Informe de avance – detalle Diagrama de Flujo

MPPT

Diagrama de Flujo / Sistema Demo TI

Fecha: 10-10-2016

Rev. 2.2

Fuentes, Cecilia

Tutor: Ing. Gonzalez, Leonardo

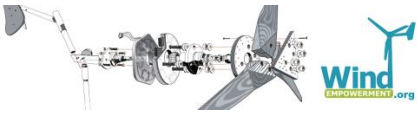
Director: Ing. Oliva, Rafael

Software development internship - Project SEPS- Modular Power for the People, Financed by Wuppertal Institute /WISIONS and WindEmpowerment.org at UNPA-UARG and L&R Ingeniería in Río Gallegos, Argentina – August to October 2016

Contenido

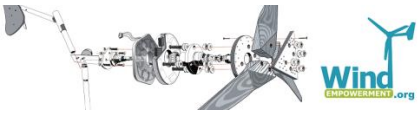
ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS



ABREVIATURAS Y ACRÓNIMOS

ADC	Conversor Analógico Digital
ADU	Unidad de Datos de Aplicación
ASCII	Código Estándar Estadounidense para el Intercambio de Información
GUI	Interfaz Gráfica de Usuario
ISR	Rutina de Servicio de Interrupción
PDU	Unidad de Datos de Protocolo
PWM	Modulación por Ancho de Pulsos
RTU	Unidad Terminal Remota
TCP	Protocolo de Control de Transmisión
WDT	WatchDog Timer o Temporizador Perro guardián



INTRODUCCION

Desde hace algunos años el Área Energías Alternativas de UNPA participa en la red WindEmpowerment (WE, 2016), que es una organización civil internacional que fomenta el conocimiento compartido y la utilización de fuentes renovables (sobre todo eólica de baja escala) para resolver problemas de electrificación rural.

Uno de los problemas planteados ha sido la falta de sistemas electrónicos de potencia de conversión de energía de diseño abierto, que sean adoptables y modificables por particulares o empresas, en un estándar común.

A fines de 2015, WE decidió lanzar un Proyecto conjunto de investigación y aplicación de electrónica de potencia denominado *SEPS - Modular Power for the People*, que obtuvo financiamiento del Instituto Wuppertal /WISIONS (SEPS, 2016) a inicios de 2016, para crear un producto de hardware y software abierto que cubriera las necesidades de sus organizaciones miembros.

Este proyecto se viene ejecutando desde abril 2016 con aportes de grupos de Francia, Reino Unido, Perú y Argentina, y trabajos de becarios de ingeniería y otras especialidades en instituciones y empresas de dichos países.

El trabajo de la presente beca se realiza como parte del financiamiento SEPS, y es una continuación del desarrollo iniciado por el becario Jean Alinei (Universidad de Grenoble) en Río Gallegos entre junio y agosto de 2016. Específicamente, las tareas incluyen el desarrollo del software de uno de los módulos previstos (solar MPPT) en lenguaje C y su interface con el controlador general.

En el presente informe se detalla el Diagrama de Flujo de MPPT de un sistema abierto (diseño MPPT-TI Firmware PMP7605) (TI, 2015) que se utiliza como referencia, y el funcionamiento del método principal main(). Además se profundizará en las subrutinas e interrupciones llamadas desde él.

En la segunda parte del informe, se describe el funcionamiento de un programa básico para el controlador dsPIC33EP que se implementó en una placa de prueba, realizado por el becario Jean Alinei. Dicho programa sólo realiza acciones elementales y su función es ensayar la funcionalidad básica del equipo.

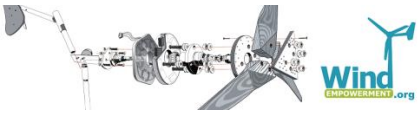
En la segunda parte del informe, se describe el funcionamiento de un programa básico para el controlador dsPIC33EP que se implementó en una placa de prueba, realizado por el becario Jean Alinei. Dicho programa sólo realiza acciones elementales y su función es ensayar la funcionalidad básica del equipo.

1 CAPÍTULO I: DIAGRAMA DE FLUJO / FIRMWARE SISTEMA TI

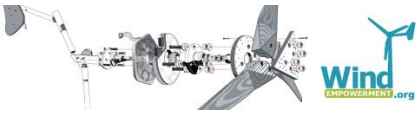
1.1 Inicializaciones y declaraciones

A continuación se mencionan las variables declaradas en el método principal.

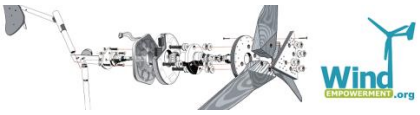
```
unsigned int        Is_Load_On=0,
                    GUI_Battery_Charge_Current_State=0,
                    GUI_Battery_Charge_Previous_State=0,
                    Is_Cal_Load_Management=0,
                    Load_I_Limit = 0x02;

unsigned int i;
unsigned int        ADC_Readings [6];
unsigned int        Panel_Voltage,
                    Battery_Voltage,
                    Panel_Current,
                    Battery_Current,
                    Panel_Current_S,
                    Battery_Current_S,
                    Prev_Battery_Current;

unsigned int        Panel_Voltage_Avg=0,
                    Panel_Current_Avg=0,
                    Battery_Voltage_Avg=0,
                    Battery_Current_Avg=0;

unsigned long       Panel_Power = 0,
                    Prev_Power = 0;
```




```
unsigned int        Duty = 210;
unsigned char       Reading_Captured = 1,
                    Phase_Shifting_Done = 0;

char                POB_Direction  = 1;

unsigned int        V_negate = 0,
                    I_negate = 0,
                    Idle_Count = 0;

signed long         PV_Volts_Prev = 0,
                    PV_Amps_Prev = 0;

float               Delta_PV_Volts = 0,
                    Delta_PV_Amps = 0,
                    Inc_Condctnce = 0,
                    Condctnce = 0;

unsigned int        Avg_ctr = 0, ctr_th = 64;
unsigned char       MPPT_Loop = 1,
                    CV_Mode_ON = 0,
                    MPP_Loop_Exit_Counter = 0,
                    CC_Loop_Exit_Counter = 0;

unsigned long       Charging_Voltage_Reqd = 0;
```



```
unsigned char       Hysterisis_ON = 0,
                    Cutoff_Counter = 0,
                    Reconnect_Counter = 0,
                    OC_Trigger_Counter,
                    OC_Triggered = 0,
                    Battery_Current_Counter = 0,
                    Start_Delay = 0,
                    Blanking_Time = 0;
#define Cutoff_Counter_Th           10
#define Reconnect_Counter_Th        10
#define OC_Trigger_Counter_Th       100

#define P_V                         5
#define B_V                         4
#define P_I                         3
#define B_I                         2
#define P_I_S                       1
#define B_I_S                       0

#define MIN_TOLERANCE_VOLTS         3       // Necesita ser sincronizado con el hardware
#define MIN_TOLERANCE_AMPS          1       // Necesita ser sincronizado con el hardware
#define MIN_TOLERANCE_CONDUCTANCE   0  // Necesita ser sincronizado con el hardware

#define PANEL_DISABLE               P3OUT |= BIT3
#define PANEL_ENABLE                P3OUT &= ~BIT3
```




```
#define LOAD_DISABLE                P3OUT |= BIT2

#define LOAD_ENABLE                 P3OUT &= ~BIT2

#define POWER_GOOD                  P1IN & BIT6

unsigned int CC_LIMIT          =    300;          // 210 - 10A, 300 - 15A, 350 - 20A

unsigned int CC_TO_CV_LIMIT    =    305;          // 305 - 14.2     , 610 - 28.4

unsigned int FLOAT_VOLTAGE     =    295;          // 295 - 13.8     , 590 - 27.6

unsigned int BATTERY_CUTOFF    =    220;          // 220 - 10.2     , 440 - 20.4

unsigned int BATTERY_RECONNECT =    240;          // 240 - 11.2     , 480 - 22.4

#define LOOP_EXIT_LIMIT             5
```

1.2 Diagrama main() simplificado

A continuación se visualiza el Diagrama simplificado de main() para su mejor comprensión, luego se detallan cada una de las partes.

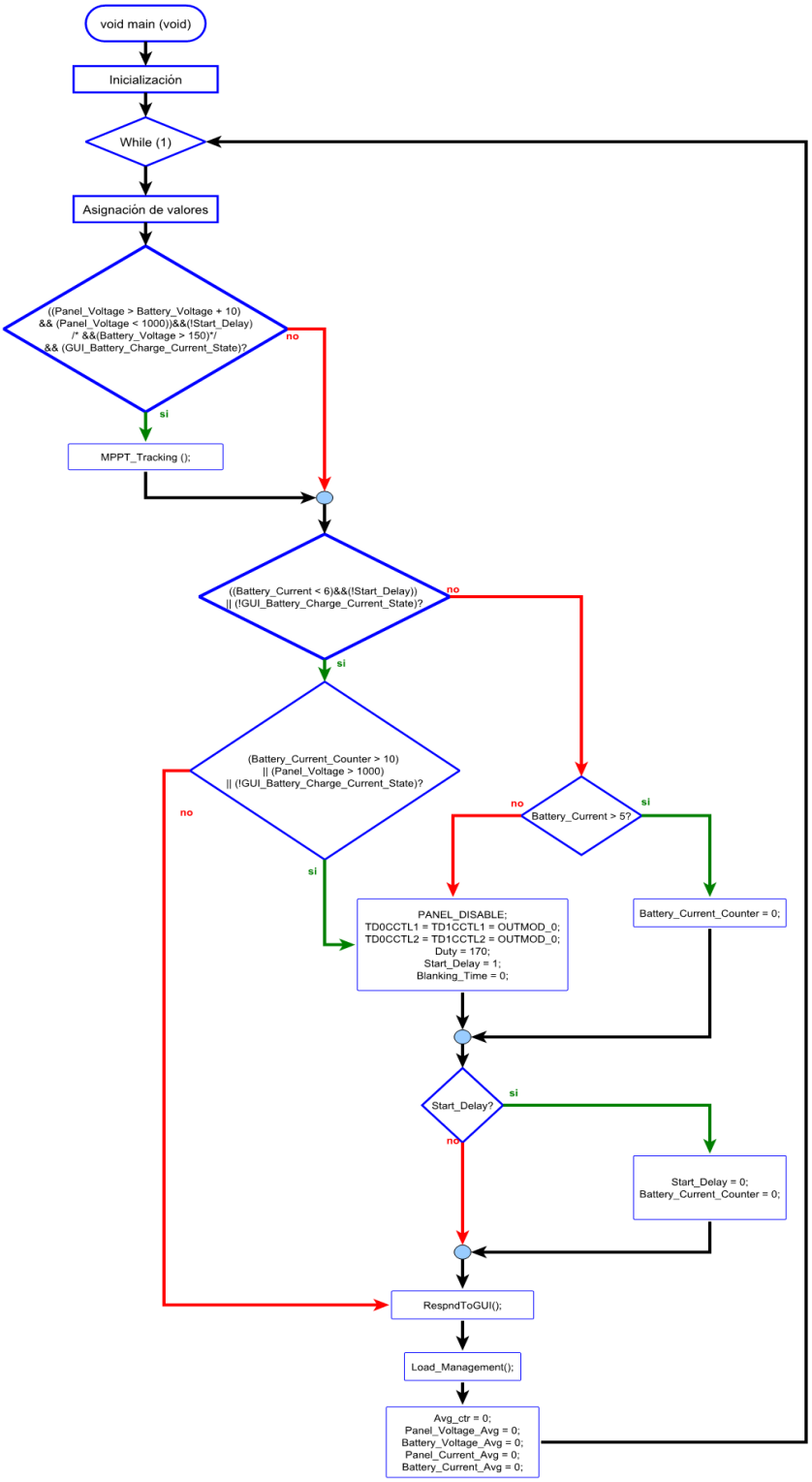


Diagrama de Flujo General - main()

1.3 Inicio main()

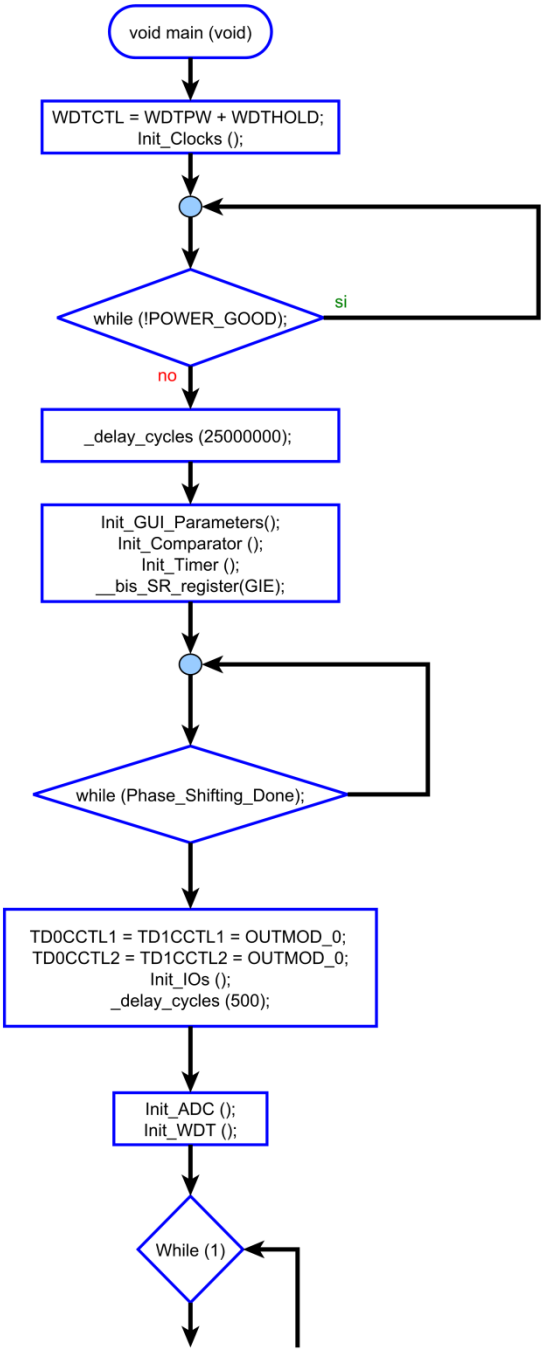


Diagrama de Flujo - inicio main()

A continuación se visualiza el Diagrama de la primera parte del main() con sus respectivas aclaraciones:

En primer lugar se inicializan los relojes para 25 MHz con el método Init_Clocks (este método se encuentra detallado más adelante).

Mientras POWER_GOOD (definido como P1IN & BIT6) no sea verdadero no se realizan acciones.

Una vez que POWER_GOOD==1 realiza un retardo de ciclo de 25000000.

Inicializa los parámetros para la GUI.

Habilita salidas para PWM de 200 KHz. Y habilita las interrupciones.

Espera flag.

Este flag desconecta las salidas del temporizador, inicializa puertos con el método Init_IOs.

Nuevamente se realiza un retardo esta vez de 500.

Inicialización de la ADC y WDT.

1.4 Ciclo principal de main() Parte A

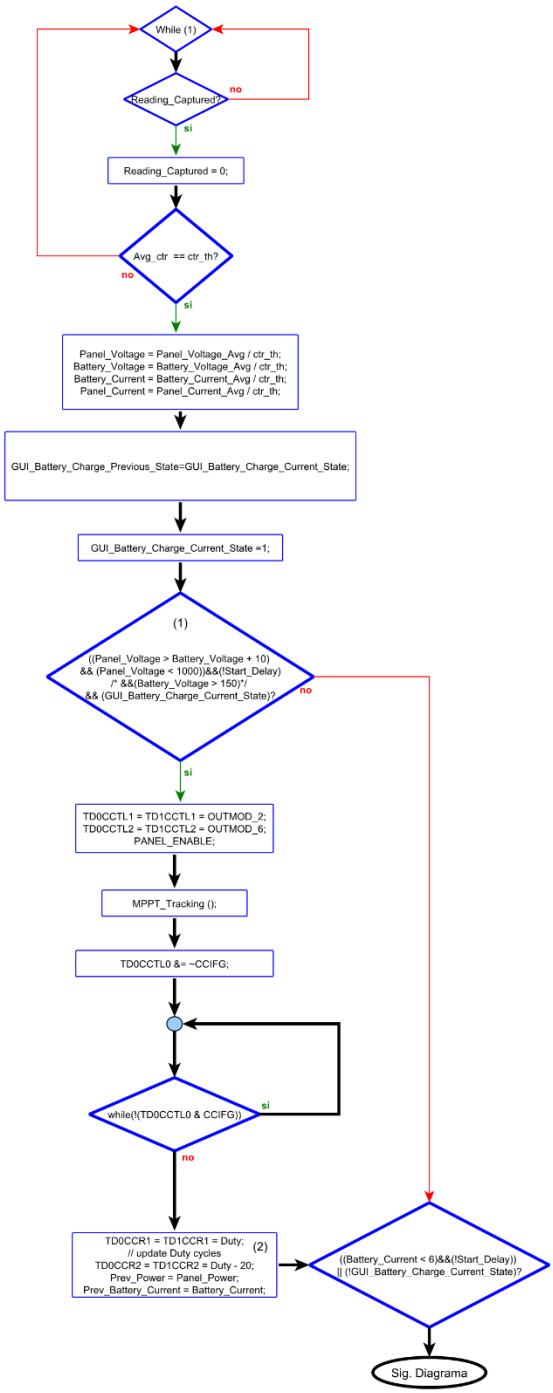


Diagrama de Flujo- main() Parte A

Comienza el ciclo principal de main().

ctr_th = 64 . i.e se calcula promedio de 64 lecturas ADC.

Cada 1.3ms se toma un conjunto de lecturas ADC

El tiempo de bucle es de 64 * 1,3 = ~ 83ms.

(1) La carga comienza si se dan las siguientes condiciones:

1. Tensión de Panel es mayor que voltaje que la batería +10

2. Panel de tensión es menor que el valor umbral máximo predefinido 1000.

3. Tensión de la batería en GUI es mayor que cierto valor. (esto es para asegurarse de que la batería no está muy descargada)

Start_delay es una variable utilizada para retardo 5s.

Esperar hasta que el temporizador finalice su ciclo actual.

Ciclos de actualización de servicio. (2).

1.5 Ciclo principal de main() Parte B

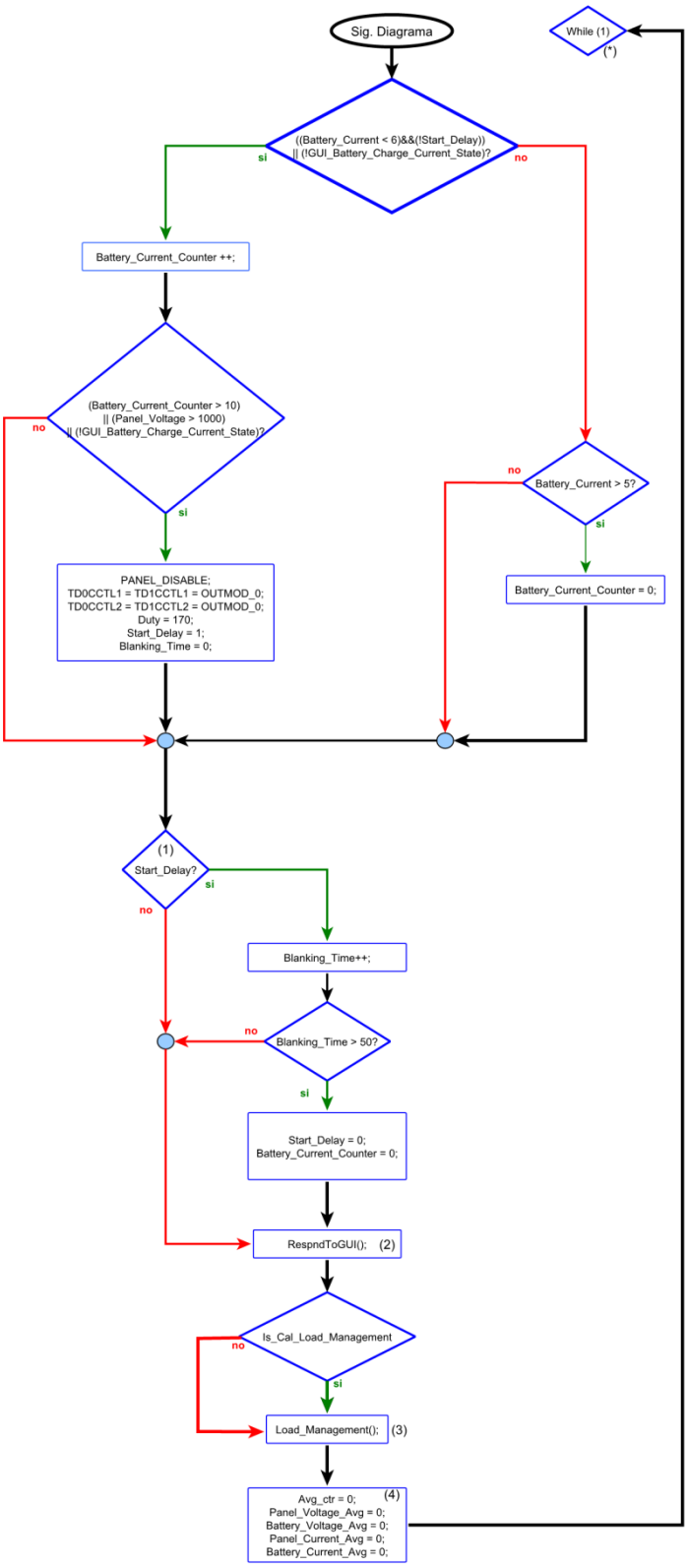


Diagrama de Flujo - main() Parte B

Continúa Ciclo principal de main() parte A

(*) Se puede observar que desde la última instrucción vuelve el ciclo del while principal (ver Inicio main).

(1) Este bloque genera demora de aproximadamente 5 segundos. Siempre que la corriente de la batería desciende por debajo de un valor umbral, el panel se apaga y se reinicia después de 5 segundos.

(2) Si la GUI esta habilitada, las variables del sistema se actualizan en la GUI.

(3) Si no se utiliza GUI, la gestión de carga se hace todo el tiempo.

(4) Reajustar todos los valores de ADC.

1.6 Interrupciones

__interrupt

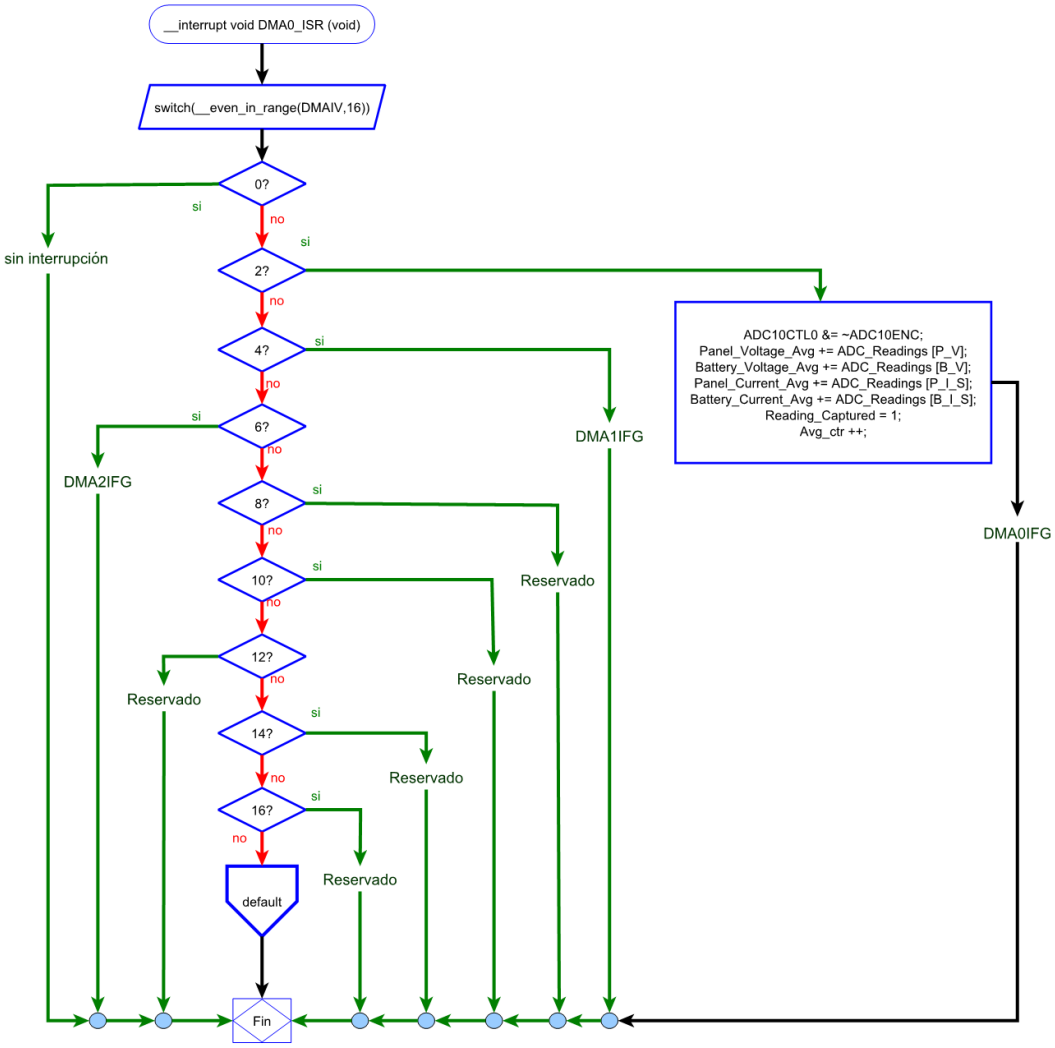
Las funciones para atención de interrupciones para Sistemas Embebidos se declaran con la palabra clave "__interrupt".

Se diferencian del resto de las funciones en que, como se pueden ejecutar en cualquier momento, al entrar deben guardar en la pila no sólo la dirección de retorno, sino también el estado del procesador, a fin de restaurar el estado original al salir.

1.6.1 DMA0_ISR

Esta interrupción se llama cuando DMA pone todos los valores de ADC en sus respectivas variables.

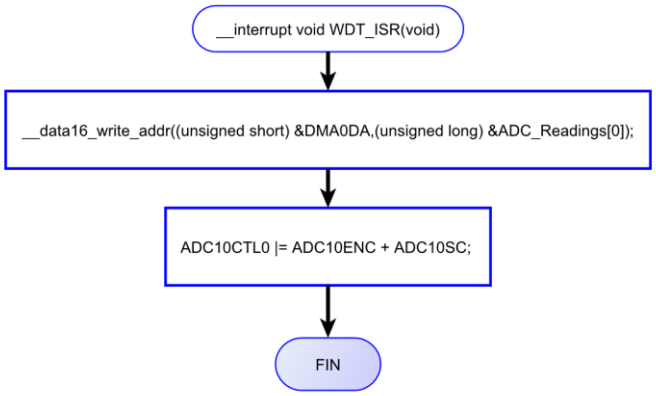
En el Caso 2 se completan las conversiones.



Rutina de Interrupción - DMA0_ISR

1.6.2 WDT_ISR

La interrupción del timer WDT se utiliza para disparar el ADC (inicio de lecturas) y también para iniciar el DMA que copia las lecturas ADC dentro de sus correspondientes variables.



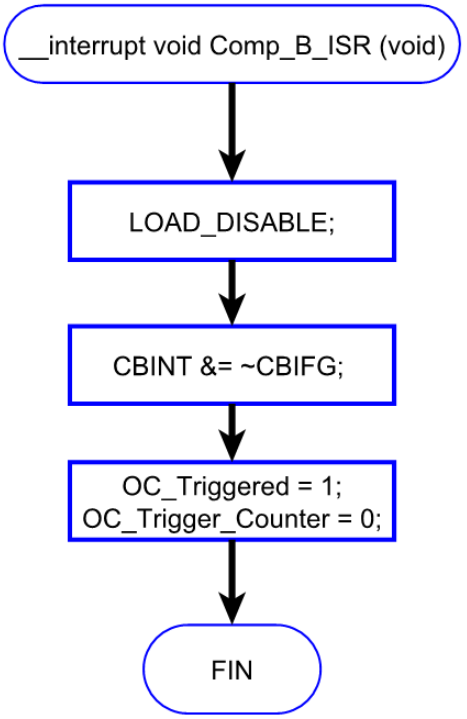
Rutina de interrupción del servicio WDT (Temporizador de vigilancia).

Comienzo de Conversión y muestreo.

Rutina de Interrupción - WDT_ISR

1.6.3 Comp_B_ISR

Este ISR se llama cada vez que se detecta una sobrecarga, la carga es deshabilitada y la Variable OC_Triggered es actualizada.

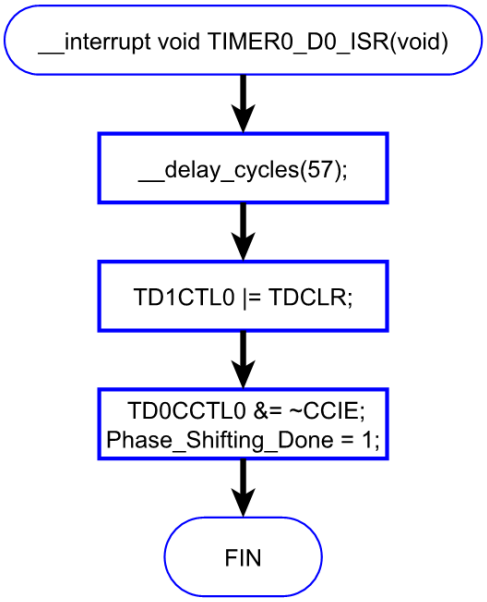


Limpia el flag de interrupción.

Rutina de Interrupción - Comp_B_IS

1.6.4 TIMER0_D0_ISR

Esta ISR se llama cuando se desborda TD0. TD1 se pone a cero para proporcionar la diferencia de fase de 180 grados.

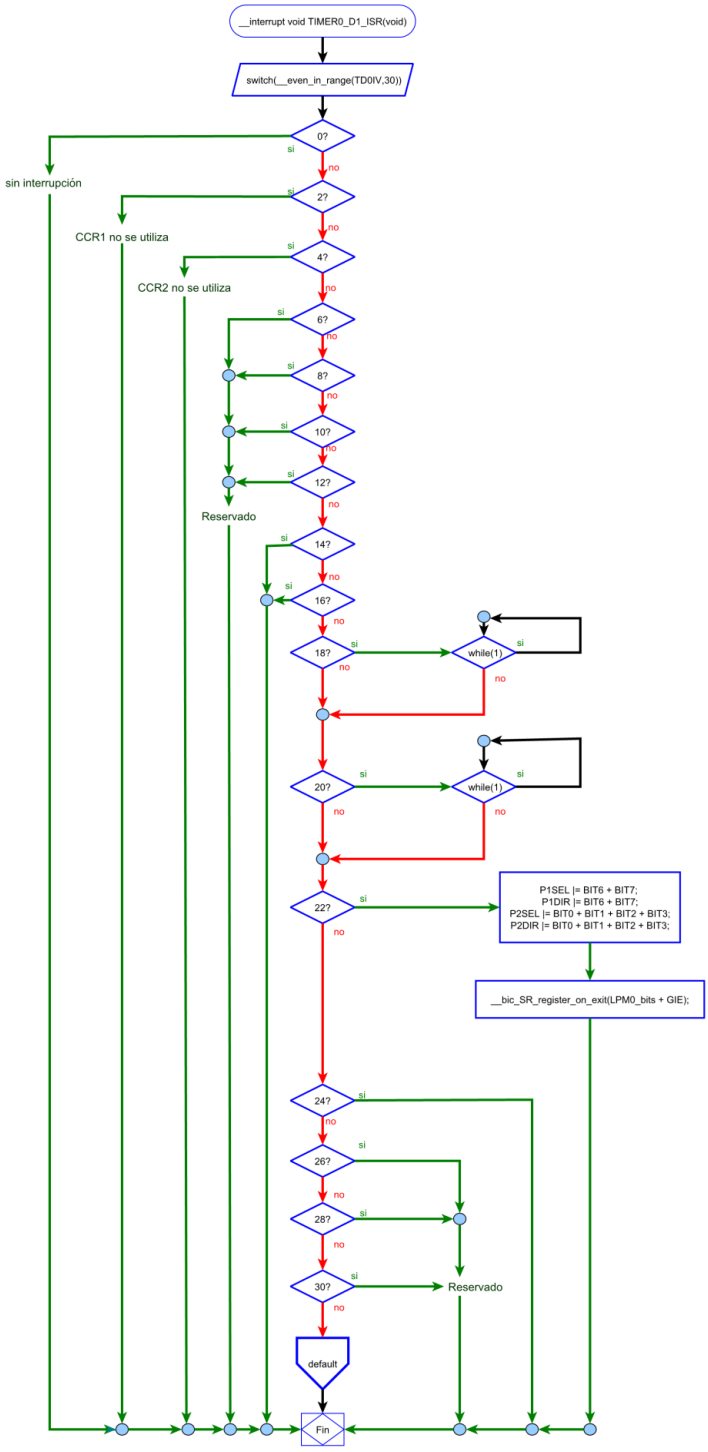


Rutina de Interrupción - TIMER0_D0

Retardo (57) ajustado para un perfecto cambio de fase de 180 grados.

1.6.5 TIMER0_D1_ISR

Vector de interrupción (TDIV) handler.



Rutina de Interrupción - TIMER0_D1

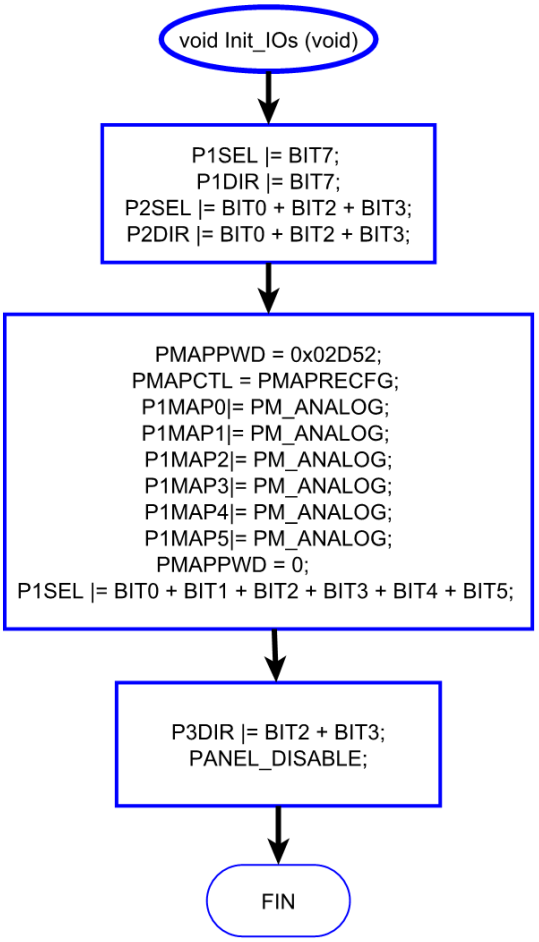
Configurar los puertos de salida PWM TD0 / TD1

1.7 Subrutinas

Detalle de las rutinas internas del método main().

1.7.1 Init_IOs()

En este método de realizan las inicializaciones de los puertos. Se configuran los puertos de salida PWM en TD0 / TD1.



Subrutina - Init_IOs()

Seleccionar opciones

Dirección de salida

P2.0 / TD0.2, P2.2 / TD1.1, P2.3 / TD1.2, seleccionar opciones.

dirección de salida.

Habilitar el acceso de escritura para modificar registros de asignación de puertos.

Permitir la configuración en tiempo de ejecución.

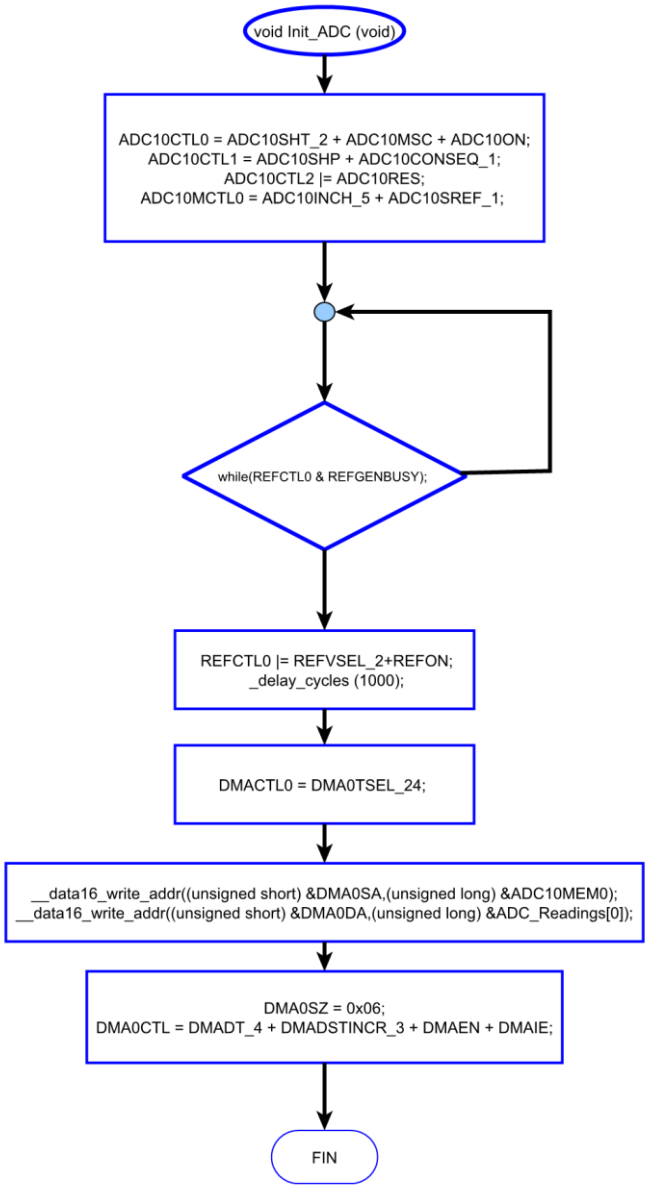
Modificar todos los registros PxMAPy – P1MAP0| a P1MAP5|

Deshabilitar el acceso de escritura para modificar registros de asignación de puertos escribiendo clave incorrecta.

Establecer el puerto PxMAPy registro de mapeo para PM ANALÓGICA junto con PxSEL.y = 1 aplicación de señales analógicas

1.7.2 Init_ADC()

Configurar ADC10.



Subrutina - Init_ADC()

16ADCclks, MSC, ADC ON.

contador de tiempo de muestreo.

Disparo por software (s/w trig.), secuencia única.

resultados de la conversión de 10 bits.

A0, A1, A2, A3, A4, A5 (EoS), Vref.

De forma predeterminada, REFMSTR=1 => REFCTL se utiliza para configurar la referencia interna.

Si el generador ref (REFGEN) está ocupado, debe esperar.

Seleccionar ref interna = 2.5V.

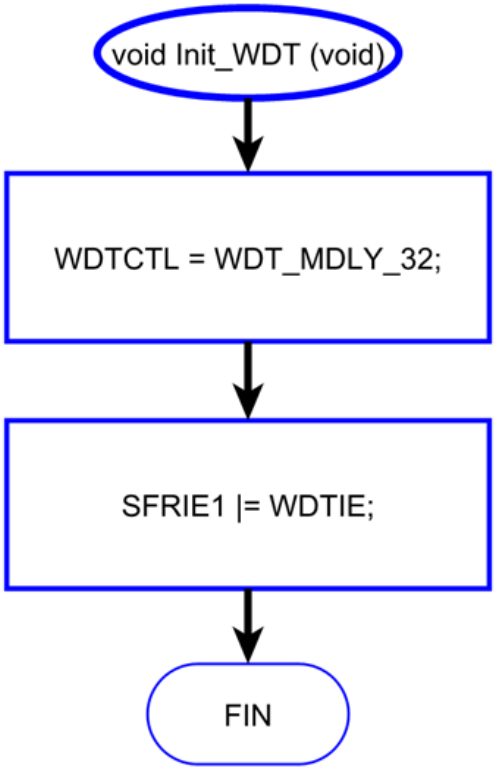
Fuente única dirección.

dirección de destino.

3 conversiones.

1.7.3 Init_WDT()

Temporizador WATCHDOG es utilizado para generar interrupciones para las líneas de lectura ADC.



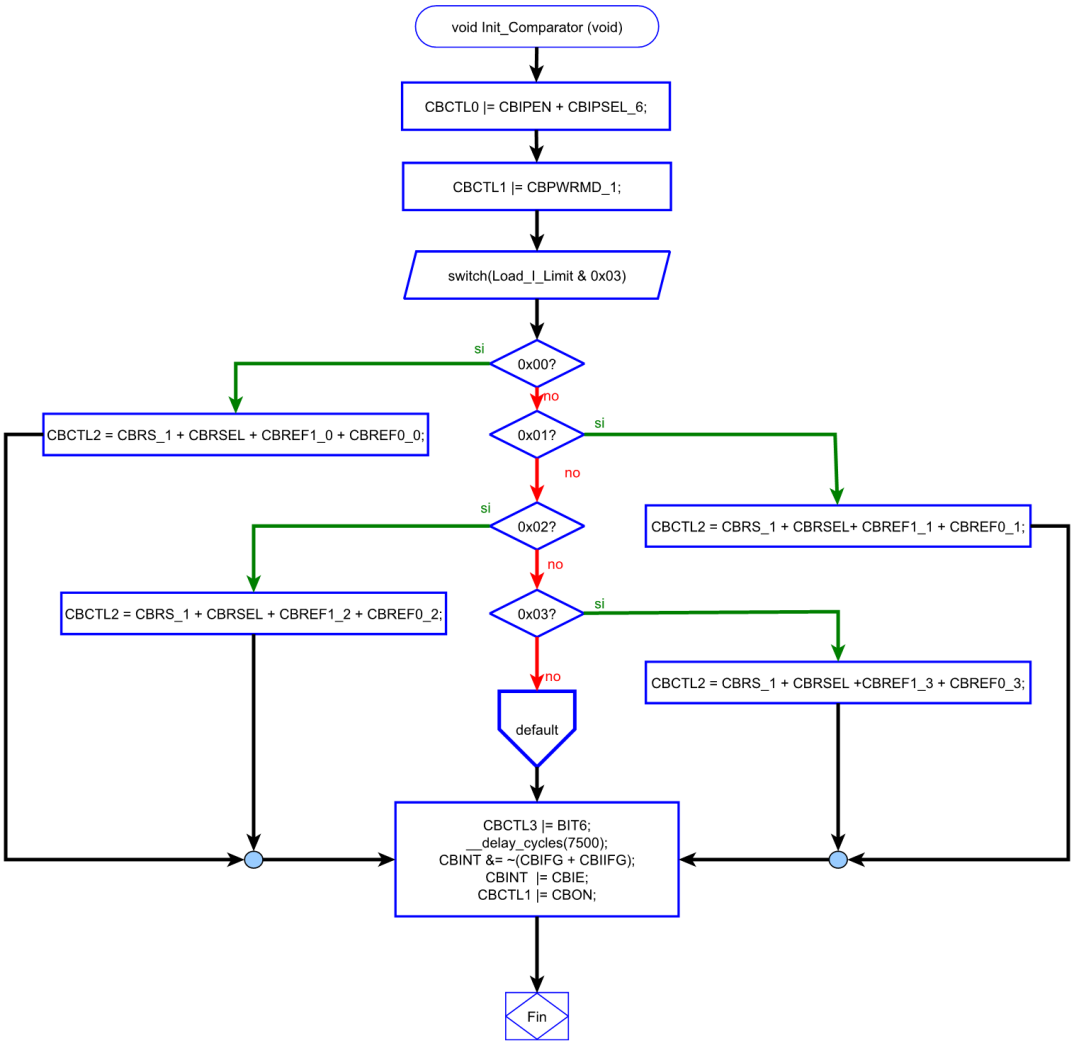
habilitar WDT.

Subrutina - Init_WDT()

1.7.4 Init_Comparator()

Conversión B es utilizado para detectar sobrecarga.

Cuando cualquier carga actual va más allá de este límite, genera una interrupción.



Subrutina - Init_Comparator()

CBCTL0 = Habilitar V +, canal de entrada CB6.

CBCTL1 = modo de potencia normal

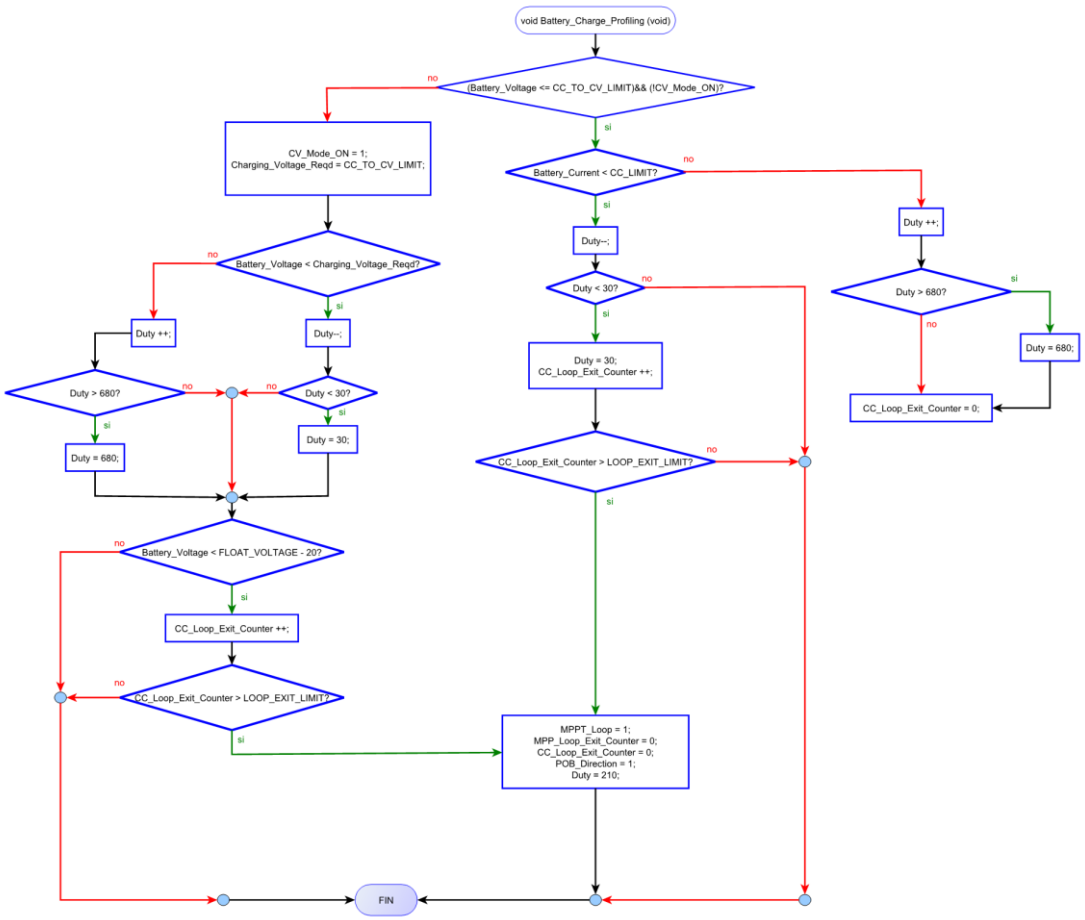
case 0x00=3 Amps.

case 0x01=6 Amps.

case 0x02= 9 Amps.

case 0x03= 12 Amps

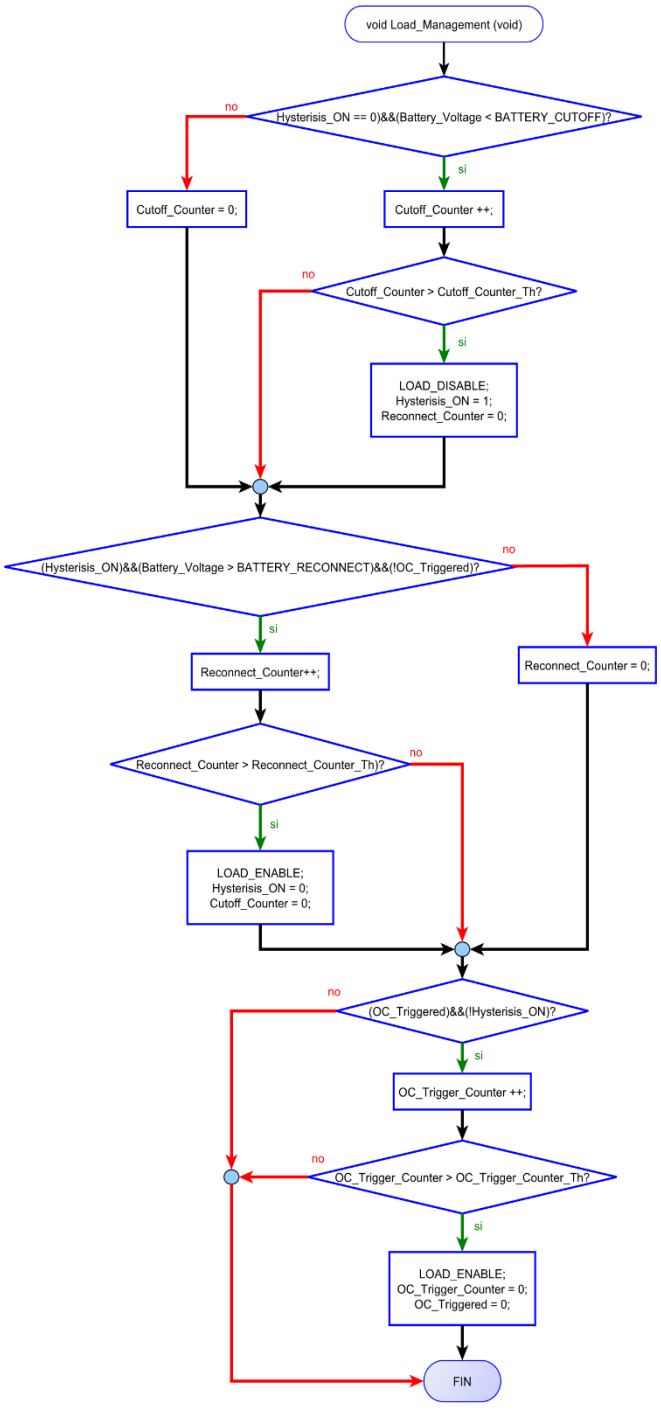
1.7.5 Battery_Charge_Profiling()



Subrut.- Battery_Charge_Profiling()

Esta rutina realiza las decisiones respecto al control de la variable Duty (ciclo de trabajo del PWM) dependiendo de la tensión en el banco de baterías, la corriente inyectada a dicho banco y los tiempos, tomando o no la decisión de regresar al modo lazo MPPT.

1.7.6 Load_Management()



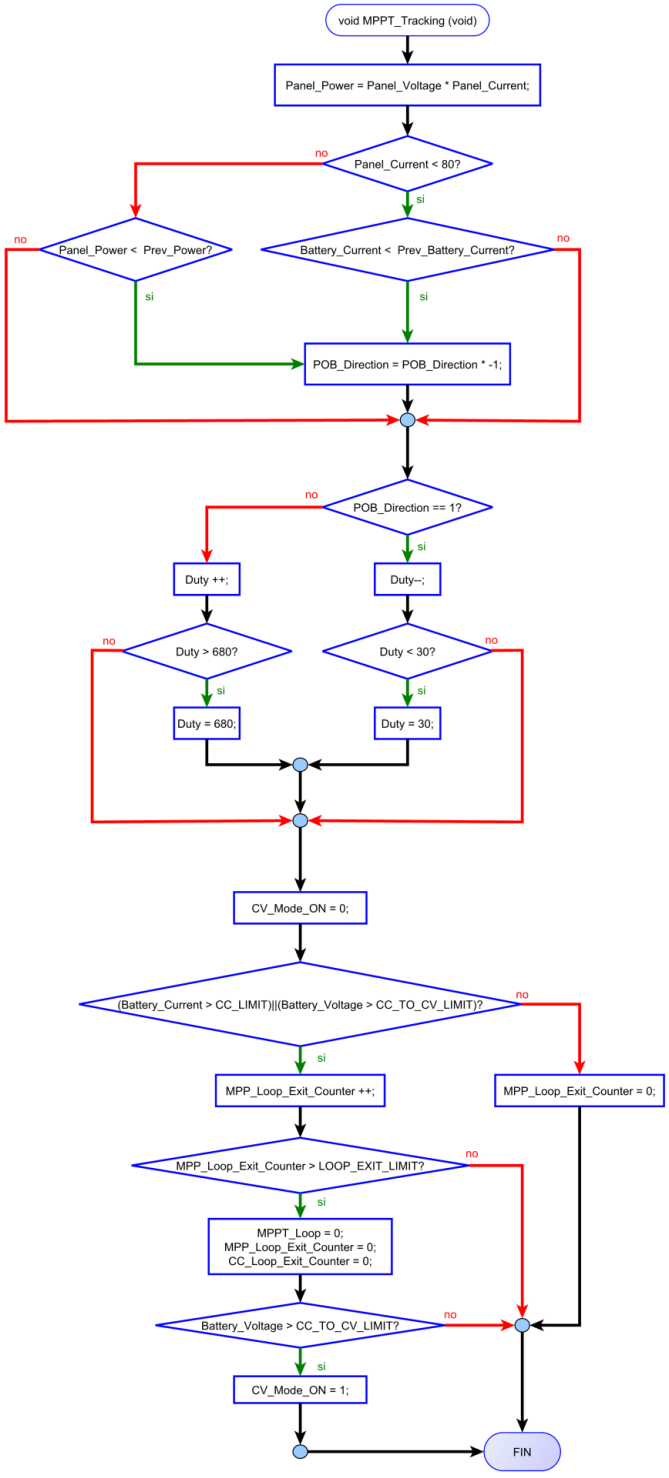
Subrutina - Load_Management()

Si el voltaje de la batería desciende por debajo de la tensión de corte de batería predefinido durante más de 1 segundo, entonces la carga se desactiva y se introduce el modo de histéresis.

Si la tensión de la batería sube por encima de la tensión (predefinida) de reconexión de la batería durante más de 1 segundo, entonces la carga se activa y se sale del modo de histéresis.

Cuando la carga se desactiva debido a un exceso de corriente, aquí la carga se habilita después de aproximadamente 10 segundos.

1.7.7 MPPT_Tracking()



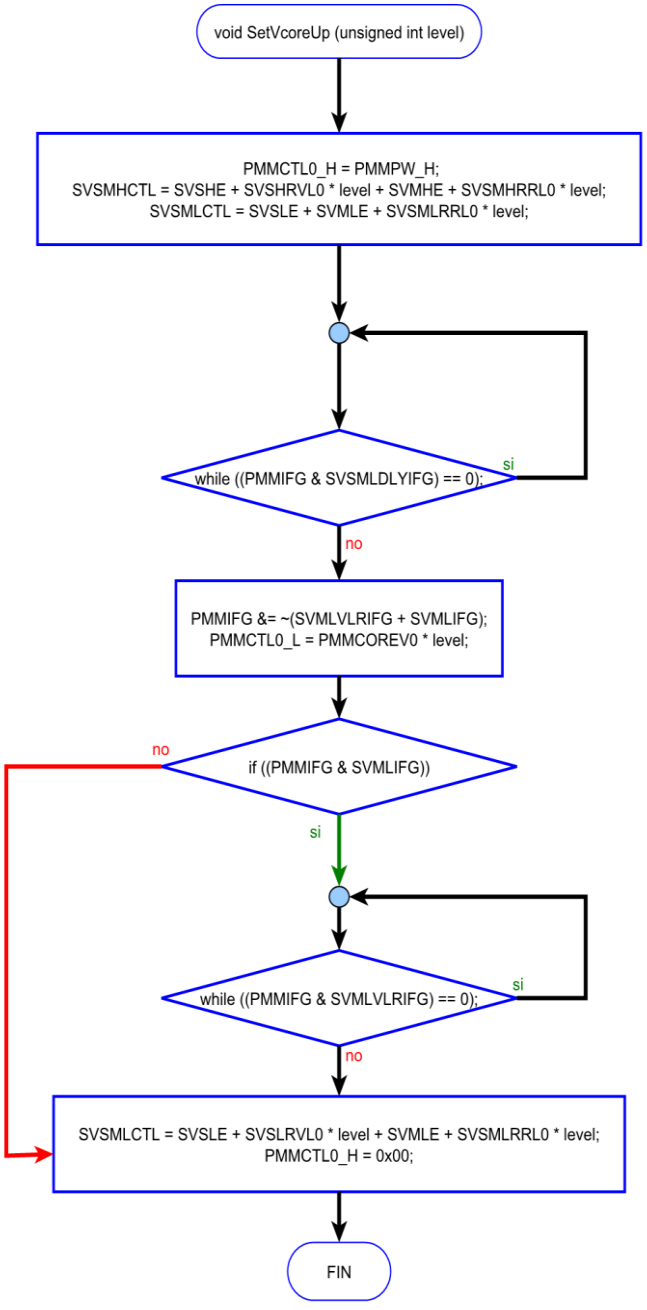
Subrutina - MPPT_Tracking()

Esta rutina implementa el seguimiento del punto de máxima potencia (MPPT) de los paneles fotovoltaicos, a través de una rutina de búsqueda de pico de la variable Panel_Power (producto de Vpanel x Ipanel), controlando la variable Duty adecuadamente.

Se mide la potencia del panel en comparación con el valor anterior.

La corriente de la batería es monitoreado para determinar la dirección de seguimiento.

1.7.8 SetVcoreUp()



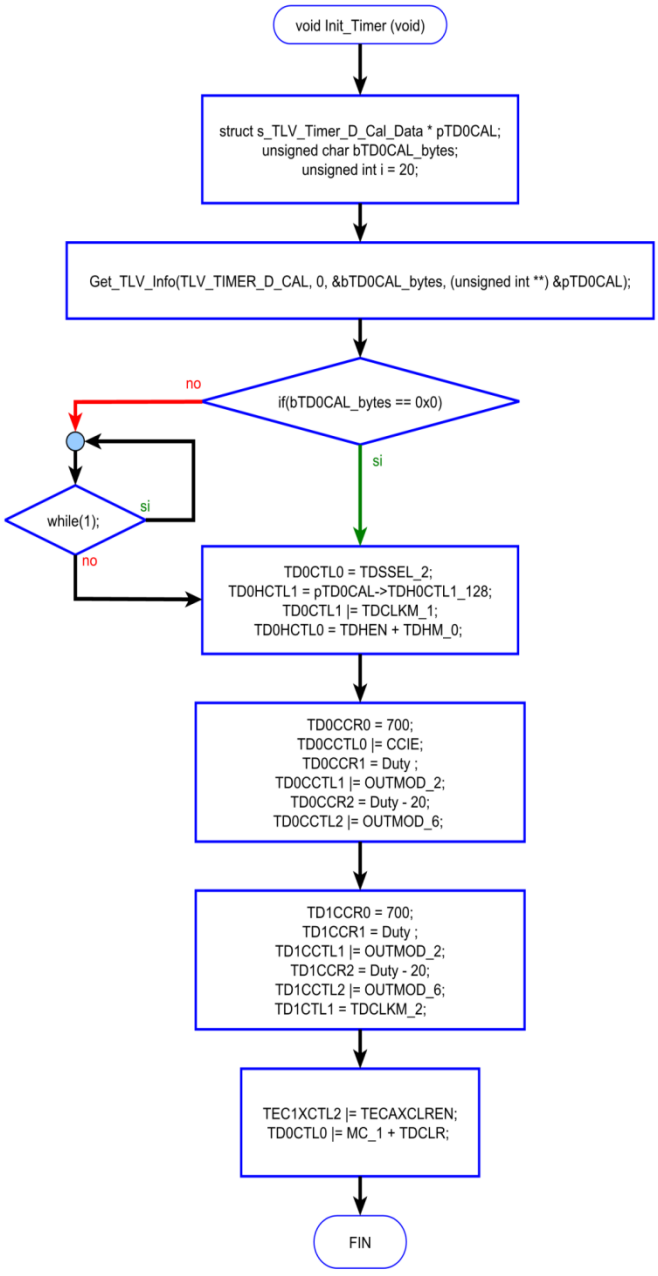
Abre registros PMM para escribir.

Establece SVS/SVM a nuevos niveles.

Bloquear registros PMM para acceso de escritura.

Subrutina - SetVcoreUp()

1.7.9 Init_Timer()

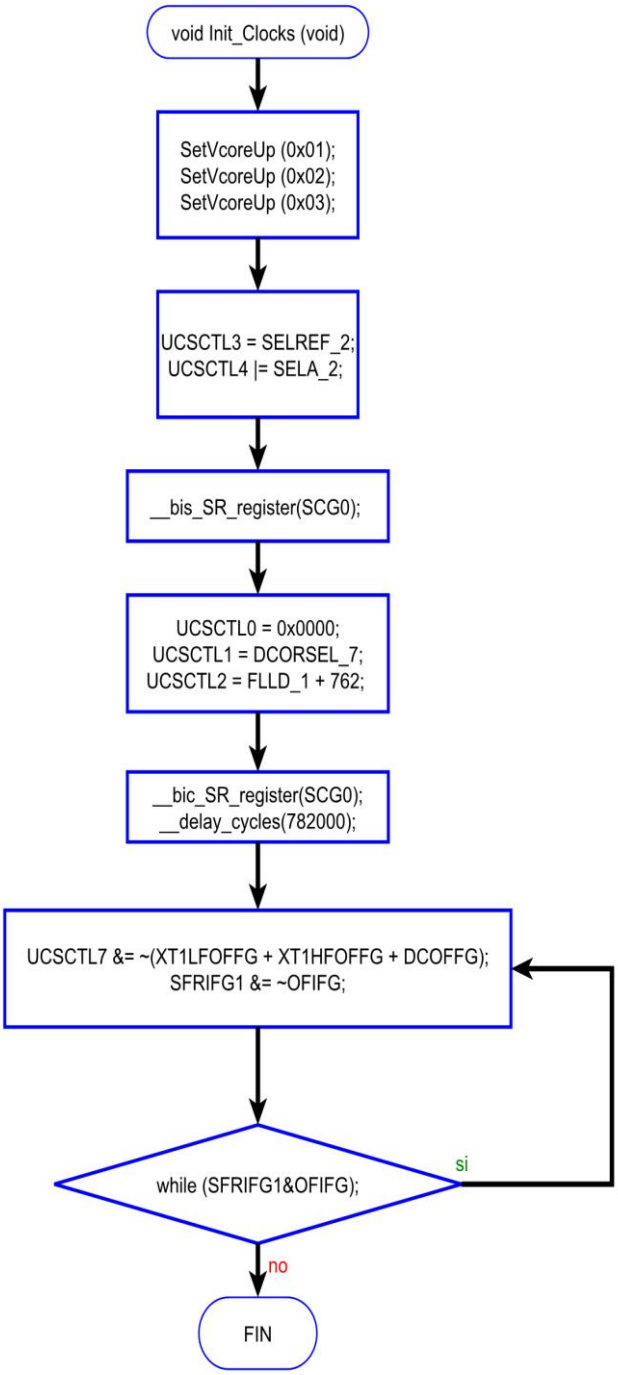


Subrutina - Init_Timer()

Esta rutina realiza la inicialización de registros para los Timers

Temporizador D es operado a 128mhz utilizando Generador de alta resolución.

1.7.10 Init_Clocks()



Configura DCO = 25Mhz.

(762 + 1) * 32768 = 25MHz

Subrutina - Init_Clocks()

2 CAPÍTULO II: PROGRAMA DE ENSAYO CONTROLADOR PROTOTIPO dsPIC33

En esta segunda parte del informe, se describe el funcionamiento de un programa básico para el controlador dsPIC33EP que se implementó en una placa de prueba, realizado por el becario Jean Alinei. Dicho programa sólo realiza acciones elementales y su función es ensayar la funcionalidad básica del equipo. Dicho programa se escribió en lenguaje C, utilizando las herramientas gratuitas como el entorno MPLAB X y el compilador XC16.

Inicializa sistema: Esto incluye puerto serie, ADC y módulo PWM.

Establecer parámetros de PWM 180kHz, frecuencia, ciclos de trabajo y deadtime.

Inicializa módulo ADC.

Define variables para intensidad y voltaje leidos con el ADC, define ganancias fijas para cada canal, todas en punto flotante (float) .

Espera que finalice la conversión A/D de los cuatro canales.

Define los contenidos de los canales de corriente y tensión, multiplicando cada ganancia por el respectivo valor entero leído del ADC.

Consulta por recepción de un carácter por el puerto serie a 9600 baud, si es un carácter válido lo “imprime” por el mismo canal.

Retardo elemental

Imprime por puerto serie las variables analógicas, y cierra el lazo.

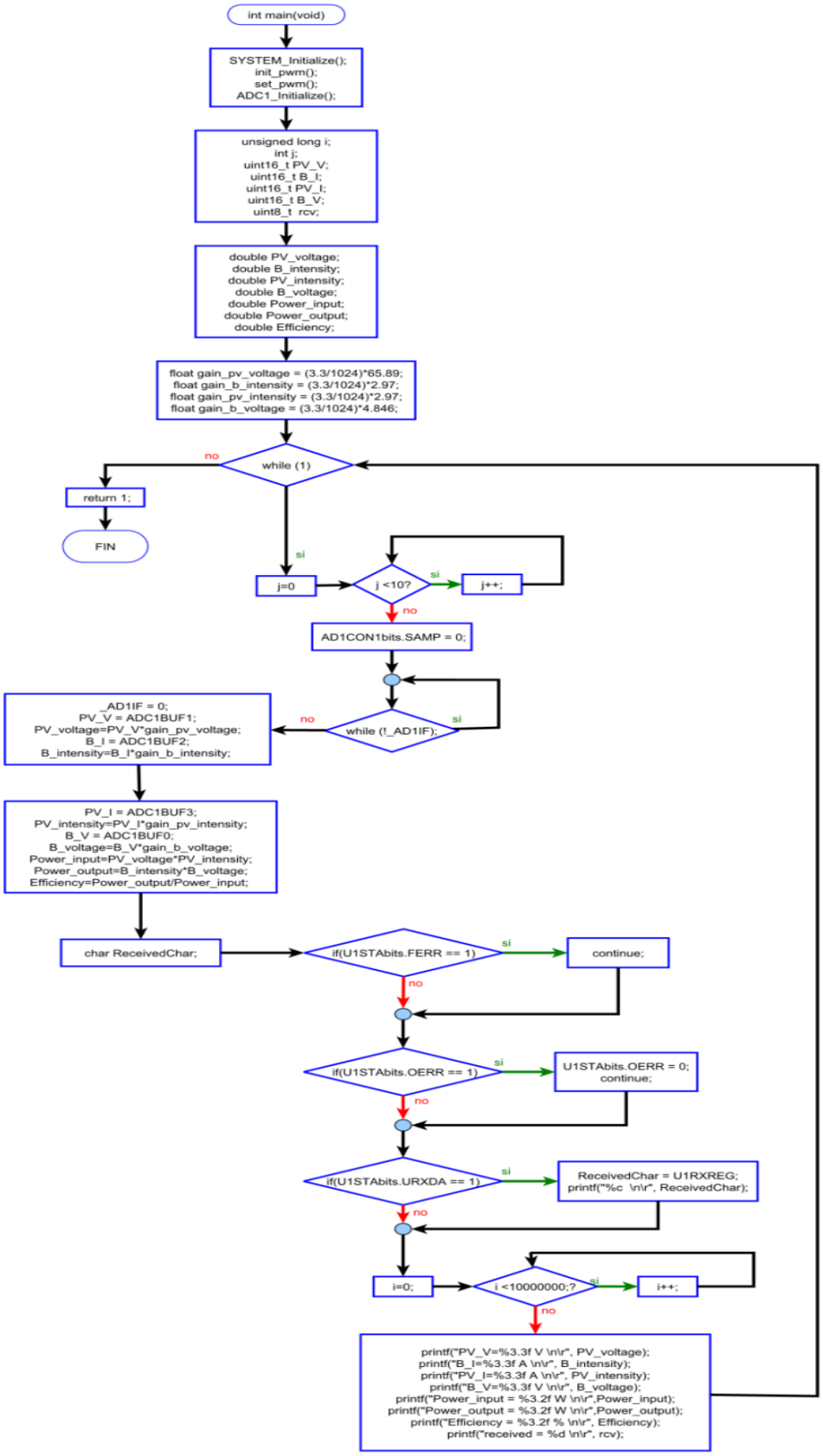


Diagrama de Flujo - Controlador dsPIC33

3 CAPÍTULO III: PROTOCOLO Modbus ASCII

Modbus es un protocolo de comunicaciones industriales estándar (de facto). En nuestro contexto, especifica el procedimiento que el controlador y el esclavo utilizarán para intercambiar datos.

En la comunicación maestro-esclavo, un dispositivo debe iniciar una solicitud y luego esperar una respuesta. El maestro es responsable de iniciar cada interacción.

3.1 Capas del Protocolo Modbus

En su implementación inicial, Modbus era un solo protocolo de comunicaciones serie, por lo que no podía ser dividida en múltiples capas.

Con el tiempo, se introdujeron diferentes unidades de datos de aplicación, ya sea para cambiar el formato del paquete utilizado por un puerto serie o para permitir el uso de redes TCP/IP y UDP (User Datagram Protocol). Esto llevó a una separación del protocolo principal, el cual define la unidad de datos de protocolo (PDU) y la capa de red, que define la unidad de datos de aplicación (ADU).

3.2 Unidad de Datos de Protocolo (PDU)

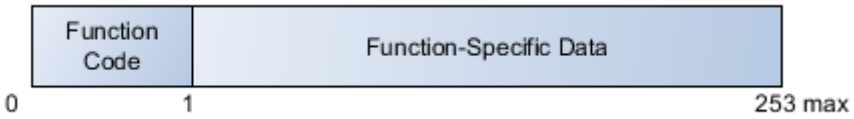
La PDU y el código que la maneja consisten en el núcleo de la Especificación del Protocolo de Aplicación Modbus. Esta especificación define el formato de la PDU, los diversos conceptos de datos utilizados por el protocolo, el uso de los códigos de función para tener acceso a esos datos y la implementación específica y restricciones de cada código de función.

El formato de Modbus PDU está definido como un código de función seguido por un conjunto de datos asociado.

El tamaño y el contenido de estos datos son definidos por el código de función y la PDU completa (código de función y datos) no puede exceder de 253 bytes de tamaño.

Cada código de función tiene un comportamiento específico que los esclavos pueden implementar de manera flexible en base al comportamiento de la aplicación deseada. La especificación de la PDU define conceptos básicos para el acceso y manipulación de datos; sin embargo, un esclavo puede manejar datos de una manera que no esté definida explícitamente en la especificación.

La PDU consta de un código de función de un byte seguido de hasta 252 bytes de datos de funciones específicas.



PDU Modbus

El código de función es el primer elemento que será validado. Si el código de función no es reconocido por el dispositivo que recibe la solicitud, responde con una excepción. Si se acepta el código de función, el dispositivo esclavo comienza a descomponer los datos de acuerdo con la definición de la función.

Debido a que el tamaño del paquete está limitado a 253 bytes, los dispositivos están limitados a la cantidad de datos que pueden ser transferidos. Los códigos de función más comunes pueden transferir entre 240 y 250 bytes de datos del modelo de datos de esclavos, dependiendo del código.

3.3 Unidad de Datos de Aplicación (ADU)

Para transmitir los datos necesarios para Modbus a través de estas capas, Modbus incluye un conjunto de variantes ADU que son diseñadas para cada protocolo de red.

3.4 Formatos Estándares

Los tres formatos ADU estándares son TCP, unidad terminal remota (RTU), y ASCII.

Los formatos RTU y ASCII ADUs normalmente se usan a través de una línea serie, mientras que el protocolo TCP se usa a través de redes TCP/IP o UDP/IP modernas.

3.5 Formato RTU

Modbus- Formato RTU

A diferencia de la ADU de TCP/IP más compleja, esta ADU incluye solamente dos piezas de información, además de la PDU principal.

Primero, una dirección se usa para definir para qué esclavo está diseñada la PDU. En la mayoría de las redes, la dirección 0 define la dirección de "broadcast". Es decir, un maestro puede enviar un comando de salida a la dirección 0 y todos los esclavos deben procesar la solicitud pero ningún esclavo debe responder. Además de esta dirección, se usa un CRC para asegurar la integridad de los datos.

Encapsulando el paquete hay un par de intervalos de silencio, es decir, periodos en los que no hay comunicación en el bus. Para una velocidad de transferencia de 9600 bps, este tiempo es alrededor de 4 ms. El estándar define una longitud mínima de silencio, independientemente de la velocidad de transferencia, de un poco menos de 2 ms.

Esto tiene un inconveniente referido al rendimiento ya que el dispositivo debe esperar a que el tiempo muerto se cumpla antes de que el paquete pueda ser procesado.

Más peligrosa aún es la introducción de diferentes tecnologías usadas para transferencia serial y velocidades de transferencia mucho más rápidas que cuando se introdujo el estándar.

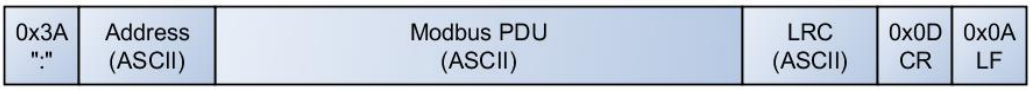
Si por ejemplo se usa un cable convertidor de USB a serie, no se tiene ningún control sobre el paquete y la transferencia de datos. Las pruebas muestran que al usar un cable convertidor de USB a serie con el controlador NI-VISA se introducen grandes intervalos de tamaño variable en el flujo de datos. Estos intervalos, períodos de silencio, engañan al código compatible con la especificación al creer que un mensaje se ha completado. Debido a que el mensaje no se completa, esto por lo general conduce a un CRC no válido y al dispositivo que interpreta la ADU como alterada.

Un método común para resolver estos problemas es romper la capa de abstracción entre la PDU de Modbus y la capa de red. Es decir, el código serie interroga al paquete de la PDU de Modbus para determinar el código de función. Combinado con otros datos en el paquete, se puede descubrir la longitud del paquete restante y usarla para determinar el final del paquete. Con esta información, puede usarse un descanso mucho más largo, lo que permite silencios de transmisión y puede ocurrir consulta a nivel de la aplicación mucho más lentamente.

Se recomienda utilizar este mecanismo para los desarrollos nuevos. El código que no emplee este método puede experimentar un número mayor de paquetes "alterados" de lo esperado.

3.6 Formato ASCII

La ADU de ASCII es más compleja que la de RTU, como se muestra en la Figura, y también evita muchos de los problemas del paquete RTU. Sin embargo, tiene algunas desventajas que le son particulares.



Modbus - Formato ASCII

3.6.1 La ADU de ASCII

La ADU de ASCII tiene un inicio y un final bien definido y único para cada paquete.

Cada paquete comienza con el carácter ":" y termina con los caracteres de retorno de carro (CR) y alimentación de línea (LF).

Además, las APIs serie como NI-VISA y el .NET Framework SerialPort Class pueden leer datos fácilmente en un búfer hasta recibir un carácter específico, como CR/LF.

Estas características hacen procesar la escritura de datos en el puerto serie sea fácil y eficiente en el código de aplicación moderno.

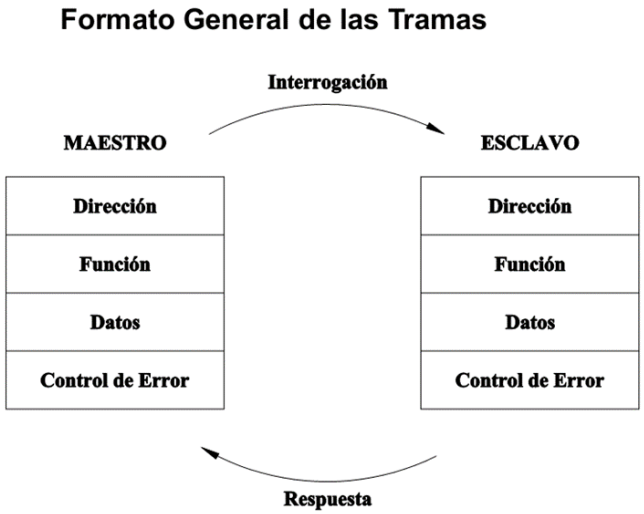
La desventaja del ADU de ASCII es que todos los datos se transfieren como caracteres hexadecimales codificados en ASCII. Es decir, en lugar de enviar un solo byte para el código de función 3, 0x03, envía los caracteres ASCII "0" y "3" o 0x30 / 0x33. Esto hace que el protocolo sea más legible, pero también significa que se debe transferir el doble de datos en seri y que las aplicaciones que envían y reciben deben ser capaces de analizar los valores ASCII.

3.7 Diferencias entre RTU y ASCII

	Modo ASCII	Modo RTU
Caracteres	ASCII '0'…'9','A'….'F'	Binario 0…255
Comprobación **Error**	LRC Longitudinal Redundancy Check	CRC Cyclic Redundancy Check
Inicio de trama	Caracter ':'	3.5 veces tiempo de carácter
Final de trama	Caracter CR/CL	3.5 veces tiempo de carácter
Distancia máx. **entre caracteres**	1 S.	1.5 veces tiempo de caracter
Bit de inicio	1	1
Bits de datos	7	8
Paridad	Par / Impar / Ninguna	Par / Impar / Ninguna
Bits de parada	1 si hay paridad; 2 si no hay paridad	1 si hay paridad; 2 si no hay paridad

3.8 Campos de las tramas MODBUS

El número de campos de las tramas MODBUS varía ligeramente dependiendo de si utilizamos la codificación ASCII o RTU.



Modbus - Formato de Tramas

3.9 Codificación ASCII (formato texto):

Modo ASCII

Comienzo de Trama	Dirección	Función	Datos	Control de Errores	Fin de Trama
:	2 bytes	2 bytes	N x 2 bytes	2 bytes	CR + LF

Modbus - Codificación ASCII

- Inicio de trama: 2 caracteres ASCII (que representan 1 byte) codificando el caracter ":" (0x3A).
- Nº Esclavo: 2 caracteres ASCII (que representan 1 byte) codificando la dirección del esclavo destino (u origen) de la trama.
- Código Operación: 2 caracteres ASCII (que representan 1 byte) con el código de operación.
- Dirección, datos y sub funciones Datos: con los parámetros necesarios para realizar la operación.
- LRC (16): H L.
- Final de trama: 4 caracteres ASCII (que representan 2 bytes) con los caracteres CR (0x0D) - LF (0x0A).

3.10 Codificación RTU (formato binario)

En este formato, el inicio de trama debería ser tras 3.5 tiempo de carácter.

Modo RTU

Comienzo de Trama	Dirección	Función	Datos	Control de Errores	Fin de Trama
Tiempo de 3 bytes	1 bytes	1 bytes	N x 1 bytes	2 bytes	

Modbus - Codificación RTU

- Nº Esclavo: 1 byte con la dirección del esclavo destino (u origen) de la trama.
- Código Operación: 1 byte con el código de operación.
- Sub funciones Datos: con los parámetros necesarios para realizar la operación.
- CRC(16): H L. (Control de errores, codificado en 2 bytes – 16 bits).

3.11 Detalle de las funciones.

Código	Acción	Significado
01	Leer Bobinas (0:xxxx)	Obtiene el estado actual ON/OFF de un grupo de bobinas lógicas.
02	Leer Entradas (1:xxxx)	Obtiene el estado actual ON/OFF de un grupo de entradas lógicas.
03	Leer Registros (4:xxxx)	Obtiene el valor binario de uno o más registros de almacenamiento.
04	Leer Registros (3:xxxx)	Obtiene el valor binario de uno o más registros de entrada.
05	Escribir Bobina (0:xxxx)	Fuerza el estado de una bobina.
06	Escribir Registro (4:xxxx)	Escribe el valor binario de un registro de almacenamiento.
15	Escribir Bobinas (0:xxxx)	Fuerza el estado de un grupo de bobina.
16	Escribir Registros (4:xxxx)	Escribe el valor binario de un grupo de registro de almacenamiento.

A continuación se muestran algunos ejemplos de las funciones “3” y “4”.

Función 03 → Leer Registros (4:xxxx)

Interrogación:

Dirección	Función	Registro comienzo (alto)	Registro comienzo (bajo)	Cantidad de Registros (alto)	Cantidad de Registros (bajo)	Control de Error
11	03	00	6B	00	03	7E

Respuesta:

Dirección	Función	Cuenta de bytes	Reg. 40108 (alto)	Reg. 40108 (bajo)	Reg. 40109 (alto)	Reg. 40109 (bajo)	Reg. 40110 (alto)	Reg. 40110 (bajo)	Control de Error
11	03	06	02	2B	00	00	00	64	55

Modbus ASCII - Función 03

Función 04 → Leer Registros (3:xxxx)

Interrogación:

Dirección	Función	Registro comienzo (alto)	Registro comienzo (bajo)	Cantidad de Registros (alto)	Cantidad de Registros (bajo)	Control de Error
11	04	00	08	00	01	E2

Respuesta:

Dirección	Función	Cuenta de bytes	Registro 30009 (alto)	Registro 30009 (bajo)	Control de Error
11	04	02	05	39	AB

Modbus ASCII - Función 04

ANEXO I.- DOCUMENTOS REFERENCIADOS

- Andrés F. Ruiz Olaya, Asfur Barandica López, Fabio G. Guerrero Moreno, (2004)- Implementación de una Red MODBUS/TCP.
- Isabel Martins Cruz, (Sartenejas Febrero 2006) – Desarrollo de un controlador de comunicaciones basado en microcontrolador para tecnología GSM.
- http://www.ecured.cu/Protocolo_de_Comunicaci%C3%B3n_Modbus, fecha consulta 26/7/2016.
- Roger Torres Salazar, (2006) – Modbus RTU. Implementación del Protocolo en Microcontrolador.

ANEXO II.- HERRAMIENTAS UTILIZADAS

yEd Editor Gráfico

Se puede descargar desde https://www.yworks.com/downloads#yEd

Visustin Generador de Diagrama de Flujo

Convierte el código fuente en diagramas de flujo y diagramas de actividad UML de manera automática.

Versión de prueba se puede descargar desde http://www.aivosto.com/visustin-es.html

Code2flow Generador Gráfico Online

Es una versión de prueba gratuita de código interactivo de Diagrama de flujo, los cuales pueden ser exportados en formatos de PDF, SVG y PNG.

Se puede utilizar desde http://code2flow.com/

Notepad++ 7.0

Editor de código fuente, soporta diversos lenguajes de programación, Funciona en entorno MS Windows y su uso se rige por la licencia GPL.

Se puede descargar desde https://notepad-plus-plus.org/