145A como "*La capa que provee al client driver para la clase mass storage, la cual es requerida para interfaz con dispositivos de almacenamiento masivo, tales como las USB Flash Drive*".

La capa **FILE SYSTEM SCSI COMMAND SUPPORT** está provista por la librería de sistema de archivos descrita en la AN1145. En verdad, este documento no trata de ahondar en temas demasiado específicos, pero si se desea hacer una investigación más exhaustiva, se recomienda la lectura de la AN1145 y las funciones de low-level. Para información más detallada acerca de la API de interface USB Embedded Host Mass Storage SCSI, refiérase a la documentación provista en la ayuda de la librería[41].

A continuación, se presentará una tabla tomada del documento AN1145A indicando los archivos .c y .h que corresponden a cada capa. No todos los archivos son usados, pero si es necesario hacer mención de ellos ya que podrían ser útiles en el desarrollo de alguna otra aplicación. La Tabla 3.42 indica los archivos que residen en esta librería.

[41] **AN1145A, página 2, Application Architecture.**

Layer	File Name	Description
USB Embedded Host Driver	usb_host.c	Provides USB embedded host support for all devices. Does not provide class support.
	usb_host.h	Header file with definitions required for USB embedded hosts. Defines the interface to the USB embedded host driver.
	USBCore.h	Header file with definitions common to both USB embedded hosts and USB peripherals
Mass Storage Client Driver	usb_host_msd.c	Provides mass storage class support for a USB embedded host.
	usb_host_msd.h	Header file with definitions for USB embedded hosts supporting the mass storage class. Defines the interface to the mass storage client driver.
SCSI Command Support	usb_host_msd_scsi.c	Provides SCSI command support for a USB embedded host utilizing the mass storage client driver.
	usb_host_msd_scsi.h	Header file with definitions for USB embedded hosts using the mass storage client driver and SCSI commands. This file also includes definitions to link the AN1045 function name requirements to this implementation.
File System Support (installed as a part of AN1045)	FSIO.c	Provides simple commands to perform file activities, such as open a file, read from a file, write to a file, close a file, etc.
	FSIO.h	Header file with prototypes for the file system functions.
	FSDefs.h	Header file with constants and data structures required by the file system functions.
Application	uart2.c	Provides an interface to UART2 to provide RS-232 input and output to the application.
	uart2.h	Header file for the UART2 functions.
	FSConfig.h	Configures the file system library for this application.
	usb_config.c	Configures the USB Stack for this application. Generated by the configuration tool.
	usb_config.h	Configures the USB Stack for this application. Generated by the configuration tool.
	system.h	Contains system level constants for libraries to reference.
	usb_data_logger.c	Main application code.

Tabla. 3.42. Archivos usados en aplicaciones USB

El tema del proyecto incluye las palabras “manejo de archivos”. Según el documento, AN1145A, que se ha mencionado repetidamente en esta última sección, el manejo de archivos incluye:

- Ver los archivos en el directorio actual
- Crear y remover directorios
- Cambiar el directorio actual
- Crear archivos
- Eliminar archivos

El capítulo cuarto de este proyecto, en la sección 4.1.4 hay un tutorial sobre la librería MCHPSUSB que incluye el uso de las funciones FS que son responsables del manejo de archivos.

Entonces, el estudio de las funciones FS que permiten las operaciones “crudas” en la memoria flash serán estudiadas en el tutorial de la librería y en esta sección se estudiará la estructura la aplicación en el firmware.

La aplicación se desarrollo según indica la Figura 3.50:

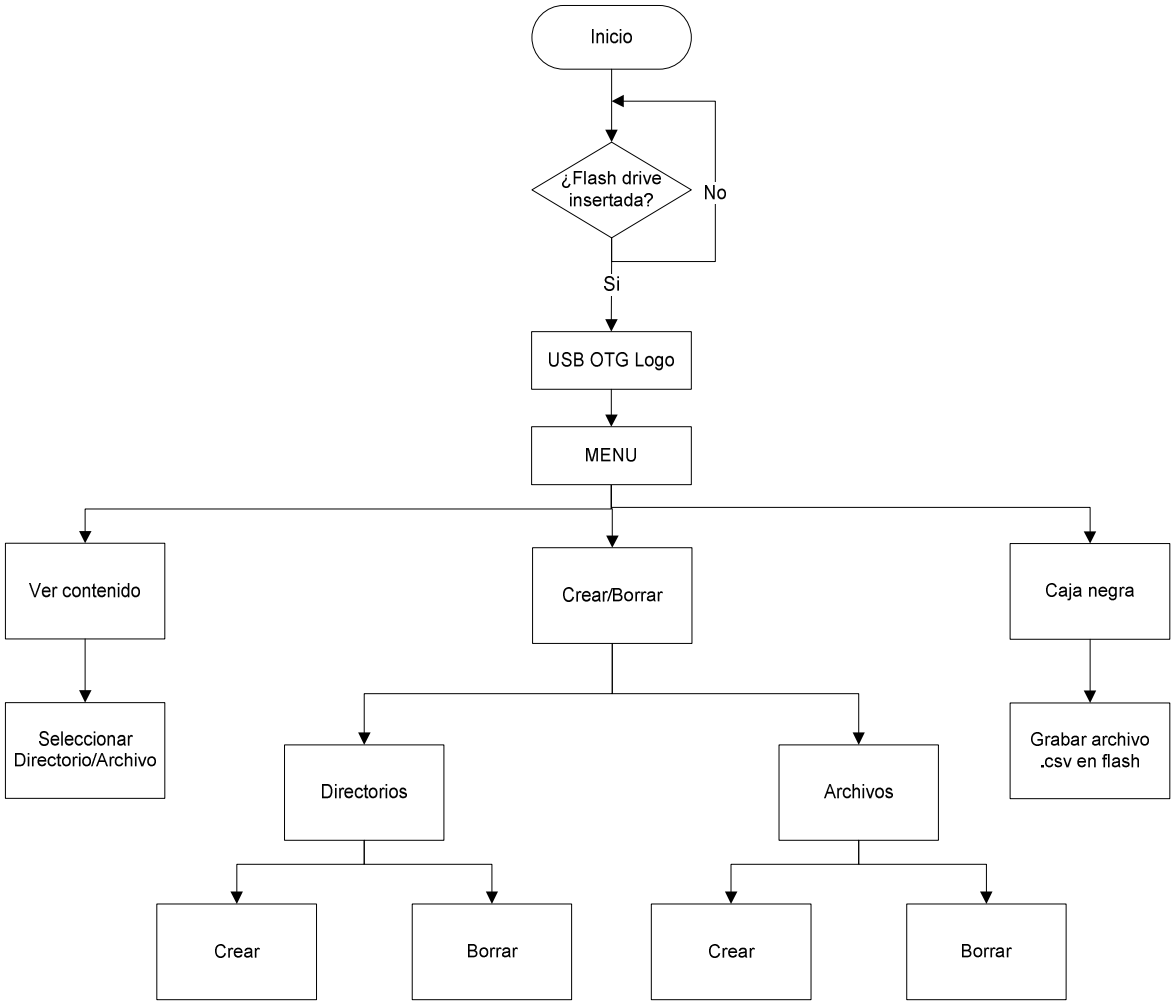


Figura.3.50. Estructura de la aplicación USB

Es decir, la aplicación empieza en un estado de espera a que una flash drive sea insertada, del cual se puede salir presionando la flecha a la izquierda (botón 4). Una vez

que la flash drive ha sido ingresada, automáticamente se presentará en la pantalla el logo de USB OTG, luego se presentará el menú con las tres opciones.

Primero, se recomienda, ir a la opción “Ver contenido”. Esta opción permite que en la pantalla se desplieguen los archivos que actualmente están almacenados. Una vez que se listan los directorios primero y luego los archivos, se puede navegar entre ellos con las teclas 1(arriba) y 3(abajo) y además se permite seleccionar un directorio o archivo al presionar la tecla 5 cuando el archivo este remarcado. Cuando la tecla 5 sea presionada el archivo o directorio será escrito en la parte superior derecha después de un visto.

Cuando ya se ha seleccionado el archivo y se presiona el botón 4 para regresar al menú, el archivo o directorio permanece seleccionado; entonces, al ingresar a la opción Crear/Borrar, un puntero sigue apuntando al directorio/archivo seleccionado.

Al ingresar a la opción *Crear/Borrar>>Directorios>>Crear* en la pantalla se presentarán cuatro cuadros con una letras A, B, C y D. Sí se escoge la opción A se creará un directorio con el nombre “A”, en el directorio que se encuentre escogido o en el directorio raíz, si no se ha escogido ninguno. Por ejemplo, sí en “Ver contenido” se selecciona el directorio “Ejemplo 1” y luego se ingresa a la opción para crear un nuevo directorio y se selecciona el nombre A, el directorio creado será E:/Ejemplo 1/A.

Lo mismo se aplica para la creación de archivos en la opción *Crear/Borrar>>Archivos>>Crear*. Para esta opción los archivos creados son A.txt, B.txt, C.txt y D.txt. Estos archivos están vacios, solo son creados para demostrar esta capacidad.

Las opciones de borrar funcionan de igual manera, primero debe seleccionarse el directorio o archivo con la opción “Ver contenido” y luego acceder a la respectiva opción para borrar directorio o archivo. La pantalla que se presentará incluye una confirmación que mostrará el nombre del archivo o directorio que va a ser borrado presentando la opciones “Si” o “No”.

Es importante remarcar que si se ha seleccionado un archivo y se ingresa a la opción *Crear/Borrar>>Directorios>>Borrar* y se presiona el botón "Si" el archivo no se borrará, de igual manera si se hace intenta borrar un directorio desde la opción para borrar archivos.

La tercera opción corresponde a la capacidad de la librería para crear un archivo y escribir en él. Esta opción se denomina "Caja Negra" y simula la caja negra de un avión. Esta aplicación, al ser ingresada crea automáticamente un archivo llamado LOG.csv[42] y graba, también automáticamente, la fecha y el día, la hora, el minuto, el segundo y que acción (de entre tres opciones) se realizó en ese instante. Por ejemplo:

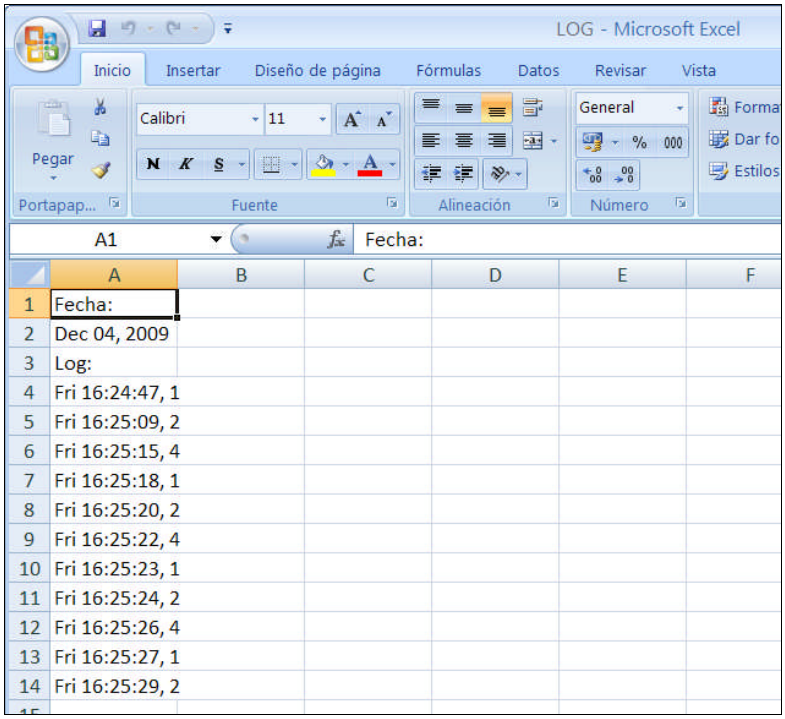


Figura. 3.51. Archivo LOG.csv generado por aplicación "Caja Negra"

Como se puede apreciar en la Figura 3.51 el formato con el que se graba la información es:

- DIA HORA:MINUTO:SEGUNDO, ACCIÓN

[42] **LOG: bitácora o registro en inglés**

Ya que el módulo RTCC viene programado en inglés, los días de la semana, el mes y el formato de la fecha vienen en este idioma.

3.1.2.4. Aplicación secundaria: RGB

La aplicación RGB[43] pone en funcionamiento un elemento de hardware del Starter Kit PIC24F. De acuerdo a la teoría expuesta ya en la sección 2.1.5, este LED tiene la capacidad de generar casi cualquier color a partir de los colores primarios azul, verde y rojo y su diferente saturación.

Para esta aplicación no se pretende generar cualquier color variando la saturación de cada uno, se propone crear los colores secundarios partiendo de la mezcla de dos o tres colores primarios. Esta aplicación fue inspirada en la Figura 3.52:

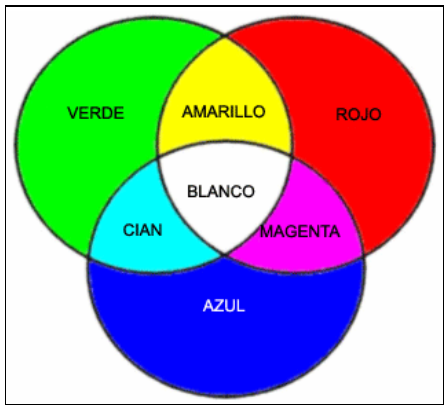


Figura. 3.52. Colores primarios

Para desarrollar la aplicación se usó esta Figura, solo que el círculo de color azul esta en la parte superior. Además, se imprime un cursor de forma redonda que sirve para navegar entre los tres círculos y según el lugar donde se encuentre, el LED RGB proyecta el color correspondiente. Por ejemplo, el cursor comienza en el centro del dibujo y por lo tanto el color que debería generar el LED RGB es blanco. Para salir se presiona el botón 5.

[43] **RGB: significa Red-Green-Blue, es un LED que proporciona los tres colores a diferentes saturaciones para ofrecer la combinación de estos tres colores. Ver sección 2.1.5.**

Para determinar que color debe producirse primero se debe conocer en que posición está el cursor con respecto a cada círculo; es decir, sí el cursor está dentro del círculo rojo, del verde, del azul o en una intersección. La geometría nos indica que la distancia entre dos puntos es:

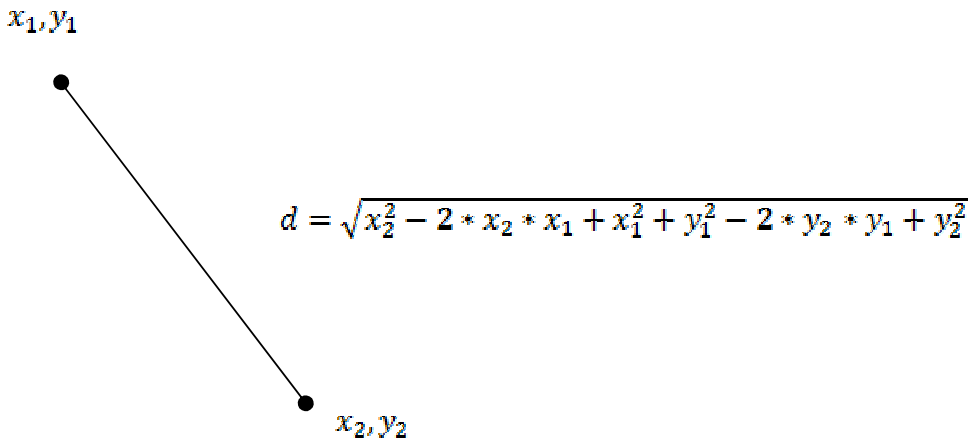


Figura. 3.53. Distancia entre dos puntos

Se conoce la posición del cursor, el centro del círculo y también su radio. ***Sí se obtiene la distancia que existe entre el centro del círculo y el cursor y resulta que esta distancia es menor al radio; entonces, el cursor está dentro del círculo***.

Con esta idea se siguen los siguientes pasos para determinar donde se encuentra el cursor:

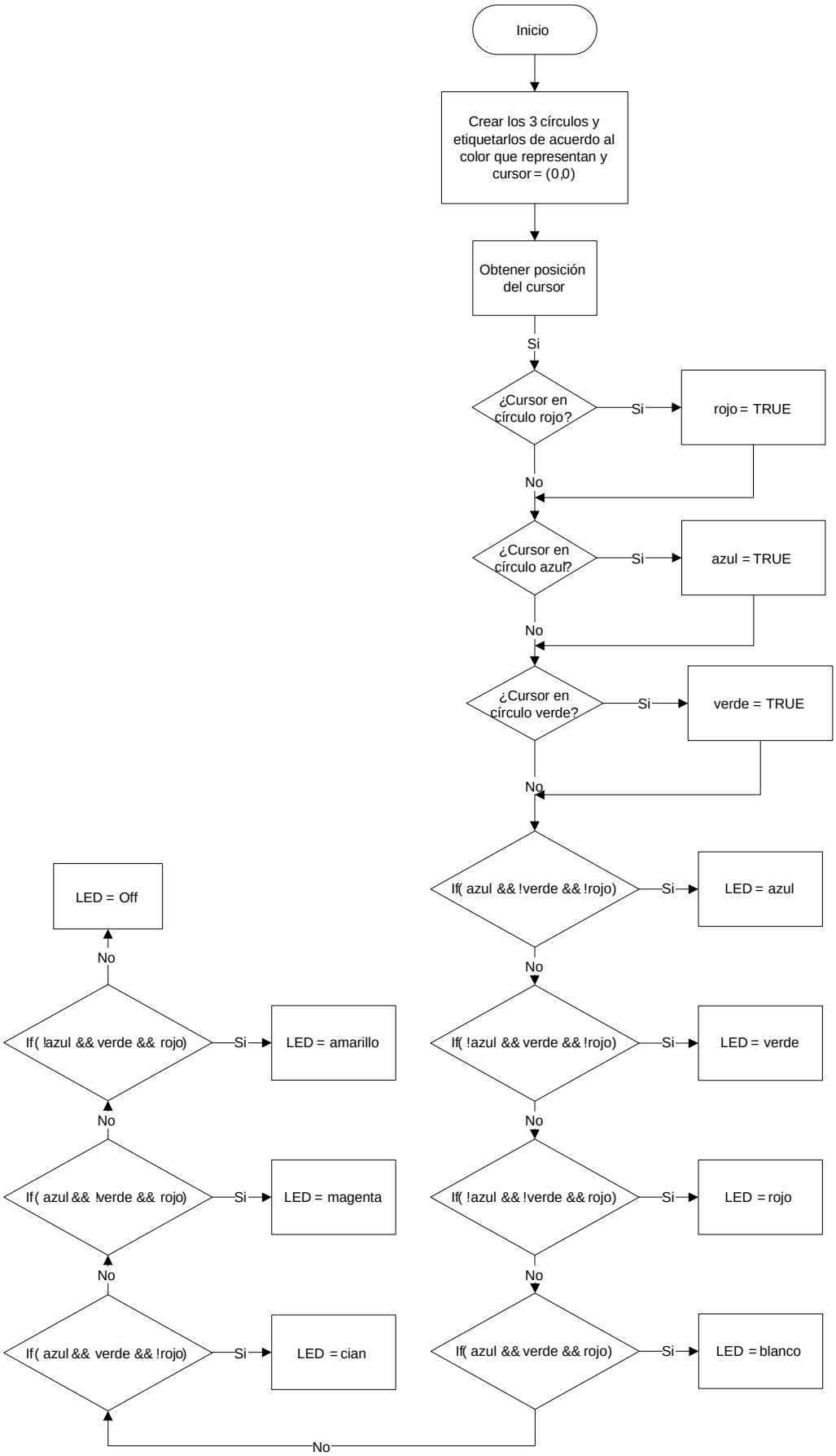


Figura. 3.54. Diagrama de flujo de la aplicación RGB

La Figura 3.54 explica el flujo que sigue esta aplicación. Se comienza por obtener la posición del cursor, luego se calcula si el cursor esta dentro de alguno de los círculos. Las variables tipo BOOL[44]: azul, rojo y verde son igualadas a TRUE cuando el cursor está dentro de uno de los círculos que representan. Una vez que se conoce en que círculos se encuentra el cursor, es fácil determinar que color se debe producir. Por ejemplo, sí las variables verde y azul son iguales a TRUE y la variable rojo es igual a FALSE la condición del color cian[45] (`if (azul && verde && !rojo)`) se cumple y por lo tanto el LED se prende de este color.

Para comprender más de esta función se recomienda leer el código fuente del archivo rgb.c, él cual esta debidamente comentado para su compresión.

3.1.2.5. Aplicación secundaria: RTCC

RTCC o Real Time Clock and Calendar (Reloj calendario de tiempo real) es una aplicación para demostrar la utilidad de este módulo explicado su base teórica en la sección 2.2.6.

La aplicación consiste mostrar una pantalla para que el usuario pueda igualar el reloj a la fecha y hora que corresponden para que la aplicación "Caja negra" grabe el archivo LOG.csv adecuadamente.

Primero, se debe escoger con los botones "+" y "-" correspondientes a cada campo, con los botones 2(derecha) y 4(izquierda) y usar el botón 5 para ejecutar la acción de incrementar (+) o decrementar (-) el valor. Cuando se desee salir de la aplicación se debe llegar hasta el botón "Igualar" y presionar 5. La fecha y la hora serán actualizadas a los registros correspondientes y se retornará al menú principal.

[44] La variable del tipo BOOL solo puede tener dos valores: TRUE o FALSE. Las variables son iniciadas todas en FALSE.

[45] Condición azul = TRUE, verde = TRUE y rojo = FALSE significa que el cursor esta en la intersección de los círculos azul y verde y fuera del círculo rojo.

CAPITULO 4

TUTORIALES

4.1 TUTORIAL PARA EL STARTER KIT PIC24F

Usar el Starter Kit PIC24F para realizar una aplicación que pretenda usar las poderosas capacidades y novedosas ventajas de la familia PIC24F, es una decisión acertada.

Las opciones que se presentan frente al Starter Kit PIC24F son: la tarjeta Explorer 16 junto con las tarjetas de expansión Graphics PicTail y USB OTG PicTail. El Explorer 16 no contiene un ICD incluido como lo hace el Starter Kit. Sumando el costo de todos los componentes que requiere el Explorer 16 y su mismo precio, el costo asciende a alrededor de 550 dólares.

Como contraparte, empezar con el Starter Kit PIC24F es un tanto más complicado y tedioso que con el Explorer 16. Por razones obvias Microchip ® presenta más cantidad de artículos y documentación orientados al uso del Explorer 16. Por esta razón, este documento presenta un corto tutorial completo para desarrollar una aplicación sencilla en el Starter Kit PIC24F.

Se presentan tres tutoriales:

- Tutorial para desarrollar aplicaciones con el Starter Kit PIC24F
- Tutorial del Lenguaje C30 para microcontroladores
- Tutorial sobre el uso de las librerías de gráficos y de USB (Graphics Library v1.65 y MCHPFUSB v2.4).

Los temas que serán tratados en cada tutorial se indican a continuación:

- Tutorial para desarrollar aplicaciones con el Starter Kit PIC24F
    - Creación de un proyecto en MPLAB IDE
    - El archivo .c y .h principales
    - Depurar, quemar y probar la aplicación.
- Tutorial básico de MPLAB C30
    - El MPLABC30
    - Tipos de datos
    - Interrupciones
- Tutorial sobre el uso de las librerías de gráficos y de USB Graphics Library v1.65 y MCHPFUSB v2.4.
    - Librería de Gráficos
        - Instalación y reconocimientos de las herramientas que ofrece la librería
        - Archivos .c y .h necesarios
        - Creación de objetos
        - Programación de objetos
        - Las funciones GOLDraw() y GOLMsg()
    - Librería de USB OTG
        - Archivos .c y .h necesarios
        - Las funciones FS

4.1.1. Tutorial para desarrollar aplicaciones con el Starter Kit PIC24F

4.1.1.1 Creación de un proyecto en MPLAB IDE. Para crear un proyecto en MPLAB IDE[46] se necesita usar la herramienta *Project Wizard*.

Project>>Project Wizard

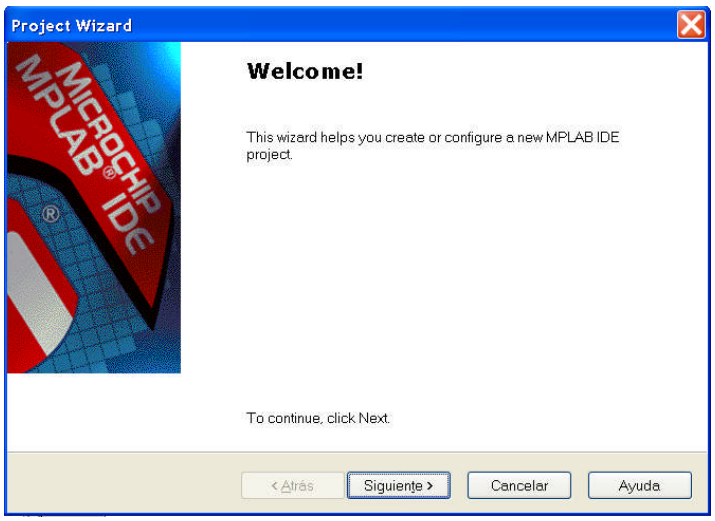


Figura. 4.1. Pantalla de Project Wizard

Presionar "Siguiente". Seleccionar el microcontrolador que se va a usar.

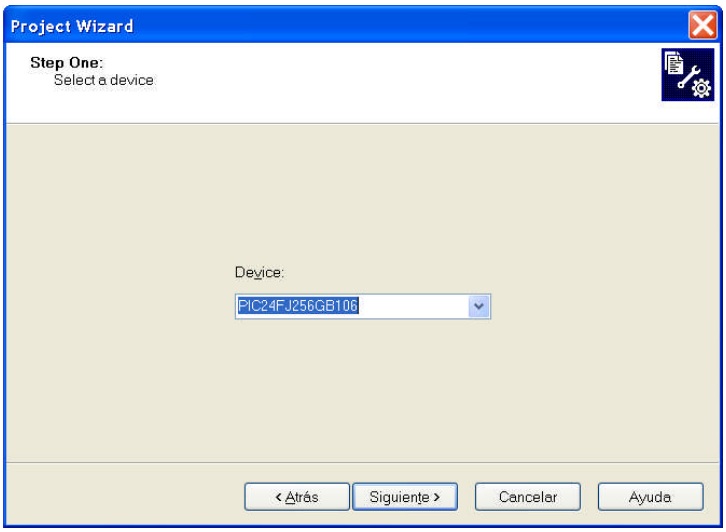


Figura. 4.2. Selección de microcontrolador

Seleccionar PIC24FJ256GB106 y presionar "Siguiente". Escoger el ToolSuite que se va a usar. Escoger la opción MPLAB C30 Compiler (pic30-gcc.exe).

[46] **Se recomienda usar la versión de MPLAB IDE 8.20 o superior.**

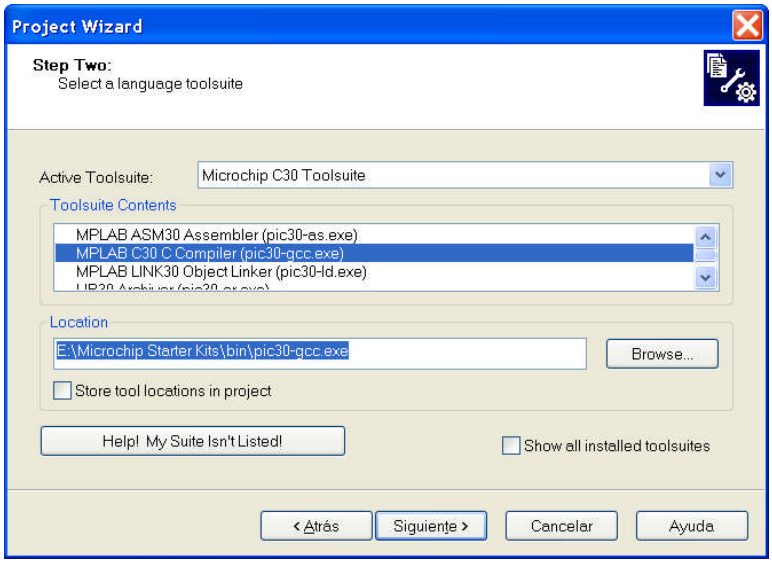


Figura. 4.3. Seleccionar el ToolSuite.

Presionar siguiente. La siguiente pantalla es para escoger el directorio donde se guardará el archivo.

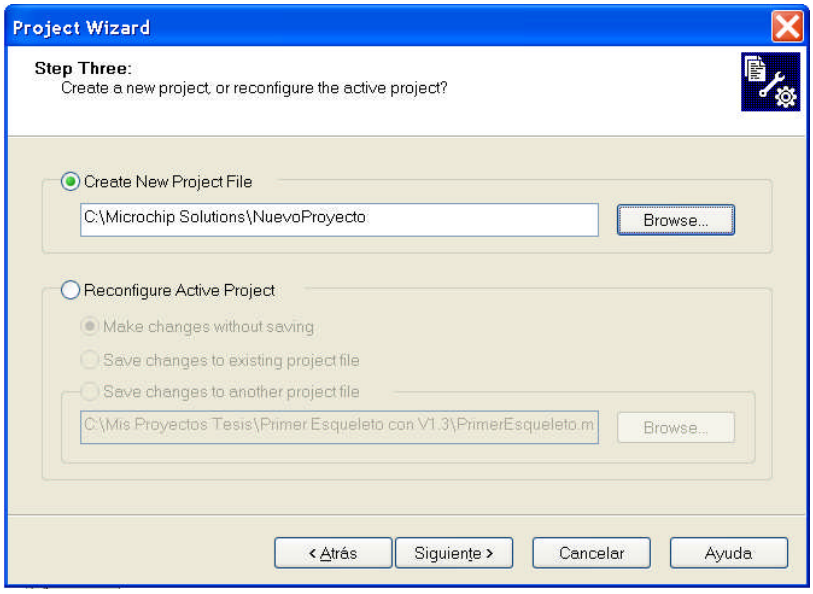


Figura. 4.4. Seleccionar directorio

Una vez seleccionado el directorio y escrito el nuevo nombre para el proyecto, presionar “siguiente”. Se llegará a la nueva pantalla en la cual se agregan los archivos que se requieren para el nuevo proyecto. En este momento no se listarán los archivos que se necesitan de las librerías de gráficos ni de USB, esto se hará en sus respectivos tutoriales.

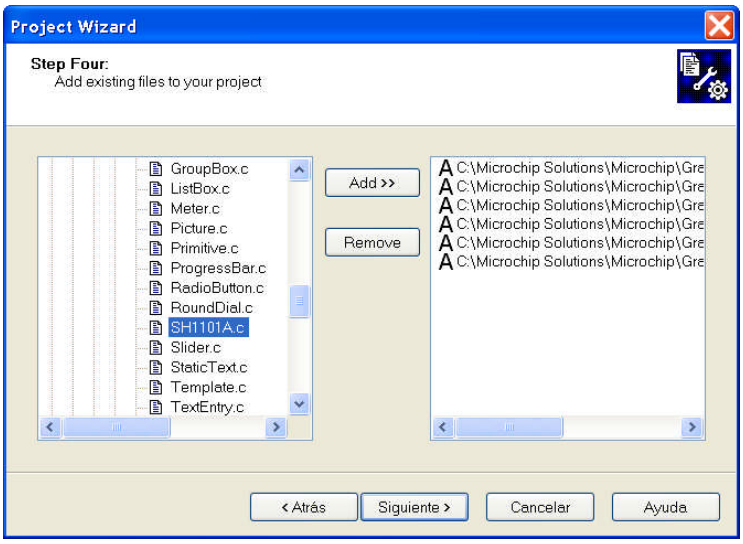


Figura. 4.5. Escoger los archivos de las API que se necesiten

Finalmente, se llega a la pantalla final de la creación del proyecto. Esta pantalla es solo una confirmación de toda la configuración que se ha escogido.

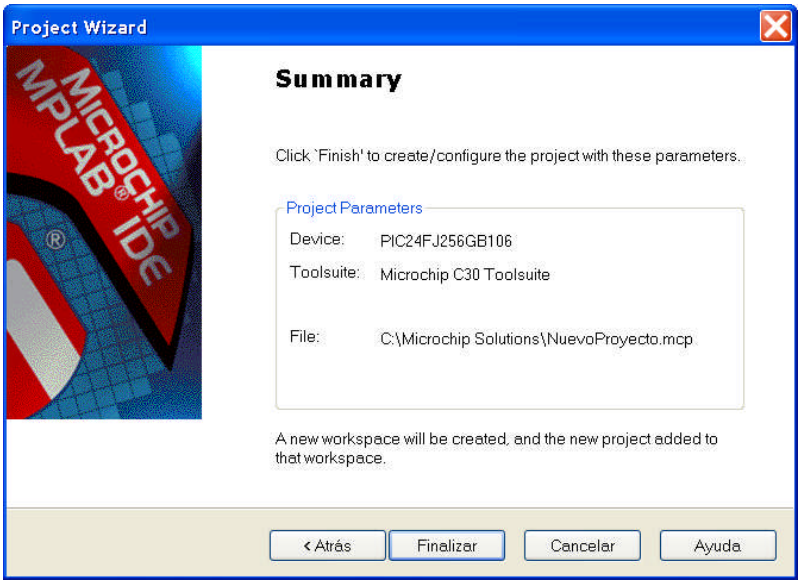


Figura. 4.6. Sumario.

4.1.1.2. El archivo .c y .h principales. El archivo principal .c es aquel que contiene la función principal int main (void) {}; el archivo principal .h es aquel que incluye la librería del mismo: #include<pic24fj256gb106.h>.

Además, el archivo main.c (archivo principal) también contiene las declaraciones, definiciones, prototipo de funciones, inicialización de variables globales, etc. Refiérase a la sección 3.2.1.1 para una mayor comprensión.

En la sección de configuración del archivo main.c antes de iniciar la función int main (void) {} se incluirá el texto de la Figura 4.7.

```
/*********************************
Bits de Configuracion
*********************************/
#define PLL_96MHZ_OFF   0xFFFF
#define PLL_96MHZ_ON    0xF7FF


#if defined( __PIC24FJ256GB110__ ) || defined( __PIC24FJ256GB106__ )
    #ifdef USE_FRC
        // Using the FRC (8MHz), no clock from the PIC18F67J50.
        // NOTE: USB operation is not guaranteed
        _CONFIG2(IESO_OFF & PLL_96MHZ_ON & PLLDIV_DIV2 & FNOSC_FRCPLL & POSCMOD_NONE) // Primary osc disabled, FRC OSC
        _CONFIG1(JTAGEN_OFF & ICS_PGx2 & FWDTEN_OFF)        // JTAG off, watchdog timer off
    #else
        // Using the 12MHz clock provided by the PIC18F67J50
        _CONFIG2(IESO_OFF & PLL_96MHZ_ON & PLLDIV_DIV3 & FNOSC_PRIPLL & POSCMOD_HS)   // Primary HS OSC with PLL, USBP
        _CONFIG1(JTAGEN_OFF & ICS_PGx2 & FWDTEN_OFF)        // JTAG off, watchdog timer off
    #endif
#else
    #error This code is designed for the PIC24FJ256GB1
#endif
```

Figura. 4.7. Bits de configuración

Este fragmento, es recomendable copiarlo textualmente ya que provee la configuración del microcontrolador para usar su oscilador a 32 MHz, no usar el puerto Jtag, etc.

Luego se pueden añadir aquí las declaraciones tipo #define, que serán explicadas en el tutorial de lenguaje C30, la declaración de variables locales y la inicialización de variables globales.

Un prototipo de funciones es sugerido también antes de entrar a la función principal. En esta función se recomienda pegar el siguiente código tal como está. Este fragmento sirve para la configuración del oscilador y configurar los pines del debugger.

```
/*********************************************************

                Aplicacion Principal Main
                ~~~~~~~~~~~~~~~~~~~~~~~~~

*********************************************************/
int main ( void)
{

GOL_MSG msg;

    RGBMapColorPins();

    TRISBbits.TRISB1 = 0;       // Debug
    LATBbits.LATB1   = 0;

#ifdef USE_FRC
        OSCCON = 0x1102;        // Enable secondary oscillator, use FRC oscillator
        CLKDIV = 0x0000;        // FRC post-scaler (1:1), USB postscaler (1:1), CPU postscaler (1:1)
    #else
        #ifdef GO_SLOW
            OSCCON = 0x3302;    // Enable secondary oscillator, use HS oscillator
            CLKDIV = 0x0080;    // (not needed - FRC post-scaler (1:1)), USB postscaler (4:1), CPU postscaler
        #else
            OSCCON = 0x3302;    // Enable secondary oscillator, use HS oscillator
            CLKDIV = 0x0000;    // (not needed - FRC post-scaler (1:1)), USB postscaler (1:1), CPU postscaler
        #endif
    #endif

    RCON = 0;
```

Figura. 4.8. Inicio de función principal

El siguiente paso es llamar a la función GOLInit() encargada de inicializar la API de gráficos y la función USBInitialize(0) encargada de iniciar el framework de USB OTG para anfitrión.

El bucle principal que se propone para cualquier aplicación, es el mismo de esta tesis:

```
    while( 1)
    {

        if (GOLDraw())
        {
            TouchSenseButtonsMsg( &msg );        // Get a raw touchpad message
            if ((msg.uiEvent != EVENT_INVALID) && ((tick - displayChangeTime) > MESSAGE_DEAD_TIME))
            {
                TraducirTouchpad( &msg );       // Translate the raw message
                GOLMsg( &msg );                 // Process the message
            }

        }

    }
```

Figura. 4.9. Bucle principal

El bucle principal se basa en la arquitectura de la API de gráficos explicada a lo largo del capítulo 3. La arquitectura API, según el documento AN1136A, demanda que el usuario sea el creador de las funciones:

- GOLDrawCallback
- GOLDrawMsgCallback
- TraducirTouchpad

Debe recordarse que estas funciones deben trabajar de acuerdo a la máquina de estados. La máquina de estados, para cada situación, debe tener un estado específico para la función GOLDrawCallback y otro para GOLDrawMsgCallback y TraducirTouchpad. Por ejemplo, si se debe crear un estado para presentar un menú principal en el que el usuario deba actuar, dos estados deben ser creados: PANTALLA_MENU_PRINCIPAL para la función GOLDrawCallback y MENU_PRINCIPAL para las funciones GOLDrawMsgCallback y TraducirTouchpad.

En cada estado, no se recomienda escribir el código, sino llamar a una función que lo contenga. Esta función puede ser escrita en el mismo archivo .c o en otro archivo según a la aplicación a la que corresponda.

La creación de la máquina de estados se detallará en el tutorial de Lenguaje C30.

A continuación, se sintetiza los pasos recomendados para escribir el código del archivo main.c y main.h:

Main.c:

1. Incluir las librerías:
    a. Stdlib.h
    b. String.h
    c. Stdio.h
    d. Main.h
2. Incluir los bits de configuración
3. Declarar e inicializar variables
4. Prototipo de funciones

5. Función principal:
    a. Iniciar con el código de la Figura 4.8
    b. Llamar a las funciones:
        i. GOLInit()
        ii. USBInitialize(0)
    c. Crear el bucle principal como la Figura 4.9
6. Crear la máquina de estados
7. Crear la función GolDrawCallBack, GOLDrawMsgCallback y TraducirTouchPad.
8. Crear las funciones de cada estado correspondiente a las funciones GolDrawCallBack, GOLDrawMsgCallback y TraducirTouchPad.

Main.h:

1. Incluir los archivos .h de la librería de gráficos y USB que se vayan a usar[47].
2. Incluir otros archivos cabecera que hayan sido creados para las diferentes aplicaciones.
3. Incluir el código de la Figura 4.10 necesario para la configuración del hardware y uso de las librerías:

```
// Hardware
#if defined USE_SH1101A
    #define NUM_TOUCHPADS                5                    // Number of capacitive TOUCHPADs
    #define STARTING_ADC_CHANNEL         8
#elif defined USE_SSD1303
    #define NUM_TOUCHPADS                4                    // Number of capacitive TOUCHPADs
    #define STARTING_ADC_CHANNEL         9
#else
    #error Graphics board not defined
#endif

// Necesario para libreria de TouchSense
// ID's for mapping the touchpads to buttons

#define ID_BUTTON_UP                ID_TOUCH_BUTTON_01
#define ID_BUTTON_RIGHT             ID_TOUCH_BUTTON_02
#define ID_BUTTON_DOWN              ID_TOUCH_BUTTON_03
#define ID_BUTTON_LEFT              ID_TOUCH_BUTTON_04
#define ID_BUTTON_CENTER            ID_TOUCH_BUTTON_05
```

Figura. 4.10. Sección de código para archivo main.h

4. Declarar los tipos de datos que se van a crear para la máquina de estados y otras aplicaciones.

[47] **Estos archives serán indicados en el tutorial correspondiente a cada librería**

5. Declarar las variables globales.

4.1.1.3. Depurar, programar y probar la aplicación. Es momento de depurar el trabajo que se ha realizado. Una vez completado los pasos indicados en la sección 4.1.1.2 se procede a depurar con la herramienta "Build All" (tecla F10).

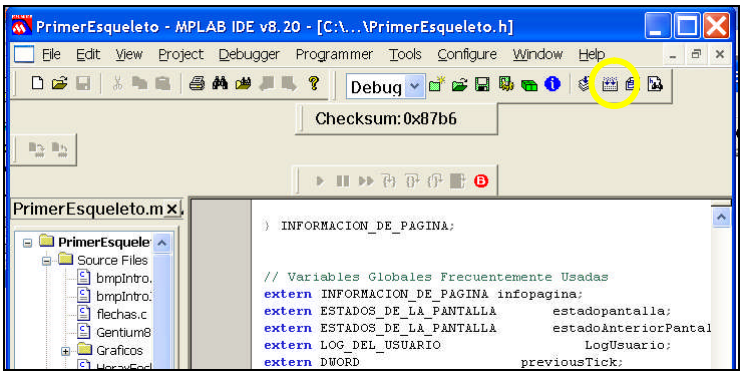


Figura. 4.11. Herramienta Build All

La parte inferior de la pantalla es una ventana llamada Output. Tiene algunas pestañas como lo indica la Figura 4.11. Entre estas, están:

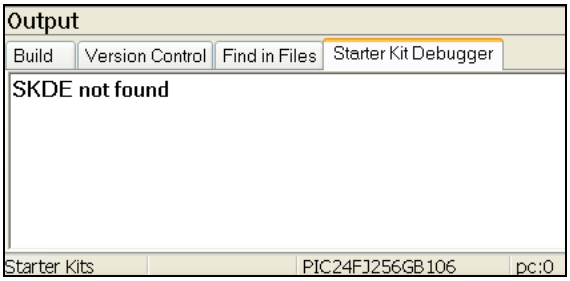


Figura. 4.12. Ventana Output

Una vez presionado el botón F10 o al hacer click en el botón Build All la pestaña Build se seleccionará automáticamente y se presentará el resultado del proceso. Pueden existir dos resultados: BUIL SUCCEEDED o BUILD FAILED. Se presenta también un resumen de los warnings o errors, de existir, en letras azules. Haciendo click sobre la línea de error, el IDE seleccionará automáticamente la línea de código correspondiente al error o al warning.

En caso de no existir errores, se puede proseguir al siguiente paso el cual es quemar el firmware al microcontrolador. Para esto el Starter Kit debe estar conectado al PC y el IDE debe reconocerlo mostrando en la ventana de Output en la pestaña Starter Kit Debugger, el mensaje "SKDE[48] connected".

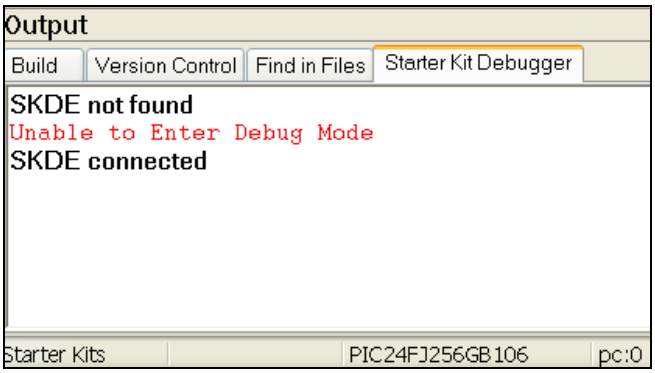


Figura. 4.13. Mensaje al conectar el SKDE

Ahora, la aplicación se programa con la herramienta "Program" en *Debbuger>>Program* o haciendo click en:

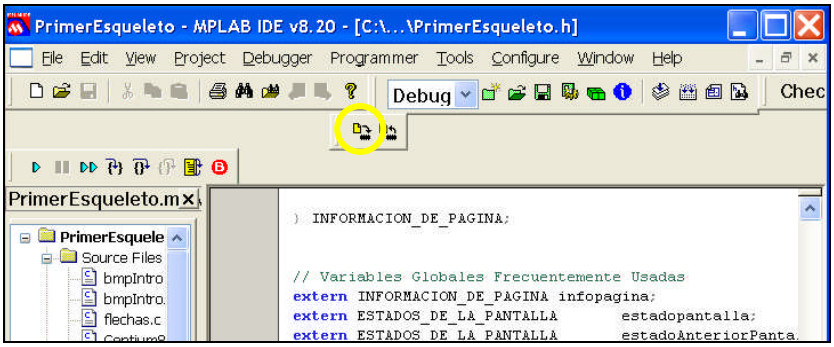


Figura. 4.14. Herramienta Program

El mensaje que se desplegará en la ventana Output, mientras se está programando el microprocesador, será:

[48] **SKDE significa Starter Kit Debugger**

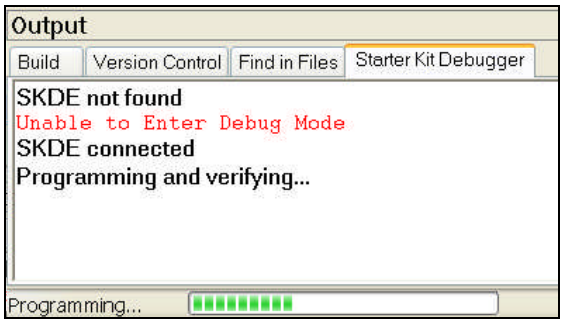


Figura. 4.15. Mensaje mientras se programa

Una vez finalizada la programación se presentará el mensaje:

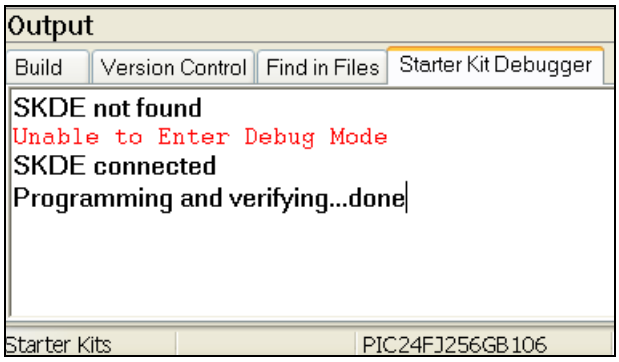


Figura. 4.16. Mensaje al finalizar la programación

Ahora, es momento de ejecutar el firmware, para esto usaremos la barra de herramientas Debug:

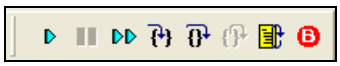


Figura. 4.17. Barra de Herramientas Debug

Esta barra incluye las herramientas:

- Play: ejecuta la aplicación normalmente
- Pause: pausa la ejecución en la línea que se encuentre
- Animate: ejecuta la aplicación a una velocidad establecida en la configuración del Debugger. Generalmente es lo suficientemente lenta para que sea visible para el usuario.
- Step Into: ejecuta las líneas de código una a una ingresando al interior de las funciones.

- Step Over: ejecuta línea a línea pero sin ingresar a las funciones.
- Reset: ejecuta un reset general al microprocesador
- Breakpoints: establece un breakpoint en la línea en que se encuentre el cursor en ese momento.

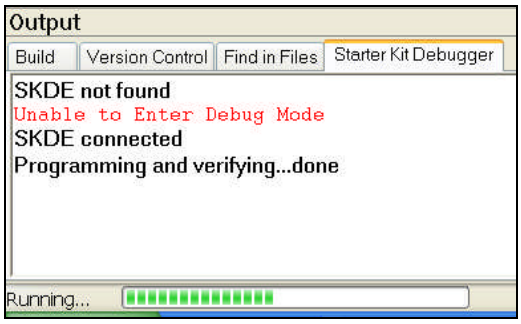


Figura. 4.18. Ventana Output durante la ejecución de la aplicación

En este punto, el usuario ya es capaz de correr el firmware grabado en la tarjeta para comprobar su funcionamiento. La pestaña indica la salida mostrada en la Figura 4.18.

4.1.2. TUTORIAL DEL LENGUAJE C30 PARA MICROCONTROLADORES

El presente tutorial contiene solo temas que han sido considerados los más importantes recalcar; sin embargo, hay variados temas que deben ser estudiados para tener un completo entendimiento del compilador. Esto no implica que de no conocer estos temas no se van a poder desarrollar aplicaciones. De hecho, un conocimiento del lenguaje C30 y algo de experiencia en el software de MPLAB es suficiente para empezar a desarrollar aplicaciones.

Entre los temas tratados estará el como declarar un servicio de interrupciones, que tipos de datos se pueden definir, mezclar assembler con módulos C y como funciona el compilador. Temas como el manejo de memoria, el uso de la heap C o las convenciones para nombrar a los archivos no son de primera necesidad para empezar a trabajar en aplicaciones.

Como último punto antes de empezar con el tutorial es importante mencionar que esta información ha sido producto en cierta parte adquirida empíricamente y otra parte ha sido extraída del documento *MPLAB30 User's guide* de la empresa Microchip ®.

4.1.2.1. El MPLAB C30. El MPLAB C30 es un compilador C optimizado sumizo a ANSI x3.159-1989, que incluye extensiones para aplicaciones de control embebido de dsPIC DSC. El compilador es una aplicación para consola Windows ® que provee una plataforma para desarrollar código en C. El compilador C30 es un puerto del compilador GCC de la Free Software Foundation.[49]

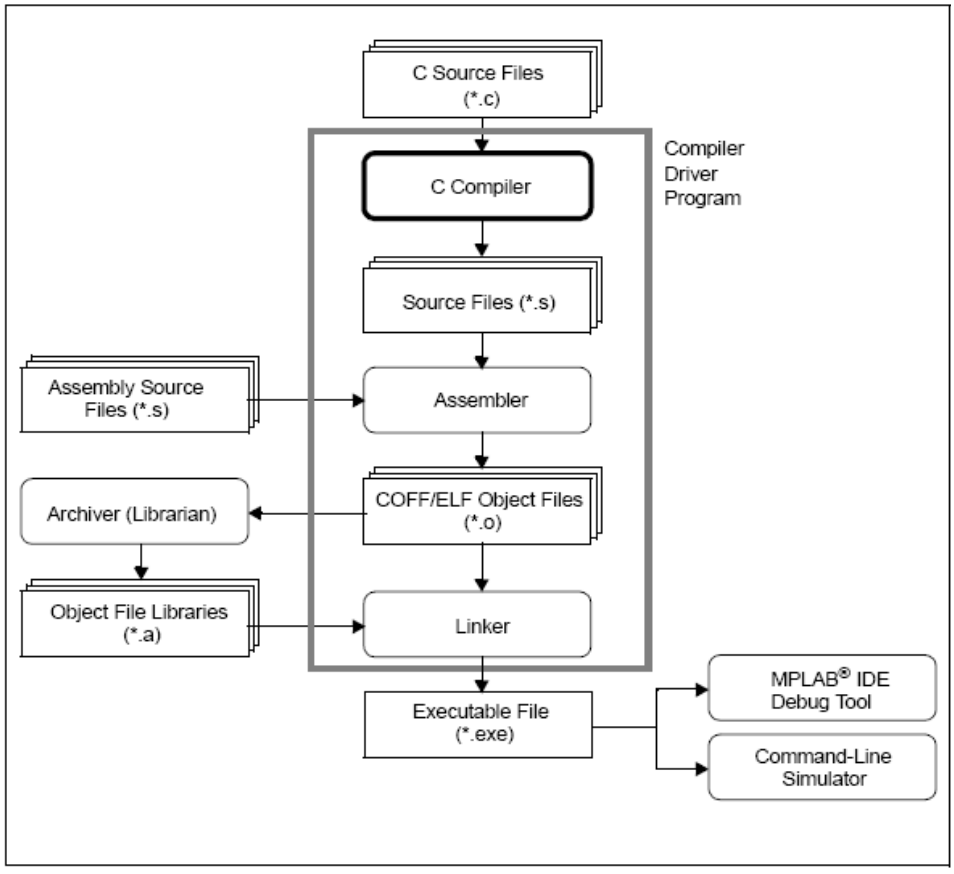


Figura. 4.19. Data flow y herramientas de desarrollo de software

La Figura 4.19 representa el flujo que sigue el firmware creado por el usuario desde el archivo .c al pasar por el compilador, transformándose en .s, crea un archivo .o para ir hasta el Archiver y conectarse con el Linker y finalmente conjugarse en archivo .exe, útil para la herramienta de Debug y el simulador de las líneas de comando.

[49] **MPLAB C30 User's guide**

Todo este proceso es transparente para el desarrollador, pero ayudará a comprender el porque se crean los archivos .o, .s y .a en la carpeta del proyecto.

4.1.2.2. Tipos de datos. Los tipos de datos enteros que se presentan en el documento *MPLABC30 User's Guide* son mencionados en este cuadro:

Type	Bits	Min	Max
`char, signed char`	8	-128	127
`unsigned char`	8	0	255
`short, signed short`	16	-32768	32767
`unsigned short`	16	0	65535
`int, signed int`	16	-32768	32767
`unsigned int`	16	0	65535
`long, signed long`	32	-2^{31}	$2^{31} - 1$
`unsigned long`	32	0	$2^{32} - 1$
`long long**, signed long long**`	64	-2^{63}	$2^{63} - 1$
`unsigned long long**`	64	0	$2^{64} - 1$

** ANSI-89 extension

Figura. 4.20. Tipos de datos de MPLAB C30 User's Guide de la extensión ANSI-89

Además, se presenta también los tipos de datos de tipo flotante:

Type	Bits	E Min	E Max	N Min	N Max
`float`	32	-126	127	2^{-126}	2^{128}
`double`*	32	-126	127	2^{-126}	2^{128}
`long double`	64	-1022	1023	2^{-1022}	2^{1024}

E = Exponent
N = Normalized (approximate)
* `double` is equivalent to `long double` if `-fno-short-double` is used.

Figura. 4.21. Tipos de datos de MPLAB C30 User's Guide de la extensión ANSI-89

La declaración de variables es un proceso que no debe ser subestimado. De una correcta declaración depende el no desperdicio de la memoria, el que no se infiltre "junk" o basura en los bits destinados a la variable, etc. En resumen, se puede decir que de la declaración depende mucho la buena calidad de programación, ya que si no se hace adecuadamente se dará entrada a "bugs", desperdicio de memoria y se dificultará el hacer cambios en los programas.

MPLAB C30 User's Guide dice: "*Los servicios de interrupción son un importante aspecto de la mayoría de microcontroladores. Las interrupciones pueden ser usadas para sincronizar operaciones de software con eventos que ocurren en tiempo real. Cuando las interrupciones ocurren, el flujo normal de la ejecución de software se suspende y funciones especiales son invocadas para procesar dicho evento. Al completar el proceso de la interrupción, la información del contexto previa al llamado de la interrupción es restablecida y la ejecución normal se toma de nuevo".*

El compilador C30 soporta por completo los procesos de interrupción en C, incluso para las variadas fuentes de interrupción que traen los dsPIC y los PIC más actuales.

Las guías que se presenta para escribir interrupciones son:

- Declarar ISR[50] sin parámetros y un de tipo void Return (obligatoriamente)
- No dejar que la ISR sea llamada por la línea principal de código (obligatoriamente)
- No dejar que ISR llame a otras funciones (opcional)[51]

Un importante hecho es recalcado en el manual de usuario de MPLAB C30: *"Las ISR son como cualquier otra función en la que se puede declarar variables locales y se puede usar variables globales. Sin embargo, una ISR necesita ser declarada sin parámetros y no retornar ningún valor. Esto es necesario porque la ISR, en respuesta a una interrupción por Hardware, es invocada asincrónicamente a la mainline del programa en C (esto significa que no es llamada en la manera normal, así que los parámetros y valores de retorno no aplican)".*

La sintaxis dictada por Microchip para la declaración de un ISR se define como:

[50] ISR (Interrput Service Routine): Servicio para generar una interrupción
[51] MPLAB C30 User's Guide

```
__attribute__((interrupt [(
                          [ save(symbol-list)]
                          [, irq(irqid)]
                          [, altirq(altirqid)]
                          [, preprologue(asm)]
                          )]
              ))
```

Figura. 4.22. Sintaxis de ISR

Las interrupciones pueden ser llamadas por distintos motivos, más de 100 fuentes en los PIC24F, algunas de estas son los timers. A continuación se presenta la pantalla de una interrupción usando el Timer 4 usada en el programa demo de la tarjeta Starter Kit para controlar la variable tick:

```
/*****************************************************************************
  Function:
    void __attribute__((interrupt, shadow, auto_psv)) _T4Interrupt(void)

  Description:
    This function updates the tick count and calls ReadCTMU() to monitor the
    touchpads.

  Precondition:
    Timer 4 and the Timer 4 interrupt must be enabled in order for
                    this function to execute.  CTMUInit() must be called before
                Timer 4 and the Timer 4 interrupt are enabled.

  Parameters:
    None

  Returns:
    None

  Remarks:
    None
  *****************************************************************************/
void __attribute__((interrupt, shadow, auto_psv)) _T4Interrupt(void)
{
    // Clear flag
    IFS1bits.T4IF = 0;
    tick++;

    ReadCTMU();
}
```

Figura. 4.23. Sintaxis de la ISR del Timer 4 en demo del Starter Kit PIC24F

4.1.3. TUTORIAL BÁSICO DE LA LIBRERÍA GRAPHICS LIBRARY V1.65

Este tutorial está hecho para que los desarrolladores que lo usen, estén en capacidad de crear sus primeras aplicaciones. Además, aquí se encontrarán complementados varios puntos que quedaron sin ser tratados anteriormente como los archivos que deben incluirse

por defecto en los proyectos para poder usar esta API, que herramientas usar para convertir imágenes, etc.

4.1.3.1. Instalación y reconocimiento de los recursos que ofrece la librería. El primer paso, lógicamente, es conseguir la librería. El enlace es: www.microchip.com/graphics[52]. Una vez instalada, se recomienda hacer un reconocimiento de los recursos que se disponen. En Windows, en *Inicio>>Programas>>Microchip>>Graphic's Library v1.65* se encontrará un menú con las siguientes utilidades:

- Application Notes
- Bitmap and Font Converter
- Graphics Library Help
- Image Resources
- Schematics

Application Notes: en esta opción se encuentran accesos directos a varios documentos de gran utilidad para empezar a usar la librería. El más recomendado es: "AN1136a How to use widgets". Los otros documentos que aquí se encuentran le serán útiles también, pero no son indispensables para lograr una primera aplicación.

Bitmap And Font Converter: tal vez la más útil y didáctica de las aplicaciones. Sirve para convertir un tipo de letra o una imagen formato Bitmap monocromática a un archivo .c o .hex para que pueda ser interpretado por el compilador y el microcontrolador y ser desplegado correctamente en la pantalla del Starter Kit PIC24F. Para usarlo se debe seguir los siguientes pasos:

1. Seleccionar la imagen: puede ser seleccionada de la sección Image Resources o puede ser creada por el mismo usuario, pero debe cumplir con las siguientes condiciones:

[52] Para el momento que este documento sea leído pueden existir versiones superiores de la librería de gráficos, sin embargo, el principio básico de funcionamiento es el mismo.

a. Debe ser una imagen de formato monocromático (Para el Starter Kit PIC24F).

b. No debe ser mayor a 128x64 pixeles.

2. Como segundo paso, abrir el programa:

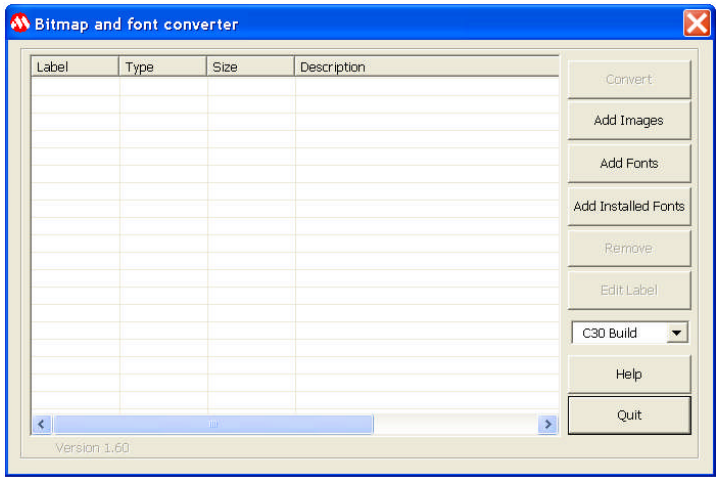


Figura. 4.24. Bitmap and Font converter

Debe cerciorarse de que la opción C30 Build este seleccionada.

3. Presione el botón Add Images, lo que desplegará la siguiente pantalla:

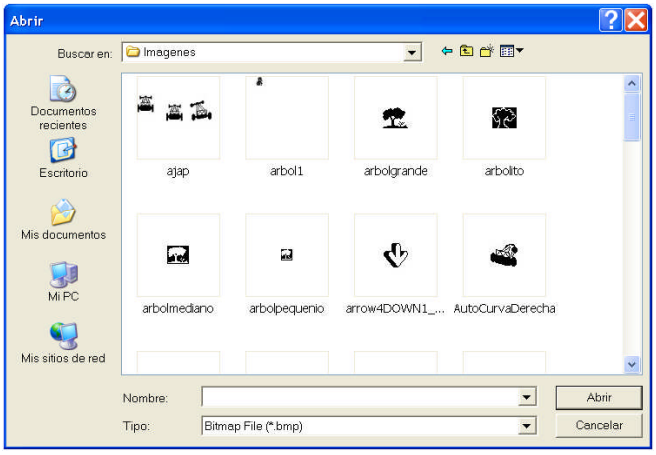


Figura. 4.25. Seleccionar la imagen

Para tener varias imágenes en un solo archivo, se puede repetir varias veces este proceso, como indica la Figura 4.26.

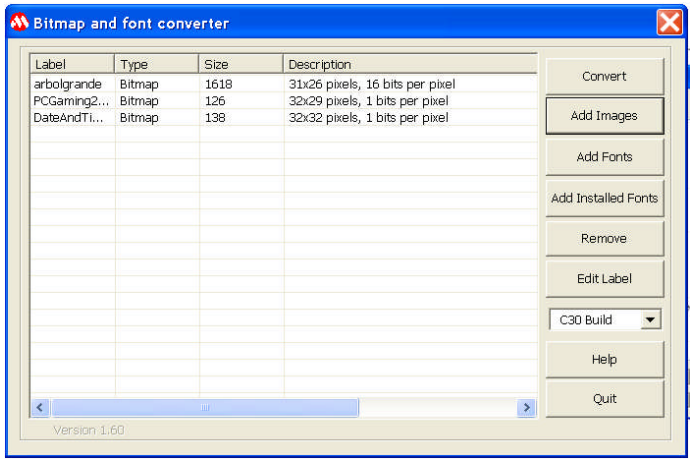


Figura. 4.26. Seleccionar varias imágenes

Esta pantalla nos indicará el número de imágenes, su tamaño y su nombre. Debe tenerse en cuenta que todas estas imágenes serán almacenadas en un solo archivo.

4. Una vez seleccionadas todas las imágenes, se contínua presionando el botón Convert:

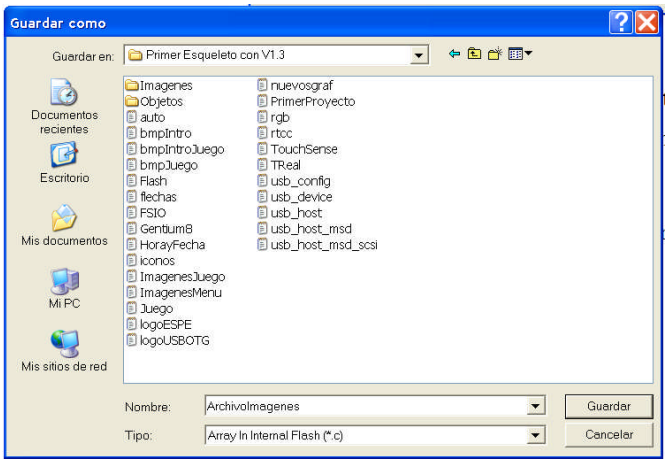


Figura. 4.26. Creacion del archivo .c

Es muy importante seleccionar la opción "Tipo:" como "Array In Internal Flash (*.c). Es posible grabar como .hex u otras opciones las cuales servirán cuando se quiera almacenar en archivo en otra memoria, pero en este caso se desea usar la memoria flash interna. El archivo ejemplo se grabará como *ArchivoImagenes*.

5. Una vez escogido el nombre, presionar Guardar:

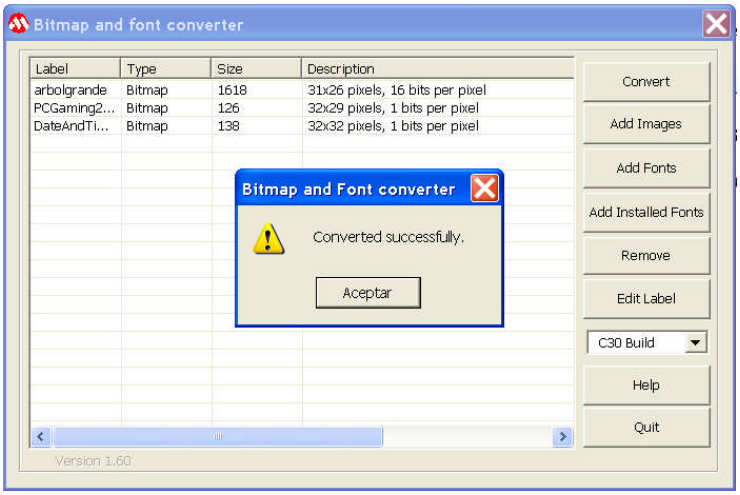


Figura. 4.27. Ventana final

En este punto se ha terminado de convertir las imágenes seleccionadas a un archivo .c que será almacenado en la memoria flash interna del PIC.

6. El siguiente paso es agregar el archivo creado al proyecto, esto se lo puede hacer de la siguiente manera:
    a. En MPLAB IDE, ir a la ventada del proyecto, hacer click derecho en la carpeta “Source Files”

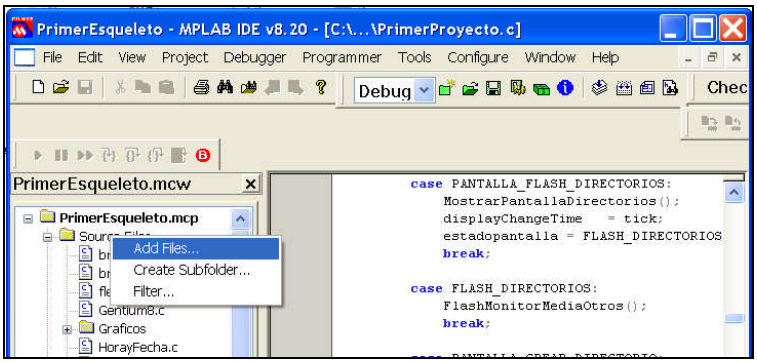


Figura. 4.28. Agregar archivos .c

    b. Seleccionar el archivo .c que se creó con la utilidad Bitmap and Font converter y presionar “Ok”. El archivo quedará añadido. Recuérdese que el

nombre dado fue “ArchivoImagenes”. Una vez que esté añadido, se presentará así:

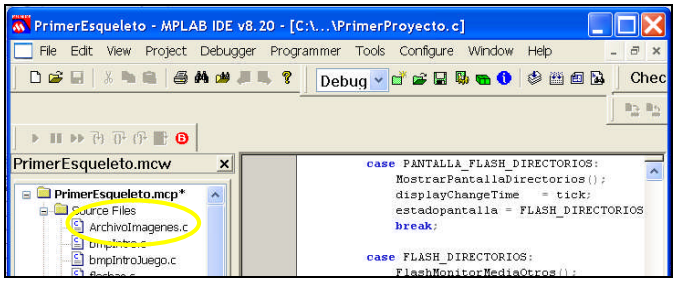


Figura. 4.29. Archivo agregado

7. El siguiente paso es “declarar” las imágenes en el archivo .c de la aplicación que vaya a usarlas. Es necesario indicar al compilador en que memoria debe buscar el archivo, en este caso en la memoria flash interna.

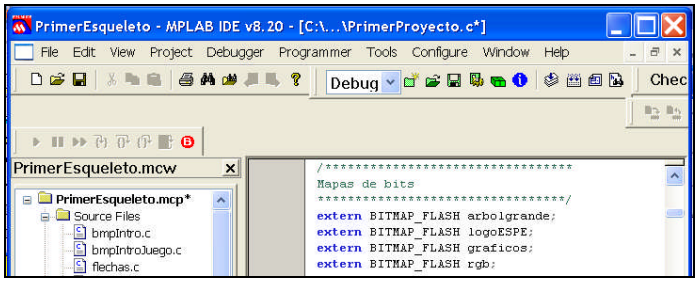


Figura. 4.30. Declaración de imagen

Obsérvese que la imagen ha sido declarada como BITMAP_FLASH (Figura 4.30). Este macro se encuentra explicado en la ayuda de la librería de gráficos.

8. Como último paso, para usar la imagen, se puede crear un objeto PICTURE o se puede agregar a cualquiera de los objetos que den la opción de agregar Bitmap, como el objeto BUTTON. Un ejemplo del uso de la imagen en un objeto PICTURE es el que se presenta en la Figura 4.31:

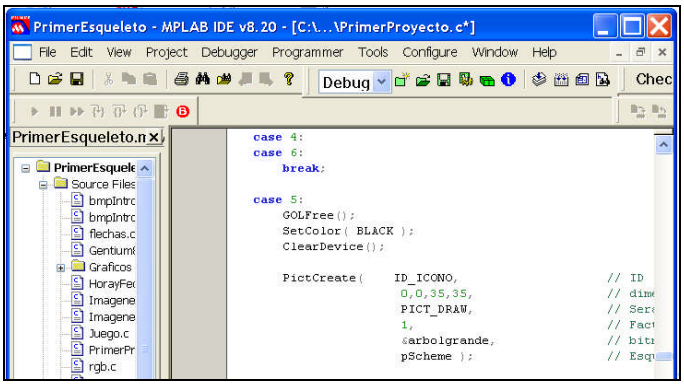


Figura. 4.31. Creación de un objeto Picture con la imagen agregada

Como puede notarse, el uso de la imagen es con la sintaxis: &nombredelaimagen. Recuérdese que el operando & accede a la memoria que representa la variable.

Graphics Library Help: la ayuda de la librería de gráficos es completa y fácil de usar. A la vez que explica la estructura de la librería, sus capas y su mensajería, explica también el uso de las funciones que son necesarias para manejo de objetos, o como son descritos “widgets”.

Image Resources: esta carpeta tiene una gran cantidad de imágenes e iconos útiles para las aplicaciones.

Schematics: este directorio contiene los diagramas esquemáticos de conexiones físicas del PIC a los diferentes dispositivos como son el driver de la pantalla y el teclado Touchpad.

4.1.3.2. Archivos que deben ser agregados para una aplicación. Una vez creado el proyecto, los archivos main.c y main.h y comprendido el funcionamiento básico de la librería de gráficos, se puede proceder a agregar los archivos que serán necesarios para que la API pueda funcionar. Estos archivos .c y .h son código que ha sido programado previamente por los desarrolladores de Microchip han escrito estos archivos comprenden las capas inferiores de la librería, como lo son: la capa Primitive y la capa GOL.

Es fuertemente recomendado, para una mejor organización del código, crear una subcarpeta llamada gráficos donde ir agregando estos archivos, esto se hace de la siguiente manera:

En MPLAB IDE en la ventada del proyecto, hacer click derecho sobre la carpeta “Source files”, seleccionar la opción “Create subfolder” , escribir el nombre indicado:

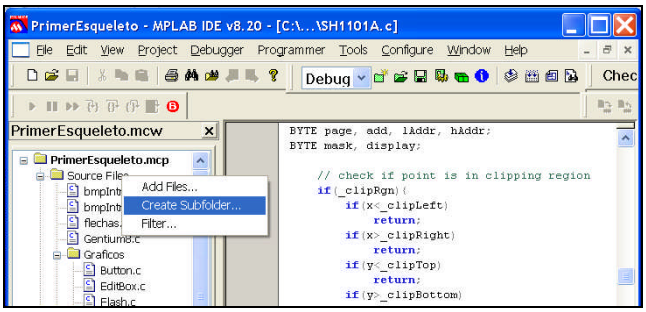


Figura. 4.32. Crear una subcarpeta

Los archivos de código (.c) que deben ser agregados son:

- GOL.c: Definiciones de las funciones de la capa GOL
- Primitive.c: Funciones primitivas (Line, Circle, etc)
- SH1101A.c: Funciones para comunicarse con el driver de la pantalla OLED
- Según los widgets que deseen usarse, deben incluirse:
    - Button.c: Funciones para crear y definir estado del botón
    - EditBox.c: Funciones para crear y definir estado de EditBox
    - Grid.c: Funciones para crear y definir estado del Grid
    - ListBox.c: Funciones para crear y definir estado del ListBox
    - Picture.c: Funciones para crear y definir estado del Picture
    - Slider.c: Funciones para crear y definir estado del Slider
    - StaticText.c: Funciones para crear y definir estado del StaticText
    - RadioButton.c: Funciones para crear y definir estado del RadioButton
- rtcc.c: Para el módulo RTCC

Los archivos cabecera (.h) que deben agregarse son:

- GenericTypeDefs.h: Definiciones generales de los tipos de datos creados
- HardwareProfile.h: Definición del tipo de hardware que se esta usando
- TouchSense.h: Funciones para detectar y traducir mensajes del teclado
- rtcc.h: Para el módulo RTCC
- Según los widgets que deseen usarse, deben incluirse:
  - Button.h
  - EditBox.c: Declaraciones de variables para EditBox
  - ListBox.c: Declaraciones de variables para ListBox
  - Picture.c: Declaraciones de variables para Picture
  - Slider.c: Declaraciones de variables para Slider
  - StaticText.c: Declaraciones de variables para StaticText
  - RadioButton.c: Declaraciones de variables para RadioButton
- ScanCodes: Contiene los macros de los códigos que pueden presentarse por el teclado
- Graphics.h: Aquí el usuario define que widgets va a usar
- GOL.h: Contiene declaraciones y variables para la capa GOL

4.1.3.3. Creación de objetos o Widgets. Para crear un objeto se recomienda seguir los siguientes pasos[53]:

1. En el archivo Graphics.h dejar sin comentar la definición del objetos que se desea y comentar aquel que no se desea usar. Por ejemplo, si se desea usar el objeto Button y no se desea usar el objeto StaticText, a StaticText se lo comenta y a Button no, así:

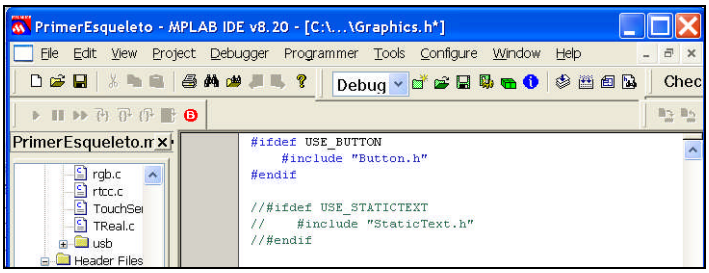


Figura. 4.33. Definición de los objetos a usarse

[53] **Para crear un nuevo tipo de objeto, se recomienda ir a www.microchip.com/grapchis**

2. Como segundo paso debe cerciorarse de que en el archivo main.h estén incluidos todos los archivos .h que se mencionan en la sección 4.1.3.2, incluyendo el correspondiente al objeto que se desea.
3. En una función que sea llamada desde la función GOLDrawCallback, se llama a la función para crear el objeto con la sintaxis indicada en Microchip Graphics Library Help. Por ejemplo, si se desea crear un botón:
    a. Encontrar la función que cree el objeto:
        i. Esta función y otras funciones correspondientes al objeto se encuentran en la sección GOL Objects de Graphics Library Help

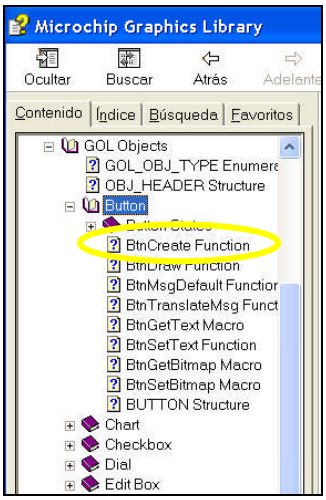


Figura. 4.34. Funciones y macros del objeto Button

        ii. Para el caso del objeto Button esta función se denomina BtnCreate
    b. Conocer la sintaxis de la función, esto se puede hacer en la misma sección

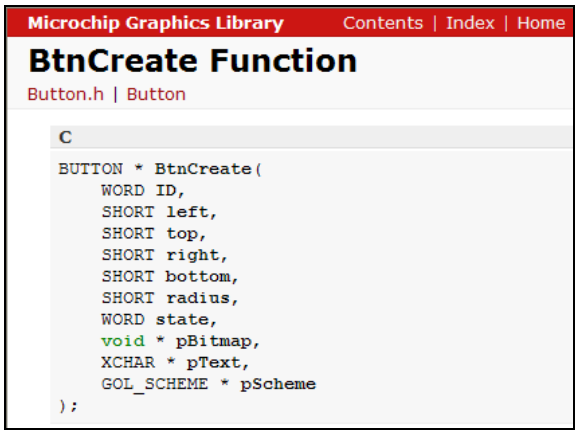
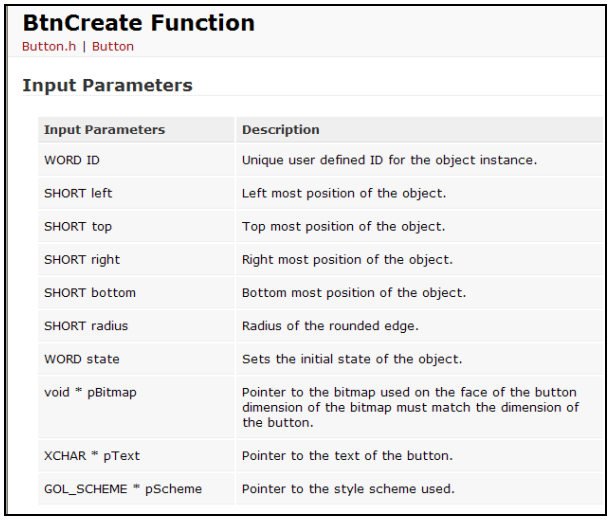


Figura. 4.35. Función para crear el objeto botón

c. Identificar para que sirve cada parámetro, de la misma manera esto se encuentra en Graphics Library



BtnCreate Function

Button.h | Button

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
SHORT radius	Radius of the rounded edge.
WORD state	Sets the initial state of the object.
void * pBitmap	Pointer to the bitmap used on the face of the button dimension of the bitmap must match the dimension of the button.
XCHAR * pText	Pointer to the text of the button.
GOL_SCHEME * pScheme	Pointer to the style scheme used.

Figura. 4.36. Parámetros para la función BtnCreate

d. Declarar los parámetros según lo deseado

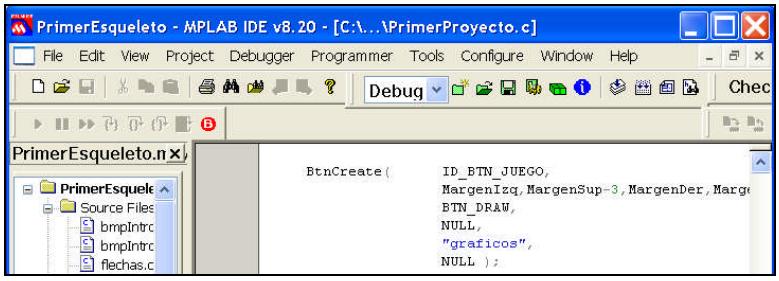


Figura. 4.37. Parámetros del usuario para la función BtnCreate

4. Depurar y grabar el firmware al Starter Kit PIC24F, correrlo y ver si el objeto se creó de acuerdo a las expectativas.

4.1.3.4. Programación de los objetos. Antes de leer esta sección, se recomienda revisar las secciones 3.1.1 y 3.1.2 para recordar como funciona el flujo de la librería y la mensajería de la misma.

Como se recordará, el ciclo que cumple el bucle principal es el siguiente:

```
// Loop principal

while( 1)
{

    if (GOLDraw())                          // Dibujar pantalla o re-dibujar objetos afectados
    {
        TouchSenseButtonsMsg( &msg );       // Obeter un mensaje crudo del touch pad
        if ((msg.uiEvent != EVENT_INVALID) && ((tick - displayChangeTime) > MESSAGE_DEAD_TIME))     // Validar el mensaje
        {
            TraducirTouchpad( &msg );       // Traducir el mensaje crudo obtenido
            GOLMsg( &msg );                 // Procesar el mensaje
        }

    }

}
```

Figura. 4.38. Funciones del bucle principal

- La función GOLDraw(), encargada de graficar la pantalla, es responsable de la creación del objeto, lo cual se estudió en la sección 4.1.3.3.[54]
- La función TouchSenseButtonMsg es transparente para el usuario
- En la función TraducirTouchpad se interpretará a que objeto afecta la acción del usuario
- En la función GOLMsg se programará la acción para el objeto que ha sido afectado[55]

Entonces, la programación del objeto se la debe hacer dentro de dos funciones:

- TraducirTouchpad: para identificar si el objeto ha sido afecto o no
- GOLMsgCallback: Para identificar que acción se debe realizar si el objeto fue el afectado

La programación en la función TraducirTouchpad debe seguir los siguientes pasos:

1. La función TraducirTouchpad lleva el parámetro &msg, explicado en la tabla 3.19 en la sección 3.1.3.2
2. Crear una nueva función para cada estado de pantalla que necesite interacción con el usuario a través del teclado, revisar la sección correspondiente a la máquina de estados 3.2.1.3.
3. Construir la función de acuerdo al siguiente ejemplo:

[54] **Esto se realiza a través de la función GOLDrawCallback**
[55] **Esto se realiza a través de la función GOLMsgCallback**

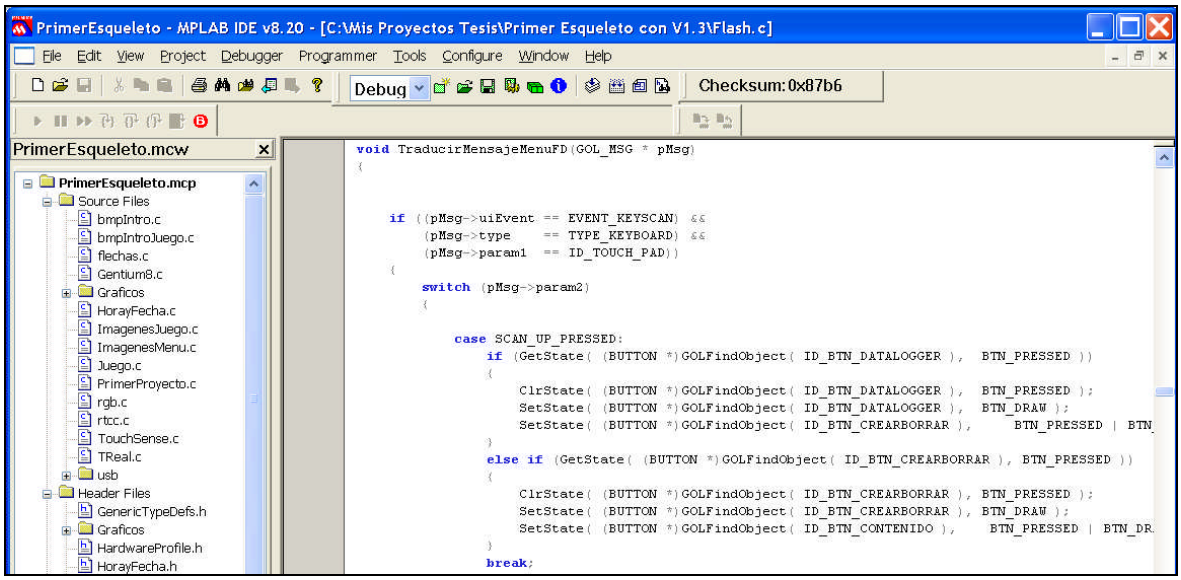


Figura. 4.39. Función dentro de TraducirTouchpad

a. Utilizar el parámetro GOL_MSG * pMsg en la declaración [56]
b. Condicionar el acceso a la función de cumplirse los siguientes pasos
    i. Se registró un evento de acción del usuario
        1. pMsg->uiEvent == EVENT_KEYSCAN
    ii. La acción registrada es del teclado
        1. pMsg->tipo == TYPE_KEYBOARD
    iii. El parámetro 1 indica que es un teclado Touchpad
        1. pMsg->==ID_TOUCH_PAD
c. Dirigir al caso de acuerdo al parámetro 2, es decir, a que acción se ejecutó
d. Sí la acción fue que el botón “Hacia abajo” (botón 3 del Touchpad) fue presionado, seleccionar el botón que corresponde.
    i. Algunos de los eventos dentro del parámetro 2 pueden ser:
        1. SCAN_UP_PRESSED
        2. SCAN_DOWN_PRESSED
        3. SCAN_RIGHT_PRESSED
        4. SCAN_LEFT_PRESSED
        5. SCAN_CR_PRESSED
        6. SCAN_UP_RELEASED
        7. SCAN_DOWN_ RELEASED
        8. SCAN_RIGHT_ RELEASED

[56] **Al leer los pasos, sígalos identificando en la Figura 4.39**

9. SCAN_LEFT_ RELEASED
10. SCAN_CR_ RELEASED

e. Para cada evento se debe:

i. Identificar que botón tiene el foco, es decir, que objeto tiene el estado de presionado(BTN_PRESSED), una vez hecho esto se debe:

1. Eliminar su estado de presionado
2. Establecer su estado como dibujado (BTN_DRAW)
3. Al nuevo botón, establecer un estado de presionado y de dibujado

La **programación en la función GOLMsgCallback** debe incluir la declaración de las funciones que se encargarán de ejecutar en sí la acción para la que han sido creados. Por ejemplo, los botones creados para el menú de gráficos al ser presionados deben ejecutar una acción, la cual es interpretar que el usuario desea ver la pantalla de los gráficos en tiempo real o desea jugar.

Para programar esta función se debe:

1. Crear la función con los parámetros indicados:

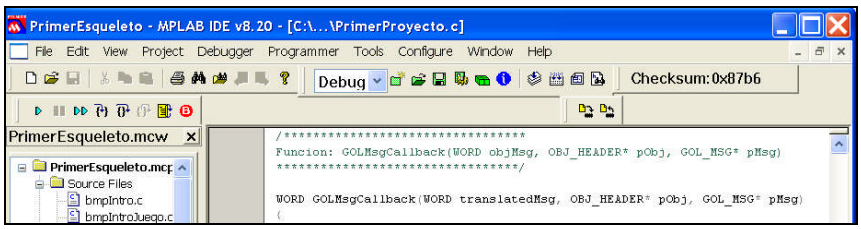


Figura. 4.40. Declaración de GOLMsgCallback

2. Iniciar la máquina de estados con una estructura switch

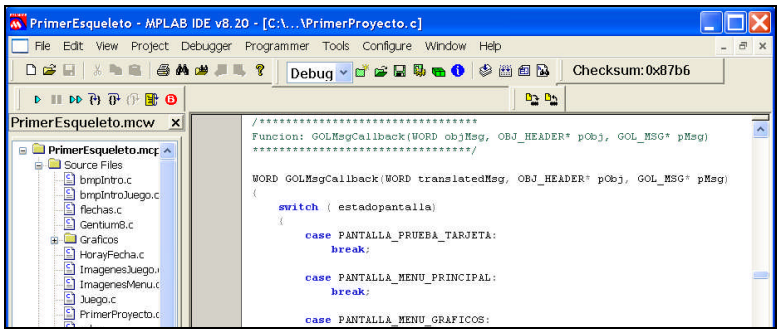


Figura. 4.41. Iniciar la máquina de estados

3. Declarar la función que se encargará de procesar el mensaje del estado correspondiente de la siguiente manera:

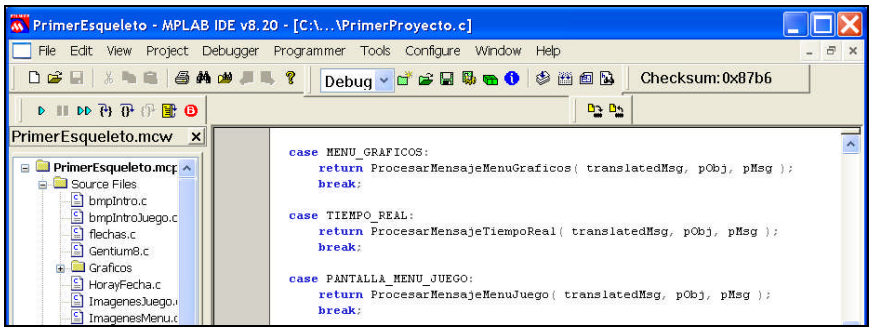


Figura. 4.42. Declaración de las funciones para procesar

4. Estructurar la función en el archivo .c correspondiente a la aplicación y declararla como prototipo en el archivo .h:

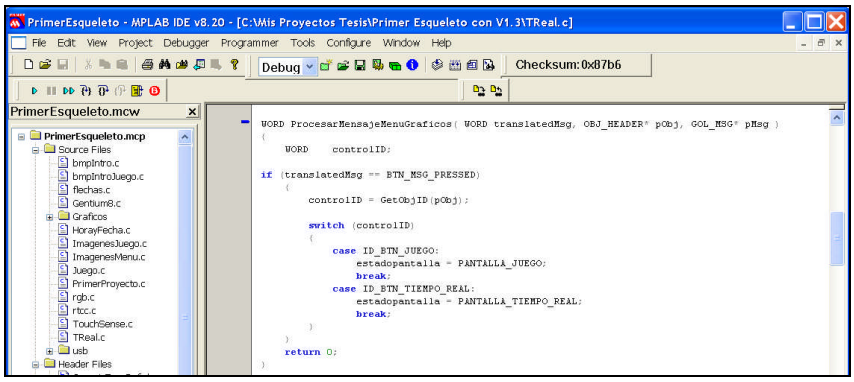


Figura. 4.43. Estructura de la función para procesar

5. Debe recordarse, que la librería de gráficos es una API que trabaja con un sistema de mensajería, por esta razón, al pasar el mensaje (GOL_MSG *pMsg) se puede saber que objeto fue seleccionado en la función TraducirTouchpad.

4.1.4. TUTORIAL BÁSICO PARA EL USO DE LA LIBRERÍA MCHPSUSB V2.4

El uso de esta librería es más sencillo, ya que esta no trabaja con un sistema de mensajería; sin embargo, es también una API así que debe cumplir ciertos requisitos. Esta librería es de uso transparente para el usuario, es decir, solo se requiere saber las definiciones de las funciones, cuales y que parámetros enviar.

4.1.4.1. Archivos que se deben agregar al proyecto. Se recomienda la lectura de los siguientes puntos:

1. Agregar los archivos de código .c:
    a. FSIO.c: aquí se encuentran escritas las funciones FS que se van a usar
    b. usb_config.c: donde se encuentra definida la Media Interface Function Pointer Table para el Mass Storage client driver
    c. usb_host.c: provee la interface de hardware para una aplicación Host USB
    d. usb_host_msd.c: este archivo es el Mass Storage Class driver para un dispostivo USB Embedded Host.
    e. usb_host_msd_scsi.c: este archivo provee la interface entre el archive del sistema y la clase USB Host Mass Storage. Traduce los requerimientos de la funcionalidad del sistema a los comandos SCSI adecuados y envía los comandos SCSI por la clase USB Mass Storage.
2. Agregar los archivos de cabecera .h:
    a. FSconfig.h: representan la configuración para el correcto uso de las funciones.
    b. FSDefs.h: incluye las declaraciones y definiciones necesarias para las funciones FS.
    c. FSIO.h: es el archivo cabecera para FSIO.c
    d. USB PIC24.h: este archivo define los ítems específicos para el PIC24.
    e. usb.h: Este archivo agrega todos los archivos cabecera necesarios para Microchip USB Host y las librerías OTG.
    f. usb_ch9.c: Este archivo define las estructuras de datos, constantes y macros que son usadas para dar soporte al protocolo framework USB para el dispositivo en el capitulo 9 de la especificación USB 2.0

g. usb_common.h: Este archivo define tipos, constantes y macros que son comunes multiples capas de Microchip USB Firmware Stack.
h. usb_config.h: es el archivo cabecera para usb_config.c
i. usb_host.h: es el archivo cabecera para usb_host.c
j. usb_host_local.h: Este archivo provee definiciones locales usadas para la interface del hardware para una aplicación Host USB.

4.1.4.2. Las funciones FS. Las funciones FS, como son referidas en este tutorial, son las funciones creadas en el framework UBS MCHPSUSB por Microchip ®. Estas funciones son lo único que se requiere conocer del framework, es lo único que no es transparente. A pesar de esto, no solo se necesita conocer su sintaxis, también se necesita saber que valores entregan y que significan dichos valores.

Es recomendable seguir estos pasos:

1. Leer el documento AN1145 Using a USB Flash Drive on an Embedded Host, disponible en www.microchip.com/usb
2. Identificar las funciones FS que van a ser usadas. Estructurarlas de acuerdo a los ejemplos dados.
   a. Por ejemplo: función FSfopen (abrir/crear un archivo) y FSFwrite (Escribir en un archivo)

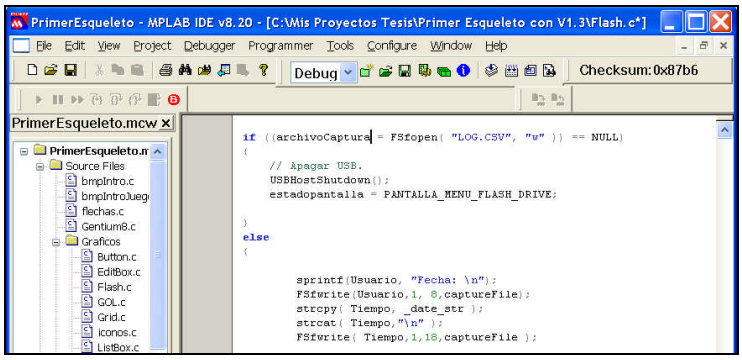


Figura. 4.44. Ejemplo funciones FSfopen y FSfwrite.

b. Llamar a las funciones de acuerdo a los ejemplos dados en estructuras if o igualándolas a punteros a archivos[57].

i. Estructuras if:

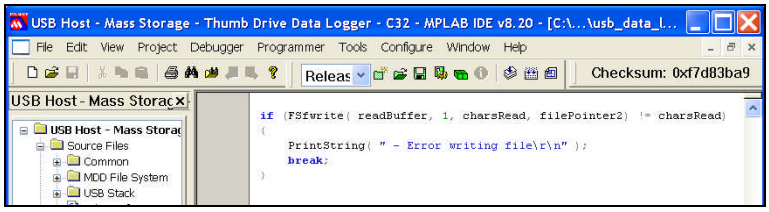


Figura. 4.45. Estructura if para función FS

ii. Igualar función FS a puntero a archivo

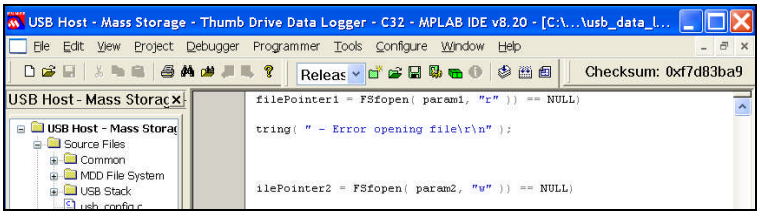


Figura. 4.46. Igualar puntero a función FS

Las funciones existentes son:

- **FSchdir().-** Cambia el directorio actual
- **FindFisrt().-** Esta función es útil para encontrar el archivo con el que se desea trabajar
- **FSremove().-** Borra el archivo especificado
- **FSmkdir().-** Crea un directorio con el nombre dado
- **FSrmdir().-** Borra el directorio especificado
- **FSfopen().-** Abre/crea un archivo
- **FSfread().-** Lee el contenido de archivos
- **FSfeof().-** Indica el final del archivo
- **FSfclose().-** Cierra el archivo

[57] **Las imágenes 4.46 y 4.45 fueron tomadas del proyecto de libre distribución creado por Microchip USB Host – Mass Storage – Thumb Drive Data Logger**

Ahora, las funciones se encuentran detalladas en el archivo FSIO.c tal como se muestra en la Figura 4.47:

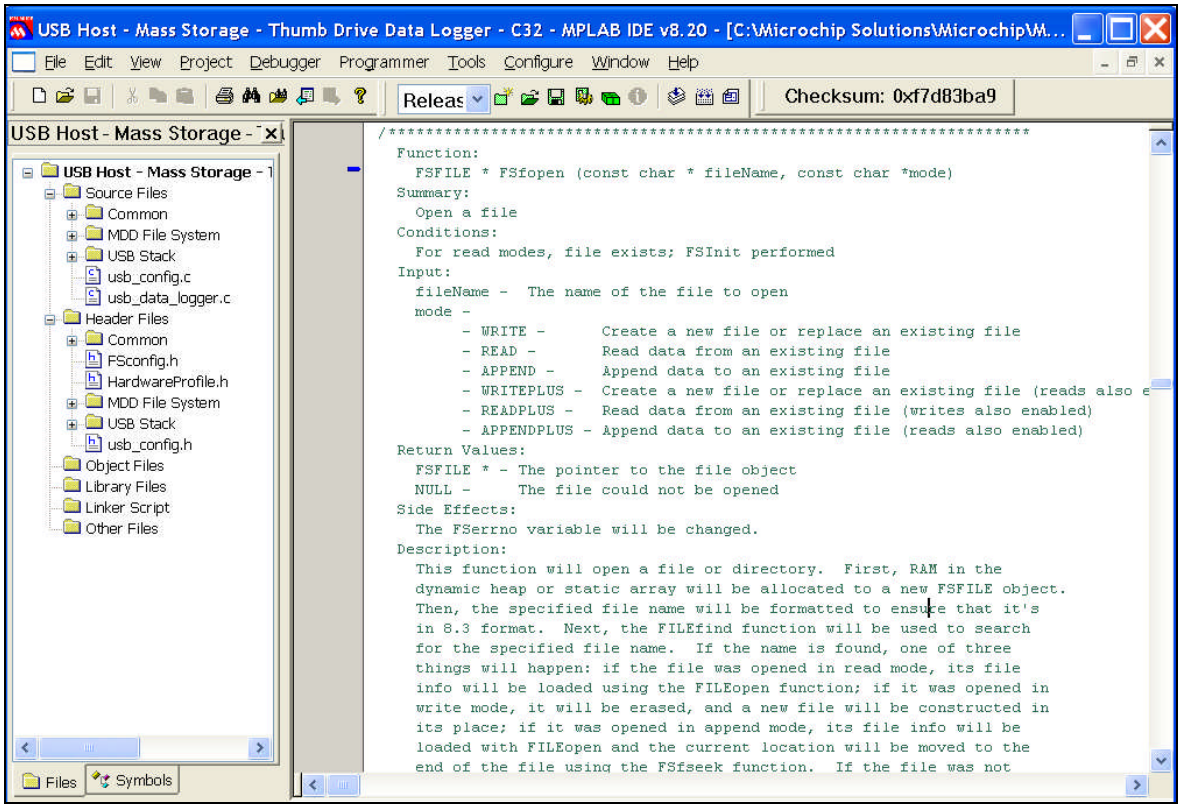


Figura. 4.47. Detalle de la función FSfopen en el archivo FSIO.c

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- Este proyecto ha demostrado que dentro del entorno universitario es posible comprender y manejar tecnologías de punta como conectividad USB On-The-Go y pantallas OLED. Además, se desarrolló un sistema de referencia para el estudio y desarrollo de futuros proyectos, que puedan incluir esta tecnología y aplicarla en otros campos.

- El trabajo con una estructura de capas de software, APIs e interfaces de mensajería brinda ventajas como: facilidad para crear nuevas aplicaciones sobre una arquitectura definida, portabilidad de la aplicación entre periféricos, robustez del diseño y mejor aprovechamiento de los recursos del microprocesador.

- La metodología Hardware/Software Partition es el primer paso para diseñar un sistema embebido. Al iniciar este proyecto, ayudó a la planificación de recursos y tiempo para el desarrollo del dispositivo.

- La mayor parte de la API de gráficos es transparente al usuario debido a su modelo por capas. Este hecho permite al usuario iniciarse sin mayor problema en el uso de las funciones para creación de objetos, traducción de la acción y procesamiento de la acción; sin embargo, una lectura y comprensión del flujo de mensajería de la librería es indispensable para elaborar una aplicación de manera eficiente que ahorre tiempos en ejecución y demás recursos.

- Las funciones callback de la función GOLDraw y GOLMsg (GOLDrawCallback y GOLMsgCallback) y la función TraducirTouchpad, son las que el usuario de la

librería debe implementar de manera obligatoria. Se concluyó que funcionan mejor con una máquina de estados que separe estados independientes para cada función; así, se puede distinguir las acciones que se debe desarrollar en cada una de estas tres funciones.

- Debido al flujo de la librería, es necesario la creación de dos estados por cada situación de la aplicación, uno para que la librería pueda graficar la pantalla y otro para que la librería pueda llamar a las funciones encargadas de procesar la acción que la situación demanda.

- Los diseñadores de sistemas embebidos consideran incluir alguna forma de interfaz gráfica como una característica de facto en sus diseños. En este proyecto se logró una retroalimentación amigable al usuario mediante la pantalla OLED. Esto fue detallado en el sistema de referencia elaborado en el proyecto.

- La demanda de memoria por las librearías Graphics Library v1.65 y MCHPFSUSB v2.4, en este proyecto fue: 13Kb, equivalente al 15% de los 87Kb disponibles para memoria de programa; 0.26Kb, equivalente al 1.4% de los 16Kb de la memoria de datos. En consecuencia, el gasto en memoria podría significar una relación costo-beneficio favorable para el desarrollo de aplicaciones, desde las más simples hasta las más complejas.

- La inclusión de la conectividad USB On-The-Go, en su configuración como Embedded Host, en este y cualquier proyecto dotará al dispositivo con capacidad de almacenamiento virtualmente ilimitada en relación a sus necesidades. Anteriormente, para los sistemas embebidos esto era casi imposible de alcanzar.

- En comparación con otras tarjetas, tales como: CY3663 - EZ-OTG / EZ-Host Development Kit, KIT USB de Digi-Key o Explorer 16 de Microchip®; el Starter kit PIC24F presenta la principal ventaja del costo económico. Los precios son $2495 (contiene muchísimos más recursos), $302 y alrededor de $150 respectivamente, y el Starter kit en $60. Obviamente, las demás tarjetas contienen

más recursos que también se exploran. En conclusión, para aprender el uso de las tecnologías y librería de Gráficos y USB, el Starter kit es la mejor opción.

- Dentro de la carrera de la tecnología, desde el punto de vista del estudiante, se puede observar una competencia por las computadoras portátiles "laptos" de cada vez abaratar costo, tamaño y demanda de energía; hasta el punto de ser casi un dispositivo de bolsillo. Por otro lado, los sistemas embebidos ahora poseen pantallas OLED a color, teclados y poderosos procesadores, asemejándose casi a una pequeña computadora. ¿Cuál será entonces la línea que marque la división entre ambos? seguramente tendrá que ver con las aplicaciones a las que se les dedique y también con quien logre acercarse más a la utopía en consumo de batería.

5.2. RECOMENDACIONES

- Se recomienda la lectura de los documentos AN1136A y AN1145A y prestar especial atención al flujo de la mensajería de la API de gráficos. Solo con este conocimiento se puede crear una nueva aplicación que funcione eficientemente, no solo para el uso del Starter Kit PIC24F, sino también para cualquier microcontrolador de 16 u 8 bits que se pretenda usar en sistema embebido que maneje gráficos.

- Para usuarios o desarrolladores que estén empezando a explorar los microcontroladores se recomienda utilizar la tarjeta Explorer 16 Development board. No se recomienda el uso de la tarjeta Starter Kit PIC24F ya que, como se mencionó anteriormente, ésta no permite la exploración de otros periféricos más que el USB OTG.

- La tarjeta Starter Kit PIC24F es un instrumento desarrollado para la exploración de los módulos USB On-The-Go, la capacidad de generar gráficos y el uso de teclados capacitivos lo que la convierte en una herramienta de estudio versátil y poderosa. No posee los conectores físicos para comunicaciones como: UART, SPI, I^2C, etc.

- Para desarrolladores que empiecen el uso de la tarjeta usada en este proyecto, se recomienda que la primera acción que traten de ejecutar es presentar algo en pantalla, ya sea un botón, un slider o cualquier otro objeto sin necesidad de que éste ejecute ninguna acción. En un sentido más técnico esto se refiere a que se logre programar la función GOLDrawCallback con una función de la librería de gráficos para creación de objetos.

- Se recomienda que se desarrolle una aplicación que grabe datos en tiempo real una Flash drive en tiempo real. Específicamente, se recomienda desarrollar el complemento a la aplicación “Tiempo real” para que las mediciones del potenciómetro se graben a un archivo .csv en la flash drive insertada a la tarjeta.

- Se recomienda el uso de lenguaje C30 sin importar cuan experto sea el desarrollador en ANSI, el uso de las librerías de gráficos y USB son mucho más eficientes en C30.

- Es recomendable mantener la misma estructura del código provisto en el demo del Starter kit PIC24F, ya que cumple con los requerimientos para trabajar con la API de gráficos. Además, presenta la facilidad de que las funciones que requieren iniciar los puertos, conversores ADC, modulación PWM, el módulo CTMU y más, ya se encuentran escritas y no es necesario ver los diagramas esquemáticos de las conexiones para saber a que pines están conectados los elementos de hardware.

- Debe estudiarse bien los tipos de variables que son definidos en el archivo GenericTypeDefs.h para comprender los tipos de datos que han sido construidos. La estructura del mensaje GOL_MSG también se recomienda que sea cuidadosamente estudiada ya que por esta se maneja la comunicación entre las capas.

REFERENCIAS BIBLIOGRÁFICAS

[1] Microchip, *Historia de los Semiconductores*, http://www.micron.com/k12, Diciembre 2009.

[2] Microchip, Appl. Note1136, pp.1-3.

[3] Inventors Inc., *Invención de los teclados touch*, http://inventors.about.com.

[4] J. Axelson, *USB Complete*, Lakeview, Madison USA 2001.

[5] USB.org, *Documento de introducción a USB OTG*, www.usb.org.

[6] Wikipedia, *Artículo sobre la Arquitectura Harvard*, www.wikipedia.org/wiki.

[7] R. Aravia, Curso PIC24, Lección 1, Microchip.

[8] Microchip, Datasheet PIC24FJ256GB106, 2008.

[9] Microchip, PIC24F User's guide, 2008.

[10] "*Foros de electrónica referente a los ICDs*", http://www.electro-tech-online.com/.

[11] Microchip, "*Pagina Web oficial de Microchip*®", www.microchip.com.

[12] "*Herramientas para depurar sistemas embebidos*", www2.electronicproducts.com.

[13] "D*epurar sistemas embebidos*", www2.electronicproducts.com/Debugging_tools.

[14] Microchip, Starter Kit PIC24F Tutorial, 2006.

[15] Microchip, Appl. Note1101A, pp.1-5.

[16] Wikipedia, "*Tecnología OLED*", www.wikipedia.org/OLED.

[17] "Fundamentos de la tecnología OLED", www.scribd.com/doc/13325893.

[18] Microchip, "*Presentación de USB OTG*", http://search.microchip.com/search OTG presentation&ac=1.

[19] Microchip, Notas sobre la familia PIC24FJ256GB110, Sección 11. Charge Time Measure Unit.

[20] Microchip, MPLAB IDE Quick Start Guide, 2007, pp 1.

[21] Microchip, MPLAB IDE Quick Start Guide, 2007, pp 6.

[22] MPLAB IDE v8.20 Help, Languaje tools.

[23] M. Barr, "*Arquitectura de firmware en cinco pasos fáciles*", Embedded.com, http://www.embedded.com/design/testissue/220100399, Agosto 2009.

[24] "*Artículo para el diseño de sistemas embebidos*", http://www.embedded.com.

[25] A. Berger, "Real Time Embedded System Desing – An Introduction to Processes, Tools and Techniques", CMP Books, Kansas USA, 2002, pag 22.

[26] A. Berger, "Real Time Embedded System Desing – An Introduction to Processes" Tools and Techniques, CMP Books, Kansas USA, 2002, pag 53.

[27] http://www.pcmag.com/encyclopedia_term/0,2542,t=application+programming+interface &i=37856,00.asp, Definición de API.

[28] www.PCMagazine.com, Definición de API.

[29] Microchip, Graphics Library Help, 2009.

[30] Wikipedia, "*Funciones callback*"www.wikipedia.org/Callback.

[31] "*Introducción a las declaraciones typedef*", http://bcook.cs.georgiasouthern.edu.

[32] http://www.ite.educacion.es/w3/eos, La construcción de la perspectiva.

[33] Microchip, "*Material sobre USB OTG*", www.microchip.com/usb.

[34] Microchip, Appl. Note1145A, pp.1-7.

[35] Microchip, MCHPFSUSB v2.3, 2008.

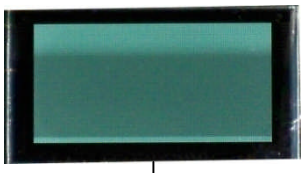
[36] Microchip, MPLAB C30 User's guide, pp. 8-10.

ANEXOS

ANEXO 1: DIAGRAMA DE LA APLICACIÓN USANDO LAS PANTALLAS

El diagrama de la aplicación usará las pantallas para indicar el flujo que tiene:

Encendido de la tarjeta



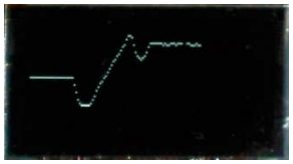
Logo de introducción

Menú principal

Opción Gráficos

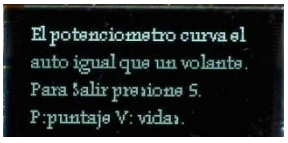
Menú Gráficos

Juego



Tiempo real

Intro



Instrucciones

En sus marcas…

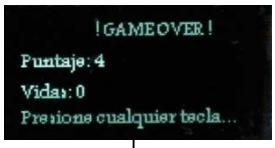
listos…

fuera!

En recta

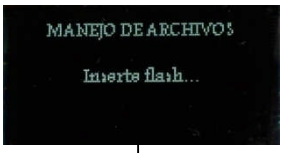
Curva Der.

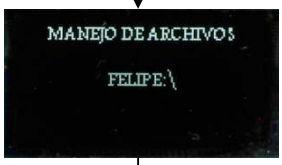
Curva Izq.

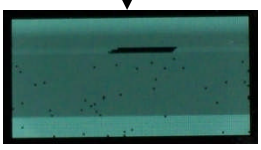


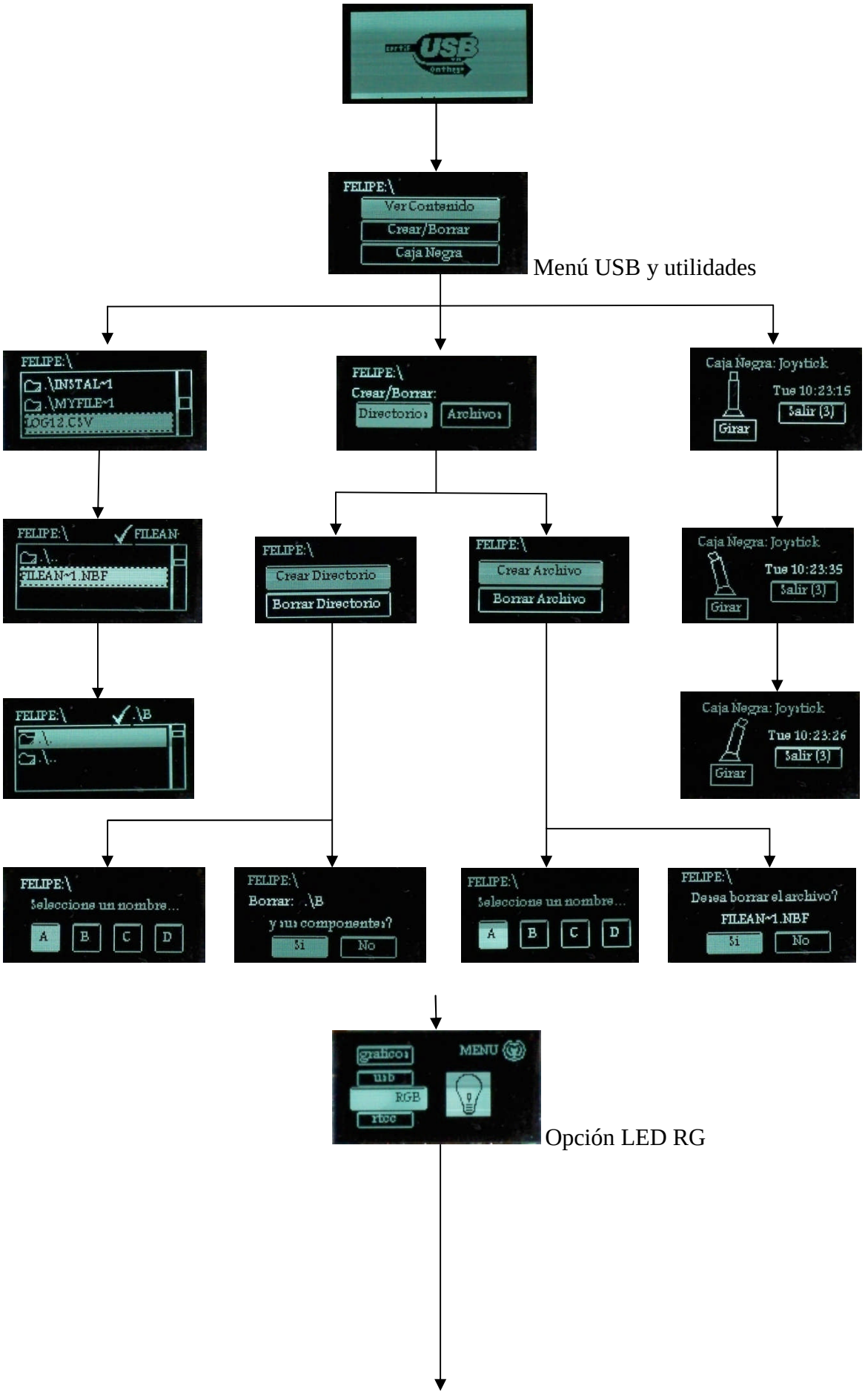
Fin

Opción USB








USB
FELIPE:\
Ver Contenido
Crear/Borrar
Caja Negra
Menú USB y utilidades
FELIPE:\
.\INSTAL~1
.\MYFILE~1
LOG12.CSV
FELIPE:\
Crear/Borrar:
Directorio
Archivo
Caja Negra: Joystick
Tue 10:23:15
Salir (3)
Girar
FELIPE:\
FILEAN~1.NBF
FELIPE:\
Crear Directorio
Borrar Directorio
FELIPE:\
Crear Archivo
Borrar Archivo
Caja Negra: Joystick
Tue 10:23:35
Salir (3)
Girar
FELIPE:\
.\B
Caja Negra: Joystick
Tue 10:23:26
Salir (3)
Girar
FELIPE:\
Seleccione un nombre...
A
B
C
D
FELIPE:\
Borrar: .\B
y sus componentes?
Si
No
FELIPE:\
Seleccione un nombre...
A
B
C
D
FELIPE:\
Desea borrar el archivo?
FILEAN~1.NBF
Si
No
grafico
MENU
RGB
Opción LED RG


Azul
Verde
Rojo

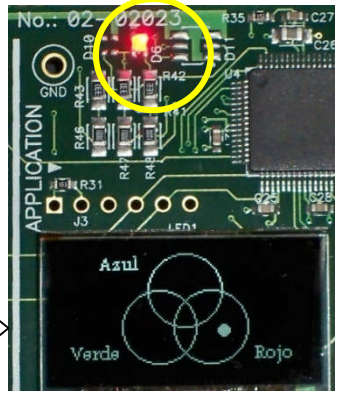

APPLICATION
Azul
Verde
Rojo


grafico1
MENU
usb
rgb
RTCC
31

Opción RTCC


2007
Nov
01
01
2
Igualar

ANEXO 2: DIAGRAMAS ESQUEMÁTICOS DEL STARTER KIT PIC24F

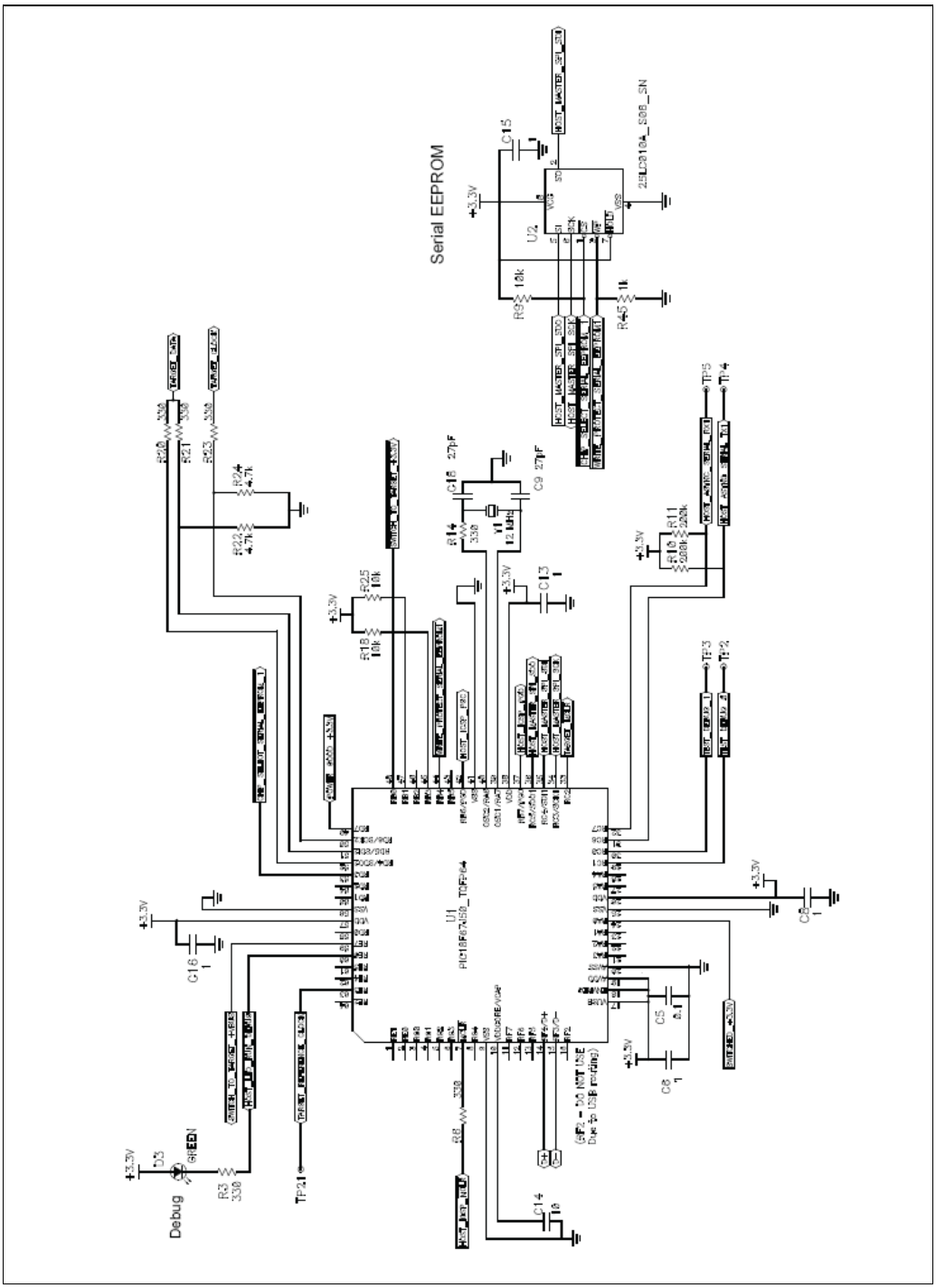
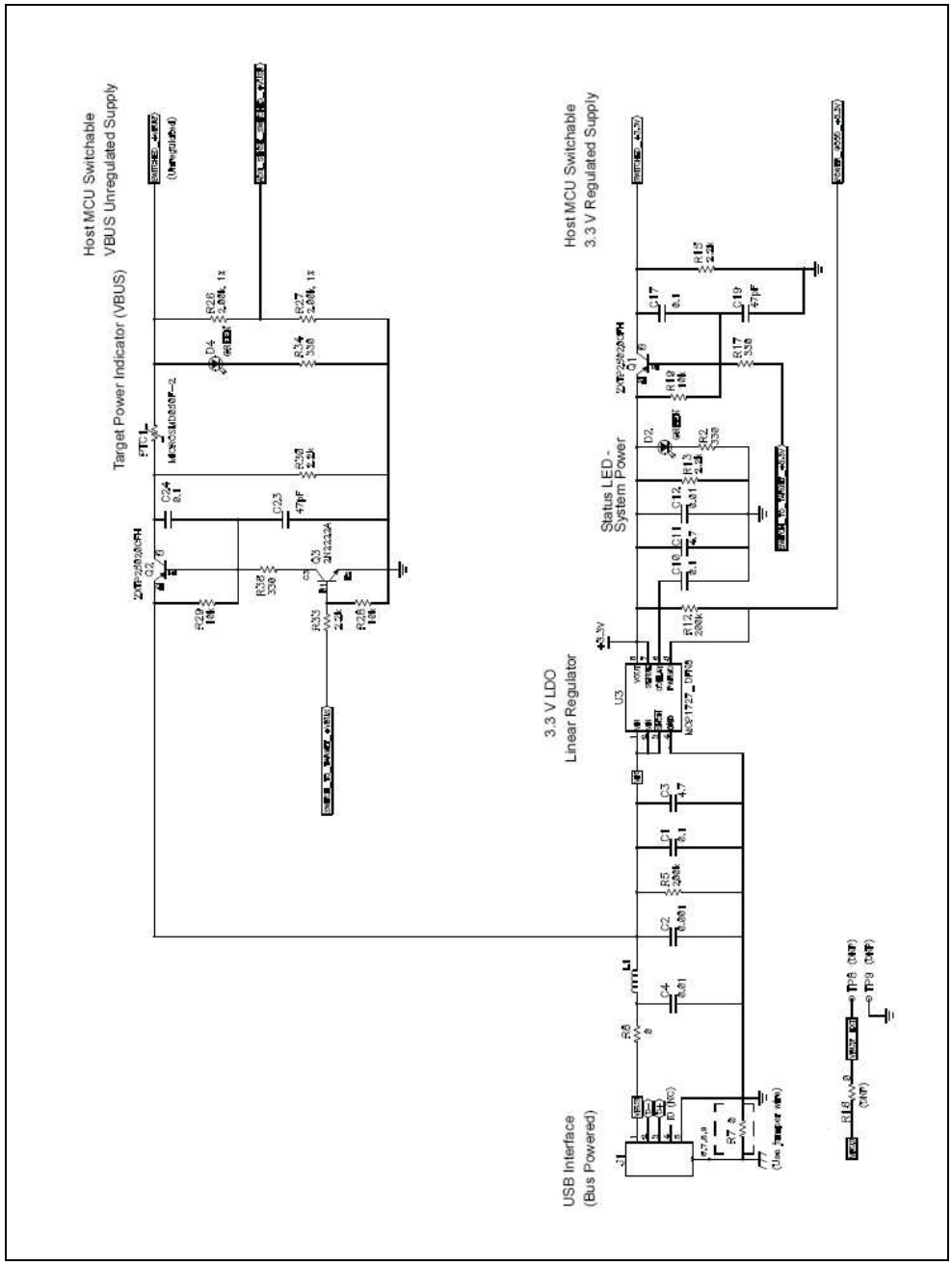


Figura. A2-1. Sistema de control del PROGRAMMER/DEBUGGER y EEPROM



Fugra. A2-2. Interface USB PROGRAMMER/DEBUGGER, regulación y switching del poder suministrado a la tarjeta

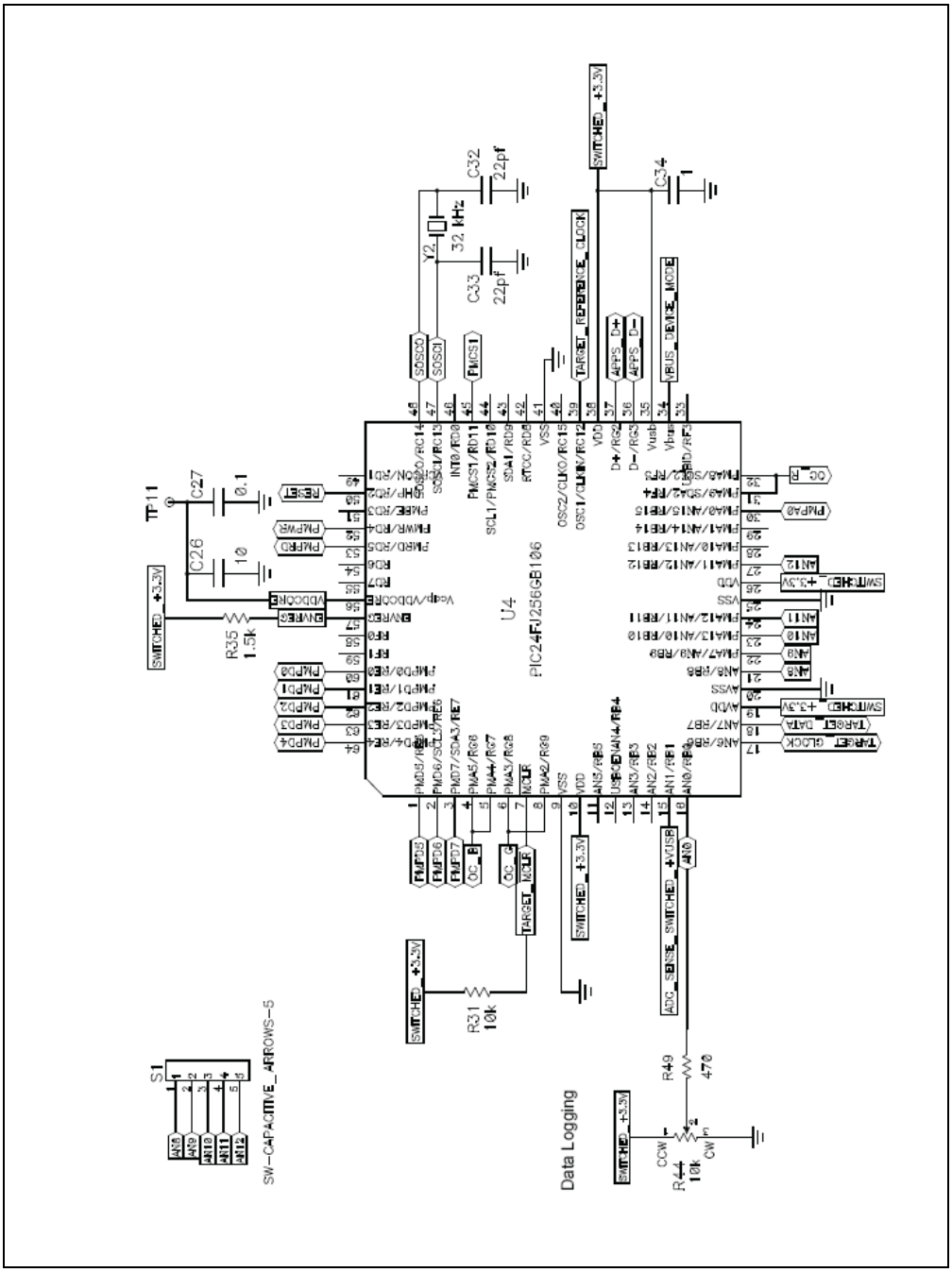


Figura. A2-3. Microcontrolador de la aplicación, touch switches y componentes asociados

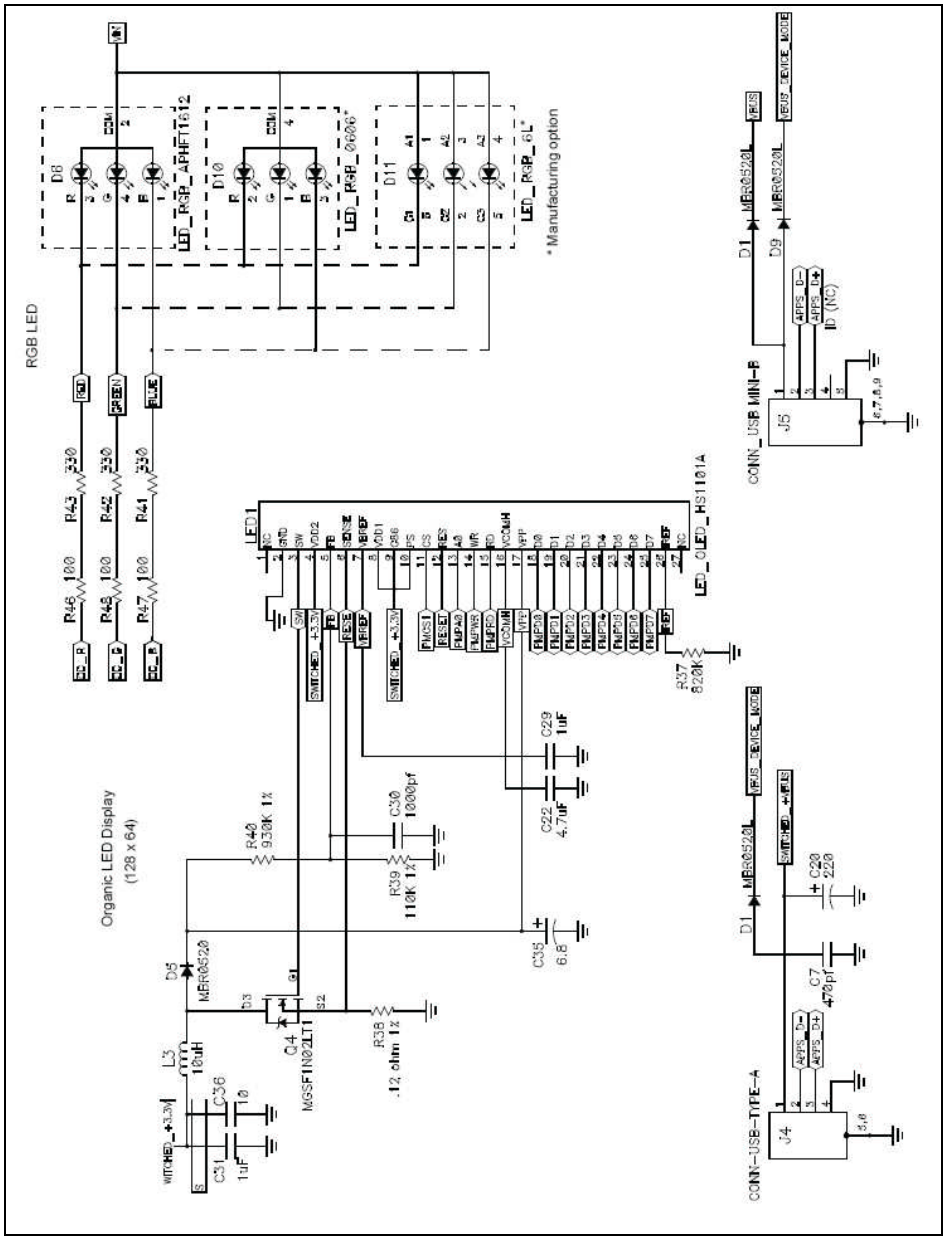


Figura. A2-4. Display OLED, LED y conectores USB del costado de la aplicación

INDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

API.- Application Programming Interface. Interface desarrollada para programar aplicaciones contando con una estructura predefinida.

Arquitectura Harvard.- Arquitectura moderna para procesadores que se caracteriza por la separación de los buses de memoria y de instrucciones.

Arquitectura Von Neuman.- Arquitectura de los primeros procesadores que utiliza un mismo bus para accesar a datos y memoria.

Bit.- Unidad de memoria mas pequeña usada en lógica binaria o discreta, donde el valor 1 representa encendido y 0 representa apagado.

Build.- Operación de software para verificar que el firmware desarrollado no tenga incongruencias ni fallas de programación.

Byte.- Unidad que representa 8 bits.

dsPIC.- Microcontrolador que posee un módulo para el procesamiento de señales (dsp).

EEPROM.- *Electrically-Eraseble Programmable Read-Only Memory*. Memoria programable solo de lectura borrable electrónicamente.

Firmware.- Es el software encargado de ejecutar, en la misma circuitería del procesador, instrucciones emitiendo órdenes a otros dispositivos del sistema.

Flash.- Tecnología de memoria no volátil. Más barata que la tecnología EEPROM, tiene la desventaja de solo grabar/borrar por bloques.

Framework.- Estructura de software para el desarrollo de aplicaciones. Puede o no constar de capas. Generalmente consta de funciones que están construidas en niveles inferiores que son transparentes al usuario.

Heap.- Es la pila de bytes que reserva el IDE para el software embebido. Se recomienda establecerla mayor a 1024.

ICD.- *In-Circuit-Debugger*. Es la herramienta que permite la depuración del software paso a paso para detección de errores y verificación del mismo.

IDE.- *Integrated-Development-Environment*. Ambiente integrado para desarrollo. Es un programa para PC que reúne en una misma pantalla varias características necesarias para el desarrollo de software.

Link.- Proceso por el cual se enlazan los archivos necesarios para el funcionamiento de un proyecto. Permite por ejemplo, que las funciones pueden ubicarse sí se encuentran en archivos .c diferentes.

MCU.- *Micro Controller Unit*. Unidad micro controladora.

Módulo.- Sub sistema que incluye las funciones necesarias para determinado proceso. Trabajan dependiente del núcleo del procesador.

Objeto.- Dentro de la librería de gráficos se refiere a un botón, slider, cuadro de texto, etc. Maneja propiedades y estados dependiendo de cada tipo.

Output.- Salida, en el IDE representa el Status del proyecto que se esta desarrollando. Se podría decir que es la salida del ICD.

PWM.- *Pulse Width Modulation*. Modulación por ancho de pulsos.

RISC.- *Reduced Instruction Set Computer*. Set de instrucciones reducido para computadora, se maneja para lenguaje assembler. Para el caso de los PIC24F, éste consta de 70 instrucciones.

Sistema embebido.- Sistema para propósito específico que casi siempre consta con los mismos componentes que una computadora, esta completamente encapsulado.

Software embebido.- Software desarrollado para ser incluido en un sistema embebido. Consta con requerimientos bien delimitados, capacidad de memoria no mayor a la necesaria y robustez en el programa. Además, su desarrollo siempre apunta al ahorro de energía.

Tiempo real.- Modo de trabajo de un programa en el que la reacción (cómputos respecto al contexto) ante una acción no demora más que el retardo permitido para una aplicación dada, de tal manera que se perciba “una reacción instantánea del dispositivo”.

USB OTG.- Interfaz USB On-The-Go. Desarrollada para dispositivos móviles que no dependan de una PC. El protocolo fue desarrollado en base a la especificación USB 2.0 haciendo ciertas modificaciones en cuanto a consumo de energía.

Widget.- Término usado informalmente para describir elementos de diseño gráfico tales como iconos, botones, slider, edit box, etc.

ÍNDICE DE DATASHEETS

AN1136A: How to Use Widgets in Microchip Graphics Library (Primera página)

MICROCHIP

AN1136

How to Use Widgets in Microchip Graphics Library

Author: *Paolo Tamayo*
Anton Alkhimenok
Microchip Technology Inc.

INTRODUCTION

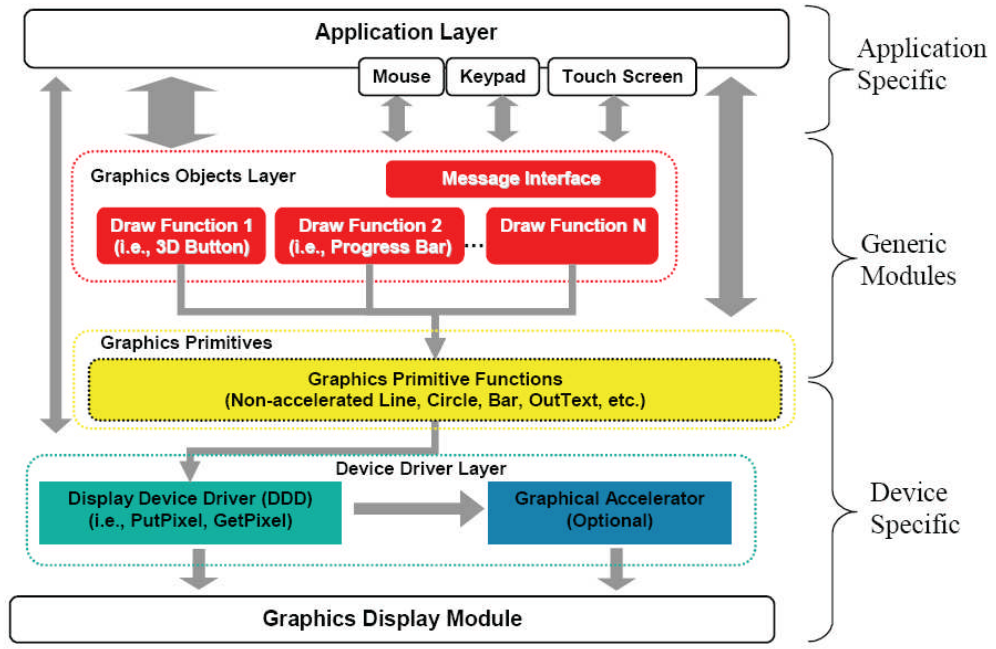
The proliferation of graphical interfaces in ordinary devices is becoming noticeable. As we go along our daily activities, more and more products we encounter have some form of graphical interface. As this feature becomes a de facto standard, the need to manufacture these devices at a lower cost becomes apparent. PIC® microcontrollers, with their reputation for low risk product development, lower total system cost solution and faster time to market, makes this realizable. The free Microchip graphics library makes it very easy to integrate graphical features in an application. This application note details how a 16-bit microcontroller with a graphical library is used to drive a QVGA display supporting 16-bit colors.

For details about the PIC24F family of microcontrollers, refer to the *"PIC24FJ128GA010 Family Data Sheet"* (DS39747). For details of the Graphic Library API, please refer to the *"Microchip Graphics Library API"* documentation included in the installer of the library.

OVERVIEW OF THE GRAPHICS LIBRARY

The Microchip Graphics Library was created to cover a broad range of display device controllers. Targeted for use with the PIC microcontrollers, it offers an Application Programming Interface (API) that performs rendering of primitive graphics objects as well as advanced widget-like objects. The library also facilitates easy integration of input devices through a messaging interface. Applications created using the library will also find a simple and straightforward process to change display devices if the need arises. The layered architectural design of the library makes all of these possible. The Microchip Graphics Library structure is shown in Figure 1.

FIGURE 1: TYPICAL SYSTEM WITH MICROCHIP GRAPHICS LIBRARY



AN1145A: Using a USB Flash Drive with an Embbeded Host (Primera página)

MICROCHIP

AN1145

Using a USB Flash Drive with an Embedded Host

Author: *Kim Otten*
Microchip Technology Inc.

INTRODUCTION

USB Flash drives are a popular, simple and inexpensive method of moving data from one PC to another. Their use in the embedded market has been limited, however, due to the requirement that a system must have USB host capability to communicate with a Flash drive.

In the past, this usually meant that the system needed to be a PC. However, the introduction of Microchip's PIC® microcontrollers with USB embedded host capability means that embedded systems can now take advantage of this popular portable media. With the ability to store data to a USB Flash drive, a PIC microcontroller-based application now has virtually unlimited data storage.

This application note demonstrates a data logger application that can run on the Explorer 16 Demo Board with the USB PICtail™ Plus Daughter Board. It implements a file system with a simple, but powerful, set of commands.

A Note About USB Flash Drives

USB Flash drives come in a wide variety of shapes and sizes. Many of the Flash drives, with up to 2 GB of memory, utilize the FAT16 file system and a Small Computer System Interface (SCSI) command interface. Microchip Application Note *AN1045, "Implementing File I/O Functions Using Microchip's Memory Disk Drive File System Library"* (DS01045), describes an implementation of this file system.

Note: Flash drives with more than 2 GB of memory utilize the FAT32 file system, which is not supported by AN1045.

THE DATA LOGGER APPLICATION

This application stores two types of data:

- Low data rate monitoring. This is done by polling the on-board potentiometer approximately once per second. The potentiometer reading and time and date stamp of the reading are saved to a file on the Flash drive.
- Higher speed time measurement accuracy. This is done by reading the temperature sensor every 10 ms. The temperature reading and the count of elapsed milliseconds are saved to a file on the Flash drive.

The application also provides a set of simple commands to interface to the Flash drive (via a serial terminal interface) and directly manipulate files on the Flash drive.

Installing the Stack

The USB data logger application is available as part of Microchip's complete USB Embedded Host Support Package (see **Appendix A: "Software Discussed in this Application Note"** for more details). To install all the necessary project files on a host PC, download the installation file from the Microchip web site and run the executable installer file. By default, the project and Stack files will be installed in the directory structure shown in Figure 1.

DATASHEET PIC24FJ256GB106 (Primera página) [58]

MICROCHIP

PIC24FJ256GB110 FAMILY

64/80/100-Pin, 16-Bit Flash Microcontrollers with USB On-The-Go (OTG)

Power Management:

- On-Chip 2.5V Voltage Regulator
- Switch between Clock Sources in Real Time
- Idle, Sleep and Doze modes with Fast Wake-up and Two-Speed Start-up
- Run mode: 1 mA/MIPS, 2.0V Typical
- Sleep mode Current Down to 100 nA Typical
- Standby Current with 32 kHz Oscillator: 2.5 µA, 2.0V typical

Universal Serial Bus Features:

- USB v2.0 On-The-Go (OTG) Compliant
- Dual Role Capable – can act as either Host or Peripheral
- Low-Speed (1.5 Mb/s) and Full-Speed (12 Mb/s) USB Operation in Host mode
- Full-Speed USB Operation in Device mode
- High-Precision PLL for USB
- Internal Voltage Boost Assist for USB Bus Voltage Generation
- Interface for Off-Chip Charge Pump for USB Bus Voltage Generation
- Supports up to 32 Endpoints (16 bidirectional):
  - USB Module can use any RAM location on the device as USB endpoint buffers
- On-Chip USB Transceiver with On-Chip Voltage Regulator
- Interface for Off-Chip USB Transceiver
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- On-Chip Pull-up and Pull-Down Resistors

High-Performance CPU:

- Modified Harvard Architecture
- Up to 16 MIPS Operation at 32 MHz
- 8 MHz Internal Oscillator
- 17-Bit x 17-Bit Single-Cycle Hardware Multiplier
- 32-Bit by 16-Bit Hardware Divider
- 16 x 16-Bit Working Register Array
- C Compiler Optimized Instruction Set Architecture with Flexible Addressing modes
- Linear Program Memory Addressing, Up to 12 Mbytes
- Linear Data Memory Addressing, Up to 64 Kbytes
- Two Address Generation Units for Separate Read and Write Addressing of Data Memory

Analog Features:

- 10-Bit, Up to 16-Channel Analog-to-Digital (A/D) Converter at 500 ksps:
  - Conversions available in Sleep mode
- Three Analog Comparators with Programmable Input/ Output Configuration
- Charge Time Measurement Unit (CTMU)

[58] **El DATASHEET del PIC24FJ256GB106 contiene 328 hojas, las AN1136 y AN1145 tienen 22 y 14 hojas respectivamente, por este motivo solo se incluyó la primera página de cada uno.**

ELABORADO POR:

_____

Sr. Luis Felipe Urresta Cueva

DIRECTOR DE CARRERA:

_____

Ing. Víctor Proaño

FECHA DE ENTREGA:

__ de Marzo del 2010