SISTEMAS DE TIEMPO REAL
Introducción al Diseño

Gallina, Sergio Hilario
Sistemas de tiempo real: Introducción al diseño / Sergio Hilario Gallina; Marcos Darío Aranda; Matías Leandro Ferraro; contribuciones de Natalia Susana Gallina. - 1a ed revisada. - Ciudad Autónoma de Buenos Aires: ACSE - Asociación Civil para la investigación, Promoción y Desarrollo de Sistemas Eléctricos Embebidos, 2015.
269 p.; 21 x 15 cm.

ISBN 978-987-45523-7-2

1. Software. 2. Microprocesadores. I. Aranda, Marcos Darío. II. Ferraro, Matías leandro. III. Gallina, Natalia Susana, colab. IV. Título.
CDD 004.33

Autores:

Aranda, Marcos Darío
Ferraro, Matías Leandro
Gallina, Sergio Hilario
Gallina, Natalia Susana

Editores:

Asociación civil para la investigación, promoción y desarrollo de los sistemas electrónicos embebidos.

Impreso en:

Editorial Cientifica Universitaria
Universidad Nacional de Catamarca
Febrero 2016

CONTENIDO

CAPITULO 1: INTRODUCCIÓN A LOS SISTEMAS DE TIEMPO REAL

1.1 CARACTERÍSTICAS DE LOS S.T.R.

En un sistema para procesamiento de aplicaciones de tipo convencional, rara vez es crítico el control del tiempo, generalmente se trata de hacer que funcione lo más rápido posible pero además se trata de optimizar el uso de los recursos internos del sistema. En cambio *en sistemas de tiempo real deben atenderse las demandas externas en el momento en que ocurren*, no se pueden poner limitaciones, esto impide organizar el sistema en base a factores internos.

Por ejemplo en un sistema de control de un avión no se puede demorar la respuesta a un evento externo, los resultados obtenidos con retraso pueden ser fatales.

En un sistema de tiempo real se deben obtener respuestas correctas en el momento preciso, de esto se infieren tres funciones básicas para estos sistemas:

Monitoreo: hay que obtener información acerca del entorno físico del sistema. Involucra el uso de sensores y opcionalmente conversores Analógico/Digital.

Control: deben realizarse los cálculos necesarios. Involucra el manejo de modelos matemáticos y algoritmos.

Actuación: se debe alterar el estado actual del sistema. Involucra el uso de transductores y opcionalmente conversores Digital /Analógico.

De lo expuesto se pueden mencionar las siguientes características:

- Deben responder a diversos eventos externos, asegurando un tiempo de respuesta máximo.
- La secuencia de ejecución de las tareas del sistema no solo están determinadas por decisiones del sistema, sino por eventos que ocurren en el mundo real.
- Deben ofrecer facilidades de interfaz con una gran cantidad de dispositivos, fundamentalmente sensores y actuadores.
- Generalmente son complejos.
- Deben presentar un alto nivel de seguridad, es crítico asegurar la confiabilidad.
- No pueden volver atrás y reiniciar la ejecución desde un contexto preexistente.
- Las demandas externas suelen ser en paralelo, provocando problemas de planificación y prioridades.
- Son de tiempo finito, lo que significa que deben estar preparados para recuperarse de situaciones de excepción.

Los metódos de medición de la eficacia de estos sistemas difieren de los utilizados para medir los sistemas tradicionales, criterios tales como rendimiento (cantidad de trabajo por unidad de tiempo), tiempo de retorno (tiempo entre el comienzo y fin de un trabajo), grado de utilización de recursos, no son indicativos. En cambio son necesarios los siguientes requisitos:

- *Predictibilidad*: se desea obtener una respuesta predecible ante eventos urgentes.
- *Alto grado de planificabilidad*: ejecutar la mayor cantidad de tareas posibles con tiempos de respuesta predecible.
- *Estabilidad frente a sobrecargas momentáneas*: aún en estas condiciones se debe garantizar el cumplimiento de tareas críticas.
- *Confiabilidad*: las restricciones de tiempo real no se cumplirán si el sistema no es confiable.

- *Adaptabilidad a cambios de configuración*: si el sistema es adaptable no se tendrá que redefinir el mismo ante cada pequeño cambio del mundo real.

Por lo dicho podemos adoptar como definición de sistema de tiempo real "sistemas en los cuales la ejecución de las tareas está determinada por el paso del tiempo u ocurrencia de eventos externos, y los resultados obtenidos pueden depender del momento en que se ejecutaron o del tiempo que se tardó en hacerlo".

En esta definición pueden surgir confusiones del significado de eventos externos, por ejemplo un cajero automático podría confundirse con un sistema de tiempo real ya que depende de eventos externos como ser el paso de la tarjeta, pero una vez que el sistema detecto la tarjeta, el sistema se comporta como un sistema interactivo y los tiempos de respuesta depende del momento en que se obtengan de una base de datos, si es rápido es mejor pero si el operador tiene que esperar esto no es crítico.

1.2 CLASIFICACIÓN DE LOS S.T.R.

Los sistemas de tiempo real se pueden clasificar según las restricciones del tiempo o por la relación entre las escalas de tiempo, de la siguiente forma:

a) Por las restricciones de tiempo
- *Hard*: (Tiempo real estricto) los cálculos siempre deben realizarse en un tiempo menor al máximo especificado.
- *Soft*: (Tiempo real flexible) el sistema debe tener un tiempo promedio inferior al especificado.
- *Firm*: (Tiempo real firme) en esta categoría se incluyen los sistemas duros que pueden tolerar perdidas, si es que existe una probabilidad de que estas ocurran.

b) Por la relación entre las escalas de tiempo
- *Basados en eventos*: es cualquier ocurrencia que causa que el contador del programa (PC) cambie ***no secuencialmente*** cambiando así el flujo de control

- *Basados en reloj*: es el tiempo en el cual una instancia de una tarea planificada esta lista para correr y esta generalmente asociada con una interrupción.
- *Interactivos*: la relación entre la computadora y el mundo exterior es vaga, típicamente se requiere que una serie de operaciones se realicen en un tiempo promedio determinado.

Por su integración con el entorno físico, los sistemas los podemos clasificar en:

- *Embebidos*: se usan para controlar un hardware especializado en el cual se instala el sistema de computación.
- *No embebidos*: estos sistemas se dividen en *orgánicos* si son completamente independientes del hardware o *débilmente acoplados* si pueden ejecutarse en otro hardware reescribiendo solo parte del código.

1.3 HARDWARE PARA TIEMPO REAL

Se describen, muy brevemente, algunos elementos de hardware que se encuentran presente en estos sistemas. Para mayor profundidad sobre estos temas se remite al lector a la bibliografía recomendada.

1.3.a Procesador

En una arquitectura CISC el tiempo de levantar y ejecutar una instrucción compleja puede ser muy alta, como una instrucción no se puede interrumpir en medio de su ejecución, podrían aparecer problemas de sincronización temporal si aparece un evento de alta prioridad. Estos problemas no existen en los procesadores RISC, en estas arquitecturas hay que considerar otros problemas provocados por el alto grado de paralelismo producto del uso intensivo del pipeline. Un evento provoca el vaciamiento del pipeline y demoras impredecibles.
La tasa de transferencia de información es crucial para las aplicaciones de tiempo real. Considerando el alto grado de iteración

con el ambiente, tasas de transferencia baja pueden disminuir el desempeño general del sistema.

1.3.b Memoria

El principal problema de la memoria en relación con los sistemas de tiempo real, son su tamaño y tiempo de acceso. En la actualidad estos tiempos se han reducido considerablemente y la tendencia es a seguir mejorando.

1.3.c Dispositivos de entrada / salida

Una característica de estos sistemas es la diversidad de dispositivos existentes y la variedad de tasas de transferencia. Además como se requiere interactuar con el mundo real será necesario utilizar conversores A/D y D/A.
Tipos de transferencia: el principal problema de sincronización reside en que la mayoría de los dispositivos operan en forma asincrónica con respecto al procesador y son mucho más lentos que este. La técnica más simple de sincronización es hacer que el microprocesador controle los dispositivos, el programa deberá reconocer los tiempos de espera y por ende desperdiciar su tiempo. Una mejora es la utilización de la técnica de *polling* (transferencia condicional). Las *interrupciones* son esenciales para la mayoría de los sistemas de tiempo real. Una técnica que combina el *polling* con las interrupciones es el uso de buffers de entrada/salida.

Si las tasas de transferencias son altas, el uso de interrupciones es ineficiente por la pérdida de tiempo que significa ejecutar la rutina de atención de la interrupción, una alternativa a esta es el uso de acceso directo a memoria (DMA), esta técnica permite que un dispositivo use los buses para transferir la información directamente a memoria sin el uso del microprocesador. En este proceso se deben atender los problemas de colisión en el bus cuando más de un dispositivo quiere transferir datos.

Dispositivos de intercambio con el ambiente: Los transductores (sensores y actuadores) son el vínculo principal entre la computadora y un proceso del mundo real.

Los sensores y actuadores deben conectarse utilizando técnicas de aislamiento y protección para no perjudicar el microprocesador.

Un *sensor* es un elemento que está en contacto con una variable física que se quiere medir y provee una salida proporcional a dicha variable. Existen diversos tipos pero básicamente hablaremos de sensores activos y pasivos, los activos autogeneran una tensión y los pasivos requieren de una fuente externa de alimentación.

Si se llama V_e a los valores de las variables de entrada y V_s a los valores de salida, se dice que el rango de medición es el conjunto de valores entre los límites que pueden ser medidos, se llama alcance al conjunto de valores de salida entre los límites generados por el sensor en todo el rango de entrada. Se llama sensibilidad a la relación de incrementos de la señal de salida con respecto a la señal de entrada. La resolución del sensor es el mínimo valor que se puede medir con certeza.

La exactitud de un sensor está dada por: *linealidad* (la mejor recta que se obtiene de una curva de calibración). *Repetitividad* (capacidad de repetir una medida en idénticas condiciones). *Corrimiento de cero* (valor entregado por el sensor al medir un valor cero). *Histéresis* (diferencia de valores en la salida para idéntica variación de la entrada, primero en forma creciente y después en forma descendente).

Corrimiento de la sensibilidad (cambios en la pendiente de la recta de aproximación de la función de sensado).

Actualmente existen sensores asociados a microprocesadores que se denominan sensores inteligentes (Smart sensor).

Un actuador es un dispositivo físico que provoca una alteración del ambiente con el cual interactúa. Existen diversas características que deben ser consideradas. Por un lado el actuador debe tener la potencia suficiente para mover el dispositivo de salida, también debe considerarse la aceleración, la capacidad de desplazamiento, la zona muerta, la saturación, la linealidad y la histéresis.

1.3.d Interfaces para dispositivos de entrada / salida

A pesar de la gran variedad de dispositivos de E/S, existe un número limitado de interfaces entre ellos y el microprocesador.
Las interfaces digitales (Figura 1-1) permiten el intercambio de señales digitales y básicamente constan de un registro digital que se conectan con los dispositivos. Debe notarse que en algunos casos el tiempo que toma transferir los datos del bus al procesador puede ser superior al tiempo durante el cual el dato es válido, por lo tanto al diseñar estas interfaces se debe tener especial cuidado con el tiempo de transferencia.

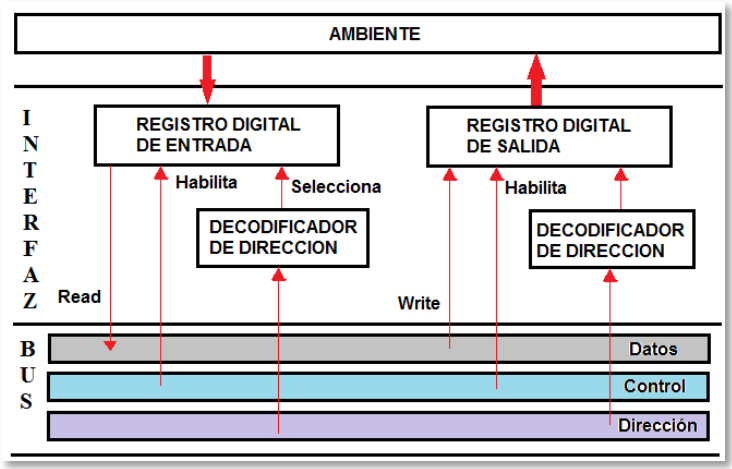


Figura 1-1: Interfaz Digital

Cuando se trata de sensores analógicos, se hace necesario incorporar conversores analógico/digital, esto implica realizar tareas de muestreo, retención y cuantificación. Normalmente estos circuitos son compartidos por múltiples sensores mediante un multiplexado de las señales de entrada. (Figura 1-2)

Para iniciar una operación de entrada, el procesador debe seleccionar el sensor, a continuación el procesador emite la señal de inicio de conversión, en respuesta a la misma el circuito inicia el muestreo y la retención durante un periodo de tiempo que permita

la cuantificación. En la actualidad existen diversas formas de cuantificación, los más utilizados son los de rampa, aproximaciones sucesivas y paralelo.

Al completar la cuantificación, el conversor activa la línea de fin de conversión que podrá ser leída por el procesador o ser utilizada como interrupción del microprocesador.

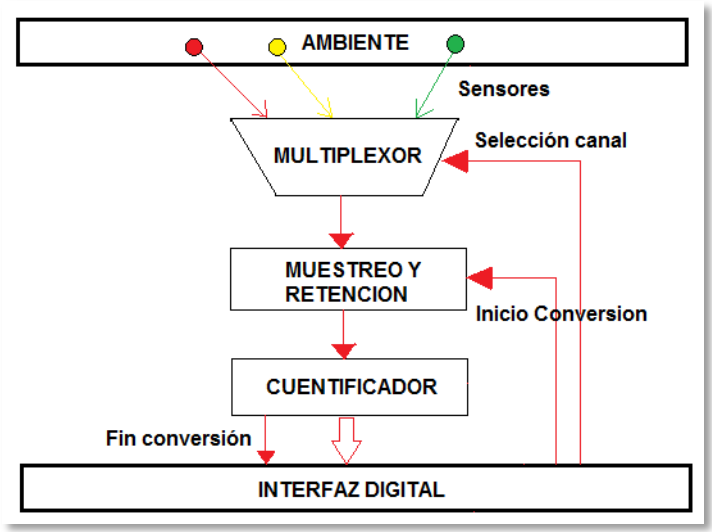


Figura 1-2: Conversión A/D

1.3.e Microcontroladores

Los microcontroladores aparecieron en la década de los 80 y han evolucionado rápidamente desde los procesadores de 8 bit a los actuales de 32 bits. Estos dispositivos de alto nivel de integración, incluyen un microprocesador, diversos tipos de memoria (EEPROM, FLASH, RAM), conversores A/D y módulos de E/S.
En la figura 1-3 se resumen algunos de los elementos que siempre se encuentran en el interior de un microcontrolador.

1.3.f Otros dispositivos

En la actualidad existe una gran variedad de dispositivos de E/S que sirven de enlace con el microprocesador: el teclado, el display, el reloj de tiempo real dispositivos RFID, entre otros.

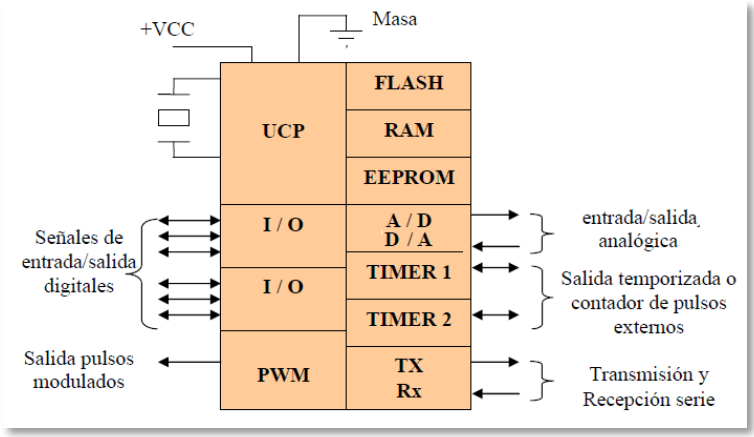


Figura 1-3: Principales bloque que integran un microcontrolador

CAPITULO 2: MODELADO Y SIMULACIÓN

2.1 ¿QUÉ ES EL MODELADO?

La creciente complejidad de los sistemas informáticos ha llegado a tal nivel que aquellos de mayor envergadura son comparables en dificultad y tamaño con las grandes obras de otras ramas de la ingeniería. Esta complejidad comporta dos cuestiones fundamentales. Por un lado, es difícil llegar a construir un sistema tan sofisticado, especialmente si no se tiene experiencia previa ni información básica sobre su composición. Por otro lado, también es difícil establecer a priori si el sistema funcionara correctamente una vez construido, lo que es especialmente grave en aquellos sistemas cuyo costo es elevado, o son especialmente difíciles de modificar una vez construidos, o llevan a cabo tareas muy delicadas o peligrosas. En estas otras ramas de las ciencias es tradicional el uso de modelos que permitan un análisis previo de las características y el funcionamiento del sistema. El uso de modelos es una herramienta básica para tratar esta complejidad, ya que permite hacer una réplica más simple del sistema, en la que se eliminan detalles que no son fundamentales, obteniendo así un objeto de estudio más sencillo de entender, manejar y que permite hacer predicciones sobre aspectos importantes del sistema real. Un buen modelo debe tener varias características:

- Debe permitir la **abstracción**, para poder obviar detalles irrelevantes en un momento dado para el análisis de ciertas propiedades concretas del sistema. Además el grado de abstracción debe ser variable y así permitir que el análisis sea a mayor o menor nivel de detalle, según sea necesario.

- Debe usar notaciones que permitan a un lector entender el sistema –**Comprensible**-. Si la notación usada es difícil de entender el modelo será de poca utilidad, incluso para sistemas con un mínimo de complejidad.
- Debe mostrar las mismas características que el sistema final –**Preciso**-, al menos en aquellos aspectos que quieran ser estudiados o analizados antes de la construcción del sistema final.
- Deben ser **Predictivo**. Pueden utilizarse para responder cuestiones sobre el sistema modelado
- Debe tener una base matemática o **formal** que permita la demostración de propiedades, con el fin de poder predecir el funcionamiento del sistema una vez construido. Debe ser significativamente más fácil y económico de construir el sistema final.

2.1.a Necesidad del modelado

Aprovechando los avances en la construcción de procesadores con cada vez mayor capacidad de cálculo y de almacenamiento de información, los sistemas informáticos son cada vez más grandes, se aplican a más campos y se depositan en ellos responsabilidades mayores. Esta situación provoca una mayor exigencia sobre los sistemas. No solo han de ser sistemas que proporcionen un resultado correcto, sino que tienen otra serie de requisitos entre los que se pueden citar la obligación de ser sistemas seguros o responder satisfactoriamente frente a situaciones no esperadas o de error. El desarrollo de sistemas es una rama de la ingeniería para la que aún no hay técnicas de desarrollo que conciten la aprobación generalizada. Sin embargo, en lo que si se está cada vez más de acuerdo es en la necesidad, y en los beneficios que conlleva, el usar modelos para guiar la construcción de los sistemas reales, de forma que se puedan analizar las propiedades del sistema final antes y durante su desarrollo. Diferentes autores han propuesto múltiples modelos distintos, influenciados por el tipo de sistemas que desarrollaban en ese momento, por el campo de aplicación para el que se proponían, etc. aún hoy en día, la diversidad es grande y se

está lejos de la unanimidad, o de la universalidad de los modelos, por lo que se sigue investigando ampliamente en el tema.

2.1.b Los sistemas de tiempo real y su modelado

Los sistemas de tiempo real son una clase concreta de sistemas que se pueden definir de manera informal como aquellos sistemas en los que el tiempo de respuesta es crucial para su funcionamiento correcto. También se dice que en un sistema de tiempo real, un dato obtenido fuera de plazo, aunque sea correcto, es un dato inválido, que incluso puede provocar que el sistema falle en su conjunto.
A este tipo de sistemas de tiempo real se les llama embebidos (embedded real-time systems). Es habitual que esos sistemas embebidos impongan fuertes restricciones en varios aspectos. Por un lado, los recursos físicos con los que se cuenta, como memoria y capacidad de cálculo, suelen estar muy ajustados, lo que incide en una mayor dificultad para encontrar una solución viable. Los recursos de software, como bibliotecas de funciones o sistemas operativos, pueden también estar limitados, ya que es habitual la ausencia de versiones para estos entornos no estándares.

El requisito temporal hace necesario el análisis de la planificabilidad del sistema, que establece si se pueden cumplir o no los plazos temporales y, si no se puede, cuáles son los que fallan. Estas particularidades de los sistemas de tiempo real impiden, o limitan, que los modelos y metodologías de desarrollo de sistemas informáticos en general se puedan aplicar a los sistemas de tiempo real o, al menos, que sean suficientes. De aquí surge la necesidad de complementar modelos generales o desarrollar otros nuevos para tener en cuenta las características adicionales que presentan los sistemas de tiempo real.

2.1.c Ventajas del diseño dirigido por modelos

Los statecharts permiten construir modelos gráficos que describen con precisión el comportamiento de un sistema. Los modelos creados no tienen ninguna relación con el lenguaje de programación que vaya a utilizarse en el desarrollo de la aplicación, sin embargo, si tienen una relación muy estrecha con el funcionamiento deseado de

la aplicación. Esta relación facilita la comunicación y el intercambio de ideas entre el cliente y el equipo de desarrollo del sistema, independientemente del tipo de formación que posean los miembros del equipo. El modelo permite simular y visualizar la aplicación desde las primeras etapas del diseño sin necesidad de construir un prototipo hardware; esta característica facilita la eliminación de errores desde el principio. Los programadores deben cambiar la forma tradicional en la que abordan la tarea de desarrollo de software trasladando su forma de pensar al dominio de la aplicación y liberándose de las limitaciones impuestas por el lenguaje de programación utilizado. Si la herramienta de modelado dispone de generadores automáticos de código y de documentación los beneficios son aún mayores, ya que los cambios que se realizan y simulan en el modelo se trasladan automáticamente al código generado y a la documentación generada, por lo que la propia herramienta se encarga de mantener el sincronismo entre el modelo, el código y la documentación. Disponer de documentación actualizada es un aspecto de enorme importancia ya que facilita el mantenimiento de las aplicaciones.

2.2 INTRODUCCIÓN A STATECHART

La idea principal de los Statechart es la extensión al formalismo de las máquinas y diagramas de estados. Esta idea fue propuesta por David Harel en la década de los 80.

"Los Statechart **extienden los diagramas de transición de estados** convencionales con tres elementos principales: *jerarquía*, *concurrencia* y *comunicación*. El uso de jerarquías permite tratar los sistemas con diferentes niveles de detalle; la concurrencia, también llamada ortogonalidad y paralelismo, posibilitando la existencia de tareas independientes entre sí o con escasa relación entre ellas, y la comunicación hace viable que varias tareas reaccionen ante un mismo evento o envíen mensajes hacia otras tareas."

2.1.a VisualSTATE IAR®

VisualSTATE de IAR Systems es un entorno de desarrollo para la creación de código en C, basado en Statecharts. Comprende las siguientes herramientas y estándares API:

- *visualSTATE Navigator*: Navegador para el manejo global de proyectos y archivos visualSTATE. Además se utiliza para correr otros módulos del software.
- *visualSTATE Designer*: Aplicación basada en gráficos para el diseño de diagramas de estado usando la notación UML.
- *visualSTATE Validator*: Aplicación basada en gráficos para simulación, análisis, depuración de sistemas visualSTATE desarrollados con visualSTATE Designer.
- *visualSTATE Verificator*: Programa de gran alcance para la verificación dinámica formal de los modelos visualSTATE.
- *visualSTATE Coder*: Puede generar automáticamente código visualSTATE, desarrollado con visualSTATE Designer.
- *visualSTATE Documenter*: Se utiliza para generar informes de documentación sobre los modelos visualSTATE.
- *visualSTATE OSEK kit*: Proporciona una interfaz de usuario amigable utilizando el software visualSTATE en un entorno OSEK.
- *visualSTATE APIs*: API básico, Api experto, DLL experto.

En los siguientes apartados se detallarán los principales componentes de los diagramas de estados (Statechart), aplicados en visualSTATE y la manera de utilizarlos. Las definiciones de este apartado fueron tomadas del documento "*Reference Guide*" de IAR visualSTATE [IAR VisualSTATE - 2008].

Máquina de estados

Una máquina de estados representa una especificación de un modelo. La máquina de estados se compone de una cantidad finita de estados y un conjunto de transiciones. Una máquina de estados puede estar solo en un estado a la vez. (Figura 2-1).

VisualSTATE se utiliza para modelar *máquinas de estados jerárquicas*. De esta manera, una máquina de estados puede contener otras máquinas de estados. Este es el caso cuando uno de los estados de la máquina es un súper estado. Un súper estado se asignará a una máquina de estados si se compone de estados mutuamente excluyentes. Si el súper estado se compone de regiones concurrentes, cada región se asignará a una máquina de estados.

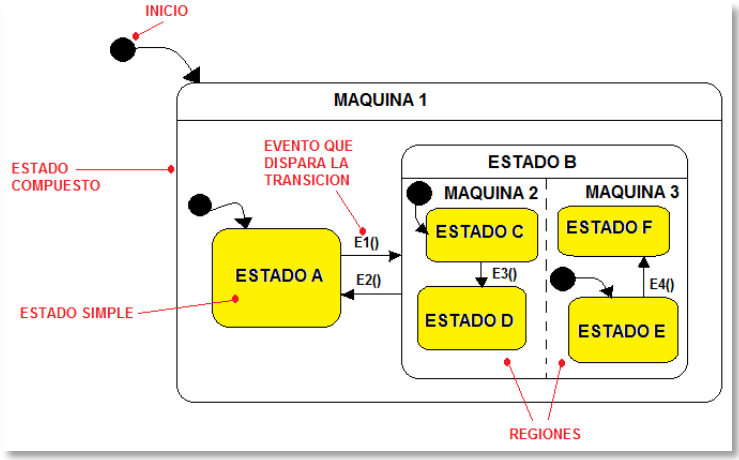


Figura 2-1: Maquina de estados en VisualState

Estados y Regiones

Un estado es una situación en la vida de determinado objeto, en la cual este cumple determinada condición, ejecuta determinada acción, o es receptivo a determinados eventos. En visualSTATE se va cambiando de un estado a otro en función de los eventos que ocurran y durante esa transición se van ejecutando las acciones asociadas a dicho cambio (transiciones) de estado

En este apartado se presentarán los distintos tipos de estados, con sus respectivos significado lógico y representación gráfica. La definición de cada uno de los estados y las imágenes de este apartado fueron tomadas del documento "Reference Guide" de IAR visualSTATE.

- **Topstate:** El topstate es el estado con mayor jerarquía de los estados. Típicamente contiene algunas regiones concurrentes. El topstate siempre está activo, por lo que las únicas reacción de estado permitidas son la de entrada (entry), cuando se resetea el sistema, y las reacciones internas. Las reacciones de salida (exit) no están permitidas, porque nunca se ejecutaría.
- **Estado simple:** El estado simple se encuentra en lo más bajo de la jerarquía de estados y no contiene otros estados o regiones.
- **Estado compuesto:** Como indica el nombre, el estado compuesto es un estado que está compuesto por otros estados, en particular por dos o más regiones concurrentes o por estados exclusivos.
- **Región:** Las regiones son usadas para representar máquinas de estados. Las regiones aparecen como regiones concurrentes en el topstate o en estados compuestos.
- **Región fuera de página ( ):** Las regiones fuera de página se utilizan para modularizar modelos complejos en diagramas de statechart. Estas hacen posible mover la lógica de control de un estado compuesto a otro Statechart. La representación gráfica de una región fuera de página es un estado con un pequeño símbolo de Statechart que indica que esta región contiene una máquina de estados. (Figura 2-2).

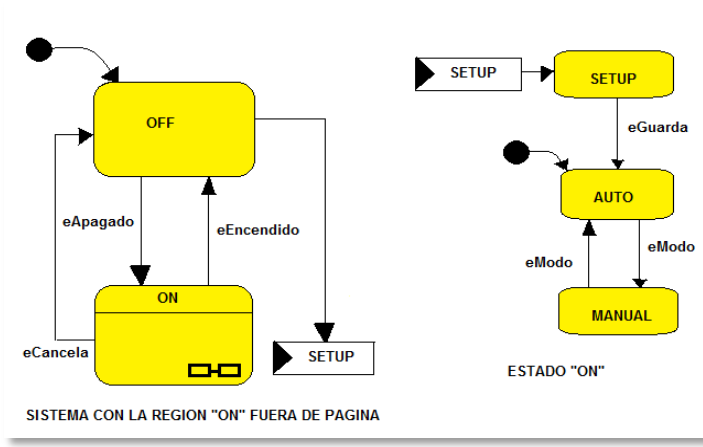


Figura 2-2: Máquina con estado fuera de página

- **Estado por defecto:** El estado por defecto queda definido por las fuentes de transición inicial. Las posibles fuentes son el estado inicial, o por los estados con historiales (superficial o profundo).
- **Estado inicial:** Cuando la fuente de transición inicial es un estado inicial se ejecuta la transición y se entra en el estado por defecto. (Figura 2-3)
- **Estado con historial superficial:** Este historial guarda el último estado dentro de una región o estado compuesto, pero no el último sub-estado dentro de alguno de los estados.
  La primera vez se toma el estado apuntado para la transición del estado con historial superficial. Antes de salir del estado guardará el último sub-estado en que estuvo. La próxima vez que entre a este estado irá al sub-estado guardado en el historial. Cabe mencionar que si alguno de los sub-estados fuera una máquina de estados, no se guardará el último estado de este, a no ser que se indique con un estado con historial superficial.

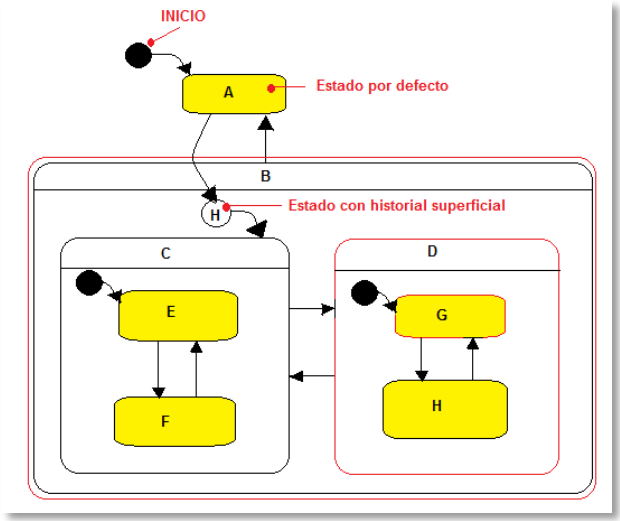


Figura 2-3: Estado inicial, por defecto y con historial superficial

La figura 2-3 ilustra un diagrama donde se muestran un estado inicial, un estado por defecto y un estado con historial

superficial. Si al salir del estado B, el último sub-estado era el H, el historial superficial recordará que estaba en el estado D, pero al volver al estado D **irá al estado por defecto G.**

- **Estado con historial profundo:** El estado con historial profundo es similar al historial superficial, solo que este recuerda el estado final de todos los sub-estados también. En la figura 2-4 se muestra un diagrama con el uso de estado con historial profundo (H*), en este caso si al salir del estado B, el último sub-estado era el H, el historial guardará esto y al volver al estado B se **volverá al estado H.**

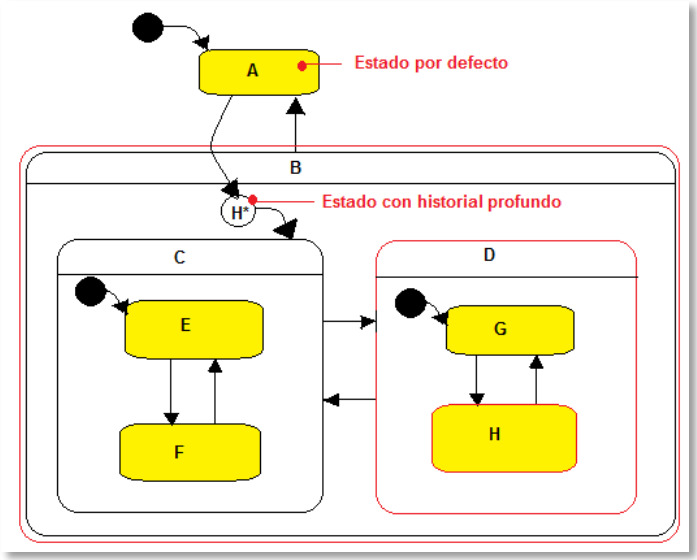


Figura 2-4: Estado con historial profundo

- **Estado final:** el estado final se utiliza para visualizar que un objeto o una máquina de estados ha terminado su operación.
- **Transiciones:** una transición es una relación entre dos estados, que indica que cuando una máquina está en un estado y ocurre

un cierto evento, la máquina realiza una o más acciones y cambia al estado destino.

En visualSTATE una transición está dividida en dos lados, a la izquierda el lado de las condiciones y a la derecha el lado de las acciones, separados por una barra (/). El lado de las condiciones indica que condiciones se debe cumplir para cambiar de estado. Mientras que el lado de las acciones indica todas las acciones que se deben ejecutar si las condiciones se cumplen.

- **Disparador (Trigger):** En visualSTATE el disparador es lo que hace que se evalúen las condiciones de una transición. Entonces cuando ocurre un disparo, y se evalúa como verdaderas las condiciones se ejecuta la transición. En visualSTATE existen dos tipos de disparadores, explícitos e implícitos.

- ***Disparadores explícitos:*** Los disparadores explícitos son los siguientes: un evento, un grupo de eventos y una señal.
  - **Evento:** Un evento es algo que sucede en un entorno externo al sistema de visualSTATE. Los eventos son procesados siempre de manera secuencial. Como se considera que un evento es una entrada momentánea (por ejemplo un botón que se presiona), debe ser capturado y almacenado antes de que pueda ser interpretado por visualSTATE. Algo que resulta interesante es que los eventos pueden tener parámetros.
  - **Grupo de eventos:** Un grupo de eventos es una colección de eventos independientes. Cuando uno de estos eventos ocurre, se ejecuta una transición específica, siempre que las condiciones de transición se cumplan. Un grupo de eventos en sí mismo nunca ocurre, solo uno de los eventos ocurre. Un evento puede ser miembro de más de un grupo de eventos.
  - **Señales:** Una señal se usa para disparar transiciones como un evento. A diferencia de los eventos, que ocurren en un entorno externo, las señales se envían de forma interna en visualSTATE. Las señales son como funciones de disparo interno, por ello se puede encontrar señales en los dos

lados de una transición. VisualSTATE cuenta con una cola para las señales, pero el usuario debe encargarse de dimensionar el tamaño de esta cola.

- ***Disparadores implícitos:*** Los disparadores implícitos son llamados también reacciones de estado y son las siguientes: entrada (**entry**), salida (**exit**) y hacer (**do**). Además existen las reacciones internas que no son disparadoras.
  - **Entry:** Es una reacción que se ejecuta cuando entra a un estado.
  - **Exit:** Esta reacción se ejecuta al salir del estado.
  - **Do Reaction:** Esta reacción, que en la especificación de UML se llama Do Activity, es una actividad que se ejecuta de forma continua cada vez que se entra al estado. En visualSTATE una reacción do es interpretada como una máquina de estados independiente. La reacción do contiene un estado simple y un estado final.

- **Internal:** Es una reacción que se ejecuta sin salir del estado, por lo que no tienen estado de destino.

- **Expresión de guarda:** La expresión de guarda se encuentra en el lado de condiciones de una transición, y para que se ejecute una transición todas las expresiones de guarda de la misma deben ser TRUE.
  Una expresión de guarda puede ser tanto una expresión lógica, como una expresión de relación. Una expresión lógica es una o más expresiones de relación separadas por operadores lógicos. Una expresión de relación se compone en tres partes, el lado de la izquierda, el operador de relación y el lado de la derecha. El lado de la izquierda puede ser una variable externa, una variable interna, el parámetro de un evento, o una constante. El lado de la derecha una expresión compleja, que involucra operadores y operandos de distintos tipos.

  Como ejemplo: (x >= 0) && (x < 10 + Action(7,3))

- **Condición de estado:** Las condiciones de estado aseguran que otra máquina de estado, dentro del mismo sistema de visualSTATE satisface condiciones específicas antes de ejecutar la transición.

Las condiciones de estado se encuentran en el lado de las condiciones de una transición. Para que se ejecute una transición todas las condiciones, incluidas las de estado, deben ser satisfechas.

Existen tres tipos de condiciones de estado: estado fuente, condiciones de estado positivo y condiciones de estado negativo. El estado fuente es el estado donde se origina la transición. La condición de estado positivo se da cuando otra máquina de estado se encuentra en un estado específico. Y la condición de estado negativo es lo opuesto al estado positivo. Cabe mencionar que el NOT se denota con el símbolo '!'.

- **Asignaciones:** Una asignación es el cambio del valor de una variable. El operador de asignación es el signo de igual (=). Del lado izquierdo del operador es la variable a la que se le va a asignar el nuevo valor. Del lado derecho puede escribirse una expresión (usando operadores de aritmética binaria, operadores lógicos o manipulación de bits, etc.) o simplemente el valor.
- **Acción de estado:** Existen dos acciones de estado: acción forzar estado y estado de destino.
- **Acción Forzar Estado:** Es un método mediante el cual se puede forzar el cambio a un estado específico de otra máquina de estados, independientemente de si el estado está activo y de las transiciones del estado. Obviamente esta acción se encuentra en el lado de las Acciónes de una transición.
- **Estado De Destino:** Es el estado en que se entra en una determinada transición.

2.3 ESPECIFICACIONES DEL SISTEMA A DISEÑAR

En este apartado se describen ejemplos destinados a ilustrar el diseño de sistemas embebidos mediante statecharts.
Algunos de los ejemplos han sido tomados de otros autores, a quienes agradecemos el aporte, por ser lo suficientes claros en la metodología.

2.3.a Modelado de un reloj digital

En este primer ejemplo [Baron Ruiz - 2010], se exploran elementos importantes de los statecharts, aunque no todos, y muestra la nueva metodología de diseño partiendo de las especificaciones del sistema.

Figura 2-5. Visualizador del reloj digital a diseñar

Se trata de diseñar un reloj digital provisto de un visualizador (Figura 2-5), y de cuatro pulsadores para controlar su funcionamiento. Las especificaciones son las siguientes:

1. El pulsador **Set/Run** servirá para conmutar entre dos modos de trabajo: la puesta en hora del reloj o el funcionamiento normal como reloj.

2. El pulsador **Hora+** servirá para incrementar la hora en modo Set, y no tendrá efecto en modo Run. Si este pulsador se mantiene pulsado durante más de 1 s. en modo Set, la hora se incrementará cada 0,5 segundos.

3. El pulsador **Minuto+** tendrá un funcionamiento idéntico al pulsador **Hora+** para permitir el ajuste de los minutos.

4. El reloj no dispone de ningún pulsador para modificar los segundos; al entrar en modo Set los segundos se pondrán a cero y comenzarán a incrementarse al pasar al modo normal de funcionamiento como reloj.

5. El pulsador **12/24** servirá para cambiar entre el formato de visualización de 12 Horas con indicación AM/PM y el formato 24 Horas.

6. El indicador **SET** se activará cuando el reloj se encuentre en el modo de puesta en hora.

7. El indicador **12h** se activará cuando el reloj se encuentre en el formato de visualización de 12 horas.

8. Los indicadores **AM** y **PM** sólo se activarán en el formato de visualización de 12H.

9. El indicador **24h** se activará cuando el reloj se encuentre en el formato de visualización de 24 horas.

10. Los indicadores **(:)** que separan las horas, minutos y segundos cambiarán de estado con una frecuencia de 1 Hz, activándose y desactivándose cada 0,5 s.

Procedimiento de diseño con Statecharts

Para diseñar sistemas con máquinas de estados jerárquicas puede seguirse el siguiente procedimiento de diseño que consta de seis pasos:

Paso1. Identificar los eventos y las acciones

Los eventos representan la influencia del ambiente sobre el sistema; son las entradas a la máquina de estados. En nuestro ejemplo los eventos los produce el usuario al presionar o soltar los pulsadores y también el oscilador que proporciona la precisión del reloj generando un tic periódico. La unidad más pequeña de tiempo que utiliza el diseño son los 0,5 segundos necesarios para conmutar el estado del indicador (:) separador de horas, minutos y segundos y también para incrementar las horas y minutos en el modo de puesta en hora, cuando se mantiene accionando el pulsador **Hora+** o el pulsador **Minuto+** durante más de un segundo. Los eventos del sistema a diseñar se muestran en la Tabla 2-1.

Tabla 2-1: Eventos del reloj digital.

NOMBRE	SE PRODUCE CUANDO
e05s	Transcurren 0,5 s desde el tic anterior
e12_24	Se pulsa el botón *12/24*
eHora	Se pulsa el botón *Hora+*
eNoHora	Se libera el botón *Hora+*
eMinuto	Se pulsa el botón *Minuto+*
eNoMinuto	Se libera el botón *Minuto+*
eSetRun	Se pulsa el botón *Set/Run*

Las acciones representan la influencia del sistema sobre el ambiente; son las salidas de la máquina de estados. **Las acciones se realizan mediante llamadas a funciones** escritas en lenguaje C, los nombres de estas funciones y el trabajo que realizan aparecen en la Tabla 2-2.

Tabla 2-2: Acciónes del reloj digital.

NOMBRE	REALIZA EL SIGUIENTE TRABAJO
aLed2ON(void)	Activa el indicador de 2 punto
aLed2OFF(void)	Desactiva el indicador de 2 punto
aLedSetON(void)	Activa el indicador de set
aLedSetOFF(void)	Desactiva el indicador de set
aMostrarHora(void)	Muestra la hora en formato 12 o 24 hs

Paso 2. Identificar los estados

Un estado es una situación en la que se satisface alguna condición, se realiza alguna actividad, o se espera la llegada de algún evento. Los estados, que se representan por rectángulos con los bordes redondeados, se identifican a partir de las especificaciones y del conocimiento del problema.

En este ejemplo de diseño, los estados pueden ser los de la Figura 2-6: los indicadores **(:)** que separan horas, minutos y segundos pueden estar activados o desactivados (estados **SeparadorOn** y **SeparadorOff)**; el reloj puede estar en modo normal de funcionamiento o en modo de ajuste de hora o de ajuste de minuto (estados **RelojRun**, **HoraSet** y **MinutoSet)**; el pulsador **Hora+** puede encontrarse liberado, pulsado o retenido durante más de un segundo (estados **BotonHoraOff, BotonHoraOn** y **BotonHoraRet)**; igualmente el pulsador **Minuto+** puede encontrarse en los estados, **BotonMinutoOff, BotonMinutoOn** y **BotonMinutoRet**.

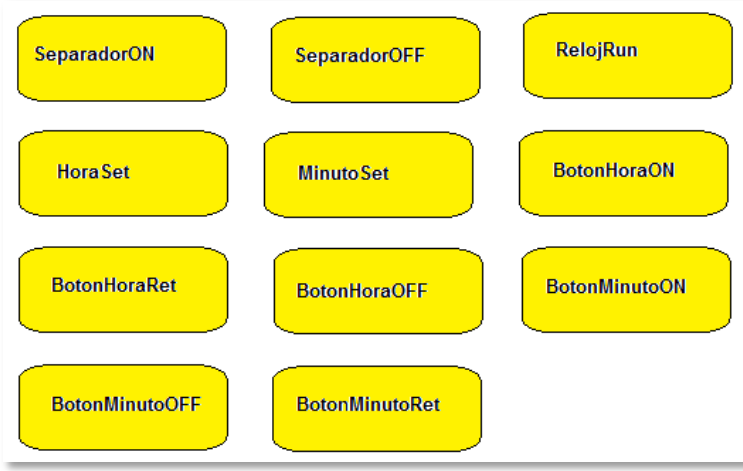


Figura 2-6. Estos estados recuerdan aspectos relevantes de la historia del reloj digital de manera muy eficiente.

Paso 3. Agrupar los estados por jerarquías

En este paso se trata de determinar los estados que tienen un comportamiento dinámico propio y los estados que sólo pueden estar activos en ciertas situaciones.

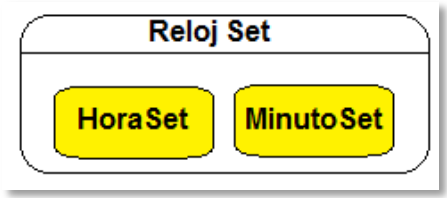


Figura 2-7: Agrupamiento de los estados del reloj digital. El ajuste de las horas y el ajuste de los minutos solo es posible si el reloj está en modo de ajuste.

En la figura 2-7, el estado **RelojSet** es un estado compuesto, o superestado, que tiene dos estados hijo, los estados **HoraSet** y **MinutoSet**. El nuevo agrupamiento permitirá pasar del modo de

ajuste al modo normal independientemente de que se esté ajustando la hora o los minutos.

Paso 4. Agrupar por concurrencia

Examinar los estados que pueden estar activos a la vez y organizar el modelo en varias máquinas de estados paralelas.

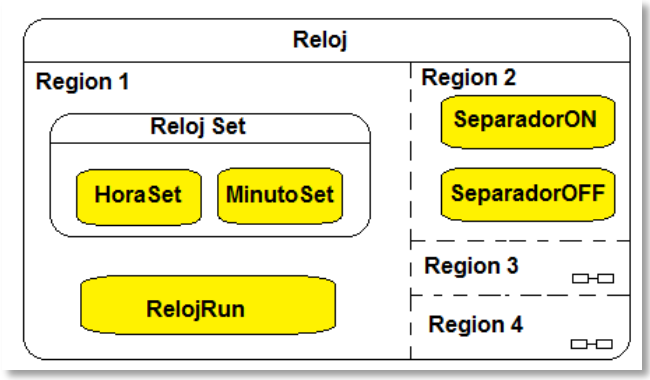


Figura 2-8: Organización del reloj digital en cuatro máquinas de estados paralelas. La Región3 contiene los tres estados correspondientes al pulsador Hora+, y la Región4 contiene los tres estados correspondientes al pulsador Minuto+; ambas regiones han sido declaradas Off-page para no complicar el diagrama y sus contenidos se despliegan en gráficos separados.

En cada instante de tiempo, una máquina de estados sólo puede encontrarse en un estado. Si un sistema puede estar en más de un estado simultáneamente hay que pensar en más de una máquina de estados. El reloj digital puede estar al mismo tiempo en modo normal de funcionamiento, con el indicador que separa horas minutos y segundos activado o desactivado, y con los pulsadores **Hora+** y **Minuto+** liberados, pulsados o retenidos, por ello el tratamiento del reloj debe hacerse con cuatro máquinas de estados concurrentes. La concurrencia se representa en los statecharts mediante regiones separadas por líneas discontinuas, tal como muestra la figura 2-8.

El contenido de cada región puede estar oculto o visible; en muchos casos, sobre todo en sistemas complejos, al mantener oculto el contenido de una región se tiene una mejor visión de la estructura

del modelo. Para mantener oculto el statechart de una región, se declara ésta *Off-page* y el contenido de la región se despliega en un gráfico separado. Las regiones offpage muestran un icono en la esquina inferior derecha (regiones 3 y 4 en figura 2-8). Para ver el contenido de la región off-page hay que hacer clic sobre el icono; pulsando la tecla retroceso se regresa al statechart que contiene la región.

El paralelismo o tratamiento separado de máquinas de estados independientes o que guardan poca relación, es un aspecto importante de los statecharts, ya que permite afrontar la complejidad sin necesidad de recurrir a sistemas operativos.

Paso 5. Añadir las acciones y transiciones

Identificar las acciones a realizar y los cambios de estados que se deben producir tras un evento.

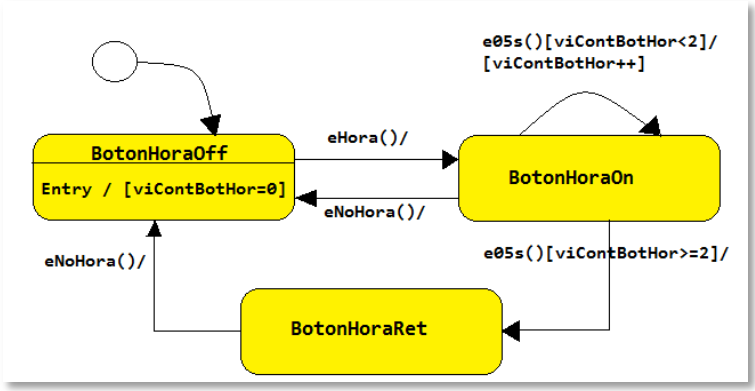


Figura 2-9. Statechart de la Región 4.

Las transiciones se representan por flechas dirigidas desde el estado origen hacia el estado destino. Cada transición consta de dos partes separadas por una barra inclinada "/"; el lado izquierdo, llamado lado de la condición, es donde aparece el disparador (evento o señal) y las guardas o condiciones exigidas, figura 2-9, para que se dispare la transición; en el lado derecho o lado de la acción aparecen las acciones a realizar en caso de que se satisfagan todas las condiciones del lado izquierdo. La figura 2-10 muestra la máquina de estados resultante después de haber añadido algunas

transiciones y Acciónes. El evento **e05s** (producido por el oscilador del reloj cada 0,5 s.) conmuta entre los estados **SeparadorOn** y **SeparadorOff**, activando y desactivando el indicador (:) y logrando que el diseño cumpla la especificación número 10; además, como el período de tiempo transcurrido entre dos entradas consecutivas a cualquiera de estos dos estados es de un segundo, se puede utilizar este hecho para incrementar la variable veSeg que cuenta los segundos. El evento **eSetRun** (producido cuando el usuario acciona el pulsador **Set/Run**) conmuta entre los dos modos de trabajo del reloj. El evento **eHora** (producido cuando el usuario acciona el pulsador **Hora+**) incrementa, módulo 24, la variable veHora que cuenta las horas, solo si el reloj se encuentra en el estado **RelojSet**.

En la figura 2-10 puede observarse que los estados **RelojSet**, **SeparadorOn** y **SeparadorOff** contienen reacciones internas disparadas por las palabras Entry y/o Exit. El nombre de reacción interna se debe a que no se trata de transiciones, ya que no hay cambio de estado, las acciones correspondientes a estas reacciones se ejecutan al entrar al (Entry) o al salir del (Exit) estado que las contiene. Así, al entrar en el estado **RelojSet** se activa el indicador SET y se ponen los segundos a cero, y al salir de este estado se desactiva el indicador SET.

También en la figura 2-10 aparecen tres pequeños círculos sin nada en su interior; se trata del pseudo-estado *inicial*, que hace que cada máquina de estados concurrente se inicie en un estado conocido. Un pseudo-estado es un estado transitorio; cuando una máquina de estados alcanza un pseudo-estado se ejecuta automáticamente la transición de salida desde ese pseudo-estado; por ello, las transiciones desde los pseudoestados no llevan eventos de disparo, aunque pueden llevar acciones asociadas. Cuando comienza a ejecutarse la máquina de estados Reloj, se dispara la transición inicial que asigna a la variable ve24H el valor 1 para que el reloj muestre la hora en formato 24 horas, e inmediatamente después se disparan tantas transiciones iniciales como máquinas de estados paralelas existan, haciendo que la máquina de la Región 1 arranque en el estado **RelojSet**, la máquina de la Región 2 se inicie en el estado **SeparadorOn** y las máquinas de las Regiones 3 y 4, figura 2-10, se inicien en sus estados correspondientes.

Las regiones 3 y 4 contienen los statecharts de las máquinas de estados asociadas al comportamiento de los pulsadores **Minuto+** y **Hora+**; ambas máquinas tienen un funcionamiento similar, ver figura 6; cuando se Accióna el pulsador **Hora+**, se pasa del estado **BotonHoraOff** al estado **BotonHoraOn** y si se retiene el pulsador durante 1 s. se pasa al estado **BotonHoraRet** en el cual la Hora del reloj se incrementa cada 0,5 s siempre que el reloj se encuentre en el estado **RelojSet**.

Para saber si el botón se ha mantenido retenido durante más de 1 s. se utiliza el evento **e05s** y la variable viContBotHor; ésta variable se inicia con valor 0 y se incrementa cada 0,5 s., cuando la variable tome valor 2 o superior será porque se ha retenido el pulsador el tiempo requerido.

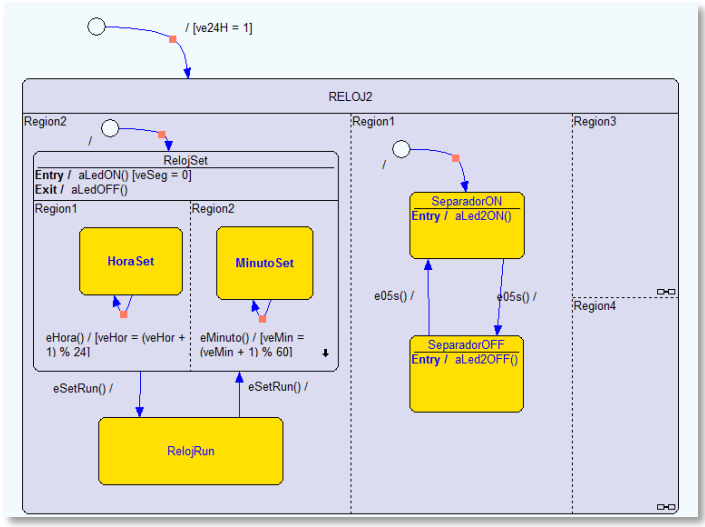


Figura 2-10. Statechart obtenido después de añadir las transiciones. Las líneas que comienzan con las palabras Entry y Exit, dentro de los estados simples, con fondo Amarillo, son reAcciónes internas que se ejecutan automáticamente al entrar al estado o al salir del estado.

De las dos transiciones disparadas por el evento e05s que parten del estado BotonHoraOn, sólo se dispara aquella que satisfaga la expresión de guarda [viContBotHor<2] o su condición contraria [viContBotHor>=2].

Paso 6. Añadir las sincronizaciones

Las sincronizaciones son los mensajes internos que una máquina de estados puede enviar a otra máquina. En visualSTATE, la sincronización entre máquinas de estados se logra utilizando **señales** y/o **condiciones de estado**. Las señales, al igual que los eventos, se usan para disparar transiciones. Las señales se diferencian de los eventos en que éstos se producen en el exterior y las señales las genera otra máquina de estados dentro de visualSTATE. Las expresiones de guarda y las condiciones de estado son efectivamente condiciones que deben cumplirse antes de que una transición se dispare; así en el lado de la condición (lado izquierdo) de una transición de una máquina de estados, puede haber condiciones que exijan que otra máquina de estados se encuentre, o no se encuentre, en otro estado. En el caso del reloj digital, para que el contador de horas se incremente cada 0,5 s. se deben cumplir tres condiciones: que se produzca el evento **e05s**, que la máquina de la Región 1 se encuentre en el estado **RelojSet** y que la máquina de la Región 4 se encuentre en el estado **BotonHoraRet**.

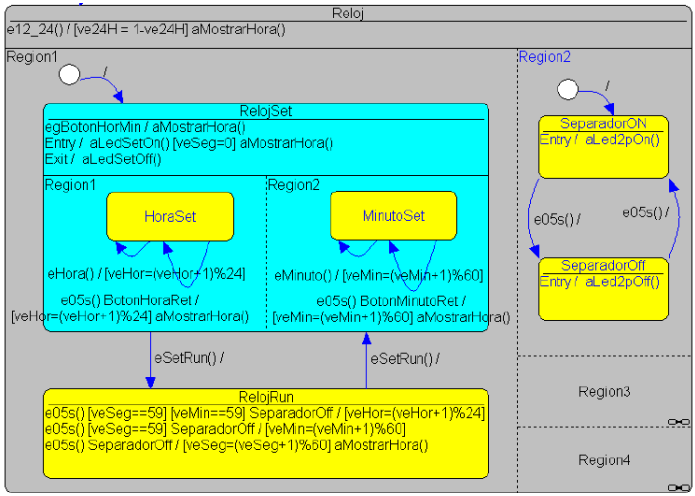


Figura 2-11. Statechart completamente terminado.

En la figura 2-11, la última línea en el estado RelojRun indica que si se produce el evento e05s y la máquina de estados de la Region2 se

encuentra en el estado SeparadorOff, lo cual sólo sucede una vez cada segundo, entonces la variable veSeg se incrementa en 1, módulo 60, y después se muestra la hora.

2.3.b Modelado y codificación de un contador

En este segundo ejemplo, tomado de las clases del Ing. Juan Manual Cruz, se quiere diseñar un contador ascendente descendente, se necesita que sea con capacidad de conteo variable y para ello debe tener la capacidad de poder definirse límites que puedan variar sin que esto signifique un rediseño del sistema. Al encender el sistema comienza una cuenta ascendente hasta un valor máximo y desde allí inicia una cuenta descendente hasta un valor mínimo. Al llegar al mínimo nuevamente empieza la cuenta ascendente.

Modelo

Paso1. Identificar los eventos y las Acciónes

No existen eventos externos, a excepción de:

Reset	Reinicia el sistema.
ePulso	Pulsos a contar.

Las Acciónes que podemos definir serán:

aCuenta++ (Incrementa el contador) viCuenta = viCuenta + 1
aCuenta-- (Decremento el contador) viCuenta = viCuenta - 1

A los efectos de parametrizar el sistema se hace necesario definir algunas constantes

cESTADOmAXIMO, cESTADOiNICIAL,
cCuentaInicial, cCuentaFinal

Paso 2. Identificar los estados

Se plantea un modelo con dos estados estables, uno cuando esta contando en forma ascentente y otro cuando cuenta en forma descendente.

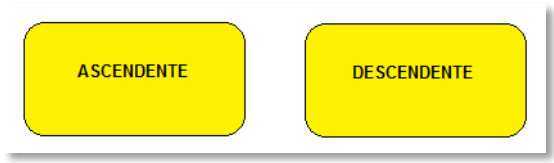


Figura 2-12: Estados estables del contador

Paso 3. Agrupar los estados por jerarquías

El problema presenta un solo nivel jerárquico que llamamos simplemente contador.

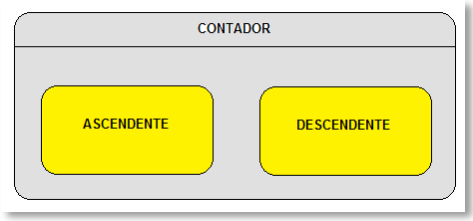


Figura 2-13: Los estados se agrupan en CONTADOR

Paso 4. Agrupar por concurrencia

En este ejemplo no existe concurrencia de tareas

Paso 5. Añadir las Acciónes y transiciones

La primera transición será la producida por el Reset, que lleva siempre al sistema contando en forma ascendente. Posteriormente con la aparición de cada pulso el sistema puede realizar una de las dos transiciones posibles, si la cCuentaFinal > viCuenta entonces permanece en el estado actual, si al llegar el pulso nos encontramos con que cCuentaFinal == viCuenta entonces cambiamos de estado y pasamos a realizar cuenta descendente.

Estando en estado descendente nuevamente el análisis es similar. Como se puede ver, la definición de constantes nos permite modelar un contador de cualquier valor con solo modificar los límites de las constantes cCuentaInicial y cCuentaFinal.

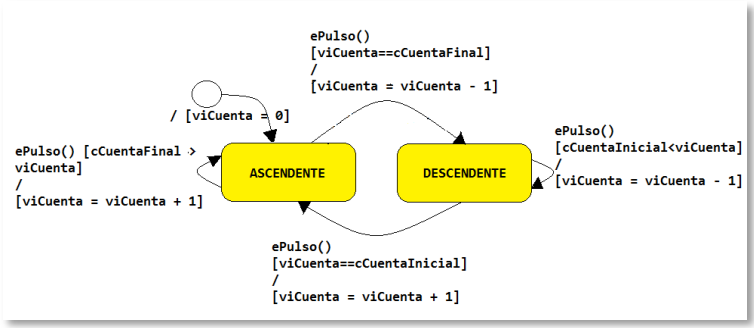


Figura 2-14: Transiciones entre estados

Paso 6. Añadir las sincronizaciones

No son necesarias

Paso 7. Simulación sobre el modelo

En este ejercicio agregamos el uso de una nueva herramienta del software. La validación se inicia llamando la herramienta Validator u oprimiendo la tecla [F8], para visualizar el proceso disponemos de dos formas: a) mediante ventanas que nos muestran el estado de los eventos, los estados y las acciones que se van ejecutando (Figura 2-15a), y/o b) mediande una animación gráfica de los estados (Figura 2-15b).

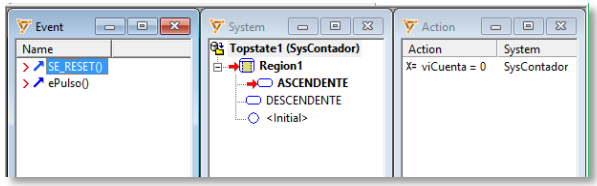


Figura 2-15(a): Ventanas del validator

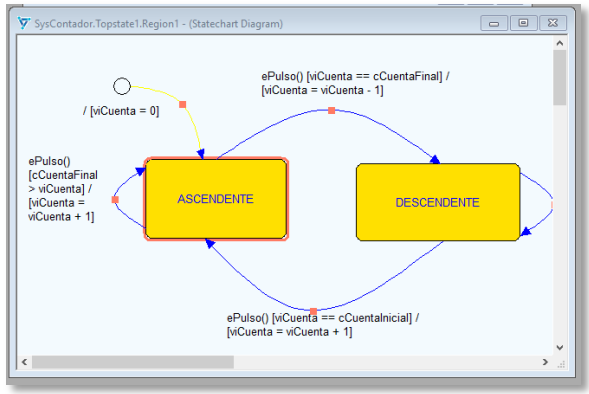
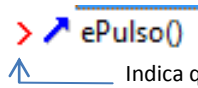


Figura 2-15(b): Ventana dinámica del sistema

Al hacer click con el mouse sobre un evento habilitado, en la ventana Event.



Indica que el evento esta habilitado

El sistema evoluciona y esa evolución es mostrada dinámicamente. Esto nos permitirá probar el modelo y determinar la exactitud del mismo.

En este caso lo que estamos viendo, en las figuras 2-15, es que se ha ejecutado un evento de reset, esto ha llevado al sistema al estado ASCENDENTE y dado el valor cero a la variable interna viCuenta.

Paso 8. Codificación

A continuación se muestra el código que responde al modelo, el mismo fuer realizado mediante el uso de sentencias SWICH – CASE.

```
/* ============================================================
Nombre: Contador Ascendente / Descendente
============================================================ */
/*Declaración de Prototipos*/
void  InicializarContador (void);
void  Contador (void);

/*Definiciones de los estados*/
```

```
#define ASCENDENTE          0
#define DESCENDENTE         1
#define ESTADOmAX           DESCENDENTE + 1
#define ESTADOiNICIAL       ASCENDENTE
/*Definiciones del límite de la cuenta inicial y final*/
#define CUENTAiNICIAL        0x00
#define CUENTAfINAL          0xFF

/*Declaración de variables*/
unsigned char  viEstado;  //Estado del Contador de 8 bits Asc/Desc
unsigned char  viCuenta;  //Contador de 8 bits Asc/Desc.

/*Programa*/
void InicializarContador (void)
{
        /*Inicializa las Variables y Salidas de Control*/
        viEstado = ESTADOiNICIAL;
        viCuenta = CUENTAiNICIAL;
        return;
}

void Contador (void)
{       /* Si el Estado está fuera de rango, reinicializa la ME y
        retorna*/
       if (viEstado >= ESTADOmAX) {
              InicializarContador();
              return;
       }

        /* En función del Estado y la Excitación Actualiza las
           Variables, las Salidas de Control y retorna*/
    switch (estado) {
         case ASCENDENTE:
              if (ePulso and viCuenta < CUENTAfINAL)
                  cuenta++;        /*incrementar el contador*/
                  if  (ePulso and viCuenta = CUENTAfINAL) {
                       viEstado = DESCENDENTE;
                       cuenta--;    /* decrementar el contador */
                  }
                  break;
         case DESCENDENTE:
                  if (ePulso and viCuenta > CUENTAiNICIAL)
                      cuenta--;
                      if (ePulso and viCuenta = CUENTAiNICIAL) {
                          viEstado = ASCENDENTE;
                          cuenta++;
                      }
                      break;
       }
}
```

2.3.c Modelado y codificación de un vehículo simple

En este tercer ejemplo, implementaremos un sistema de control de un pequeño vehículo. La unidad de control dispone de dos pulsadores (I1 y I2) como entradas del sistema y de dos salidas S0 y S1.

Se deben de cumplir las siguientes condiciones:

- En estado de reposo (I1=I2=0) el vehículo no se moverá.
- Si se pulsa el pulsador I1 el vehículo se moverá hacia adelante, continuando el movimiento al dejar de presionar dicho pulsador.
- Si se pulsan ambos pulsadores I1 y I2 a la vez el vehículo se moverá hacia atrás, continuando el movimiento al dejar de pulsarlos.
- Si se pulsa el pulsador I2 el vehículo se parará.

Las señales de salida en función del movimiento del robot deberán de ser las siguientes:

- Si el vehículo está parado S0=S1=0
- Si el vehículo se mueve hacia atrás S0=0 y S1=1
- Y si el vehículo se mueve hacia adelante S0=1 y S1=0

Resolución del ejemplo

Antes de comenzar con el VisualSTATE y para quienes han estudiado diseño de sistemas digitales secuenciales basados en los diagramas de transición de estados (DTE), veremos la similitud entre estos dos métodos de diseño.

Comenzando con el diseño de sistemas secuenciales basados en DTE, lo primero y más importante que tenemos que hacer es dibujar nuestro modelo en base al diagrama de transición de estados, lo podemos dibujar directamente en un papel ó ayudarnos de algún software de los muchos que hay, para la realización de esta tarea. Hay que tener en cuenta, que si nos equivocamos aquí, todo lo que

hagamos después no servirá de nada. Una cosa que siempre tenemos que comprobar es que sea un autómata determinista, para ello hay que comprobar que no queden posibles estados sin definir.

Como se ve en la figura 2-16 nuestro DTE tiene tres estados que hemos llamado “Para”, “Adelante” y “Atrás”, que definen los tres posibles estados de movimiento en los que se puede encontrar el vehículo.

Como vemos de cada estado salen 4 transiciones, contándose también las que salen y entran al mismo estado. Como tenemos dos entradas, nuestro autómata es determinista ($2^2 = 4$).

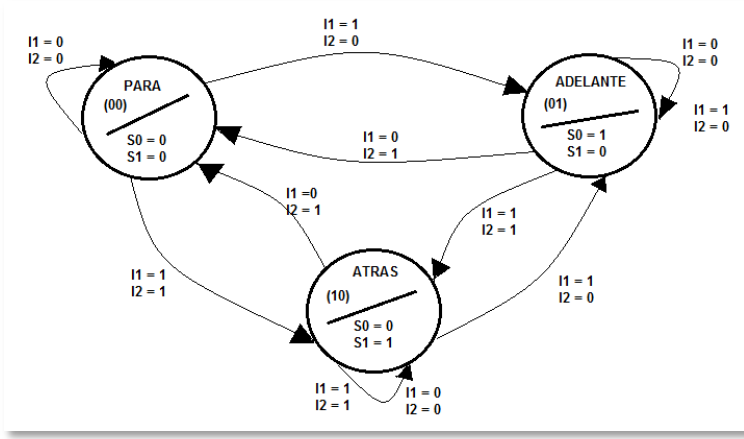


Figura 2-16: Diagrama de estados – Modelo de MOORE

Partimos del estado **Para** con entradas I1=I2=0 donde las salidas son S0=0 y S1=0, si pulsamos I1 (I1=1, I2=0), se producirá la transición al estado **Adelante** y la salida cambiará a S0=1 y S1=0 y el robot se moverá hacia adelante. Si ahora soltamos el pulsador (I1=0, I2=0), vemos que continuamos en el mismo estado y por tanto el vehículo continuará moviéndose en la misma dirección, que es como se había definido en las condiciones del ejemplo. Pues de esta forma hay que ir comprobando todos los estados que definamos y comprobando todas las posibles transiciones entre ellos.

Con este análisis y habiendo tomado la decisión de implementarlo con un microcontrolador PIC, definimos el hardware que se muestra en la figura 2-17, donde vemos que se han agregado tres led indicadores del estado en que se encuentra el sistema, además de los pulsadores, el puente H y el motor de CC.

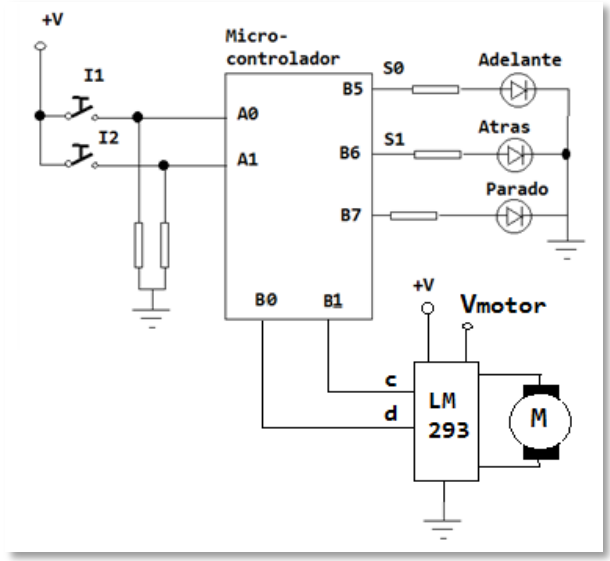


Figura 2-17: Modelo para simulación basica en Proteus®.

Modelado con VisualSTATE

En este ejercicio podremos ver la similitud entre el modelo realizado a través de las maquinas de estado para un diseño de lógica no programada, basado en biestables, y el modelado con VisualState.

Paso1. Identificar los eventos y las acciones

Existen los siguientes eventos externos:

Reset	Reinicia el sistema.
eAvanzar	Se ha pulsado I1.
eRetroceder	Se han pulsado simultáneamente I1 e I2.
eParar	Se ha pulsado I2.

Las acciones que podemos definir serán:

aAvanza:	El vehículo avanza.
aRetrocede:	El vehículo se mueve hacia atrás.
aPara:	El vehículo se para.

A los efectos de parametrizar el sistema se hace necesario definir algunas constantes

ESTADOmAXIMO, ESTADOiNICIAL,
PARA, ADELANTE, ATRAS

Paso 2. Identificar los estados

Nuestro sistema tendrá 3 estados estables PARADA: cuando el motor no funciona, ADELANTE: cuando el vehículo se mueve hacia adelante y ATRÁS: cuando el vehiculo se mueve hacia atrás.

Figura 2-18 Estados del sistema

Paso 3. Agrupar los estados por jerarquías

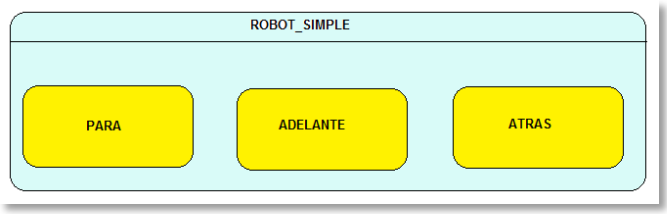


Figura 2-19: Agrupamiento de estados en una sola máquina simple

Paso 4. Agrupar por concurrencia

En este ejemplo no existe concurrencia de tareas.

Paso 5. Añadir las acciones y transiciones

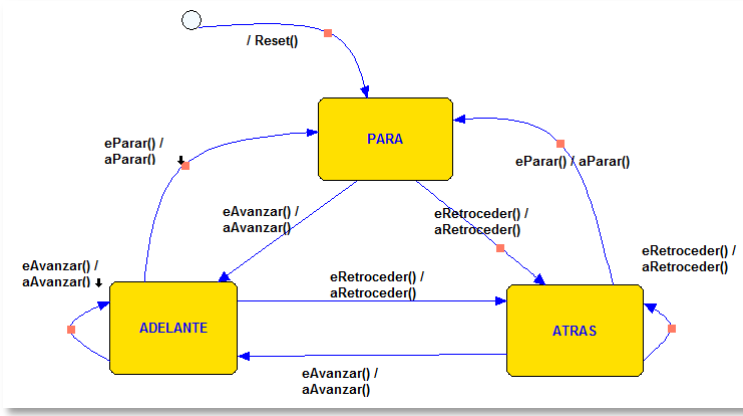


Figura 2-20: Transiciones entre estados

Podemos ver en la figura 2-20 las transiciones entre los estados, los eventos que las disparan y las acciones que ocurren por efecto de las mismas.

Despúes de un RESET el vehículo queda en el estado PARA y según el evento que ocurra –eAvanza o eRetrocede, podrá pasar al estado ADELANTE o ATRÁS respectivamente.

Seguramente el análisis no resulta todo lo completo que podría ser pero lo dejaremos asi a los efectos de no complicar el ejemplo y dejamos al lector la introducción de mejoras al modelo.

Paso 6. Añadir las sincronizaciones

No son necesarias.

Paso 7. Simulación

Como ya vimos en ejemplos anteriores, llamando la herramienta *Validator* u oprimiendo la tecla [F8], podemos ver como evoluciona el sistema a medida que simulamos la aparición de eventos.

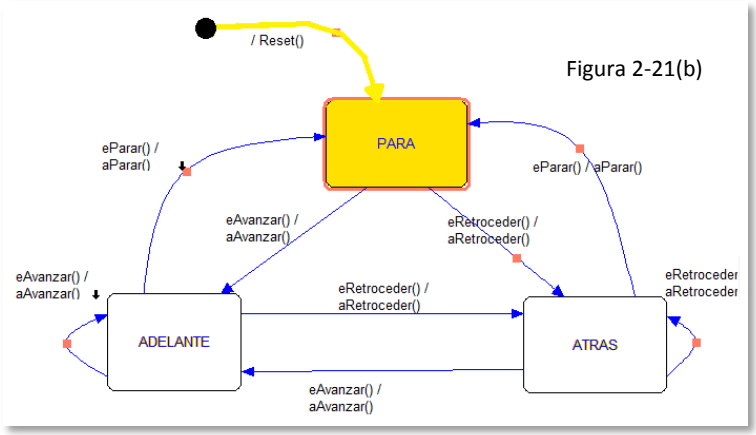


Figura 2-21(b)

Figura 2-21 (b): Simulación, Modelo dinámico

Paso 8. Codificación

El código resultante en lenguaje "C" para el compilador de CCS será el siguiente:

```
/* ================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: Vehículo simple
Autor: Sergio Gallina – Matias Ferraro
Descripción: Autómata de Moore utilizando máquina de estado - 2015
================================================================*/
#include <16F877A.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#use delay(clock=20000000)
/*Prototipos de funciones*/
void aAvanza(void);
void aRetrocede(void);
void aPara(void);
//Declaración de variables de la Maquina de estados.
int8 cPARA=0, cADELANTE=1, cATRAS=2;
int8 estado=0;
int1 eRetrocede, eAvanza, ePara;
//Funcion principal.
void main()
```

```
{
      //Inicialización de registros
      set_tris_b (0x00);   //Puerto B como salida
      while(true)
      {
        eAvanza = input(PIN_A0);  // Leo estado del interruptor I1
        ePara = input(PIN_A1);    // Leo estado del interruptor I0
        eRetrocede = (eAvanza && ePara);
        output_low(PIN_B7);//Apago los leds indicadores del estado
        output_low(PIN_B6);
        output_low(PIN_B5);

        if(estado==cPARA)
        {        // Si entramos en el estado cPARA
            output_high(PIN_B7);  // Encendemos el LED de parada
            output_low(PIN_B5); // Apagamos el LED de adelante
            output_low(PIN_B6); // Apagamos el LED de parado
            if (eAvanza == 1)
            {    // Si se produce un evento
                 aAvanza();    // ejecutamos la Acción aAvanza
                 estado = cADELANTE;  // el próximo estado
            }
            if (eRetrocede == 1)
            {    // Si pulso simult. I1 e I0
                 aRetrocede();
                 estado = cATRAS;
             }
        }
        if (estado == cADELANTE)
        {
             output_high(PIN_B5); // Encendemos el LED de adelante
             output_low(PIN_B6); // Apagamos el LED de atras
             output_low(PIN_B7); // Apagamos el LED de parado
             if (eAvanza == 1)
             {   // Se pulso I0
                 aAvanza();
                 estado = cADELANTE;
             }
               if (ePara == 1)
               {    // Se pulso I1
                    aPara();
                    estado = cPARA;
               }
               if (eRetrocede == 1)
               {
                    aRetrocede();
                    estado = cATRAS;
                }
          }
         if (estado == cATRAS)
```

```
          {
               output_high(PIN_B6); // Encendemos el LED de atras
               output_low(PIN_B5); // Apagamos el LED de adelante
               output_low(PIN_B7); // Apagamos el LED de parado
               if (eRetrocede == 1)
               {
                     aRetrocede();
                     estado = cATRAS;
               }
               if (ePara == 1)
               {
                     aPara();
                     estado = cPARA;
               }
               if (eAvanza == 1)
               {
                      aAvanza();
                      estado = cADELANTE;
               }
            }
        }
}

void aAvanza(void) {      // El motor hace avanzar el vehículo
         output_high(PIN_B0);
         output_low(PIN_B1);
}

void aRetrocede(void)  {  // El motor hace retroceder el vehículo
         output_low(PIN_B0);
         output_high(PIN_B1);
}

void aPara(void)  {    // El motor esta detenido
         output_low(PIN_B0);
         output_low(PIN_B1);
 }
```

NOTA: En este código no se han utilizado las constantes definidas en el *paso 1*, como ESTADOmAXIMO y ESTADOiNICIAL, se recomienda al lector la siguiente ejercitación: modificar el modelo y el código de forma tal que, el robot debe detenerse el sistema, por una falla, asume como estado un valor distinto de 0, 1 o 2 y solo se podrá salir de este estado con un Reset del sistema.

CAPITULO 3: MICROCONTROLADORES DE 32 BITS

3.1 REPASANDO CONCEPTOS

a. *Modos de direccionamiento*: es la forma en que el microprocesador accede a los datos que utiliza. Ortogonalidad[1] es la relación de disponibilidad de modos de direccionamiento y registros para cada instrucción.

Entre los modos de direccionamiento más comunes:

- Por Registro: El dato se encuentra en un registro.
- Inmediato: El dato se encuentra en la instrucción misma. (Figura 3-1)

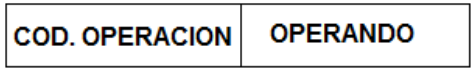


Figura 3-1: Modo de direccionamiento inmediato.

[1] Un set de instrucciones es ortogonal cuando las instrucciones pueden usarse con cualquier registro y modo de direccionamiento. Es decir, teniendo el conjunto de instrucciones que operan sobre registros, el conjunto de registros de operación y el conjunto de modos de direccionamiento, se dice que el ISA (Set de Instrucciones), posee ortogonalidad cuando puedes combinar los tres conjuntos como quieras. Si en cambio tienes instrucciones que solo pueden usarse con ciertos registros y/o ciertos modos de direccionamiento, se dice que el ISA posee poca ortogonalidad. La ortogonalidad es una característica deseada, ya que simplifica el diseño de compiladores optimizadores y la programación en lenguaje assembly.

- Directo: El dato se encuentra en la memoria y la instrucción contiene la dirección. (Figura 3-2)

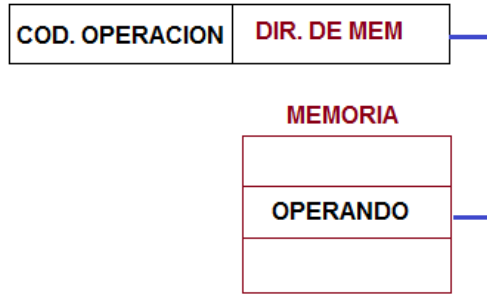


Figura 3-2: Modo de direccionamiento directo.

- Indirecto por registro: El dato se encuentra en la memoria cuya dirección está contenida en un registro que oficia de puntero, la instrucción hace referencia al registro (puntero). (Figura 3-3)

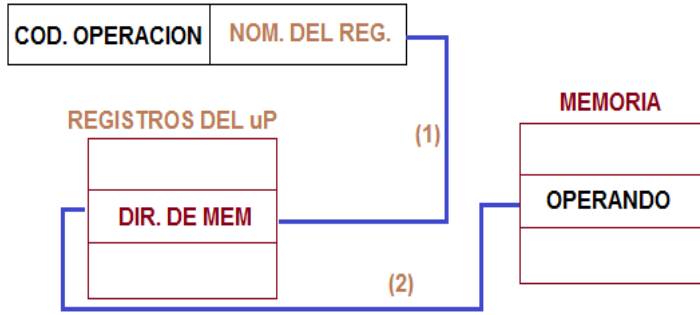


Figura 3-3: Modo de direccionamiento indirecto por registro.

- Indirecto por memoria: El dato está en la memoria y se utiliza otra dirección de memoria como puntero. (Figura 3-4)

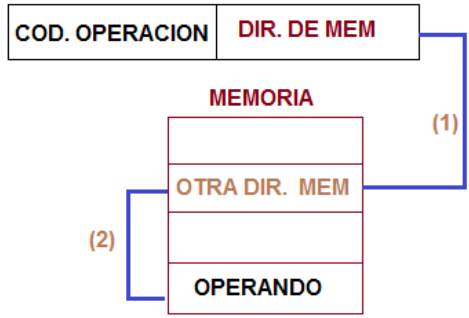


Figura 3-4: Modo de direccionamiento indirecto por memoria.

- Indexado: Similar al indirecto, se agrega un desplazamiento respecto del valor indicado en el índice.
- Relativo: Similar al indexado pero utiliza como registro al contador de programa.

b. *Byte – Half-Words – Words – Double-Words, Alineación, Endianness*: Es común llamar palabra o "*words*" cuando corresponde al contexto natural del microprocesador, así en un micro de 32 bit la palabra será de 32 bits, la media palabra "*Half-Words"* será de 16 bits y la doble "*Double-Words"* de 64 bits

El término inglés ***endianness*** designa el formato en el que se almacenan los datos de más de un byte en un computador.

No se debe confundir trivialmente el orden de escritura textual en este artículo con el orden de escritura en memoria, por ello establecemos que lo que escribimos primero lleva índices de memoria más bajos, y lo que escribimos a continuación lleva índices más elevados, que lo que lleva índices bajos es previo en memoria, y así sucesivamente, siguiendo la ordenación natural de menor a mayor, por ejemplo la secuencia {0,1,2} indicaría, -algo más allá de la intuición- que 0 es previo y contiguo en el espacio de memoria a 1, etc.

Usando este criterio el sistema ***big-endian*** adoptado por Motorola entre otros, consiste en representar los bytes en el orden "natural": así el valor hexadecimal 0x0A0B0C0D se codificaría en

memoria en la secuencia {0A, 0B, 0C, 0D}. En el sistema ***little-endian*** adoptado por Intel, entre otros, el mismo valor se codificaría como {0D, 0B, 0C, 0A}, de manera que de este modo se hace más intuitivo el acceso a datos, porque se efectúa fácilmente de manera incremental de menos relevante a más relevante. (Figura 3-5)

Algunas arquitecturas de microprocesador pueden trabajar con ambos formatos (ARM, DEC Alpha, PA-RISC, Arquitectura MIPS), y a veces son referidas como sistemas ***middle-endian***.

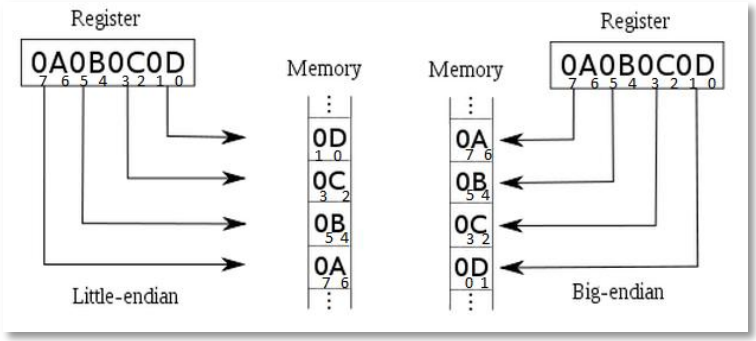


Figura 3-5: Sistema littlen-endian y big-endian

c. *Segmentación o Pipelining*: Un procesador realiza al menos dos tareas: "leer y decodificar una instrucción" y "ejecutar la instrucción", si el hardware se diseña para que estas tareas se ejecuten en secciones separadas, las acciones se podrán realizar en simultáneo, cuando se ejecuta una instrucción se puede leer y decodificar la siguiente, esto duplicara el número de instrucciones por segundo que se pueden ejecutar.

Si separamos las acciones de leer y decodificar en dos secciones separadas entonces se podrán ejecutar las instrucciones tal como se muestra en la figura 3-6.

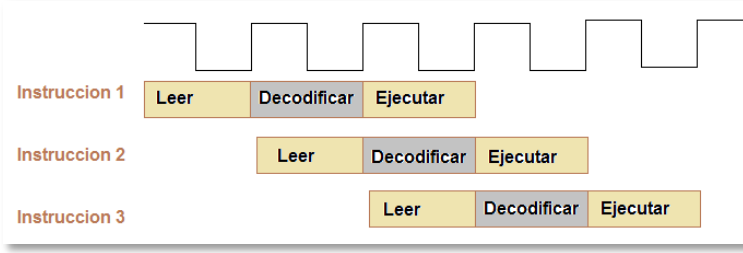


Figura 3-6: Ejecución de instrucciones

• ¿Qué facilita el pipelining? Para la implementación del pipeline es conveniente que:

– Todas las instrucciones sean del mismo largo.

– Pocos formatos de instrucciones.

– Los operandos con memoria solo aparecen en las instrucciones de carga y almacenamiento.

• ¿Qué lo hace difícil? Hay situaciones en la segmentación cuando la siguiente instrucción no se puede ejecutar en el próximo ciclo de reloj, esto se puede dividir en tres tipos de riesgos:

– Riesgos de dependencia de datos.

– Riesgos estructurales.

– Riesgos de control.

Es muy común encontrar pipeline de 5 etapas: LEER, DECODIFICAR, EJECUTAR, LEER MEMORIA, GUARDAR RESULTADOS, en estos casos suelen aparecer problemas de **dependencia de datos,** una instrucción utiliza el dato almacenado en un registro por la instrucción anterior, en estos casos la etapa EJECUTAR debe demorarse hasta que el dato sea guardado por la instrucción precedente.

Ejemplo:

NOTA: las instrucciones “add” o “sub” no escriben el resultado hasta el estado Grabar Resultados (G.Res.).

Ciclo	1	2	3	4	5	6	7
Add **$s0**,$t0,$t1	Leer	Deco	**Ejecut**	Mem	G.Res		
Sub **$t2**,$s0,$t1		Leer	**Demora**	**Demora**	**Demora**	Deco	Ejec

Necesita esperar hasta G.Res.
ya que $s0 tendrá su resultado
después de G.Res.

Algunas veces se pueden resolver (o reducir) atascamientos para los riesgos de datos utilizando la técnica adelantamiento (forwarding)

Forwarding: Como el resultado de la operación "add" ($s0) se obtiene después del estado "Ejecución", se puede adelantar el resultado ($s0) al próximo "Ejecución" para realizar la operación "sub".

Ciclo	1	2	3	4	5	6
Add **$s0**,$t0,$t1	Leer	Decod.	**Ejecut**	Mem	G.Res	
Sub **$t2**,$s0,$t1		Leer	Decod.	**Ejecut**	Mem	G.Res

Adelanta $s0 a la próxima instrucción

Además de este problema podemos encontrarnos con **riesgos estructurales**, cuando dos instrucciones intentan utilizar un mismo recurso (una instrucción que use la ALU para sumar y otra que use la ALU para calcular la dirección de destino en una instrucción de salto, colisionaran).

Ciclo	1	2	3	4	5	6	7
Instrucción 1	Leer	Deco	Ejecut	**Mem**	G.Res		
Instrucción 2		Leer	Decod	Ejecut	Mem	G.Res	
Instrucción 3			Leer	Deco	Ejecut	Mem	G.Res
Instrucción 4				**Leer**	Decod	Ejecut	Mem

Riesgo estructural

Ciclo	1	2	3	4	5	6	7
Instrucción 1	Leer	Deco	Ejecu	**Mem**	G.Res		
Instrucción 2		Leer	Deco	Ejecut	Mem	G.Res	
Instrucción 3			Leer	Decod	Ejecut	Mem	G.Res
Instrucción 4				**Demora**	**Demora**	**Demora**	Leer

Un tercer problema son los riesgos de control relacionados a los cambios de dirección en los saltos condicionados.

Ciclo	1	2	3	4	5	6
Add $4,$5,$6	Leer	Deco	Ejecut	Mem	G.Res	
beq $1,$2,40		Leer	**Decod**	Ejecut	Mem	G.Res
lw $3,300($0)			**Atasco**	Leer	Deco	Ejecut

Como no se sabe si el salto se realizara, se retarda lw hasta el estado IF (ciclo 4). Si esto se realiza, se presenta un riesgo estructural en el ciclo 4 y 5

El procesador utiliza lo que se denomina predicción, se predice que el salto no se realizará y se lee la instrucción siguiente, si el salto se produce, entonces tendremos una penalidad.

Ciclo	1	2	3	4	5	6
Add $4,$5,$6	Leer	Deco	Ejecut	Mem	G.Res	
beq $1,$2,40		Leer	Decod	Ejecut	Mem	G.Res
lw $3,300($0)			Leer	Decod	Ejecut	Mem

Si no salta

Ciclo	1	2	3	4	5	6
Add $4,$5,$6	Leer	Deco	Ejecut	Mem	G.Res	
beq $1,$2,40		Leer	**Decod**	Ejecut	Mem	G.Res
			Atasco	**Atasco**	**Atasco**	**Atasco**
40					Leer	Decod

Si salta

La figura 3-7 muestra una forma constructiva de un pipeline de 5 etapas para un procesador RISC.

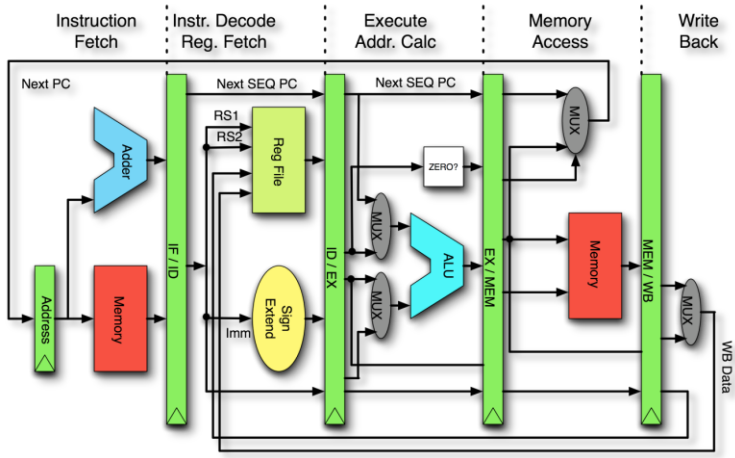


Figura 3-7: Pipeline de 5 etapas.

El pipeline es uno de los principios fundamentales de los procesadores en general. Cuando se manipula datos, tiene que pasar por una serie de pasos o etapas. La siguiente explicación debe considerarse como un punto de partida que ilustra los principios básicos.

Lo primero que tiene que ocurrir es la búsqueda de la instrucción. Con el fin de hacer esto, se realiza una llamada al contador de programa (PC). En la etapa Fetch (Instruction Fetch - IF), la dirección del PC se carga en la caché de instrucciones en la memoria. Una vez que la dirección está en la caché de instrucciones, es accesible por la Unidad de Control (CU) a través del registro.

Cuando la Unidad de Control (CU) lee el registro de control, se entra en lo que se llama la etapa de decodificación de instrucciones (Istruction Decode - ID). Mientras que la CU está decodificando la instrucción, envía diferentes señales. Estas señales pasan información como ser si se va a utilizar la unidad lógica aritmética (ALU), y muchas otras cosas. Una vez que la CU ha preparado todo en el procesador, los datos se envían, con todos los dispositivos

necesarios activados. Esta etapa se llama la etapa de ejecución (Execute - EX), y es el lugar donde se realiza todo en proceso. Dependiendo del tipo de instrucción que se está ejecutado, podría ser necesario para acceder a la memoria en la etapa de acceso a memoria (Memory Access - MEM).

Una vez finalizada la tarea, el resultado debe ser almacenada en otro registro que se utilizará para el siguiente proceso (si se necesita el resultado). Esta última fase se llama la etapa Guardar resultado (Write Back - WB), y es lo que permite que el resultado quede precargado para la siguiente instrucción. Estas 5 etapas se ejecutan una tras otra, pero eso no significa que el procesador sólo pueda hacer solo una a la vez. El flujo real de la pipeline puede ser concurrente, con cada etapa puede empezar a trabajar inmediatamente con la siguiente instrucción, en el próximo ciclo de reloj.

Aunque simplificada, este ejemplo da una breve introducción a las etapas que son comunes en la mayoría de los procesadores. Sin embargo, muchos procesadores añaden rutas adicionales a fin de acelerar la velocidad de ejecución.

¿Qué se necesita para realizar esta separación de etapas?

➔ REGISTROS, para poder almacenar la información.

d. *RISC / CISC.* RISC fue desarrollado en la Universidad de Berkeley en la década del 70, con la idea de incluir en el procesador solo las instrucciones más utilizadas, reduciendo espacio en el CORE y reduciendo la cantidad de accesos a memoria. El paradigma CISC es utilizar la menor cantidad de instrucciones posibles para resolver un problema.

Podemos descomponer el tiempo de ejecución (TE) de un programa en:

$$TE_{jecución} = Instrucciones_{tarea} * Ciclos_{instrucción} * t_{ciclo}$$

Entonces en un RISC se pretende reducir la cantidad de cliclos por instrucción ($Ciclos_{instrucción}$) y en un CISC la cantidad de instrucciones para una tarea ($Instrucciones_{tarea}$).

El paradigma RISC suele aventajar al CISC, particularmente cuando viene acompañada de una alta ortogonalidad. Además al ser el core más simple suele ser más sencillo llevarlos a altas velocidades incrementando la frecuencia de reloj y por ende disminuyendo el tiempo de ciclo (t_{ciclo}).

e. *Manejo de la memoria*. Nos referiremos a la tarea de ubicación de los procesos en la memoria de programas. En un sistema dedicado simple todas las tareas se encuentran ubicadas en la memoria y son activadas o desactivadas por el sistema operativo.

En un sistema con programas en disco rígido, estos se cargan desde el medio no volátil, en tiempo de ejecución, por lo que resulta imposible saber, al momento de compilar el programa, en qué posición de memoria va a ejecutarse, y será necesario ejecutar alguna estrategia de reubicación, dos de esas técnicas son el paginado y la segmentación.

Paginación: Es una forma de superposición automática administrada por el Sistema Operativo. El espacio de direcciones virtuales se divide en bloques de igual tamaño llamadas *páginas*, estas páginas tienen normalmente un tamaño que es múltiplo de 2^n. y la memoria real se divide igualmente y a cada partición lo llamamos *marco de página*. (Página 3-8)

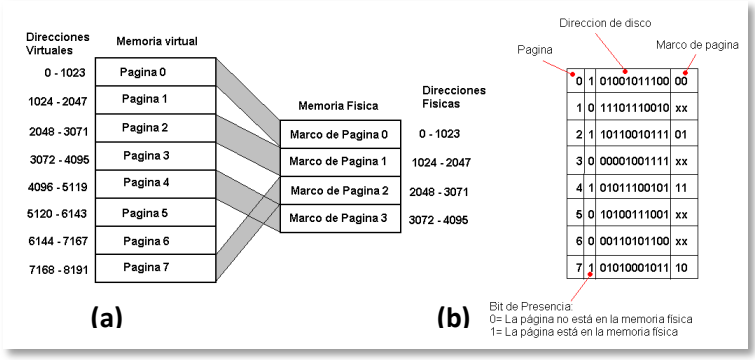


Figura 3-8: (a) Esquema de asignación entre memoria - (b) Tabla de páginas

La siguiente secuencia de eventos ocurre cuando se hace referencia a una posición de memoria virtual:

1) Se identifica una de los marcos de página, el contenido del marco se escribe en la memoria virtual si hubiese cambios.
2) Se ubica en la memoria virtual la página a la que se desea acceder y esta se escribe en el marco de página identificado en el paso 1).
3) Se actualiza la tabla de páginas.

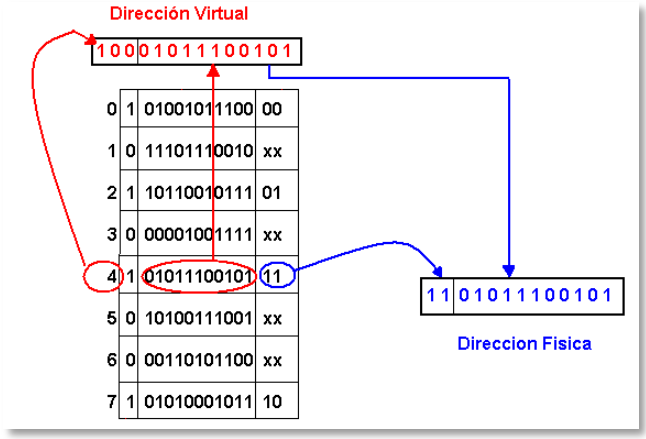


Figura 3-9: Memoria virtual

La memoria virtual de la figura 3-9 tiene 2^{13} = 8191 posiciones virtuales por lo que se necesitan 13 bits para manejar las direcciones, de estos 13 bits, 3 son utilizados para la selección de la página y 10 para el desplazamiento (posición dentro del bloque), con el objeto de llevar el control de las paginas se debe administrar una *tabla de páginas*. En esta tabla se indica para cada bloque su presencia o no en la memoria principal y su número de marco de página.

Para traducir una dirección virtual en una dirección física, se toman los bits del marco de página y se le agrega el desplazamiento.

Segmentación: Como se ha visto la memoria virtual es unidimensional, la segmentación divide el espacio de direcciones en segmentos que pueden ser de tamaño arbitrario, cada segmento tiene su propio espacio unidimensional de direcciones. Esto permite crear tablas, pilas u otra estructura de datos.

La segmentación permite esquemas de protección pudiéndose definir zonas de “lectura solamente” o de “ejecución solamente”.

Cuando se utiliza segmentación en memoria virtual, el espacio (tamaño) se define cuando se lo necesita, esto permite que sea todo lo grande que se requiera.

La figura 3-10 muestra una memoria segmentada.

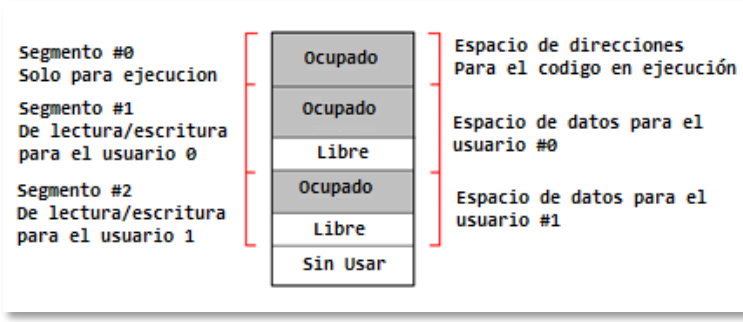


Figura 3-10: Memoria segmentada

Con el objeto de determinar una dirección en una memoria segmentada, se debe especificar una dirección de segmento y una dirección dentro del mismo, el sistema operativo traduce esta dirección a una dirección física.

f. *Programación*: Los micropocesadores y microcontroladores se pueden programar en lenguaje C/C++ o lenguaje ensamblador. *¿Es mejor programar un microcontrolador en C o en ensamblador?*

Compartiendo la respuesta a esta pregunta que nos da Raul Alvarez en “Ensamblador Versus C en micronctroladores” [Alvarez - 2014], decimos que la única respuesta correcta es: Depende.

Para una respuesta más detallada démosle una mirada a las ventajas y desventajas de cada lenguaje:

a) Ventajas del lenguaje ensamblador

- Controla con precisión la operación del microcontrolador y permite al programador precisión en las operaciones.

- Permite escribir un código más sucinto, y por lo tanto más veloz. En aplicaciones profesionales se usa sobre todo para escribir drivers para interfaces de periféricos o para escribir rutinas altamente optimizadas que requieren velocidad, tamaño reducido y/o precisión.
- Es valioso como herramienta educacional, ya que para escribir ensamblador uno debe conocer a detalle la arquitectura interna del microcontrolador, el set de instrucciones, los registros y su funcionamiento interno.

b) Desventajas del lenguaje ensamblador

- No es portable, debido a que depende de la estructura interna del microcontrolador, el código de un microcontrolador no puede correr en otro de diferente arquitectura.
- No posee estructura ni control de tipos, por lo cual, el programador debe cuidar por sí mismo de proveer una estructura adecuada a su programa y controlar los tipos de datos.
- El mantenimiento del código (revisión, modificación, ampliación) es más complicado.

c) Ventajas del lenguaje C

- Es portable. Generalmente un programa escrito para un tipo de microcontrolador puede correr con mínimas modificaciones en otro microcontrolador de diferente arquitectura.
- C proporciona estructura, abstracción y control de tipos de datos (aunque no tan estrictamente como otros lenguajes de alto nivel).
- Permite también cierto acceso de bajo nivel, similar al ensamblador, combinando en general ciertas características de bajo nivel del ensamblador y otras ventajas ofrecidas por los lenguajes de alto nivel.
- Es más rápido y eficiente que otros lenguajes de alto nivel usados también para programar microcontroladores (C++, Basic, Java, Python, etc.) y su uso está altamente difundido en aplicaciones profesionales. Con compiladores modernos puede

llegar a ser tan rápido como el ensamblador dependiendo de la habilidad y los recursos del programador.

- Los programas en C son más fáciles de mantener (revisar, modificar, ampliar).
- Existen muchísimas librerías libremente disponibles para el uso de cualquier programador, lo cual facilita el desarrollo de una aplicación.
- En aplicaciones profesionales de sistemas embebidos el 90 a 95% (quizás más) de todo el código está escrito mayormente en lenguaje C y en algunos casos en C++. Casi cualquier dispositivo embebido moderno (lavadora, teléfono celular, cámara fotográfica/video, reproductor MP3, router, microondas, sistemas de control para automóviles o aviones, etc.) con seguridad tiene su código escrito en lenguaje C/C++.

d) Desventajas del lenguaje C

- Es menos veloz que el ensamblador.
- El código escrito en C ocupa más memoria que el escrito en ensamblador, dada una misma aplicación. Sin embargo los compiladores modernos poseen algoritmos de optimización que logran reducir tremendamente esta diferencia.

Finalmente, la herramienta que funcione mejor para una persona en particular y sobre todo para la solución que se desarrolla, ésa es la más indicada, ya sea ésta ensamblador, C, Basic, Java o cualquier otro lenguaje de alto nivel. Más importante aún es la capacidad, preparación y experiencia del programador y su habilidad para abordar la solución de un problema de manera elegante, eficiente y efectiva.

En lo sucesivo, para los ejemplos incluidos en este texto se ha escrito código en lenguaje C.

3.2 ARM (Advanced RISC Machine)

A principios de los 80 la firma ACORN COMPUTERS, basados en el desarrollo RISC de la Universidad de Berkeley desarrollo el Acorn

Risc Machine (ARM1, 1983) este micro no fue más que una curiosidad de laboratorio y permitió futuros desarrollo.

Unos años después APPLE COMPUTERS, VLSI y ACORN forman la empresa ADVANCED RISC MACHINE, hoy ARM, La primera PDA de APPLE denominada NEWTON lleva el procesador ARM6 con arquitectura ARMv3, ya de 32 bits.

Una cosa que hay que entender de ARM es que es una empresa que no fabrica ni un solo procesador, ni los manda a fabricar con su marca. Es lo que se conoce como "fabless". Simplemente se limita a diseñar. No vende ningún producto final físico bajo la marca "ARM".

Lo que hace ARM es diseñar unas IP (arquitectura del set de instrucciones, microprocesador, procesador gráfico, interconexiones, cachés…) y los va licenciando a las empresas que quieran. Esas empresas toman esos diseños y los fabrican bajo sus propias marcas como es el caso de NEC-Renesas o Samsung.

En las siguientes imágenes vemos algunos Chips con licencia ARM.

Figura 3-11: Microprocesadores ARM

ARM a grandes rasgos trabaja con tres tipos de licencias: procesador, POP y arquitectura.

- **La licencia de procesador** es la que permite a un licenciatario utilizar un microprocesador o una GPU que ARM ha diseñado. En

esencia no puedes cambiar ese diseño pero sí tienes margen para implementarlo como quieras. En el caso de Samsung, que utiliza este tipo de licencia, puede usar los cortex a7 y cortex a15. Por parte de ARM recibe unas pequeñas directrices pero es Samsung la encargada de acabar el diseño para encontrar la relación óptima entre rendimiento, eficiencia energética y demás para luego pasarlo a fábrica.

- **El POP** (processor optimization pack) es un nivel de licencia superior a lo comentado en el punto anterior. ARM no solo licencia el diseño de un procesador o de una GPU, si no que ARM hace el trabajo de optimizar ese diseño en temas de rendimiento y eficiencia de cara a un proceso de fabricación concreto, con un tipo de transistores concreto, en una fábrica concreta.

- Y como tercera opción está **la arquitectura**. ARM licencia una de sus arquitecturas como puede ser ARMv7 o ARMv8 (esta última es la de 64 bits) y eres libre para implementarla como quieras. ARM simplemente se encarga de correr ciertas pruebas para comprobar que tu propio diseño es compatible con su ISA, pero es responsabilidad tuya hacerte tus propios diseños, probarlos, testearlos, hacer el POP y buscar un modo de fabricarlos.

3.2.a Arquitectura ARM a nivel de sistema

Ésta arquitectura está orientada a ejecutar un sistema operativo y diversos programas del usuario, por lo que cuenta con los medios necesarios para ello.

Algunas características a destacar son:

Excepciones: Es un evento que interrumpe el flujo normal del programa, una interrupción es una excepción. Las excepciones comúnmente soportadas son:

- Reset.
- Undefined Instruction (instrucción no definida).
- Supervisor Call (anteriormente llamada Software Interrupt).

- Prefetch Abort (Problema en acceso a instrucción en memoria).
- Data Abort (Problema en acceso a datos en memoria).
- IRQ (Interrupciones externas).
- FIQ (Rápida Interrupción externa).

Modos de operación: Los modos de operación son:

- User, modo normal de trabajo de un programa del usuario.
- FIQ, atención de una interrupción rápida.
- IRQ, atención de una interrupción.
- Supervisor, modo protegido para el Sistema Operativo, se ingresa a él luego de un *Reset* o de una instrucción *Supervisor Call.*
- Abort, se ingresa cuando se detecta un problema de acceso a memoria.
- Undefined Instruction, se ingresa cuando no se reconoce el código de operación de una instrucción.
- System, modo especial para permitir reingreso en rutinas de manejo de interrupciones.

Registros (Figura 3-12.a): ARM dispone de 16 registros (R0 a R15), de 32 bits. Entre ellos se encuentran el contador de programa (Program Counter) el puntero de Stack (Stack Pointer).

El Registro R14 es llamado Link Register y es el que preserva la dirección de retorno ante un salto *Branch With Link* o una *excepción*.

Program Status Register (Figura 3-12.b): el estado del procesador puede observarse en el CPSR (Current Program Status Register) este registro es automáticamente preservado ante una excepción, el valor anterior e accesible en el SPSR (Saved Program Status Register), existe un SPSR para cada modo de excepción.

En los bits de estos registros de estado, están los flags N, Z, V y C (Figura 3-13) que se actualizan después de operaciones aritméticas y lógicas, los flags I y F destinados a inhibir interrupciones y los flags indicadores del estado de operación.

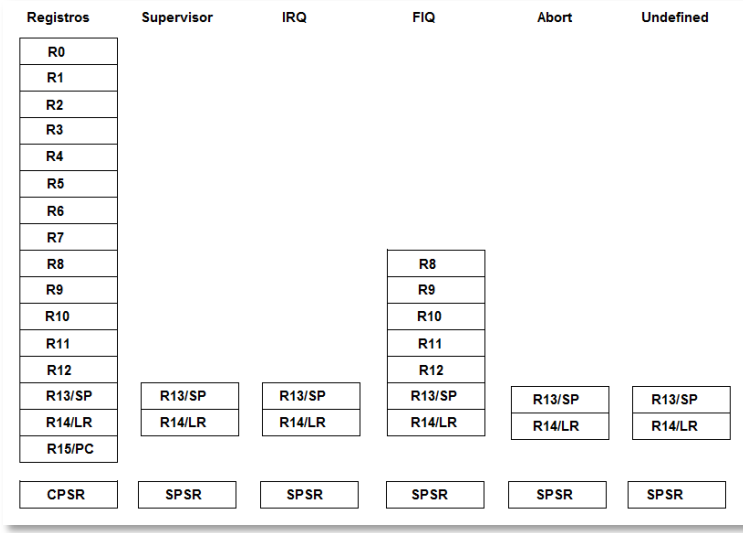


Figura 3-12(a): Registros en la mayoría de las arquitecturas ARM
Fuente: [Caprile -2012]

Figura 3-12(b): Program Status Register
Fuente: [Caprile -2012]

El acceso a los xPSR se realiza mediante instrucciones que mueven el contenido desde estos y a los registros generales (las arquitecturas ARMv6-M y ARMv7-M no emplean este esquema).

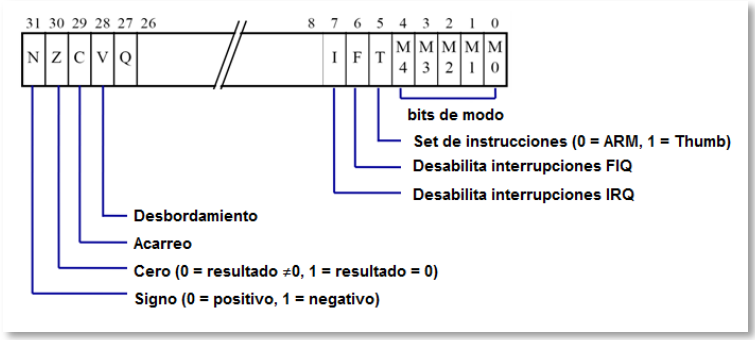


Figura 3-13: Flags del registro de estado (SPSR)

3.2.b Arquitectura a nivel de aplicación

Relación entre aplicación y sistema: una aplicación es un programa que se ejecuta en modo usuario (USR) y eventualmente ejecuta peticiones al sistema operativo (SO), el manejo del sistema en si corre por cuenta del SO y el modo usuario es una versión restringida. Si no se utiliza un SO el sistema funciona siempre en modo supervisor.

La llamada a una función API que provee el SO se realiza mediante una instrucción SVC (SuperVisor Call) que produce el ingreso a la excepción, esto coloca efectivamente al procesador en el modo supervisor.

Para reducir el riesgo de errores una aplicación accede al APSR (Application Program Status Register). Este registro no es más que el CPSR pero con acceso restringido a algunos bits.

3.2.c Set de instrucciones – arquitectura

La arquitectura ARM incorporó algunas características del diseño RISC creado por la Universidad de Berkeley, aunque no todas. Las que se mantuvieron son:

Arquitectura de carga y almacenamiento (load-store). Las instrucciones que acceden a memoria están separadas de las

instrucciones que procesan los datos, ya que en este último caso los datos necesariamente están en registros.

Instrucciones de longitud fija de 32 bits. Campos de instrucciones uniforme y de longitud fija, para simplificar la decodificación de las instrucciones.

Formatos de instrucción de 3 direcciones. Consta de "f" bits para el código de operación, "n" bits para especificar la dirección del 1er operando, "n" bits para especificar la dirección del 2do operando y "n" bits para especificar la dirección del resultado (el destino). El formato de esta instrucción en assembler, para la instrucción de sumar dos números para producir un resultado, es:

ADD d, s1, s2 ; d := s1 + s2

Características que posteriormente agregó ARM:

- Todas las instrucciones se ejecutan en un ciclo de reloj.
- Modos de direccionamiento simples, el procesamiento de datos solo opera con contenidos de registros, no directamente en memoria.
- Control sobre la unidad aritmética lógica (ALU, Arithmetic Logic Unit) y el "shifter", en cada instrucción de procesamiento de datos se maximiza el uso de la ALU y del "shifter".
- Modos de direccionamiento con incremento y decremento automático de punteros, para optimizar los lazos de los programas.
- Carga y almacenamiento de múltiples instrucciones, para maximizar el rendimiento de los datos.
- Ejecución condicional de todas las instrucciones, para maximizar el rendimiento de la ejecución.
- Set de instrucciones ortogonal, regular o simétrico. En este tipo de set no hay restricciones en los registros usados en las instrucciones, son todos registros de propósitos generales, con muy pocas excepciones (por ejemplo el contador de programa - PC), por lo que a los programadores de Assembler les resulta más fácil aprender un set con estas características, y también, a los compiladores les resulta más fácil manejarlo. Mientras

que la implementación del hardware será generalmente más eficiente.

- Técnica de "pipeline". Esta técnica consiste en comenzar la próxima instrucción antes de que la actual haya finalizado, cuyo objetivo es economizar tiempo.
- Excepciones vectorizadas. Las excepciones son condiciones inusuales o inválidas asociadas con la ejecución de una instrucción particular.
- Arquitectura "Thumb". Algunos procesadores ARM tienen esta arquitectura para aplicaciones que necesiten mejorar la densidad de código. Consiste en usar un set de instrucciones de 16 bits que es una forma comprimida del set de instrucciones ARM de 32 bits.

Categoría de instrucciones: Puede esperarse que un set de instrucciones de propósitos generales incluya instrucciones que se encuentren dentro de alguna de las siguientes categorías:

- Instrucciones de procesamiento de datos, tales como adición, substracción, multiplicación.
- Instrucciones de movimientos de datos, que copian datos de una posición de memoria en otra, o de memoria a registros del procesador, etc.
- Instrucciones de control de flujo, que conmutan la ejecución de una parte del programa por otra, posiblemente dependiendo de los valores de los datos.
- Instrucciones especiales que controlan el estado de la ejecución del procesador, por ejemplo para conmutar a modo privilegiado que lleva a funciones del Sistema Operativo.

Instrucciones de procesamiento de datos: Una instrucción que procesa datos requiere de dos operandos, uno de ellos siempre es un registro ***a)*** y el otro es un segundo registro o un valor inmediato. El segundo operando pasa a través del registro de desplazamiento *barrel shifter* ***b)***, donde sufre un desplazamiento, luego se combina con el primer operando en la ALU. Finalmente, el resultado de la ALU se escribe en el registro destino ***c)*** (y se puede actualizar el registro de código de condición). Todas estas instrucciones tienen lugar en un único ciclo de reloj, como se exhibe en la figura 3-14.

Nótese, también, cómo el valor del contador de programa, PC, en el registro de direcciones, se incrementa y se copia nuevamente, tanto en el registro de direcciones como en r15 del banco de registros ***d)***. La próxima instrucción se carga al final del pipeline de instrucciones ***e)*** (i. pipe). Cuando se requiere el valor inmediato se lo extrae de la instrucción actual ***f)***, al tope del pipeline de instrucciones. Solamente se usan los ocho bits finales, bits [7:0], de la instrucción para el procesamiento de instrucciones con el segundo operando dado con un valor inmediato.

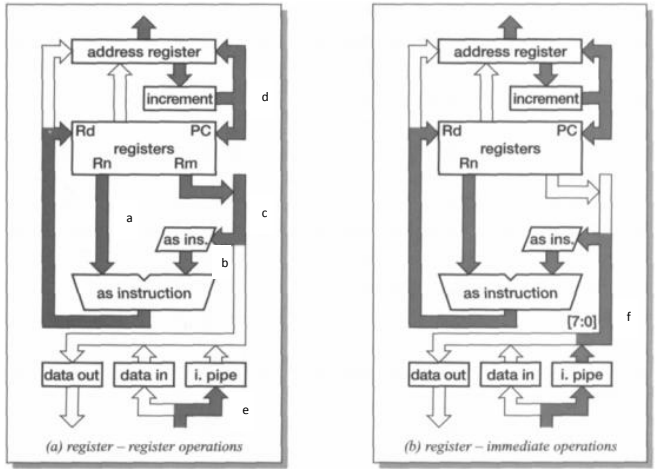


Figura 3-14: Actividad del camino de datos de las instrucciones de procesamiento de datos.
Fuente: Steve Furber. "ARM. System-on-Chip Architecture",.

Instrucciones de movimientos de datos: Una instrucción de transferencia de datos (carga o almacenamiento) calcula una dirección de memoria de una manera muy similar a la que lo hacen las instrucciones de procesamiento de datos. Se usa un registro como dirección base, al cual se le agrega (o se le substrae) un desplazamiento que puede ser otro registro o un valor inmediato. Se envía la dirección al registro de direcciones y tiene lugar un segundo ciclo de transferencia de datos. En lugar de dejar el trayecto de los datos (datapath) inactivo durante la transferencia de los datos, la ALU retiene las direcciones desde el primer ciclo y está disponible por si se requiere usarlas para calcular una

modificación de auto indexación del registro base. Si no se requiere la auto-indexación, en el segundo ciclo no se escribe el valor calculado en el registro base. En la figura 3-15 se muestran los dos ciclos de la operación del trayecto de los datos de una instrucción de almacenamiento (store, STR) con un desplazamiento inmediato. Nótese como el valor incrementado del contador de programa, PC, se almacena en el banco de registros al finalizar el primer ciclo, así el registro de direcciones está libre para aceptar la dirección del dato transferido del segundo ciclo, luego, al finalizar el segundo ciclo, el contador de programa, PC, se carga nuevamente en el registro de direcciones para permitir que continúe la pre búsqueda del código de operación de la instrucción. Nótese, que en esta etapa el valor enviado al registro de direcciones en un ciclo es el valor usado para el acceso a memoria en el ciclo próximo. En efecto, el registro de direcciones es un registro pipeline entre el trayecto de los datos del procesador y la memoria externa. Cuando se lo desea, el registro de direcciones puede producir una dirección de memoria para el próximo ciclo, un poquito antes del final del ciclo actual. Esto puede permitir que funcionen con alto rendimiento varios dispositivos de memoria. Por ahora, veremos el registro de direcciones como un registro pipeline a la memoria. Cuando la instrucción especifica el almacenamiento de un tipo de datos byte, el bloque de datos data.out extrae un byte desde el final del registro y lo repite cuatro veces a través del bus de datos de 32 bits. Luego, la lógica de control de la memoria externa, puede usar los dos bits del bus de direcciones del final para activar el byte apropiado dentro del sistema de memoria. Las instrucciones de carga siguen un patrón similar excepto porque el dato se adquiere desde la memoria solo a través del registro data.in en el segundo ciclo y se necesita un tercer ciclo para transferir el dato desde allí al registro destino.

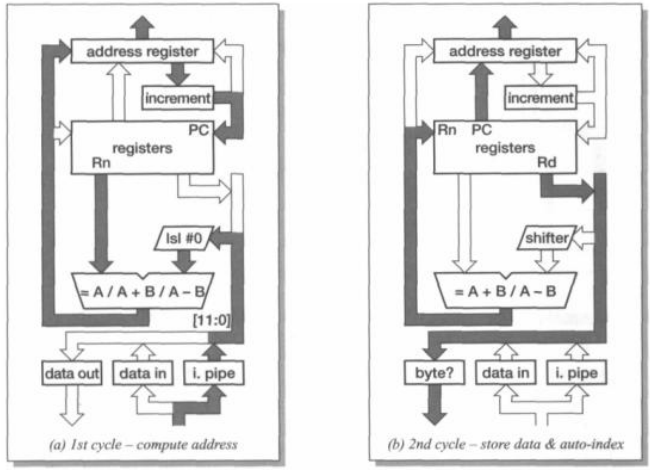


Figura 3-15: Actividad del trayecto de los datos en STR (store register)
Fuente: Steve Furber. "ARM. System-on-Chip Architecture"

Instrucciones de control de flujo: Las instrucciones de salto calculan la dirección destino en el primer ciclo, como se muestra en la siguiente figura 3-16. Se extrae un campo inmediato de 24 bits de la instrucción, luego se desplaza dos posiciones de bits hacia la izquierda para dar un desplazamiento alineado por palabras, que se agrega al contador de programa, PC. El resultado es emitido como una dirección de búsqueda de código de operación de una instrucción, y mientras se rellena el pipeline de instrucciones, si se lo requiere se copia la dirección de retorno en el link register, r14, es decir, la instrucción es un salto con retorno. El tercer ciclo, que se requiere para completar el rellenado del pipeline, también se usa para hacer una conexión pequeña al valor almacenado en el link register a fin de que éste apunte directamente a la instrucción que sigue al salto. Esto es necesario porque r15 contiene PC+8, mientras que la dirección de la próxima instrucción es PC+4. Otras instrucciones ARMs funcionan de manera similar a éstas descriptas.

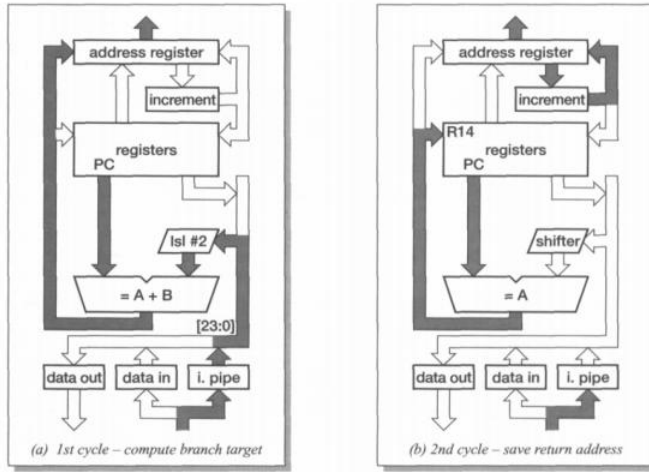


Figura 3-16: Los primeros dos (o tres) ciclos de una instrucción de salto
Fuente: Steve Furber. "ARM. System-on-Chip Architecture"

El "barrel shifter" la arquitectura ARM soporta instrucciones que ejecutan operaciones de desplazamiento en serie con una operación de la Unidad Aritmético–Lógica, ALU. De esta manera se vuelve crítica la eficacia del "shifter", ya que el tiempo de desplazamiento contribuye directamente al tiempo de los ciclos del trayecto de los datos, como se muestra en figura 3-17.

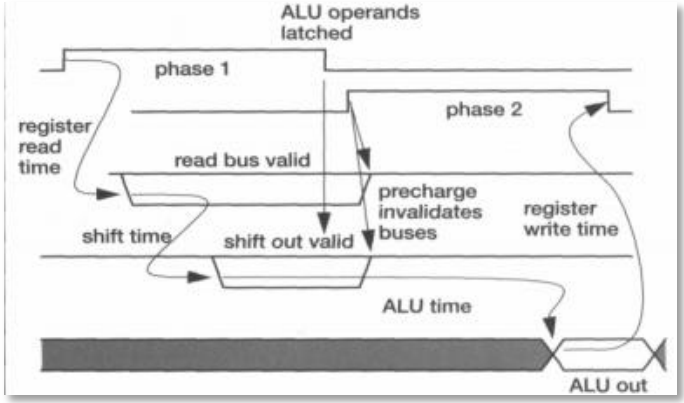


Figura 3-17: ARM Data path timing. (Pipeline de 3 etapas)

A fin de minimizar el retraso a través del "shifter", se usa una diagonal de matriz de interruptores para llevar cada entrada a la salida apropiada. El principio de cross-bar de matriz de interruptores se ilustra en la figura 3-18 para una matriz de 4 x 4 (el procesador ARM usa una matriz de 32 x 32). Cada entrada se conecta a cada salida a través de un interruptor. Si se usa lógica precargada dinámica, como la que se utiliza en el trayecto de los datos de ARM, se puede implementar cada interruptor como un único transistor NMOS. Se implementan las funciones de desplazamiento conectando los interruptores a lo largo de la diagonal a una entrada de control común.

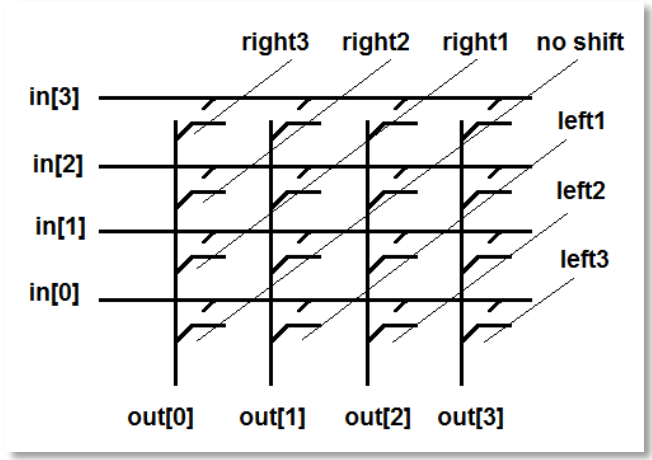


Figura 3-18: El principio de la diagonal ("travesaño") de matriz de interruptores del "barrel shifter"
Fuente: Steve Furber. "ARM. System-on-Chip Architecture"

3.2.d El sistema de memoria

Además del estado del registro del procesador, un sistema ARM tiene estado de la memoria. La memoria se puede ver como un arreglo lineal de bytes numerados desde cero hasta $2^{32} - 1$. Los datos pueden ser de byte (8 bits), de media palabra (16 bits) o palabra (32 bits). Las palabras están siempre alineadas en bandas de

4 bytes (esto es, los 2 bits de direcciones menos significativos son cero, porque son múltiplos de 4) y las medias palabras están alineadas en bandas de bytes pares (porque son múltiplos de 2).

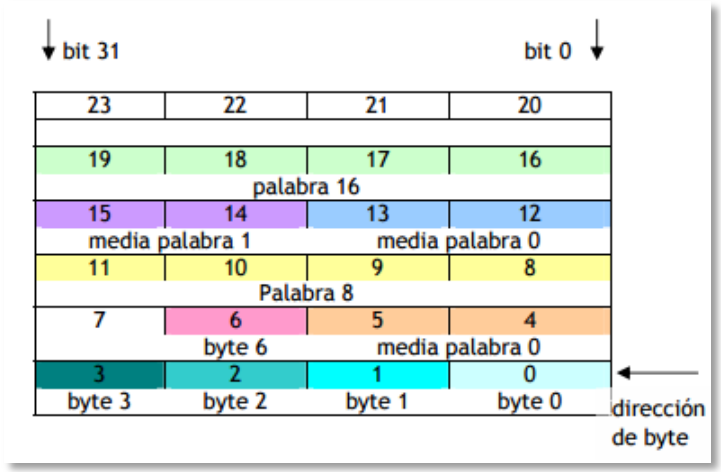


Figura 3-19: Organización de la memoria ARM, little endian

La figura 3-19 muestra una pequeña área de memoria donde la posición de cada byte tiene un único número. Un byte puede ocupar cualquiera de estas posiciones de memoria. El dato de tamaño de una palabra debe ocupar un grupo de posiciones de cuatro bytes que comienzan en una dirección que es un múltiplo de cuatro y tiene sus cuatro bits menos significativos en 0.

Esta es la organización de memoria little-endian usada por Intel y por ARM. Algunos ARM se pueden configurar para trabajar como big-endian, que es la configuración adoptada por Motorola y por los protocolos TCP, entre otros, donde los bytes se escriben en el orden natural en que se los lee.

En la forma big endian, figura 3-20, al tener primero el byte de mayor peso, se puede saber rápidamente si el número es positivo o negativo sólo comprobando el estado del bit más significativo del primer byte (recordemos que el signo se almacena en el bit más significativo) y sin necesidad de saber la longitud del número. Esta forma de representación coincide con el orden en que se escriben los números, de modo que las rutinas de conversión entre sistemas

de numeración son más eficientes que si se realizaran en little endian.

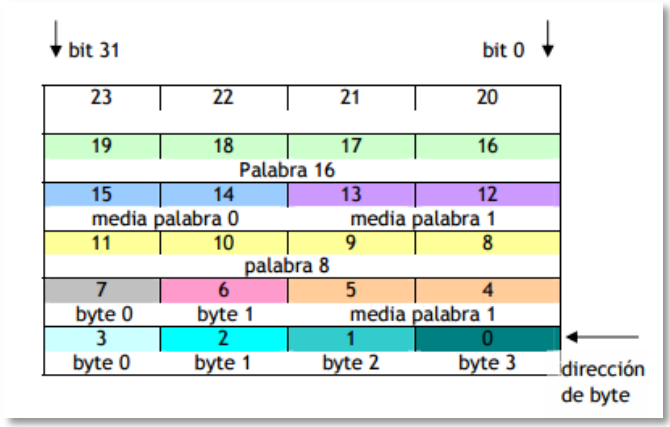


Figura 3-20: Organización de la memoria ARM, big endian

3.2.e Sistema de entrada/salida

El ARM maneja periféricos de E/S (tales como controladores de disco, interfaces de red, etc.) como dispositivos "mapeados" como memoria, con soporte de interrupciones. Los registros internos de estos dispositivos aparecen como posiciones direccionables dentro del mapa de memoria del ARM y se pueden leer y escribir usando las mismas instrucciones (Load/Store - carga / almacenamiento) como cualquier otra posición de memoria. Los periféricos puede llamar la atención del procesador haciendo un pedido de interrupción usando:

La interrupción normal (IRQ, Interrupt request).

La interrupción de alta prioridad (FIQ, Fast Interrupt Request).

Ambas entradas de interrupción son sensibles a nivel y enmascarables. Normalmente la mayoría de las fuentes de interrupción comparten la entrada IRQ. Algunos sistemas pueden incluir hardware externo para acceso directo a memoria (DMA,

Direct Memory Access) para que el procesador maneje el tráfico de E/S de banda ancha.

Un dispositivo periférico como, por ejemplo, el controlador de línea serie, contiene un número determinado de registros. En un sistema "mapeado" como memoria cada uno de estos registros aparecen como posiciones de memoria en una dirección particular (una alternativa es una organización del sistema con funciones de E/S en un espacio de direcciones separado del de los dispositivos de memoria). Un controlador de línea serie puede tener un conjunto de registros como los siguientes:

- Un registro de transmisión de datos (de escritura solamente), los datos que se escriban en este registros se enviarán por la línea serie.
- Un registro de recepción de datos (de lectura solamente), este es el destino de los datos que llegan por línea serie.
- Un registro de control (lectura y escritura), este registro "setea" la velocidad de los datos y controla la señal de solicitud de envío (request to send) y otras señales similares.
- Un registro de habilitación de interrupciones (de lectura y escritura), este registro controla qué eventos del hardware generarán una interrupción.
- Un registro de estado (de lectura solamente), este registro indica cuándo hay un dato disponible para leer, cuándo el buffer de escritura está vacío, etc.

Para recibir los datos el software debe inicializar apropiadamente al dispositivo, usualmente para generar una interrupción cuando hay un dato disponible o cuándo se detectó una condición de error. Luego, la rutina de atención de la interrupción debe copiar el dato en un buffer y verificar las condiciones de error para saber cuál es la que lo produjo y solucionarlo si es posible.

3.3 ARMv7

En la arquitectura ARMv7 encontramos tres perfiles distintos:

El perfil A, el cual está diseñado para procesadores que necesitarán manejar aplicaciones complejas, como Sistemas Operativos (Ej: Linux, Windows, etc). Estos procesadores requerirán alta capacidad de procesamiento, soporte para memoria virtual con unidad de manejo de memoria (MMU), y opcionalmente una unidad de protección de memoria (MPU).

El perfil R, el cual está diseñado para sistemas embebidos de alto espectro en el cual el desempeño en tiempo real toma un papel crítico. Equipos que usen este tipo de procesador son: controladores de disco duro (HD) los cuales necesitan baja latencia.

El perfil M, el cual está diseñado para sistemas de tipo microcontrolador, donde una baja latencia de interrupción, bajo costo y facilidad de uso son factores críticos. Son utilizados para aplicaciones de control industrial, incluyendo sistemas de control en tiempo real.

Las familias de procesadores Cortex son los primeros productos basados en la arquitectura v7 y el Cortex-M3 está basado en el perfil M de esta arquitectura.

3.3.a Cortex-M3

El Cortex-M3 es un microprocesador de 32bits. Tiene un tamaño de bus de datos de 32 bits, un banco de registros de 32 bits, e interfaces de memoria de 32 bits. El procesador tiene una arquitectura Harvard, lo que significa que su bus de instrucciones y de datos están separados. Esto permite a las instrucciones y a los datos ser accedidos simultáneamente, y como resultado de esto, el desempeño del procesador aumenta debido a que los accesos a datos no afectan el pipeline de instrucciones. Sin embargo, los buses de instrucción y datos comparten el mismo espacio de memoria

físico, por lo que se tienen 4GB de espacio direccionable tanto para datos como para instrucciones.

Para aplicaciones complejas que requieran más funciones de manejo de memoria, el Cortex-M3 puede tener un MPU opcional si fuese necesario.

También dentro del diseño del Cortex-M3 está incluido un número de componentes de debugging. Estos componentes ofrecen funciones de debugging, como así también puntos de quiebre (breakpoints) y de observación (watchpoints).

La figura 3-21 muestra un esquema simplificado de esta estructura, en línea punteada se muestra los módulos opcionales.

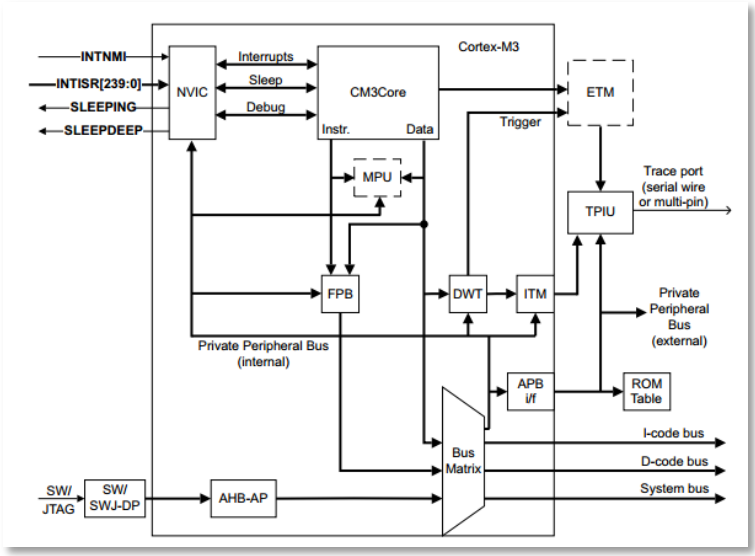


Figura 3-21: Diagrama en bloques (Fuente ARM – Cortex-M3, Technical Reference Manual, 2006)

La CPU del Cortex puede ejecutar la mayor parte de sus instrucciones en un solo ciclo, esto se logra gracias a un pipeline de 3 etapas: BUSQUEDA – DECODIFICACION - EJECUCION (figura 3-22 y 3-23).

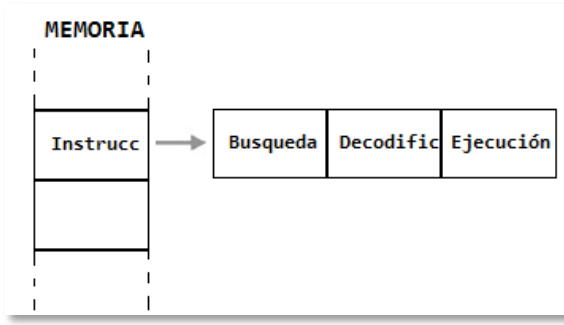


Figura 3-22: Pipeline de 3 etapas

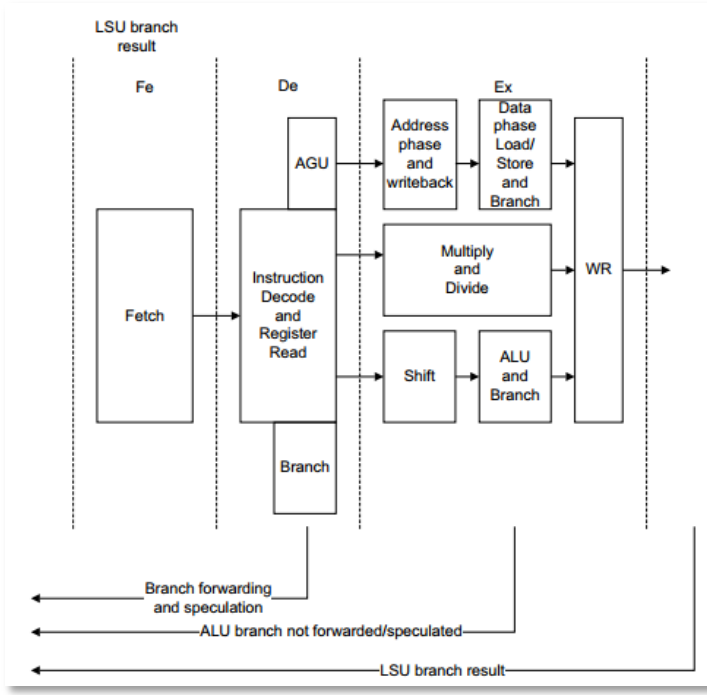


Figura 3-23: Pipeline de 3 etapas
Fuente: ARM – Cortex-M3, Technical Reference Manual, 2006

Además del pipeline, el Cortex posee una función de predicción de salto. Esto significa que al momento de ejecutar una instrucción de salto condicional, se hace un fetch (se "lee") especulativo, de tal forma que ambas direcciones están disponibles para ejecutar sin incurrir a limpiar el pipeline.

El set de instrucciones es un subconjunto del THUMB-2 donde coexisten instrucciones de 16 y de 32 bits.

El Thumb2 ofrece una mejora en densidad de código del 26% sobre el set de instrucciones ARM y una mejora de 25% en desempeño sobre el Thumb. Además posee instrucciones mejoradas de multiplicación capaces de ejecutarse en un ciclo y una división por hardware que toma entre 2 y 7 ciclos. Debido a la estás mejoras, al pipeline y a la tecnología Thumb2, el nivel de desempeño del Cortex es de 1.2 a 1.25 DMIPS/MHz.

La definición del mapa de memoria (Figura 3-24) realizado por ARM ha permitido que (en términos generales) todos los fabricantes ubiquen memoria y periféricos en las mismas direcciones, facilitándose así la migración de un fabricante a otro sin mayores complicaciones.

Dado que la arquitectura ARMv7-M fue pensada para aplicaciones que pueden requerir el uso de sistemas operativos, realizaremos una división entre sistema y aplicación.

El primer Gbyte de memoria está dividido a la mitad en una región de código (0,5GB) y una de SRAM (0,5 GB). El primer MByte de la región de SRAM y de la subsiguiente región de periféricos es direccionable por bit usando una técnica llamada *bit banding.*

La ventaja directa de tener zonas de memoria direccionable por bit es la de acceder a registros y modificar sólo algunos *flags;* en un programa toma menos tiempo ya que no sea necesario escribir código extra para hacerlo.

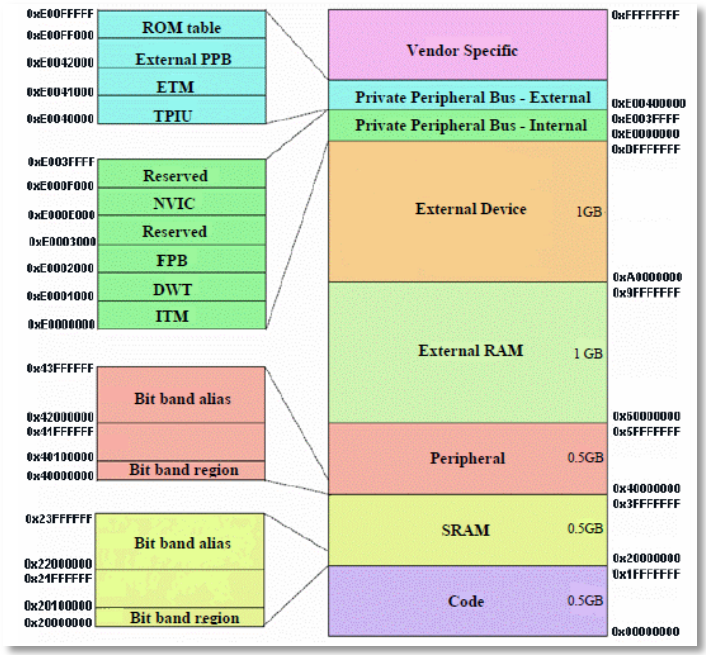


Figura 3-24: Mapa de memoria
Fuente: An Introduction to the ARM Cortex-M3 Processor

El NVIC o Nested Vector Interrupt Controller (Figura 3-25) es el controlador de interrupciones estándar del núcleo Cortex, esto significa que todos los microcontroladores basados en Cortex tendrán la misma estructura de interrupciones, independientemente del fabricante. Por lo tanto, el código de aplicación y sistemas operativos podrán ser fácilmente portables de un microcontrolador a otro y el programador no necesitará aprender nuevos registros cada vez que se utilice nuevo hardware.

El NVIC también tiene una latencia de interrupción muy corta, esto se debe tanto al diseño del NVIC propiamente como también al set de instrucciones Thumb-2 que permite instrucciones multiciclo.

Como su nombre lo indica, el NVIC soporta interrupciones anidadas, como así también las interrupciones externas y la mayoría de las excepciones del sistema pueden ser programadas con distintos niveles de prioridad. Cuando una interrupción ocurre, el NVIC revisa

el nivel de prioridad de la interrupción y lo compara con la prioridad de la interrupción activa. Si el nivel de prioridad de la nueva interrupción es mayor que el nivel actual, el handler de interrupción de la nueva interrupción desestimará la tarea activa y comenzará a ejecutar la nueva rutina.

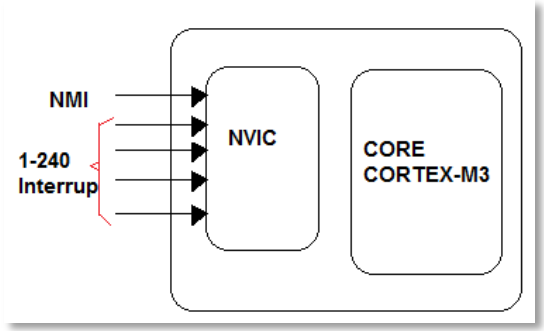


Figura 3-25: Tipos de Interrupciones del NVIC

3.2.b Arquitectura a nivel de sistema

El procesador puede operar en uno de dos modos solamente, figura 3-26. (Recordar que en la descripción general del modelo ARM se definió la existencia de 7 modos: User, FIQ, IRQ, Supervisor, Abort, Undefined Instruction).

Modo *THREAD*, modo normal. En este modo se inicia después de un reset o al terminar de procesar una excepción.

Modo *HANDLER*, modo en que se procesan las excepciones.

El modo Handler es un modo privilegiado para atención de excepciones (interrupciones), mientras que el Thread puede operar en modo privilegiado o no privilegiado, se retorna a este modo finalizada la atención de la excepción.

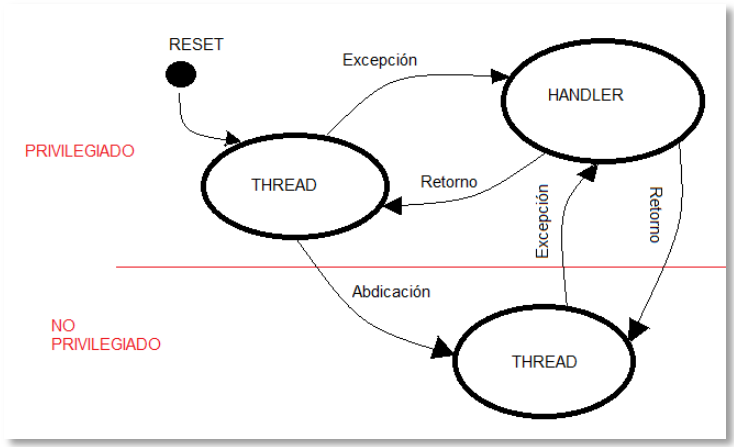


Figura 3-26: modos de operación en ARMv7-M.

En esta versión no existen las interrupciones IRQ y FIQ. Las interrupciones son separadas y anidadas gracias al SCB (System Control Block) y al NVIC. El SCB se encuentra en el area de sistema en el mapa de memoria, en la región Private Peripheral Bus (Figura 3-24) y provee funciones para la operación del procesador, fundamentalmente en el manejo de excepciones. Las más importantes son:

- Control de reset por software.
- Manejo de la tabla de vectores de excepciones.
- Manejo de las excepciones.
- Definición del esquema de prioridades.
- Control de la energía.

La dirección de la rutina para atender una excepción se obtiene leyendo una posición de memoria dentro del vector de excepciones. Esta arquitectura recupera la interrupción no enmascarable NMI.

Manejo de excepciones: Esto también difiere del modelo ARM, ante una excepción, el procesador suspende su actividad y guarda los registros R0 a R3, R12, el PC, el PSR y el LR (R14) en el STACK. Al momento de ingresar a la función que atiende a esta excepción, el LR se carga con un valor que no corresponde a una posición valida

del mapa de memoria, de modo que ante una instrucción de retorno de subrutina, el procesador identifica que se trata de un retorno de excepción y recupera del stack los registros salvados.

Las excepciones del sistema se manejan en un bloque Sistem Control Block (SCB), ubicado en el Sistem Control Space (SCS).

Cada excepción puede estar en alguno de los siguientes estados:

- Inactiva: La excepción no está activa ni pendiente.
- Pendiente: La excepción está esperando por un servicio del procesador.
- Activa: La excepción es atendida por el procesador y todavía no se completó la tarea.

Las excepciones del sistema son:

- Reset: excepción de prioridad más alta (-3)

  Se invoca en el encendido o un reinicio en caliente. Es una forma especial de excepción. Cuando se produce un Reset, la operación del procesador se detiene, potencialmente en cualquier punto en una instrucción. Cuando se sale del reinicio, la ejecución se inicia desde el dirección proporcionada por el vector de reset (ver tabla de vectores) y en modo Thread privilegiado

- NMI: prioridad siguiente al Reset (-2)

  Una interrupción no enmascarable (NMI) sólo puede ser desencadenada por software. Esta es la excepción de prioridad más alta cuando no se produce el Reset. Está habilitada de forma permanente y tiene una prioridad fija de -2. No puede ser:

  - enmascarada o impedida la activación por cualquier otra excepción
  - precedida por ninguna excepción que no sea el Reset.

- HardFault: genérica para atender cualquier tipo de falla prioridad (-1)

  Es una excepción que se produce debido a un error durante el procesamiento de excepciones, o porque una excepción no

puede ser manejada por cualquier otro mecanismo de excepción. Las *HardFault* tienen una prioridad fija de -1, lo que significa que tienen mayor prioridad que cualquier excepción con prioridad configurable.

- MemManageFault: generada por error de protección de la memoria.

  Es una excepción que se produce a causa de una falla relacionada con la protección de memoria. El MPU o las restricciones de protección de memoria determinan esta falla, tanto para transAcciónes de la memoria de instrucción como de datos. Este fallo se usa para abortar la instrucción en la cual se accede a la región de memoria *Execute Never* (XN), incluso si la MPU está desactivada.

- BusFault: error de acceso a memoria para instrucciones o datos.

  Es una excepción que se produce debido a un fallo de memoria relacionada con una transacción de instrucciones o en la memoria de datos. Esto podría ser de un error detectado en un bus del sistema de memoria.

- UsageFault: fallos en la ejecución (ejemplo instrucciones no reconocidas).

  Es una excepción que se produce debido a un fallo relacionado con la ejecución de la instrucción. Esto incluye:

  - Una instrucción indefinida.
  - Un acceso ilegal.
  - Estado no válido en ejecución de la instrucción.
  - Un error en la declaración de excepción.

  Este tipo de fallas puede ser causada por:

  - Una dirección no alineados en la palabra y media palabra.
  - División por cero.

- SVCall: respuesta a una instrucción SVC, para realizar llamas al sistema operativo.

Es una excepción que se desencadena por la instrucción SVC. En un entorno de Sistema Operativo, las aplicaciones pueden utilizar las instrucciones SVC para acceder a funciones del kernel del Sistema Operativo y los controladores de dispositivos.

- DebugMonitor: soporta al esquema de depuración.
- Interrupt (IRQ): Es una excepción señalada por un periférico, o generados por un pedido de software. Todas las interrupciones son asíncronas.

Las interrupciones del sistema son:

- PendSV: se genera por software y es parte del proceso de atención a una llamada al sistema operativo mediante una instrucción SVC, la prioridad se puede configurar pero no se puede inhabilitar.
- SysTick: generada por el timer del mismo nombre, la prioridad se puede configurar pero no se puede inhabilitar.

Las interrupciones externas:

La arquitectura ARM v7, dentro del SCB pose el registro ICTR (Interrup Controler Type Register), el cual contiene un campo de 4 bit llamado INTLINESNUM que se utiliza para especificar la cantidad de interrupciones posibles divididas por 32, así el numero máximo de interrupciones externas será 2^4 x 32 = 512, de estas 16 corresponden a las excepciones del sistema por lo que nos quedan 496 interrupciones externas posibles.

En Cortex M3, solo 3 de estos bit se utilizan por lo que la cantidad de interrupciones externas se reduce a 2^3 X 32 = 256 menos 16 tendremos 240 interrupciones externas posibles (Figura 3-25).

En la siguiente tabla 3-1, se muestran las características de las diferentes excepciones para Cortex M3 (Fuente manual LPC17xx manual del usuario).

Tabla 3-1: Resumen de excepciones

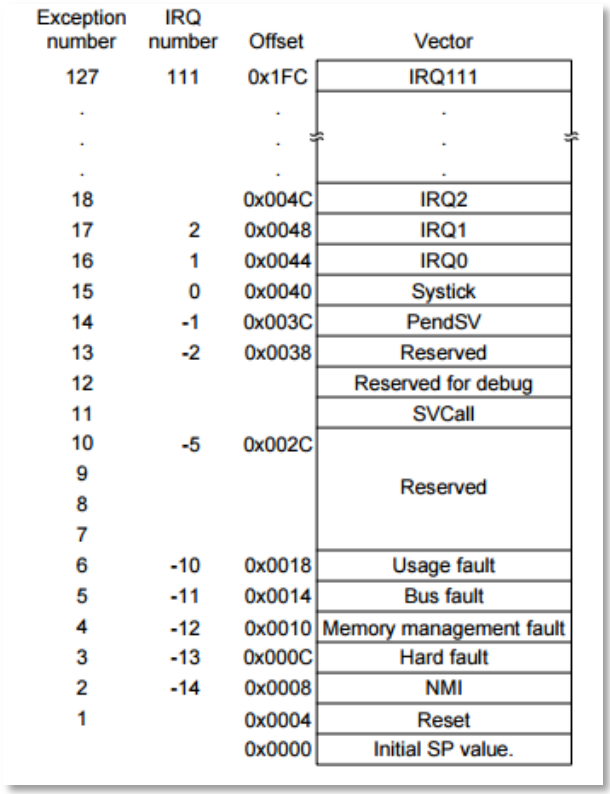
Nro Excepción	Nro IRQ (1)	Tipo	Prioridad	Vector
1	-	Reset	-3	0x00000004
2	-14	NMI	-2	0x00000008
3	-13	Falla Hardware	-1	0x0000000C
4	-12	Falla Memoria	Configurable	0x00000010
5	-11	Falla Bus	Configurable	0x00000014
6	-10	Falla Manejo	Configurable	0x00000018
7-10	-	-	-	Reservado
11	-5	SVCall	Configurable	0x0000002C
12-13	-	-	-	Reservado
14	-2	PendSV	Configurable	0x00000038
15	-1	SysTick	Configurable	0x0000003C
16	0 y hacia arriba	Interrup	Configurable	0x00000040 y hacia arriba(2)

(1) Para simplificar la capa de software, el CMSIS sólo utiliza el número de IRQ y por lo tanto utiliza valor negativo para más excepciones que las interrupciones.

(2) Se incrementa en saltos de 4.

3.3.c Tabla de vectores

La tabla de vectores contiene el valor de reposición del *stack pointer* (puntero de pila), y las direcciones de inicio, también llamados vectores de excepción, para todos los controladores de excepciones. La figura 3-27 muestra el orden de los vectores de excepción en la tabla de vectores. El bit menos significativo de cada vector debe ser 1, que indica que el controlador de excepciones es un código *Thumb*.



Exception number	IRQ number	Offset	Vector
127	111	0x1FC	IRQ111
.	.	.	.
.	.	.	.
.	.	.	.
18		0x004C	IRQ2
17	2	0x0048	IRQ1
16	1	0x0044	IRQ0
15	0	0x0040	Systick
14	-1	0x003C	PendSV
13	-2	0x0038	Reserved
12			Reserved for debug
11			SVCall
10	-5	0x002C	Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value.

Figura 3-27: Tabla de vectores

Prioridades de excepción

Como muestra la Tabla 3-1, todas las excepciones tienen una prioridad asociada con:

• Un valor de prioridad más baja → indica una mayor prioridad

• Prioridades configurables para todas las excepciones excepto *Reset*, *Hard-fault* y *NMI*.

Si el software no configura ninguna prioridad, todas las excepciones con una prioridad configurable tienen una prioridad de 0.

Observación: valores de prioridad configurables están en el rango de 0 a 31. Esto significa que el *Reset*, *Hard-fault* y *NMI*, con valores de prioridad negativos fijos, siempre tienen mayor prioridad que cualquier otra excepción. Por ejemplo, la asignación de un valor mayor prioridad a IRQ [0] y un valor de prioridad inferior a IRQ [1] significa que IRQ [1] tiene mayor prioridad que IRQ [0]. Si tanto IRQ [1] e IRQ [0] aparecen, IRQ [1] se procesa antes de IRQ [0].

Si hay varias excepciones pendientes con la misma prioridad, la excepción pendiente con el número de excepción más bajo tiene prioridad. Por ejemplo, si IRQ [0] y IRQ [1] están pendientes y tienen la misma prioridad, entonces IRQ [0] se procesa antes de IRQ [1].

Cuando el procesador está ejecutando un manejador de excepciones, este se adelanta (preempted) si se produce una excepción de prioridad más alta. Si se produce una excepción con la misma prioridad que la excepción que se maneja, el controlador no se adelanta, con independencia del número de excepción. Sin embargo, el estado de la nueva interrumpir cambia a pendiente.

Agrupación de interrupción por prioridad

Para aumentar el control prioritario en los sistemas con interrupciones, la NVIC soporta agrupación por prioridad.

Esto divide cada entrada de registro de prioridad de interrupción en dos campos:

- Un campo superior que define la prioridad de grupo.
- Un campo inferior que define una sub-prioridad dentro del grupo.

Sólo la prioridad de grupo determina preeminencia sobre las excepciones de interrupción. Cuando el procesador está ejecutando un manejador de excepción de interrupción, otra interrupción de las mismas prioridades de grupo que la interrupción que está manejando no adelanta al controlador, si varias interrupciones pendientes tienen la misma prioridad de grupo, el campo sub-prioridad determina el orden en que se procesan. Si varias interrupciones pendientes tienen la misma prioridad de grupo y sub-prioridad, la interrupción con el número IRQ más bajo se procesa primero.

Entrada y salida de excepción

Para describir el manejo de excepciones se utilizan los siguientes términos:

Preemption (adelantamiento) o Anticipación: Cuando el procesador está ejecutando un manejador de excepciones, una excepción puede adelantarse si su prioridad es más alta que la prioridad de la excepción que está siendo atendida. Cuando una excepción se adelanta a otra, las excepciones se llaman excepciones anidadas.

Return: Esto ocurre cuando se completa el gestor de excepciones, y:

- No hay ninguna excepción pendiente con suficiente prioridad que deba ser atendida.
- El manejador de excepciones completado no estaba manejando una excepción *late-arriving* (llegada tardía).

El procesador restaura el estado que tenía antes de producirse la interrupción.

Tail-chaining (cola de encadenamiento): Este mecanismo acelera el servicio de excepción. Al término de un manejador de excepciones, si hay una excepción pendiente que cumple con los requisitos de entrada, se recupera la pila (pop) y se transfiere el control al nuevo manejador de excepciones.

Late arriving (llegada tardía): Si una excepción de prioridad más alta se produce durante el proceso de guardar los registros en el stack para una excepción anterior, el procesador pasa a controlar la excepción de mayor prioridad e inicia la búsqueda del servicio de la excepción. El proceso de guardar los registros en el stack no se ve afectada por la llegada con retraso debido a que el estado guardado es el mismo para ambas excepciones.

Entrada a excepción

La entrada a una excepción se produce cuando hay una excepción con suficiente prioridad y, o bien:

• El procesador está en el modo Thread.

• La nueva excepción es de mayor prioridad que la excepción que se maneja, en cuyo caso la nueva excepción se antepone a la excepción original.

Cuando una excepción se adelanta a otra, las excepciones se anidan.

Cuando el procesador tiene una excepción, a menos que la excepción es una *tail-chaining* o una excepción de *late-arriving*, el procesador actualiza la información en la pila (push). Esta operación se conoce como *stacking* (apilamiento) y la estructura de ocho palabras de datos se conoce como *stack frame* (marco de pila). El marco de pila contiene la siguiente información:

- R0-R3, R12.
- Dirección de retorno.
- PSR.
- LR.

Inmediatamente después del stacking, el stack pointer (puntero de pila) indica la dirección más baja en el stack frame (marco de pila). A menos que la alineación de la pila esté desactivada, el stack frame está alineado a una dirección de palabra doble. Si el bit STKALIGN del registro de control de configuración (CCR) se establece en 1, el ajuste del alineamiento se realiza durante el apilamiento.

El stack frame incluye la dirección de retorno. Esta es la dirección de la siguiente instrucción en el programa interrumpido. Este valor se restaura en el PC al regreso de la excepción para que el programa interrumpido se reanude.

En paralelo a la operación de apilamiento, el procesador realiza una búsqueda leyendo la dirección de inicio del manejador de excepción (ver vectores). Cuando el stacking se completa, el procesador comienza a ejecutar el manejador de excepciones. Al mismo tiempo, el procesador escribe un valor EXC_RETURN a la LR. Esto indica que el stack pointer se corresponde al stack frame.

Si no se produce una excepción con prioridad más alta durante la entrada a la excepción, el procesador comienza a ejecutar el controlador de la excepción y automáticamente cambia al estado activo. Si otra excepción de prioridad más alta se produce durante la entrada de excepción, el procesador comienza a ejecutar el

manejador de excepción para esta excepción y no cambia el estado pendiente de la excepción anterior. Este es el caso de llegada tardía.

Retorno de una excepción

Un retorno de excepción se produce cuando el procesador está en el modo de *Handler* y ejecuta una de las siguientes instrucciones para cargar el valor EXC_RETURN en el PC:

- una instrucción POP que incluye el PC
- una instrucción BX con cualquier registro.
- una LDR o instrucción LDM con el PC como destino.

EXC_RETURN es el valor cargado en el LR a la entrada de la excepción. El mecanismo de excepción cuenta con este valor para detectar cuando el procesador ha completado un manejador de excepción. Los cuatro bits más bajos de este valor proporcionan información sobre la pila de retorno y el modo del procesador. La Tabla 3-2 muestra los valores EXC_RETURN [3: 0] con una descripción del comportamiento de retorno excepción.

Tabla 3-2: Comportamiento de retorno de una excepción.

EXEC_RETURN [3:0]	Descripción
bXXX0	Reservado
b0001	Retorno del modo HANDLER. Retorno de la excepción que captura el estado del MSP. Ejecución utiliza por MSP después del retorno.
b0011	Reservado
b01X1	Reservado
b1001	Retorno del modo THREAD Retorno de la excepción que captura el estado del MSP. Ejecución utiliza por MSP después del retorno.
b1101	Retorno del modo THREAD Retorno de la excepción que captura el estado del PSP. Ejecución utiliza por PSP después del retorno.
b1X11	Reservado

Fuente: LPC17xx User Manual (2010)

Administración de energía

El procesador Cortex M3 utiliza los modos de suspensión (sleep) para reducir el consumo de energía:

- Modo "*Sleep"* (Modo de Energía 1) detiene el reloj del procesador.
- Modo *"Deep Sleep"* (sueño profundo) (Modo de Energía 2/3) detiene los osciladores de alta frecuencia poner en off el PLL y la memoria flash.

El bit SLEEPDEEP del System Control Register (SCR) selecciona cuál modo de reposo se utiliza. Tabla 3-3.

Tabla 3-3: Asignación de bits en el SCR.

Bit	Nombre	Función
[31:5]	--	Reservado
[4]	SEVONPEND	0 = Solo las interrupciones habilitadas o los eventos pueden despertar al procesador. 1 = Eventos habilitados y todas las interrupciones, incluidas las deshabilitadas, pueden despertar al procesador. Cuando un evento o una interrupción entran en estado pendiente, la señal del evento despierta el procesador de WFE. Si el procesador no está esperando un evento, el evento es registrado y afecta la próxima WFE. El procesador también se despierta en la ejecución de una instrucción SEV o un evento externo.
[3]	--	Reservado
[2]	SLEEPDEEP	Indica a que modo de reposo entra el procesador 0 = sleep 1 = deep sleep
[1]	SLEEPONEXIT	Indica cuando se retorna del modo Handler al modo Thread: 0 = no entra en modo de reposo al retornar del modo Thread. 1 = entra en sleep, o deep sleep, al retornar de una ISR. Establecer este bit a 1 habilitando un interrupción a la aplicación, para evitar retornar una aplicación principal vacía.
[0]	--	Reservado

Fuente: LPC17xx User Manual (2010)

Entrando en modo de reposo (Sleep Mode)

El sistema puede generar eventos de activación espurios, por ejemplo, una operación de depuración despierta el procesador. Por lo tanto el software debe ser capaz de poner el procesador de nuevo en modo de reposo después de tal evento. Un programa puede tener un bucle de inactividad para poner el procesador de nuevo a modo de reposo.

Espera por una interrupción (WFI)

La instrucción espera una interrupción, WFI, provoca la entrada inmediata al modo de reposo. Cuando el procesador ejecuta una instrucción WFI, este detiene la ejecución de instrucciones y entra en modo de reposo.

Espera de evento (WFE)

La instrucción espera de eventos, WFE, provoca la entrada en modo de suspensión condicional según el valor de un bit del registro de eventos. Cuando el procesador ejecuta una instrucción de WFE, comprueba este registro:

- Si el registro es 0 el procesador deja de ejecutar instrucciones y entra en modo de reposo.
- Si el registro es 1 el procesador borra el registro a 0 y continúa ejecutando instrucciones sin entrar en modo de reposo.

Si el registro de eventos es 1, esto indica que el procesador no debe entrar en el modo de reposo de la ejecución de una instrucción de WFE. Típicamente, esto es porque aparece una señal de evento externo, o un procesador en el sistema ha ejecutado una instrucción Send Event (SEV). El software no puede acceder a este registro directamente.

Sleep-on-exit

Si el bit SLEEPONEXIT (Table 3-3) del SCR se establece en 1, cuando el procesador completa la ejecución de un manejador de excepción se vuelve al modo *Thread* y de inmediato entra en modo de suspensión. Utilice este mecanismo solo en las aplicaciones que

requieren del procesador para actuar cuando se produce una excepción.

Despertar del modo de reposo

Las condiciones para que el procesador despierte (*wakeup*) dependen del mecanismo que provoca que entre en modo de suspensión.

Despertar del Wair For Interrupt (WFI) o sleep-on-exit

Normalmente, el procesador se despierta (*Wake-up*) sólo cuando se detecta una excepción con la suficiente prioridad para ser ejecutada.

Algunos sistemas embebidos pueden tener para ejecutar tareas de restauración del sistema después de que el procesador se despierta, y antes de que ejecute un manejador de interrupciones. Para ello establece el bit PRIMASK a 1 y el bit FAULTMASK a 0. Si llega una interrupción que está habilitada y tiene una prioridad más alta que la prioridad de la excepción actual, el procesador se despierta, pero no ejecuta el manejador de interrupciones hasta que el procesador establece PRIMASK a cero.

NOTA: Para acceder a los registros especiales se utilizan las instrucciones.

- **MRS:** mueve el contenido de un registro especial a un registro de propósitos generales.
- **MSR:** mueve el contenido de registro propósitos generales a un registro especial.
- **CPS:** Cambia el estado del procesador.

El **Priority Mask Register (**PRIMASK) evita la activación de todas las excepciones con la configuración de prioridades. Los bits de este registro son:

- bits [0] – PRIMASK – si esta en 0 no tiene efectos, si esta en 1 evita la activación de todas las excepciones con la configuración de prioridades.
- bits [31:1] – Reservados.

El **Fault Mask Register (**FAULTMASK) evita la activación de todas las excepciones, excepto de **Non-Maskable Interrupt** (NMI). Los bits son:

- bits [0] – FAULTMASK – si esta en 0 no tiene efectos, si esta en 1 evita la activación de todas las excepciones, excepto de NMI.
- bits [31:1] – Reservados.

El procesador pone en cero la FAULTMASK al salir de un manejador de excepciones, a excepción del manejador de NMI.

Despertar del Wait For Event (WFE)

El procesador se despierta si:

- Detecta una excepción con la suficiente prioridad para causar la atención de la excepción.

Además, si el bit SEVONPEND en el System Control Register (SCR) se establece en 1, cualquier nueva interrupción pendiente desencadena un evento y despierta el procesador, incluso si la interrupción está deshabilitada o tiene prioridad insuficiente para causar entrada de excepción. (Tabla 3-3).

El controlador de despertador de interrupción (Wake-up Interrupt Controller - WIC)

El controlador de despertador de interrupción (WIC) es un periférico que puede detectar una interrupción y despertar al procesador del modo *Deep sleep*, *Power-down* o *Deep Power-down*. El WIC sólo se activa cuando el bit SLEEPDEEP en el SCR se establece en 1. (Tabla 3-3).

El WIC no es programable, y no tiene ningún registro o interfaz de usuario. Opera enteramente por señales de hardware.

Cuando el WIC está activado y el procesador entra en modo de *Deep sleep* o el modo de *power-down*, la unidad de administración de energía del sistema puede apagar la mayor parte del procesador Cortex-M3. Esto tiene el efecto secundario de detener el

temporizador *SysTick*. Cuando el WIC recibe una interrupción, se necesita un número de ciclos de reloj para despertar al procesador y restaurar su estado, antes de que pueda procesar la interrupción. Esto significa latencia de interrupción y se incrementa en el modo de *Deep sleep*. Despertar de modo *Power-down* requiere el inicio de muchas otras partes del dispositivo, y toma más tiempo e incluye el restablecimiento del regulador de voltaje en el chip.

Observación: Si el procesador detecta una conexión con un depurador (*debugger*) deshabilita el WIC.

Consejos de programación de administración de energía

ANSI C no puede generar directamente el WFI e instrucciones WFE. El CMSIS proporciona las siguientes funciones intrínsecas de estas instrucciones:

```
void __WFE (void)          // Esperar Evento
void __WFI (void)          // Espere interrupción
```

CAPITULO 4: CMSIS (Cortex Microcontroller Software Interface Standard)

4.1 EL CMSIS - COMPONENTES

La evolución del hardware debe tener un acompañamiento similar por el software. Además de los lenguajes de programacion (C/C++), pasan a ser de importancia otros elementos del software tales como:

- Los administradores de tareas (RTOS).
- Los servicios de abstracción entre las capas de software y hardware.

El software de interfaz estándar para microntroladores cortex (CMSIS) facilita el desarrollo de software cuando se utiliza microcontroladores Cortex. El CMSIS se define en estrecha cooperación con varios proveedores de silicio y software y proporciona un enfoque común para interconectar a los periféricos, sistemas operativos en tiempo real, y los componentes de middleware.

ARM ofrece como parte de la CMSIS las siguientes capas de software (figura 4-1):

- Core Peripheral Access Layer (CPAL).
- Middleware Access Layer (MWAL).
- Device Peripheral Access Layer (DPAL).

• **Core Peripheral Access Layer**: contiene definiciones de nombres, de direcciones y funciones para acceder a los registros centrales y

periféricos. Define también una interfaz independiente para núcleo RTOS que incluye un canal de depuración.

Estas capas de software son ampliadas por terceros:

•**Device Peripheral Access Layer**: En esta capa se definen direcciones de registros de hardware y otras definiciones, así como funciones de acceso específicos a los dispositivos.

•**Middleware Access Layer**: Definida por ARM pero completada por los fabricantes del silicio y adaptada a las particularidades de sus dispositivos. Define una API que proporciona funciones de ayuda adicionales para aceder a los periféricos.

En el caso de la propuesta Cortex-M se ha definido la norma CMSIS, que define:

- Un modo común de acceso a periféricos.
- Una forma estándar de definir los vectores de excepción.
- Acuerdo en el uso de denominaciones para los periféricos propios de Cortex y sus vectores.
- Una interfase estándar e independiente del dispositivo (HAL) con el RTOS, y el debugger.
- Interfases con aplicaciones típicas (TCP/IP Stack, Flash File System).

Estas definiciones son enriquecidas por cada fabricante con la información propia de sus periféricos especiales.

Mediante el uso de componentes de software compatible con CMSIS, el usuario puede fácilmente reutilizar el código. CMSIS tiene por objeto permitir la combinación de componentes de software de múltiples proveedores de middleware.

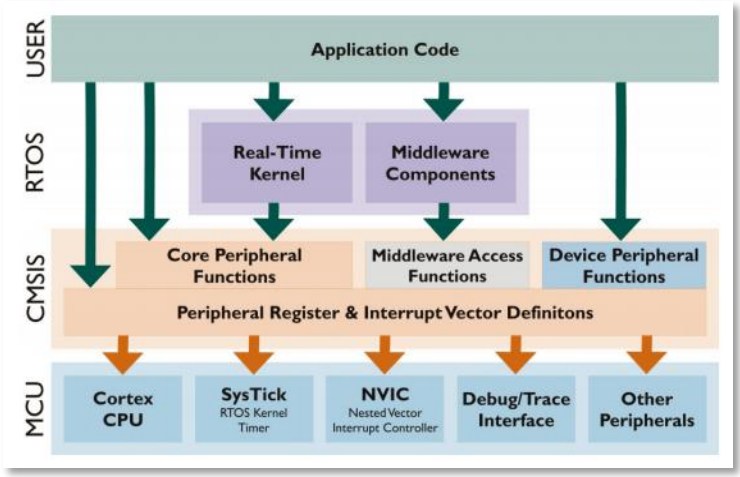


Figura 4-1: Estructura de capas de los sistemas embebidos
Fuente: Getting started with CMSIS.
https://www.doulos.com/knowhow/arm/CMSIS/CMSIS_Doulos_Tutorial.pdf

El objetivo principal de CMSIS (Cortex Microcontroladores Software Interfaz Stándar) es mejorar la portabilidad y reutilización del softwareen diferentes microcontroladores y cadenas de herramientas. Esto permite que el software de diferentes fuentes pueda integrarse perfectamente. CMSIS ayuda a acelerar el desarrollo de software a través del uso de las funciones estandarizados.

CMSIS consta de varias especificaciones entrelazadas que apoyan el desarrollo de código a través de todos los microcontroladores basados en Cortex-M.

- **CMSIS-CORE:** API para el núcleo del procesador Cortex-M y los periféricos. Proporciona una interfaz estandarizada para Cortex-M0, Cortex-M3, Cortex-M4, SC000 y SC300. Incluye también funciones intrínsecas para Cortex-M4 SIMD.
- **CMSIS-Driver:** define las interfaces de controladores periféricos genéricos de middleware por lo que es reutilizable a través de dispositivos compatibles. El API es RTOS independiente y conecta periféricos del microcontrolador con el middleware que

implementa para pilas ejemplos de comunicación, sistemas de archivos o interfaces gráficas de usuario.

- **CMSIS-DSP:** DSP Colección de la biblioteca con más de 60 funciones para los distintos tipos de datos: punto fijo (q7 fraccional, Q15, Q31) y simple precisión de punto flotante (32 bits). La biblioteca está disponible para Cortex-M0, Cortex-M3, y Cortex-M4. La aplicación Cortex-M4 está optimizado para el conjunto de instrucciones SIMD.

- **CMSIS-RTOS API:** API común para los sistemas operativos en tiempo real. Proporciona una interfaz de programación estándar que es portable a muchos RTOS y por lo tanto permite que las plantillas de software, middleware, bibliotecas y otros componentes que pueden trabajar a través de los sistemas soportados por RTOS.

- **CMSIS-Pack:** describe el paquete basado en XML (PDSC) presentar los partes de usuario y dispositivo relevante de una colección de archivos (llamado paquete de software) que incluye la fuente, archivos de cabecera y bibliotecas, documentación, algoritmos de programación flash, plantillas de código fuente, y proyectos de ejemplo. Las herramientas de desarrollo e infraestructuras web utilizan el archivo PDSC para extraer los parámetros del dispositivo, los componentes de software y configuraciones de la junta de evaluación.

- **CMSIS-SVD:** Describe los periféricos de un dispositivo en un archivo XML y se puede utilizar para crear conciencia periférica en depuradores o archivos de cabecera con registro periférica e interrumpir las definiciones.

- **CMSIS-DAP:** Depurar puerto de acceso. Firmware normalizado para una Unidad de depuración que se conecta a la CoreSight depuración puerto de acceso. CMSIS-DAP se distribuye como paquete independiente y muy adecuado para la integración en las juntas de evaluación. Este componente se ofrece como descarga independiente

4.2 REGLAS Y NORMAS DE CODIFICACIÓN

Para un mejor entendimiento del código se resumen algunas reglas y normas de codificación utilizadas en la implementación del CMSIS.

Reglas elementales

- El código "C" CMSIS se ajusta a las reglas MISRA 2004 [MISRA-C - 2004].
- Los tipos de datos estándar ANSI están definidos en el archivo de cabecera**<stdint.h>.**
- Se utiliza la expresión #define para definir constantes.
- Las variables tienen un tipo de dato completo.
- Todas las funciones de **Core Peripheral Access Layer** son de re-entrada.
- La capa del **Core Peripheral Access Layer** tiene un código no bloqueante.
- Para cada excepción/interrumpir existen las siguientes definiciones:
  - Una excepción/interrupción utiliza el sufijo **_Handler** (para excepciones) o **_IRQHandler** (para interrupciones).
  - Una excepción /interrupción por default utiliza la palabra **weak.**
  - Un #define en el número de interrupción utiliza el sufijo **_IRQn.**

El CMSIS recomienda las siguientes reglas de escritura:

- LETRA CAPITAL nombre para identificar registros del núcleo, de los registros, e instrucciones de CPU.
- CamelCase nombre para identificar funciones de acceso a los periféricos y las interrupciones (ejemplo *InitLed* para simbolizar inicialización de led)
- PERIPHERAL_ prefijo para identificar las funciones que pertenecen a los periféricos.
- Comentarios utilizan el estilo ANSI C90 (/ * comentario * /) o el estilo de C ++ (// comentario).

- Comentarios de funciones, proporcionan para cada función la siguiente información:
    - Breve resumen en una línea.
    - Explicación detallada de parámetros.
    - Información detallada sobre los valores de retorno.
    - Descripción de la función real.

Ejemplo:

```
/*
* @brief habilita interrupción en el controlador NVIC
* @param IRQn número de interrupción para una interrup. específica
* @return ninguno.
* habilita interrupción en el controlador de interrupciones  NVIC
* Otros seteos de las interrupciones no son afectados.
*/
```

Tipos de datos y tipos de calificadores de IO

El **CortexM HAL** utiliza los tipos estándar del archivo de cabecera estándar ANSI C **<stdint.h>**. Los tipos de **calificadores de IO** se utilizan para especificar el acceso a las variables de los periféricos.

Tabla 4-1: Calificadores de IO

IO Type Qualifier	#define	Descripción
__I	volatile const	Accede sólo a lectura.
__O	volatile	Accede sólo a escritura.
__IO	volatile	Accede a lectura y escritura.

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Número de versión de la CMSIS

El archivo **core_cm3.h** contiene el número de versión de la CMSIS con las siguientes definiciones:

```
#define __CM3_CMSIS_VERSION_MAIN (0x00)/* [31:16] main version */
#define __CM3_CMSIS_VERSION_SUB (0x03)/* [15:0] sub version */
#define  __CM3_CMSIS_VERSION((__CM3_CMSIS_VERSION_MAIN  <<  16)  |
__CM3_CMSIS_VERSION_SUB)
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

El archivo **core_cm0.h** contiene el número de versión de la CMSIS con las siguientes definiciones:

```
#define __CM0_CMSIS_VERSION_MAIN (0x00) /*[31:16] main version*/
#define __CM0_CMSIS_VERSION_SUB (0x00) /*[15:0] sub version*/
#define __CM0_CMSIS_VERSION ((__CM0_CMSIS_VERSION_MAIN << 16) |
__CM0_CMSIS_VERSION_SUB)
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Núcleo de CMSIS Cortex

El archivo **core_cm3.h** contiene el tipo de CMSIS Cortex-M3 con las siguientes definiciones:

```
#define __CORTEX_M (0x03)
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

El archivo **core_cm0.h** contiene el tipo de CMSIS Cortex-M0 con las siguientes definiciones:

```
#define __CORTEX_M (0x00)
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

4.3 ARCHIVOS CMSIS

A continuación se describen los archivos proporcionados por el CMSIS para acceder al hardware y a los periféricos del Cortex-M.

device.h: (Provisto por el fabricante del silicio) Especifica el dispositivo y define los periféricos del dispositivo actual. Al ser incluido en el código de la aplicación. Incluye *core_cm3.h* y *system_<device>.h*

core_cm#.h: (# = 0/3/4) ARM Declaraciones globales de Cortex-M. Define el núcleo del periférico de la CPU del Cortex-M0/M3/M4.

core_cmFunc.h: ARM. Define el acceso a las funciones de los registro del núcleo del Cortex-M.

core_cmInstr.h: ARM. Define las instrucciones del núcleo del Cortex-M.

core_cm3.c: ARM Declaraciones globales de Cortex-M. Provee la ayuda para las funciones de acceso de registro del núcleo.

startup_device: ARM (adaptado por los fabricantes como compilador). Provee el código de inicialización y la tabla de vector de interrupción completa.

system_device: ARM (adaptado por los fabricantes de silicio). Provee la configuración de archivos específicos del dispositivo.

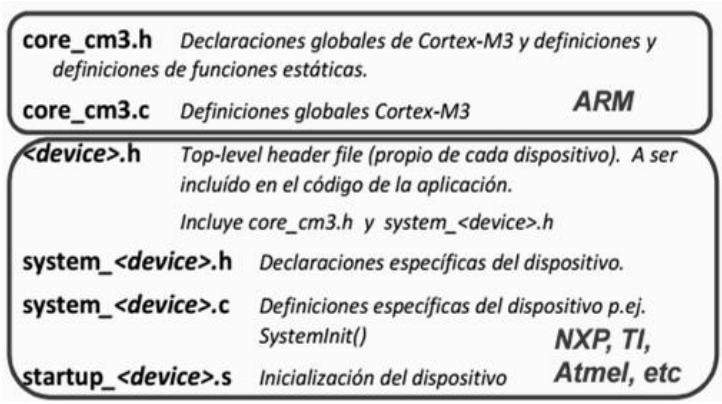


Figura 4-2*: Estructura de archivos del CMSIS*
Fuente *http://www.dsi.fceia.unr.edu.ar/*

device.h

El archivo *device.h* es proporcionado por el proveedor de silicio y es el archivo central que el programador debe incluir, éste archivo contiene:

• *Número de interrupción:* proporciona los números de interrupción (IRQN) para todas las excepciones e interrupciones.

• *Configuración para core_cm0.h / core_cm3.h:* refleja la configuración real del procesador CortexM que es parte del dispositivo real.

• *Device Peripheral Access Layer:* proporciona definiciones para todos los periféricos del dispositivo. Contiene todas las estructuras de datos y el mapeo de direcciones para los periféricos específicos del dispositivo.

• *Access Functions for Peripherals (optional):* proporciona funciones de ayuda adicionales para los periféricos que son útiles para la programación de estos periféricos.

Número de interrupción

Para acceso a la interrupción el archivo **<device>.h** define los números **IRQn** como se indica en la siguiente definición.

```
Typedef enum IRQn
{
     /*** Cortex-M3 Processor Excptions/Interrupt Numbers  ***/
     NonMaskableInt_IRQn = -14,/* !< 2 Non Maskable Interrupt */
     MemoryManagemen_IRQn = -12,/* !< 4 Memory Manag. Interr. */
     BusFault_IRQn       = -11,/* !< 5 Bus Faul Interrupt    */
     UsageFault_IRQn     = -10,  /* !< 6 Usage Fault Interrupt */
     SVCall_IRQn         = -5,/* !< 11 SV Call Interrupt    */
     DebugMonitor_IRQn   = -4, /* !< 12 Debug Monitor Interr.*/
     PendSV_IRQn         = -2, /* ¡< 14 Pend SV Interrupt    */
     Systick_IRQn        = -1 /* ¡< 15 System Tick Interrupt*/
}
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Definición de especificaciones de dispositivos

Las definiciones específicas del dispositivo se encuentran en el archivo de cabecera del dispositivo y se utilizan para las opciones de configuración de CortexM. Algunas opciones de configuración se reflejan en la capa CMSIS utilizando los ajustes #define que se describen a continuación.

Varias características en *core_cm# .h* están configurados por los siguientes *#define* (Tabla 4-2), que deben escribirse antes del comando de preprocesamiento *#include <core_cm# .h>.* Si los define faltan, se utilizaran los valores por defecto.

Tabla 4-2: Configuración de características

#define	Núcleo	Valor por defecto	Descripción
__CM0_REV	M0	0x0000	Nro de versión del núcleo.
__CM3_REV	M3	0x0101 | 0x0200	Nro de versión del núcleo.
__CM4_REV	M4	0x0000	Nro de versión del núcleo.
__NVIC_PRIO_BITS	M0, M3, M4	2 .. 8	2 (M0) 4 (CM3, CM4) Número de bits de prioridad implementadas en el CNTV (dispositivo específico).
__MPU_PRESENT	M0, M3, M4	0 | 1	Informa sobre la presencia de la Unidad de Protección de Memoria.
__FPU_PRESENT	M4	0 | 1	Informa sobre la presencia de la Unidad de Punto Flotante.
__Vendor_SysTickConfig	M0, M3, M4	0 | 1	Cuando este valor esta en 1 la función de configuración del SysTick en el *core_cm3.h* se excluye. En este caso el archivo *device.h* contiene las especificaciones del fabricante del silicio para el Systick.

Capa de acceso al periférico

Cada periférico utiliza un prefijo que consiste en *<abreviatura del dispositivo>* _ *<nombre periférico>* _ para identificar registros de este periférico específico. La intención de esto es evitar conflictos de nombres, causados por la utilización de nombres cortos. Si existe más de un periférico del mismo tipo, los identificadores tienen un sufijo (dígito o letra).

<device abbreviation>_UART_Type: define la presentación de registro genérico para todos los canales UART en un dispositivo.

```
Typedef struct
{
      Union {
      .
      .
} LPC_UART_TypeDef;
```

<device abbreviation>_UART1: es un puntero a una estructura de registro que se refiere a una UART específica

```
#define LPC_UART2 ((LPC_UART_TypeDef *) LPC_UART2_BASE )
#define LPC_UART3 ((LPC_UART_TypeDef *) LPC_UART3_BASE )
```

Requerimientos mínimos

Para acceder a los registros y funciones relacionadas con un periférico, en los archivos *device.h* y *core_cm0.h* / *core_cm3.h* se define como mínimo:

El registro *typedef* para cada periférico que define todos los nombres del registro. Nombres que comienzan con RESERVE se utilizan para introducir espacios en la estructura para ajustar las direcciones de los registros periféricos.

Por ejemplo:

```
typedef struct {
    __IO uint32_t CTRL;   /* SysTick Control and Status    */
    __IO uint32_t LOAD;   /* SysTick Reload Value Register */
    __IO uint32_t VAL;    /* SysTick Current Value Register*/
    __I  uint32_t CALIB;  /* SysTick Calibration Register  */
} SysTick_Type;
```

Dirección base para cada periférico (en caso de que varios periféricos utilicen la misma disposición registro typedef, se definen múltiples direcciones de base). Por ejemplo:

```
#define SysTick_BASE(SCS_BASE+0x0010)  /*SysTick Base Address*/
```

Definición de acceso para cada periférico (en caso de varios periféricos que utilizan la misma disposición registro typedef, existen multiples definiciones de acceso, es decir LPC_UART0, LPC_UART2).

Por ejemplo:

```
/*SysTick access definition */
#define SysTick((SysTick_Type *)SysTick_BASE)
```

Esta definición permite acceder a los periféricos desde el código del usuario con una simple asignación:

```
SysTick-> CTRL = 0;
```

Además el archivo *device.h* puede definir:

- #define constantes que simplifican el acceso a los registros periféricos. Estas constantes definen bits de posiciones u otros patrones específicos que son utilizados para la programación de los registros de los periféricos. Los identificadores utilizados comienzan con <abreviatura del dispositivo> _ y <nombre periférica> _. Se recomienda el uso de mayúsculas para esas constantes.
- Las funciones que desempeñan tareas más complejas con el periférico. Una vez más estas funciones comienzan con <abreviatura del dispositivo> _ y <nombre periférica> _.

core_cm0.h, core_cm0.c / core_cm3.h, core_cm3.c / core_cm4.h, core_cm4.c, core_cm4_simd.h

En los archivos *core_cm0.h* / *core_cm3.h* / *core_cm4.h* de describen la estructura de datos de los periféricos de CortexM0 / M3 / M4 y hace la asignación de direcciones de estas estructuras. También prevé acceso a los registros del núcleo y periféricos a través de funciones eficientes.

En los archivos *core_cm0.c/ core_cm3.c / core_cm4.c* se definen varias funciones que ayudan a acceder a los registros del procesador. En conjunto, estos archivos implementan la capa de acceso a los periféricos (Core Peripheral Access Layer) para Cortex M0 / M3 / M4

El archivo *core_cm4_simd.h* define las instrucciones para el CortexM4 SIMD.

La definición __CMSIS_GENERIC permite usar en librerías genéricas de proyectos, independiente de los dispositivos. Solo se usan tipos y definiciones del núcleo.

core_cmFunc.h and core_cmInstr.h

En el archivo *core_cmFunc.h* se definen funciones de acceso a los registros del núcleo.

En el archivo *core_cmInstr.h* se definen instrucciones del núcleo CortexM.

Estos archivos forman parte de la capa de acceso a los periféricos (Core Peripheral Access Layer) para los procesadores CortexM.

startup_*device*

ARM provee, para cada compilador compatible, un archivo plantilla para *startup_device*. El mismo está adaptado por el proveedor de silicio para incluir vectores de interrupción para todos los manejadores de interrupción específicos del dispositivo. El controlador de interrupción se puede utilizar directamente en el software de aplicación sin ningún requisito para adaptar el archivo *startup_device*.

Los siguientes nombres de excepción se fijan y definen en el inicio de la tabla de vectores de CortexM0:

```
__Vectors   DCD __initial_sp           ; Top of Stack
            DCD Reset_Handler          ; Reset Handler
            DCD NMI_Handler            ; NMI Handler
            DCD HardFault_Handler      ; Hard Fault Handler
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD SVC_Handler            ; SVCall Handler
            DCD 0                      ; Reserved
            DCD 0                      ; Reserved
            DCD PendSV_Handler         ; PendSV Handler
            DCD SysTick_Handler        ; SysTick Handler
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Los siguientes nombres de excepción se fijan y definen en el inicio de la tabla de vectores de CortexM3:

```
__Vectors   DCD __initial_sp           ; Top of Stack
            DCD Reset_Handler          ; Reset Handler
            DCD NMI_Handler            ; NMI Handler
            DCD HardFault_Handler      ; Hard Fault Handler
            DCD MemManage_Handler      ; MPU Fault Handler
            DCD BusFault_Handler       ; Bus Fault Handler
            DCD UsageFault_Handler     ; Usage Fault Handler
            DCD 0                       ; Reserved
            DCD 0                       ; Reserved
            DCD 0                       ; Reserved
            DCD 0                      ; Reserved
            DCD SVC_Handler            ; SVCall Handler
            DCD DebugMon_Handler       ; Debug Monitor Handler
            DCD 0                      ; Reserved
            DCD PendSV_Handler         ; PendSV Handler
            DCD SysTick_Handler        ; SysTick Handler
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

En los siguientes ejemplos, se muestran para interrupciones específicas de un dispositivo

```
    ; External Interrupts
        DCD WWDG_IRQHandler   ; Window Watchdog
        DCD PVD_IRQHandler ; PVD through EXTI Line detect
        DCD TAMPER_IRQHandler ; Tamper
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Interrupciones específicas del dispositivo deben tener una función ficticia que puede ser sobrescrita en el código de usuario. A continuación se muestra un ejemplo de esta función ficticia.

```
    Default_Handler     PROC
                    EXPORT WWDG_IRQHandler [WEAK]
                    EXPORT PVD_IRQHandler [WEAK]
                    EXPORT TAMPER_IRQHandler [WEAK]
                    :
                    WWDG_IRQHandler
                    PVD_IRQHandler
                    TAMPER_IRQHandler
                    :
                    B .
                    ENDP
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

La aplicación del usuario puede definir una función del manejador de interrupciones de la forma en que se muestra a continuación:

```
void WWDG_IRQHandler(void)
{
   :
   :
}
```

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

system_*device*.c

ARM proporciona una plantilla de archivo para *system_device.c*, la misma es adaptada por el proveedor de silicio para que coincida con su dispositivo real. Como requisito mínimo de este archivo, se debe proporcionar una función de la configuración del sistema de dispositivo específico y una variable global que contiene la frecuencia del sistema. Ello configura el dispositivo e inicializa el oscilador (PLL) que es parte del microcontrolador.

El archivo *system_device.c* debe ofrecer como requisito mínimo la función *SystemInit* como se muestra a continuación.

void SystemInit (void) Esta función configura el oscilador (PLL). Para sistemas con velocidad de reloj variable también se actualiza la variable SystemCoreClock. *SystemInit* se llama desde un archivo *startup_device*.

void SystemCoreClockUpdate (void) Actualiza la variable SystemCoreClock y debe ser llamado cada vez que se cambia el reloj durante la ejecución del programa. SystemCoreClockUpdate () evalúa la configuración de registro de reloj y calcula el reloj del núcleo actual.

También forma parte del archivo system_device.c la variable SystemCoreClock que contiene la velocidad de reloj de la CPU

uint32_t SystemCoreClock Contiene el reloj del núcleo del sistema (que es la frecuencia de reloj del sistema suministrado al temporizador SysTick y el reloj de núcleo de procesador). Esta variable puede ser utilizada por la aplicación del usuario para configurar el temporizador SysTick o configurar otros parámetros. También puede ser utilizado por el depurador para consultar la frecuencia del temporizador de depuración o configurar la velocidad de reloj.

SystemCoreClock se inicializa con un valor predefinido correcto.

El compilador debe estar configurado para evitar la remoción de esta variable en caso de que el programa de aplicación no lo está utilizando. Es importante para los sistemas de depuración que la variable este físicamente presente en la memoria de modo que pueda ser examinado para configurar el debugger.

4.4 CAPA DE ACCESO A LOS PERIFÉRICOS

Acceso a los registros del núcleo CortexM

Las funciones que se resumen en la tabla 4-3 se encuentran definidas en los archivos *core_cm0.h* / *core_cm3.h* y proveen acceso a los registros del núcleo CortexM.

Tabla 4-3: Funciones con acceso a los registros del núcleo

Función	Núcleo	Registro	Descripción
void __enable_irq (void)	M0, M3, M4	PRIMASK = 0	Habilitación global de interrupciones.
void __disable_irq (void)	M0, M3, M4	PRIMASK = 1	Inhabilitación global de interrupciones.
uint32_t __get_CONTROL (void)	M0, M3, M4	return CONTROL	Leer el registro de control.
void __set_CONTROL (uint32_t value)	M0, M3, M4	CONTROL = value	Setear el registro de control.
uint32_t __get_IPSR (void)	M0, M3, M4	return IPSR	Leer el valor del registro IPSR.
uint32_t __get_APSR (void)	M0, M3, M4	return APSR	Leer el valor del registro APSR.
uint32_t __get_xPSR (void)	M0, M3, M4	return xPSR	Leer el valor del registro xPSR.
uint32_t __get_PSP (void)	M0, M3, M4	return PSP	Leer el valor del registro PSP (Stack Pointer).
void __set_PSP (uint32_t TopOfProcStack)	M0, M3, M4	PSP = TopOfProcStack	Setea el vlor del Stack Pointer.
uint32_t __get_MSP (void)	M0, M3, M4	return MSP	Leer el MSP (Main Stack Pointer).
void __set_MSP (uint32_t TopOfMainStack)	M0, M3, M4	MSP = TopOfMainStack	Setea el MSP (Main Stack Pointer).
uint32_t __get_PRIMASK (void)	M0, M3, M4	return PRIMASK	Leer el registro PRIMASK.
void __set_PRIMASK (uint32_t value)	M0, M3, M4	PRIMASK = value	Asigna valor a la máscara de prioridades PRIMASK.

void __enable_fault_irq (void)	M3, M4	FAULTMASK = 0	Habilitación global de excepciones e interrupciones por fallas.
void __disable_fault_irq (void)	M3, M4	FAULTMASK = 1	Inhabilitación global de excepciones e interrupciones por fallas
uint32_t __get_BASEPRI (void)	M3, M4	return BASEPRI	Leer el registro BASEPRI (Base Priority).
void __set_BASEPRI (uint32_t value)	M3, M4	BASEPRI = value	Setea el registro BASEPRI.
uint32_t __get_FAULTMASK (void)	M3, M4	return FAULTMASK	Leer el registro FAULTMASK (Fault Mask Register).
void __set_FAULTMASK (uint32_t value)	M3, M4	FAULTMASK = value	Asignar valor al registro FAULTMASK (Fault Mask Register).
uint32_t __get_FPSCR (void)	M4	return FPSCR	Retorno del estado del registro de control de punto flotante.
void __set_FPSCR (uint32_t value)	M4	FPSCR = value	Asigna valor al registro de control de punto flotante.

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

CortexM Instruction Access

Las siguientes funciones que se resumen en la tabla 4-4, están definidas en *core_cm0.h* / *core_cm3.h* y generan instrucciones específicas de CortexM. Las funciones se implementan en los archivos *core_cm0.c* / *core_cm3.c*.

Tabla 4-4: Instrucciones de Coretex-M

Función	Núcleo	CPU Instrucción	Descripción
void __NOP (void)	M0, M3, M4	NOP	No Opera.
void __WFI (void)	M0, M3, M4	WFI	Espera interrupción.
void __WFE (void)	M0, M3, M4	WFE	Espera evento.
void __SEV (void)	M0, M3, M4	SEV	Setea Evento.
void __ISB (void)	M0, M3, M4	ISB	Sincronización del Barrier.
void __DSB (void)	M0, M3, M4	DSB	Sincronización de datos del Barrier.
void __DMB (void)	M0, M3, M4	DMB	Memoria de datos del Barrier
uint32_t __REV (uint32_t value)	M0, M3, M4	REV	Orden de byte inversa en valor entero.
uint32_t __REV16 (uint16_t value)	M0, M3, M4	REV16	Orden de byte inversa en valor corto sin signo.
sint32_t __REVSH (sint16_t value)	M0, M3, M4	REVSH	Orden de byte inversa en valor corto con signo con extensión de signo a entero
uint32_t __RBIT (uint32_t	M3, M4	RBIT	Orden de bit inversa.

value)			
uint8_t __LDREXB (uint8_t *addr)	M3, M4	LDREXB	Cargar byte (8 bits).
uint16_t __LDREXH (uint16_t *addr)	M3, M4	LDREXH	Cargar media palabra (16 bits).
uint32_t __LDREXW (uint32_t *addr)	M3, M4	LDREXW	Cargar palabra (32 bits).
uint8_t __STREXB (uint8_t value, uint8_t *addr)	M3, M4	STREXB	Almacena byte (8bits).
uint16_t __STREXH (uint16_t value, uint16_t *addr)	M3, M4	STREXH	Almacena media palabra (16 bits).
uint32_t __STREXW (uint32_t value, uint32_t *addr)	M3, M4	STREXW	Almacena palabra (32 bits).
void __CLREX (void)	M3, M4	CLREX	Quita el bloqueo exclusive creado por __LDREXB, __LDREXH, o __LDREXW
void __SSAT (void)	M3, M4	SSAT	Saturar un valor con signo.
void __USAT (void)	M3, M4	USAT	Saturar un valor sin signo.

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

NVIC Access Functions

El CMSIS provee acceso al NVIC a través de registros y funciones que facilitan su seteo. El CMSIS HAL usa los números de IRQ (IRQn) para identificar las interrupciones. El primer dispositivo que interrumpe tiene el valor 0. Los valores negativos de IRQn son ustilizados por las excepciones del núcleo del procesador.

Para los números de IRQ de las excepciones del núcleo, el archivo *device.h* provee los siguientes valores (Tabla 4-5).

Tabla 4-5: Excepciones del núcleo

Core Exception enum Value	Núcleo	IRQn	Descripción
NonMaskableInt_IRQn	M0, M3, M4	14	Interrupción no enmascarable.
HardFault_IRQn	M0, M3, M4	13	Interrucción por falla de Hard.
MemoryManagement_IRQn	M3, M4	12	Interrupción por gestión de memoria.
BusFault_IRQn	M3, M4	11	Interrupción por falla de bus.
UsageFault_IRQn	M3, M4	10	Interrupción por falla de uso.
SVCall_IRQn	M0, M3, M4	5	Interrupción por SV Call.
DebugMonitor_IRQn	M3, M4	4	Interrupción por depuración del monitor.
PendSV_IRQn	M0, M3, M4	2	Interrupción por Pend SV.
SysTick_IRQn	M0, M3, M4	1	Interrupción por System Tick.

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Las siguientes funciones simplifican la configuración de la NVIC (Tabla 4-6).

Tabla 4-6: Funciones del NVIC

Funcion	Núcleo	Parámetro	Descripción
void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)	M3, M4	Grupo de prioridad	Valor establecido por el grupo de prioridad (Grupos.Subgrupos).
uint32_t NVIC_GetPriorityGrouping (void)	M3, M4	No requiere	Captura el grupo de prioridad (Grupos.Subgrupos).
void NVIC_EnableIRQ (IRQn_Type IRQn)	M0, M3, M4	Número IRQ	Habilita IRQn.
void NVIC_DisableIRQ (IRQn_Type IRQn)	M0, M3, M4	Número IRQ	Inhablita IRQn.
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)	M0, M3, M4	Número IRQ	Retorna 1 si IRQn esta pendiente sino 0.
void NVIC_SetPendingIRQ (IRQn_Type IRQn)	M0, M3, M4	Número IRQ	Setea a pendiente IRQn.
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)	M0, M3, M4	Número IRQ	Borra el estado pendiente de IRQn.
uint32_t NVIC_GetActive (IRQn_Type IRQn)	M3, M4	Número IRQ	Retorna 1 si IRQn esta activado sino 0.
void NVIC_SetPriority (IRQn_Type IRQn,uint32_t priority)	M0, M3, M4	Número IRQ, Prioridad	Setea la prioridad de IRQn (not threadsafe for CortexM0).
uint32_t NVIC_GetPriority (IRQn_Type IRQn)	M0, M3, M4	Número IRQ	Captura la prioridad de IRQn.
uint32_t NVIC_EncodePriority (uint32_t PriorityGroup,uint32_t PreemptPriority,uint32_t SubPriority)	M3, M4	Número IRQ, Grupo de prioridad, Prioridad preventiva, Sub-prioridad	Prioridad de codificación para el grupo dado, preventivo y sub prioridad.
void NVIC_DecodePriority (uint32_t Priority,uint32_t PriorityGroup,uint32_t* pPreemptPriority,uint32_t* pSubPriority)	M3, M4	Prioridad, Grupo de prioridad, Puntero al Preempt.Prioridad, Puntero a la subprioridad	Decodificación dada al grupo de prioridad, preventivo y subprioridad.
void NVIC_SystemReset (void)	M0, M3, M4	No requiere	Reset el sistema.

Fuente: http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm

Función de configuración del SysTick

El SYSTCK es un temporizador de 24 bits de cuenta descendente, que produce una interrupción cuando el registro interno llega a cero desde el valor de recarga inicial [Capella Hernández, 2013].

Su mayor ventaja es su sencillez, dado que no hay que configurar nada excepto el valor de recarga. El inconveniente es que al no poder configurar preescalas ni otros parámetros las temporizaciones que se pueden realizar son bastante básicas.

Para configurar el SysTick deben seguirse los siguientes pasos:

– Desactivar el contador. ENABLE=0.

– Cargar el valor de RELOAD.

– Escribir cualquier valor en la cuenta para que se ponga a 0.

– Configurar los registros de control y estado, incluyendo la activación.

Esta funcionalidad es proporcionada por el CMSIS, que configura y pone en marcha el temporizador SysTick mediante la función.

```
uint32_t SysTickConfig (uint32_t ticks)
```

Concretamente, esta función CMSIS realiza las siguientes Acciónes:

– Configura el registro de recarga del SysTick con el valor pasado como parámetro.

– Configura la prioridad de la interrupción del SysTick al valor más bajo (IRQ priority).

– Configura la fuente de reloj para el contador del SysTick a HCLK - Core Clock Source.

– Habilita la interrupción del timer.

– Inicia el contador.

Para ajustar el tiempo base del SysTick se utiliza la siguiente fórmula:

Valor de recarga = SysTick Counter Clock (Hz) x Temporización deseada (s)

El SysTick Counter Clock es el reloj que le llega al temporizador SysTick. El valor de recarga es el parámetro que le pasamos a la

función SysTick_Config(), que no debe exceder 0xFFFFFF. Si fuera necesario, se puede cambiar la prioridad del temporizador usando la función NVIC_SetPriority(SysTick_IRQn,...) después de invocar la función SysTick_Config().

Instrumented Trace Macrocell (ITM)

El Cortex-M3 incorpora la ITM que proporciona capacidades de rastreo para el sistema microcontrolador. El ITM tiene 32 canales de comunicación que son capaces de transmitir valores de 8/16/32 bits; dos canales de comunicación ITM son utilizados por CMSIS a la salida de la siguiente información:

- ITM Canal 0: implementa la función ITM_SendChar que se puede utilizar para la salida printfstyle través de la interfaz de depuración.
- ITM Canal 31: se reserva para el kernel RTOS.

CMSIS ofrece siguientes funciones de depuración:

- ITM_SendChar (ITM utiliza el canal 0)
- ITM Recibe Char (utiliza variable global)
- ITM CheckChar (utiliza variable global)

```
uint32_t ITM_SendChar(uint32_t chr) /* Parámetro: character a enviar */
```

Esta función transmite un carácter por el ITM channel 0. Retorna cuando ningún depurador ha reservado la salida. Se bloquea cuando un depurador está conectado.

Parámetros:

[in] *chr* Carácter a transmitir

Retorna:

Carácter transmitido.

```
uint32_t ITM_ReceiveChar(void)
```

Esta función recibe un carácter a través de la variable externa ITM Rx_Buffer. Retorna cuando ningún depurador ha reservado la salida. Se bloquea cuando un depurador está conectado.

Retorna:

Carácter = 1 → Ningún character recibido

```
uint32_t ITM_CheckChar(void)
```

Esta función lee una variable externa ITM Rx_Buffer, y comprueba si un carácter está disponible o no.

Retorna:

0 No hay carácter disponible

1 Carácter disponible

CAPITULO 5: PERIFÉRICOS

5.1 LPC17XX CORTEX-M3 PERIFÉRICOS

5.1.a Nested Vectored Interrupt Controller(NVIC)

El controlador de interrupción NVIC es una parte integral del Cortex-M3. La integración en la CPU permite una baja latencia por interrupción y procesamiento eficiente de las interrupciones *late arriving* (de llegada tardía).

Cada periférico de un dispositivo puede tener una o más líneas de interrupción en el NVIC. Cada línea puede representar más de una fuente de interrupción. Números de excepción se refieren a donde las entradas se almacenan en la tabla de vectores de excepción. Los números de interrupción se utilizan en algunos otros contextos, tales como interrupciones de software. Además, la NVIC se encarga de la interrupción no enmascarable (NMI). Para que NMI opere a partir de una señal externa, la función NMI debe estar conectado al pin del dispositivo relacionado (P2.10 / EINT0n / NMI). Una vez conectado, un 1 lógico en el pin hará que la NMI sea procesada.

La tabla 5-1, resume los registros de la NVIC tal como se aplican en el LPC17xx.

El software para el NVIC utiliza las instrucciones CPSIE I y CPSID I para activar y desactivar las interrupciones. El CMSIS presenta las siguientes funciones intrínsecas de estas instrucciones:

void __disable_irq (void) // Inhabilitar Interrupciones

void __enable_irq (void) // Habilitar interrupciones

Tabla 5-1: Registros utilizados por el NVIC

Nombre	Descripción	Acceso	Reset	Dirección
ISERO hasta ISER1	Registros de interrupción Set-Enable. Estos 2 registros permiten activar interrupciones y posteriormente leer la interrupción para especificar la función del periférico.	RW	0	ISER0 - 0xE000 E100 ISER1 - 0xE000 E104
ICER0 hasta ICER1	Registros de interrupción Clean-Enable. Estos 2 registros permiten la inhabilitar las interrupciones y leer la interrupción para especificar la función del periférico.	RW	0	ICER0 - 0xE000 E180 ICER1 - 0xE000 E184
ISPR0 hasta ISPR1	Registros de interrupción Set - Pending. Estos 2 registros permiten cambiar el estado de las interrupciones a pendiente y leer la interrupción para especificar la función del periférico.	RW	0	ISPR0 - 0xE000 E200 ISPR1 - 0xE000 E204
ICPR0 hasta ICPR1	Registros de interrupción Clear - Pending. Estos 2 registros permiten cambiar el estado de interrupción a pendiente y no volver a leer el estado pendiente de l interrupción.	RW	0	ICPR0 - 0xE000 E280 ICPR1 - 0xE000 E284
IABRO hasta IABR1	Registros de interrumpción Active Bit. Estos 2 registros permiten siempre la lectura del estado activo de interrupción para las funciones específicas de los periféricos.	RO	0	IABR0 - 0xE000 E300 IABR1 - 0xE000 E304
IPR0 hasta IPR8	Registros de interrupción Priority. Estos 9 registros permiten la asignación de una prioridad a cada interrupción. Cada registro contiene la prioridad en el 5-bits para 4 interrupciones.	RW	0	IPR0 - 0xE000 E400 IPR1 - 0xE000 E404 IPR2 - 0xE000 E408 IPR3 - 0xE000 E40C IPR4 - 0xE000 E410 IPR5 - 0xE000 E414 IPR6 - 0xE000 E418 IPR7 - 0xE000 E41C IPR8 - 0xE000 E420
STIR	Registro de interrupción de software de activación. Este registro permite al software generar una interrupción.	WO	0	STIR - 0xE000 EF00

Fuente: Tabla 51- LPC17xx manual del usuario

Además, el CMSIS proporciona una serie de funciones para el control de NVIC, incluyendo:

- ✓ void **NVIC_SetPriorityGrouping**(uint32_t priority_grouping) // fija la prioridad del grupo
- ✓ void **NVIC_Enable IRQ**(IRQn_t IRQn) // habilita una IRQ
- ✓ void **NVIC_Disable IRQ**(IRQn_t IRQn) // inhabilita una IRQ
- ✓ uint32_t **NVIC_GetPendingIRQ**(IRQn_t IRQn) // retorna verdadero si la IRQ esta pendiente
- ✓ void **NVIC_SetPendingIRQ**(IRQn_t IRQn) // estable una IRQ como pendiente
- ✓ void **NVIC_ClearPendingIRQ** (IRQn_t IRQn) // borra el estado de IRQ pendiente
- ✓ uint32_t **NVIC_GetActive**(IRQn_t IRQn) // retorna el número de IRQ de la interrupcion activa
- ✓ void **NVIC_SetPriority**(IRQn_t IRQn, uint32_t priority) // establecer la prioridad de una IRQ
- ✓ uint32_t **NVIC_GetPriority**(IRQn_t IRQn) // leer la prioridad de una IRQ
- ✓ void **NVIC_SystemReset**(void) // reset del sistema

5.1.b. Puertos Input/Output de propósitos generales (GPIO)

Los GPIOs se configuran con los siguientes registros:

1. Power: siempre activado.
2. Pins: pines GPIO y sus modos.

El registro PINSEL controla las funciones de los pines del dipositivo como se muestra abajo. Los pares de bits en estos registros corresponden a los pines específicos del dispositivo.

Tabla 5-2: Selección de función en los pines

Valores desde PINSEL0 a PINSEL9	Función	Reset
00	Función por defecto (típicamente GPIO).	00
01	Primera función alternativa.	
10	Segunda función alternativa.	
11	Tercera función alternativa.	

Fuente: Tabla 75 – LPC17xx manual del usuario

3. Wake-up: se pueden utilizar los puertos GPIO 0 y 2 para despertar, si es necesario.

4. Interrupts: Habilitar interrupciones GPIO en IO0 / 2IntEnR o IO0 / 2IntEnF. Las interrupciones se habilitan en el NVIC utilizando apropiadamente la interrupción Set Enable Register.

5.1.b.1 Características

- Puertos E / S Digital

• Funciones de aceleración GPIO:

- Los registros GPIO se encuentran en un bus de periféricos AHB para una rápida I/O.

- Los registros de máscara permiten el tratamiento de conjuntos de bits de puertos como grupo, dejando a otros bits sin cambios.

- Todos los registros GPIO son direccionable como bytes, media palabra, y palabra completa.

- El valor del puerto completo se puede escribir en una instrucción.

- Los registros GPIO son accesibles por el GPDMA.

• Seteo o borrado del nivel de bits, en una sola instrucción se puede setear o borrar cualquier número de bits en un puerto.

• Todos los GPIO soportan el manejo de bits con bit-banding.

• Registros GPIO son accesibles por el controlador GPDMA para permitir datos desde la DMA y hacia los GPIOs, sincronizado a cualquier solicitud de DMA.

• Control de Dirección de los bits de puerto individuales.

• Todas las E / S se fijan como entradas por defecto, con pull-up después de un reset.

- Generación de Interrupciones de puertos digitales

• Los puertos 0 y 2 pueden proporcionar una única interrupción para cualquier combinación de pines del puerto.

• Cada pin del puerto se puede programar para generar una interrupción por flanco ascendente, descendente, o ambos.

• La detección por flancos es asincrónica, por lo que puede funcionar cuando los relojes no están presentes, como durante en modo power-down.

• Cada interrupción habilitada contribuye a una señal de activación que se puede utilizar para llevar al modo power-down.

• Los registros proporcionan una visión de software pendiente de interrupciones por flanco ascendente y descendente, y en general a la espera pendiente de interrupciones de GPIO.

• Las interrupciones GPIO0 y GPIO2 comparten la misma posición en la NVIC con interrupción externa 3.

5.1.b.2 Aplicaciones

- Uso general de E / S
- Manejo de LED u otros indicadores
- Control de habilitación de dispositivos
- Sensado de entradas digitales, detección de bordes
- Salida en modo Power-down.

5.1.b.3 Registros

El LPC17xx implementa cinco porciones de 32 bits para puerto de propósito general de I / O.

Los registros de la Tabla 5-3 representan las características GPIO mejoradas disponibles en todos los puertos del GPIO. Estos registros se encuentran en un bus AHB para una rápida lectura y escritura.

A todos ellos se puede acceder como byte, media palabra o palabra completa. Un registro de máscara permite el acceso a un grupo de bits en un único puerto GPIO independientemente de otros bits en el mismo puerto.

Tabla 5-3: Registos para manejo de GPIO

Nombre	Descripcion	Acceso	Reset	PORTn registro Nombre&direcc.
FIODIR	Este registro controla individualmente la dirección de cada pin del puerto.	R/W	0	FIO0DIR - 0x2009 C000 FIO1DIR - 0x2009 C020 FIO2DIR - 0x2009 C040 FIO3DIR - 0x2009 C060 FIO4DIR - 0x2009 C080
FIOMASK	Escribir, fijar, borrar, y leer un puerto (a través de escrituras en FIOPIN, FIOSET, y FIOCLR) sólo los bits habilitados por ceros en este registro.	R/W	0	FIO0MASK - 0x2009 C010 FIO1MASK - 0x2009 C030 FIO2MASK - 0x2009 C050 FIO3MASK - 0x2009 C070 FIO4MASK - 0x2009 C090
FIOPIN	El estado actual de los pines del puerto digital se pueden leer desde este registro, independientemente de la dirección o de la función alternativa (siempre y cuando los pines no se configuren como una entrada del ADC). El valor leído se enmascara haciendo una AND con FIOMASK invertida. Escribiendo en los lugares habilitados por ceros en FIOMASK.	R/W	0	FIO0PIN - 0x2009 C014 FIO1PIN - 0x2009 C034 FIO2PIN - 0x2009 C054 FIO3PIN - 0x2009 C074 FIO4PIN - 0x2009 C094
FIOSET	Este registro controla el estado de los pines de salida. Escribir unos (1) produce altos en los pines del puerto correspondiente. Escribir ceros (0) no tiene ningún efecto. Leer este registro devuelve el contenido actual de la salida del puerto. Sólo los bits habilitados por 0 en FIOMASK pueden ser alterados.	R/W	0	FIO0SET - 0x2009 C018 FIO1SET - 0x2009 C038 FIO2SET - 0x2009 C058 FIO3SET - 0x2009 C078 FIO4SET - 0x2009 C098
FIOCLR	Este registro controla el estado de los pines de salida. Escribir unos (1) produce bajos en los pines del puerto correspondientes. Escribir ceros (0) no tiene ningún efecto. Sólo los bits habilitados por 0 en FIOMASK pueden ser alterados.	WO	0	FIO1CLR - 0x2009 C03C FIO2CLR - 0x2009 C05C FIO3CLR - 0x2009 C07C FIO4CLR - 0x2009 C09C

Fuente: Tabla 101 del LPC17xx User manual

La dirección del puerto en el registro FIOxDIR: se utiliza para controlar la dirección de las señales en los pines cuando se configuran como pines del puerto GPIO. La dirección del pin se debe ajustar a su funcionalidad.

Tabla 5-4: Fast GPIO Dirección puerto registro FIO0DIR a FIO4DIR - direcciones 0x2009 C000 a 0x2009 C080) Descripción de bits

Bit	Simbolo	Valor	Descripcion	Reset
31:0	FIO0DIR		Bit de control de la dirección del PORTx. Bit 0 en FIOxDIR controla pin Px.0, bit 31 controla pin Px.31.	0x0
	FIO1DIR			
	FIO2DIR			
	FIO3DIR	0	Pin definido como entrada.	
	FIO4DIR	1	Pin definido como salida.	

Fuente: Tabla 103 del LPC17xx User manual

La salida de puerto en el registro FIOxSET: se utiliza para producir una salida de nivel ALTO en los pines del puerto configurados como salida (set bits). Escribir 1 produce un nivel ALTO en los pines del puerto correspondientes. Escribiendo 0 no tiene efecto. Si cualquier pin se configura como una entrada o una función secundaria, escribiendo 1 en el bit correspondiente en el FIOxSET no tiene ningún efecto.

Leyendo el registro FIOxSET devuelve el valor de este registro, según lo determinado por las escrituras anteriores a FIOxSET y FIOxCLR. Este valor no refleja el efecto de cualquier influencia exterior en los pines de E / S.

Tabla 5-5: Fast GPIO port output Set register (FIO0SET to FIO4SET - addresses 0x2009 C018 to 0x2009 C098) Descripción de bits

Bit	Simbolo	Valor	Descripcion	Reset
31:0	FIO0SET		Establece el valor del bit de salida. Bit 0 en FIOxSET controla pin Px.0, bit 31 controla pin Px.31.	0x0
	FIO1SET			
	FIO2SET			
	FIO3SET	0	Pin no se modifica.	
	FIO4SET	1	Pin seteado en valor ALTO.	

Fuente: Tabla 105 del LPC17xx User manual

La salida de puerto en el registro FIOxCLR: se utiliza para producir una salida de nivel BAJO en pines del puerto configurados como salida (clean bits). Escribir un 1 produce un nivel BAJO en el pin de puerto correspondiente y borra el bit correspondiente en el registro FIOxSET. Escribiendo un 0 no tiene efecto. Si cualquier pin se configura como una entrada o una función secundaria, escribiendo FIOxCLR no tiene ningún efecto.

El acceso a un pin del puerto a través del registro FIOxCLR está condicionada por el bit correspondiente del registro FIOxMASK.

Tabla 5-6: Fast GPIO port output Clear register (FIO0CLR to FIO4CLR- addresses 0x2009 C01C to 0x2009 C09C) Descripción de bits

Bit	Simbolo	Valor	Descripcion	Reset
31:0	FIO0CLR		Establece el valor del bit de salida en cero. Bit 0 en FIOxSET controla pin Px.0, bit 31 controla pi. Px.31	0x0
	FIO1CLR			
	FIO2CLR			
	FIO3CLR	0	Pin no se modifica.	
	FIO4CLR	1	Pin seteado en valor BAJO.	

Fuente: Tabla 107 del LPC17xx User manual

El valor del puerto del pin del registro FIOxPIN: proporciona el valor de los pines del puerto que están configurados para realizar sólo las funciones digitales. El registro dará el valor lógico del pin, independientemente de si el pin está configurado como entrada o salida, o como GPIO o una función digital de alternativa. Por ejemplo, un pin GPIO configurado como entrada o salida, entrada UART, salida de PWM funciona como seleccionable. Cualquier configuración de ese pin permitirá leer su estado lógico actual desde el registro FIOxPIN correspondiente.

Si un pin tiene una función analógica como una de sus opciones, el estado pin no se puede leer. En ese caso, el valor leer pin en el registro FIOxPIN no es válido.

El acceso a un pin del puerto a través del registro FIOxPIN está condicionada por el bit correspondiente del registro FIOxMASK.

Tabla 5-7: Fast GPIO port Pin value byte and half-word accessible register description

Nombre Registro	Descripcion	Bits & acceso	Reset	Dir & nombre
FIOxPIN0	Bit 0 en registro FIOxPIN0 corresponde al pin Px.0 ... bit 7 al pin Px.7	8(byte) R/W	0x00	FIO0PIN0 - 0x2009 C014 FIO1PIN0 - 0x2009 C034 FIO2PIN0 - 0x2009 C054 FIO3PIN0 - 0x2009 C074 FIO4PIN0 - 0x2009 C094
FIOxPIN1	Bit 0 en registro FIOxPIN1 corresponde al pin Px.8 … bit 7 al pin Px.15.	8(byte) R/W	0x00	FIO0PIN1 - 0x2009 C015 FIO1PIN1 - 0x2009 C035 FIO2PIN1 - 0x2009 C055 FIO3PIN1 - 0x2009 C075 FIO4PIN1 - 0x2009 C095
FIOxPIN2	Bit 0 en registro FIOxPIN2 corresponde al pin Px.16 ... bit 7 al pin Px.23	8(byte) R/W	0x00	FIO0PIN2 - 0x2009 C016 FIO1PIN2 - 0x2009 C036 FIO2PIN2 - 0x2009 C056 FIO3PIN2 - 0x2009 C076 FIO4PIN2 - 0x2009 C096
FIOxPIN3	Bit 0 en registro FIOxPIN3 corresponde al pin Px.24 ... bit 7 al pin Px.31	8(byte) R/W	0x00	FIO0PIN3 - 0x2009 C017 FIO1PIN3 - 0x2009 C037 FIO2PIN3 - 0x2009 C057 FIO3PIN3 - 0x2009 C077 FIO4PIN3 - 0x2009 C097
FIOxPINL	Bit 0 en el registro FIOxPINL corresponde al pin Px.0 … bit 15 al pin Px.15.	16(half -word) R/W	0x0000	FIO0PINL - 0x2009 C014 FIO1PINL - 0x2009 C034 FIO2PINL - 0x2009 C054 FIO3PINL - 0x2009 C074 FIO4PINL - 0x2009 C094
FIOxPINU	Bit 0 en el registro FIOxPINU corresponde al pin Px.16 … bit 15 al Px.31	16(half -word) R/W	0x0000	FIO0PINU - 0x2009 C016 FIO1PINU - 0x2009 C036 FIO2PINU - 0x2009 C056 FIO3PINU - 0x2009 C076 FIO4PINU - 0x2009 C096

Fuente: Tabla 110 del LPC17xx User manual

La máscara del puerto de registro FIOxMASK: se utiliza para seleccionar los pines del puerto que no se verán afectados por accesos de escritura al registro FIOxPIN, FIOxSET o FIOxCLR. El registro de máscara también filtra el contenido del puerto cuando se lee el registro FIOxPIN.

Un cero en el bit de este registro permite un acceso al correspondiente pin físico a través de un acceso de lectura o escritura. Si un bit en este registro es un uno, el pin correspondiente

no se puede cambiar como acceso de escritura y si se lee, no se reflejará en el registro actualizado FIOxPIN.

Ejemplo: Se lee el estado del P0 [22] para invertir, utilizando el registro FIOPIN.

LPC_GPIO0->FIOPIN ^= (1<<22);

Tabla 5-8: Fast GPIO port Mask register (FIO0MASK to FIO4MASK - addresses 0x2009 C010 to 0x2009 C090) Descripción de bits

Bit	Simbolo	Valor	Descripcion	Reset
31:0	FIO0MASK		Control de acceso a un pin.	0x0
	FIO1MASK			
	FIO2MASK			
	FIO3MASK	0	Pin controlado se ve afectada por las escrituras en registro FIOxSET, FIOxCLR, y FIOxPIN del puerto (s). Estado actual del pin se puede leer desde el registro FIOxPIN.	
	FIO4MASK	1	Pin controlado no se ve afectada por las escrituras en FIOxSET, FIOxCLR y FIOxPIN registro (s) del puerto. Cuando se lee el registro FIOxPIN, este bit no se actualizará con el estado del pin físico.	

Fuente: Tabla 111 del LPC17xx User manual

5.1.b.4 Funciones del CMSIS

void **FIO_ByteClearValue**(uint8_t portNum, uint8_t byteNum, uint8_t bitValue)

Acción: Borra los bits del puerto FIO en byte accesibles.

Parámetros:

[in] *portNum:* Número de puerto, su valor puede estar entre 0 y 4

[in] *byteNum*: Número de Byte, su valor debe estar en el rango entre 0 y 3.

[in] *bitValue:* Valor que contiene todos los bits a borrar, en el rango de 0 a 0xFF.

uint8_t **FIO_ByteReadValue** (uint8_t portNum, uint8_t byteNum)

Acción: Leer el estado actual en el puerto y pin que tiene la dirección de entrada del GPIO para el byte accesible.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *byteNum*: Número de Byte, su valor debe estar en el rango entre 0 y 3.

Devuelve: El valor actual del puerto y pin de la FIO especificando los bytes.

void **FIO_ByteSetDir** (uint8_t portNum, uint8_t byteNum, uint8_t bitValue, uint8_t dir)

Acción: Setea la dirección del puerto FIO en bytes accessibles.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *byteNum*: Número de byte, su valor debe estar en el rango entre 0 y 3.

[in] *bitValue*: Valor que contiene todos los bits a borrar, en el rango de 0 a 0xFF.

[in] *dir*: Valor de la dirección, puede ser: 0: Entrada; 1: Salida.

void **FIO_ByteSetMask** (uint8_t portNum, uint8_t byteNum, uint8_t bitValue, uint8_t maskValue)

Acción: Valor de la máscara de los bits en el puerto FIO en bytes accesible.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *byteNum*: Número de Byte, su valor debe estar en el rango entre 0 y 3.

[in] *bitValue*: Valor que contiene todos los bits a borrar, en el rango de 0 a 0xFF.

[in] *maskValue*: Valor de la máscara, contiene el valor para cada bit: 0: No enmascara; 1: Enmascara.

void **FIO_ByteSetValue** (uint8_t portNum, uint8_t byteNum, uint8_t bitValue)

Acción: Setea los bits del puerto FIO en bytes accessibles.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *byteNum*: Número de Byte, su valor debe estar en el rango entre 0 y 3.

[in] *bitValue*: Valor que contiene todos los bits a borrar, en el rango de 0 a 0xFF.

void **FIO_HalfWordClearValue** (uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue)

Acción: Borra los bits del puerto FIO para media palabra accesible.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *halfwordNum*: Número de HalfWord, debe ser 0 (inferior) o 1(superior).

[in] *bitValue*: Valor que contiene todos los bits a borrar , en el rango de 0 a 0xFFFF.

uint16_t **FIO_HalfWordReadValue**(uint8_t portNum, uint8_t halfwordNum)

Acción: Lee el estado actual del puerto y pin que tiene la dirección de entrada del GPIO accesible a la media palabra.

Parámetros:

[in] *portNum*: Número del puerto, su valor puede estar entre 0 y 4

[in] *halfwordNum*: Número de HalfWord, debe ser 0 (inferior) o 1(superior).

Devuelve: El valor actual del puerto y pin de la FIO especificando la media palabra.

void **FIO_HalfWordSetDir** (uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue, uint8_t dir)

Acción: Setea la dirección del puertoFIO para la media palabra accesible.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *halfwordNum*: Número de HalfWord, debe ser 0 (inferior) o 1(superior).

[in] *bitValue*: Valor que contiene todos los bits a borrar , en el rango de 0 a 0xFFFF.

[in] *dir*: Valor de la dirección, puede ser: 0: Entrada; 1: Salida.

void **FIO_HalfWordSetMask** (uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue, uint8_t maskValue)

Acción: Valor de la máscara en el puerto FIO accesible para media palabra.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *halfwordNum*: Número de HalfWord, debe ser 0 (inferior) o 1(superior).

[in] *bitValue*: Valor que contiene todos los bits a borrar , en el rango de 0 a 0xFFFF.

[in] *maskValue*: Valor de la máscara, contiene el valor para cada bit: 0: No enmascara.; 1: Enmascara.

void **FIO_HalfWordSetValue** (uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue)

Acción: Setea los bits del puero FIO para media palabra accesible.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *halfwordNum*: Número de HalfWord, debe ser 0 (inferior) o 1(superior).

[in] *bitValue*: Valor que contiene todos los bits a borrar , en el rango de 0 a 0xFFFF.

void **FIO_SetMask** (uint8_t portNum, uint32_t bitValue, uint8_t maskValue)

Acción: Setea el valor de la máscara para los bits en el puerto FIO.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *bitValue*: Valor que contiene todos los bits a borrar , en el rango de 0 a 0xFFFFFFFF.

[in] *maskValue*: Valor de la máscara, contiene el valor para cada bit: 0: No enmascara.; 1: Enmascara.

void **GPIO_ClearInt** (uint8_t portNum, uint32_t bitValue)

Acción: Borra la interrupción de los GPIO (es usado para los puertos 0 y 2).

Parámetros:

[in] *portNum*: Número de puerto, su valor puede ser 0 o 2.

[in] *bitValue*: Contiene todos lo bits del GPIO, en el rango de 0 a 0xFFFFFFFF.

void **GPIO_ClearValue** (uint8_t portNum, uint32_t bitValue)

Acción: Borra el valor de los bits que tienen dirección de salida en el puerto GPIO .

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *bitValue*: Contiene el valor de todos los bits a borrar en los GPIO, en el rango de 0 a 0xFFFFFFFF. Ejemplo: valor 0x5 borra el bit 0 y el bit 1.

FunctionalState **GPIO_GetIntStatus** (uint8_t portNum, uint32_t pinNum, uint8_t edgeState)

Acción: Captura el estado de la interrupción del GPIO (usado para los puerto 0 y 2).

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 o 2.

[in] *pinNum*: Número de pin, puede ser: 0..30(para el puerto 0) y 0..13 (para el puerto 2).

[in] *edgeState*: Estado de flanco, puede ser: 0: Flanco ascendente; 1: Flanco descendente.

Devuelve: Un Booleano que puedes ser:

ENABLE: La Interrupción se ha generado debido a un flanco ascendente en P0.0.

DISABLE: Un flanco ascendente no ha sido detectado en P0.0.

void **GPIO_IntCmd** (uint8_t portNum, uint32_t bitValue, uint8_t edgeState)

Acción: Habilita la interrupción de los GPIO (usado para los puertos 0 y 2).

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 o 2.

[in] *bitValue*: Valor que contiene todos los bits de los GPIO para habilitarlos , en el rango de 0 a 0xFFFFFFFF .

[in] *edgeState*: Estado de flanco, puede ser: 0: Flanco ascendente; 1: Flanco descendente.

uint32_t **GPIO_ReadValue** (uint8_t portNum)

Acción: Lee el estado actual del puerto y pin que tiene la dirección de entrada del GPIO.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

Devuelve: El valor del puerto del GPIO.

void **GPIO_SetDir** (uint8_t portNum, uint32_t bitValue, uint8_t dir)

Acción: Setea la dirección del puerto GPIO.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *bitValue:* Valor que contiene todos los bits para ajustar la dirección , en el rango de 0 a 0xFFFFFFFF .

Ejemplo: valor 0x5 para establecer la dirección de bit 0 y el bit 1.

[in] *dir*: Valor de la dirección, puede ser: 0: Entrada; 1: Salida.

void GPIO_SetValue (uint8_t portNum, uint32_t bitValue)

Acción: Establece el valor para los bits que tienen dirección de salida en el puerto GPIO.

Parámetros:

[in] *portNum*: Número de puerto, su valor puede estar entre 0 y 4.

[in] *bitValue*: Valor que contiene todos los bits en GPIO para establecer , en el rango de 0 a 0xFFFFFFFF.

Ejemplo: valor 0x5 para establecer el bit 0 y el bit 1.

5.1.c. Interfaces serie:

El LPC17xx cuenta entre sus perifericos con una serie de interfaces para las comunicaciones serie, como ser:

- Puerto Ethernet con interfaz RMII y controlador DMA dedicado.
- Puerto USB 2.0, puede operar en modo Host o en modo OTG y un controlador DMA dedicado.
- Cuatro UART con generación fraccional de velocidad de transmisión, FIFO interna, IrDA, y DMA. Uno de estos puertos UART tiene un control de módem y soporte para RS-485 / EIA-485.
- Dos canales de controlador CAN.
- Dos controladores de SSP con FIFO y capacidades multi-protocolo. Las interfaces de SSP se puede utilizar con el controlador GPDMA.
- Controlador sincrónico SPI, comunicación full duplex y longitud de datos programable.
- Tres de interfaces I2C, una con una salida de open-drain apoyo a la especificación I2C completa y modo Fast plus con velocidades de datos de 1 Mbit / s.
- I2S (Inter-IC Sound) interfaz para entrada/salida de audio digital.

En los siguiestes apartados describiremos en forma resumida algunos de estos periféricos y muy sinteticamente las funciones del CMSIS que facilitan su uso.

5.1.c.1 UART (Transmisor Receptor Asíncrono Universal)

El UART es un bus asíncrono para el intercambio de datos. Un UART toma bytes de datos en paralelo y los transmite bit a bit, en forma secuencial. En el destino, un segundo UART vuelve a ensamblar los bits en bytes completos. Cada UART contiene un registro de desplazamiento para la conversión entre las formas de transmisión de serie y paralelo.

Caracteríticas de la UART0/2/3

Estos puertos tienen las siguientes caracteristicas:

• Tamaños de datos de 5, 6, 7, y 8 bits.

• Generación de paridad y comprobación: par, marca impar, espacio o ninguno.

• Uno o dos bits de parada.

• 16 bytes de recepción y transmisión FIFO.

• Generador de velocidad de transmisión, incluyendo un divisor fraccional de gran versatilidad.

• Soporta DMA para transmitir y recibir.

• Selector de velocidad.

• Generación y detección de rotura.

• Multiprocesador modo de direccionamiento.

• Modo de IrDA para apoyar la comunicación por infrarrojos.

• Soporte para el control de flujo por software.

Para su configuración se debe trabajar sobre los siguientes registros:

1. Energía: En el registro PCONP, se setea los bits PCUART0/2/3.

Registro PCONP – dirección 0x400F C0C4

2. Reloj: En el registro PCLKSEL0, seleccione PCLK_UART0; en el registro PCLKSEL1, seleccione PCLK_UART2/3.

Registro PCLKSEL0 – dirección 0x400F C1A8

Registro PCLKSEL1 – dirección 0x400F C1AC

Tabla 5-9: Bits del registro “Power Control for Peripherals register” (PCONP - address 0x400F C0C4)

Bit	Nombre	Descripción	RESET
●			
3	PCUART0	Bit de control de energía de la UART 0.	1
●			
24	PCUART2	Bit de control de energía de la UART 2.	0
25	PCUART3	Bit de control de energía de la UART 3.	0
●			

Fuente: Tabla 46 del LPC17xx manual del usuario.

Tabla 5-10: Descripción del registro “Peripheral Clock Selection register 0” (PCLKSEL0 - address 0x400F C1A8)

Bit	Nombre	Descripción	RESET
●			
6:7	PCLK_UART0	Selección del clock para el periférico UART0.	00
●			

Fuente: Tabla 40 del LPC17xx manual del usuario.

Tabla 5-11: Descripción del registro “Peripheral Clock Selection register 1” (PCLKSEL1 - address 0x400F C1AC)

Bit	Nombre	Descripción	RESET
●			
16:17	PCLK_UART2	Selección del clock para el periférico UART2.	00
18:19	PCLK_UART3	Selección del clock para el periférico UART3	00
●			

Fuente: Tabla 41 del LPC17xx manual del usuario.

3. Velocidad de transmisión: En registro U0/2/3LCR, setea el bit DLAB = 1. Esto permite el acceso a los registros DLL y DLM para

ajustar la velocidad en baudios. Además, si es necesario, ajusta la velocidad de transmisión frAcciónal en el registro divisor fraccional.

Registros U0LCR - dirección 0x4000 C00C,

U2LCR - dirección 0x4009 800C,

U3LCR - dirección 0x4009 C00C

Tabla 5-12: Descripción del registro "UARTn Line Control Register" (U0LCR - address 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR -0x4009 C00C)

Bit	Nombre	Valor	Descripción	RESET
●				
7	Divisor Latch Access Bit (DLAB)	0	Inhabilita el acceso al Divisor Latches.	0
		1	Habilita el acceso al Divisor Latches.	
●				

Fuente: Tabla 279 del LPC17xx manual del usuario.

4. UART FIFO: Usa el bit FIFO habilitado (bit 0) en el registro U0/2/3FCR para habilitar la FIFO.

Registros U0FCR - dirección 0x4000 C008,

U2FCR - dirección 0x4009 8008,

U3FCR - dirección 0x4009 C008

5. Pines: Selecciona los pines de la UART a través de los registros PINSEL y modos de los pines a través de los registros PINMODE.

Observación: La UART puede recibir cuando no tiene habilitado la resitencia pull-down.

6. Interrupciones: Para habilitar la interrupción por UART se setea el bit DLAB = 0 en el registro U0/2/3LCR

Las interrupciones están habilitadas en el NVIC utilizando apropiadamente el registro de interrupción Set Enable.

7. DMA: Las funciones de transmisión y recepción las realiza por la UART0/2/3 donde puede operar con el controlador GPDMA.

Tabla 5-13: Descripción del registro “UARTn FIFO Control Register” (U0FCR - address 0x4000 C008, U2FCR - 0x4009 8008, U3FCR -0x4007 C008)

Bit	Nombre	Valor	Descripción	RESET
0	FIFO enable	0	Se inhabilita la UARTn FIFOs. No debe ser usado en la aplicación.	0
		1	Activado en alto, permitirá tanto UARTn Rx y Tx FIFO y UnFCR [7: 1] el acceso. Este bit debe estar configurado para un funcionamiento correcto de la UART. Cualquier transición en este bit, borra automáticamente las FIFO UART relacionada.	
•				

Fuente: Tabla 278 del LPC17xx manual del usuario.

Los pines que utilizan las UART0/2/3 son:

Tabla 5-14: UARTn Descripción de Pin

Pin	Tipo	Detalle
RXD0, RXD2, RXD3	Entrada	Recepción de datos
TXD0, TXD2, TXD3	Salida	Transmisión de datos

Fuente tabla 269 del LPC17xx manual del usuario.

Caracteríticas de la UART1

Esta UART incluye las mismas cararteríticas de las UART0/2/3 vistas anteriormente, incluyendo además:

- Control completo del módem handshasking.
- Soporte para RS-485.

Los pines que utiliza la UART1 son:

Tabla 5-15: UART1 Descripción de Pin

Pin	Tipo	Detalle
RXD1	Entrada	Recepción de datos
TXD1	Salida	Transmisión de datos

CTS1	Entrada	Cancelar el envío. Señal activa (0) indica si el módem externo está listo para aceptar datos transmitidos a través de la UART1 TXD1
DCD1	Entrada	Data Carrier Detect. Señal activa (0) indica si se ha establecido un enlace de comunicación con el UART1 y los datos pueden ser intercambiados.
DSR1	Entrada	Conjunto de datos preparado. Señal activa (0) indica si el módem externo está listo para establecer un enlace de comunicaciones con el UART1.
DTR1	Salida	Terminal de datos preparado. Señal activa (0) indica que el UART1 está listo para establecer la conexión con el módem externo. El pin DTR también se puede utilizar como una señal de habilitación en RS-485 / EIA-485.
RI1	Entrada	Indicador de llamada. Señal activa (0) indica que una señal de llamada telefónica haya sido detectada por el módem.
RTS1	Salida	Solicitud de envío. Señal activa (0) indica que el UART1 esta lista para transmitir datos. El pin RTS también se puede utilizar como una señal de habilitación en RS-485 / EIA-485

Fuente: Tabla 288 del LPC17xx manual del usuario.

Archivos del CMSIS

Los siguientes archivos de cabecera definen la interfaz de programación de aplicaciones (API) para la interfaz USART:

Driver_USART.h: Driver de API para *Universal Synchronous Asynchronous Receiver/Transmisor*

Funciones del CMSIS

Se pretende dar una idea de las funciones que la CMSIS pone a disposición del desarrollo de aplicaciones, solo mencionaremos algunas de las funciones y se aconseja la lectura de la documentacion disponible en la página web de CMSIS.

int32_t ***USART_Initialize***(*USART_SignalEvent_t cb_event*)

Acción: La función USART_Initialize inicializa la interfaz USART. Se la llama cuando el componente middleware inicia la operación.

Devuelve: Un código de estado de error.

Parámetro:

[in] cb_event: Puntero a USART_SignalEvent.

El parámetro cb_event es un puntero a la función de devolución de llamada USART_SignalEvent; utilizar un puntero NULL cuando no se requieren señales de devolución de llamada.

int32_t ***USART_Send*** *(const void* data, uint32_t num)*

Acción: Inicia el envío de datos al transmisor USART.

Utiliza el modo asíncrono para enviar datos al transmisor USART. También se puede utilizar en modo síncrono al enviar datos solamente (datos recibidos se ignora).

Devuelve: Un código de estado de error.

Parámetros:

[in] data: Puntero al buffer de datos que se van a enviar al transmisor USART.

[in] num: Número de elementos de datos para enviar.

Tipo de datos:

uint8_t cuando se configura para 5..8 bits de datos.

uint16_t cuando se configura para 9 bits de datos.

El llamando a la función *USART_Send* se inicia con la operación de envío. La función es no bloqueante y regresa tan pronto como el driver ha comenzado la operación.

Después que la operación de envío se ha completado, podría haber algunos datos que quedan en el buffer del driver que todavía se están transmitiendo. Cuando todos los datos se han transmitido físicamente se genera el evento *USART_EVENT_TX_COMPLETE*. En ese momento también se borra el campo de datos *tx_busy* en *USART_STATUS*.

Estado de la transmisión se puede controlar mediante una llamada al USART_GetStatus y la comprobación de la bandera que indica si tx_busy todavía está tranmitiendo.

La operación de envío puede ser abortada llamando *USART_Control* con *USART_ABORT_SEND* como parámetro de control.

int32_t ***USART_Receive*** *(const void* data, uint32_t num)*

Acción: Iniciar la recepción de datos desde el receptor USART.

Devuelve: Un código de estado de error.

Parámetros:

[out] data: Puntero de datos para recibir desde el receptor USART.

[in] *num:* Número de elementos de datos para recibir.

Se utiliza en modo asíncrono para recibir datos desde el receptor USART. También se puede utilizar en modo síncrono cuando se reciben datos solamente.

El receptor debe estar habilitado llamando *USART_Control* con *USART_CONTROL_RX* con un parámetro de control y un argumento.

Los parámetros de la función especifican el buffer de datos y el número de elementos a recibir. El tamaño del artículo es definido por el tipo de datos que depende del número configurado de bits de datos.

Tipo de datos es:

uint8_t cuando se configura para 5..8 bits de datos.

uint16_t cuando se configura por 9 bits de datos.

Al llamar a la función *USART_Receive* se inicia la operación de recepción. La función es no bloqueante y regresa tan pronto como el driver ha comenzado la operación.

Al completarse la operación se genera el evento *USART_EVENT_RECEIVE_COMPLETE*. El progreso de la operación de recepción también se puede controlar mediante la lectura del número de datos ya recibidos llamando *USART_GetRxCount*.

El estado del receptor se puede controlar mediante una llamada al *USART_GetStatus* y comprobando la bandera *rx_busy* que indica si la recepción está aún en curso.

Durante la recepción se pueden generar los siguientes eventos (en modo asíncrono):

USART_EVENT_RX_TIMEOUT: Tiempo de espera superado (opcional).

USART_EVENT_RX_BREAK: Rotura detectada (no se genera el error de encuadre para la condición Break).

USART_EVENT_RX_FRAMING_ERROR: Error de encuadre detectado.

USART_EVENT_RX_PARITY_ERROR: Detecta el error de paridad.

USART_EVENT_RX_OVERFLOW: Desbordamiento de datos detectada (también en modo esclavo síncrono).

La operación de recepción se puede abortar llamando *USART_Control* con *USART_ABORT_RECEIVE* con un parámetro de control.

USART_STATUS ***USART_GetStatus*** *(void)*

Acción: La función *ARM_USART_GetStatus* recupera el estado actual de la interfaz USART.

Devuelve:

El estado de la USART, según la estructura *USART_STATUS*

Tabla 5-16: Bits de estados de la USART

Nro	Tipo	Nombre	Nro	Tipo	Nombre
1	uint32_t	Tx_busy: 1	5	uint32_t	Rx_break: 1
2	uint32_t	Rx_busy: 1	6	uint32_t	Rx_framing_error:1
3	uint32_t	Tx_underflow: 1	7	uint32_t	Rx_parity_error: 1
4	uint32_t	Rx_overflow: 1			

Fuente: www.keil.com/pack/doc/CMSIS/drivers

Código de estado de error

La siguiente lista proporciona los códigos de error de estado que son comunes en todos los DRIVERS.

Un código de error negativo indica que ocurrió durante la ejecución.

Tabla 5-17: Código de error

Nombre	Valor	Indica
DRIVER_OK	0	Operación exitosa.
DRIVER_ERROR	-1	Error no especificado.
DRIVER_ERROR_BUSY	-2	Driver ocupado.
DRIVER_ERROR_TIMEOUT	-3	Tiempo sobrepasado.
DRIVER_ERROR_UNSUPPORTED	-4	Operación no soportada.
DRIVER_ERROR_PARAMETER	-5	Error en parámetro.
DRIVER_ERROR_SPECIFIC	-6	Código especifico del periférico.

5.1.c.2. CAN1/2

Controller Area Network (CAN) es un protocolo de comunicación serie de alto rendimiento. El controlador CAN está diseñado para proporcionar una implementación completa del Protocolo CAN version 2.0B.

Permite la implementación de redes locales con un nivel muy alto de seguridad. Las aplicaciones utilizadas son para: entornos de automatización industriales y redes de alta velocidad, como así también para cableado de bajo costo multiplexado.

El módulo CAN se compone de dos elementos: el controlador y el filtro de aceptación. Todos los registros y la memoria RAM se acceden como palabras de 32 bits.

Características generales de la CAN

• Compatible con la especificación CAN 2.0B, ISO 11.898-1.

• Arquitectura multi-master con arbitraje de bits no destructiva.

• Prioridad de acceso al bus determinado por el identificador de mensaje (11 bits o 29 bits).

• Garantía de tiempo de latencia para los mensajes de alta prioridad.

• Velocidad de transferencia programable (hasta 1 Mbit/s).

• Multicast y broadcast para facilitar mensaje.

• Tamaño de los datos desde 0 hasta 8 bytes.

• Capacidad de gestión de errores de gran alcance.

• No retorno a cero (NRZ) codificación/decodificación con relleno de bits.

Característcas del controlador CAN

• 2 controladores CAN y buses.

• Compatible con identificador de 11 bits, así como identificador de 29 bits.

• Doble buffer de recepción y triple buffer de transmisión.

• Límite de error programable y contadores de error con accesos a lectura/escritura.

• Perdida de captura y captura de código de error con la posición detallada del bit.

• Un solo disparo de transmisión (sin retransmisión).

• Mode de sólo escucha (sin reconocer, no activa los flags de error).

• Recepción de mensajes "propios" (Auto recepción de petición).

Características de filtro aceptación

• Implementación de Hardware Fast, es un algoritmo de búsqueda de apoyo de gran número de identificadores CAN.

• Filtro global de aceptación reconoce 11 bits y 29 bits Rx Identificadores para todos buses CAN.

• Permite la definición explícita y de grupos de identificadores CAN de 11 bits y 29 bits.

• Filtro de aceptación proporciona la recepción automática de estilo para FullCAN seleccionado identificador estándar.

Los periféricos CAN1/2 se configuran con los siguientes registros:

1. Energía: En el registro PCONP, setea los bits PCCAN1/2.

Registro PCONP – dirección 0x400F C0C4.

Tabla 5-18: Bits del registro “Power Control for Peripherals register” (PCONP - address 0x400F C0C4)

Bit	Nombre	Descripción	RESET
●			
●			
13	PCCAN1	Bit de control de energía del CONTROLADOR CAN 1.	
14	PCCAN2	Bit de control de energía del CONTROLADOR CAN 2.	
●			

Fuente: Tabla 46 del LPC17xx manual del usuario.

2. Reloj: En el registro PCLKSEL0, selecciona PCLK_CAN1, PCLK_CAN2, y, para el filtro de aceptación, PCLK_ACF. Tenga en cuenta que estos deben ser todos el mismo valor.

Registro PCLKSEL0 – dirección 0x400F C1A8

3. Despertador: El controlador CAN es capaz de despertar al microcontrolador desde el modo Power-down.

4. Pins: Seleccionar los pines CAN1/2 a través de los regitros PINSEL y sus modos a través de los registros PINMODE.

Tabla 5-19: Eventos del reloj digital. Descripción del registro “Peripheral Clock Selection register 0” (PCLKSEL0 - address 0x400F C1A8)

Bit	Nombre	Descripción	RESET
●			
26:27	PCLK_CAN1	Selección de clock para el canal 1 del bus CAN.	00
28:29	PCLK_CAN2	Selección de clock para el canal 2 del bus CAN.	00
●			

Fuente: Tabla 40 del LPC17xx manual del usuario

5. Interrupciones: Las interrupciones CAN están habilitadas utilizando los registros CAN1/2IER. Las interrupciones se habilitan en el NVIC utilizando apropiadamente la interrupción del regitro Set Enable.

CAN1IER - dirección 0x4004 4010

CAN2IER - dirección 0x4004 8010

Tabla 5-20: Descripción del registro “CAN Interrupt Enable Register” (CAN1IER - address 0x4004 4010, CAN2IER - address 0x4004 8010)

Bit	Nombre	Descripción	RESET
0	RIE	Habilita la interrupción en recepción. El controlador interrumpe cuando el buffer de recepción esta lleno	0
1	TIE1	Habilita la interrupción en transmisión. Cuando un mensaje se ha transmitido con éxito de TXB1 o Transmit Buffer 1 es accesible de nuevo, el controlador CAN envía una interrupción.	0
2	EIE	Habilita interrupción por error.	0
3	DOIE	Habilita interrupción por Saturacion. Si el bit de estado de saturación se establece, el controlador CAN envía una interrupción.	0
4	WUIE	Habilita interrupción por Wake-Up. Si el controlador CAN despierta del modo sleep, el controlador CAN envía una interrupción.	0
5	EPIE	Habilita interrupción por Error Passive. Si el estado de error del controlador CAN cambia de error activa a error pasivo o viceversa, se solicita la correspondiente interrupción.	0
6	ALIE	Habilita interrupción por Arbitration Lost. Si el controlador ha perdido el arbitraje, se solicita la correspondiente interrupción.	0
7	BEIE	Habilita interrupción por error en el Bus. Si un error de bus es detectado, se solicita la correspondiente interrupción.	0
8	IDIE	Habilita interrupción por ID Ready. Cuando se ha recibido un identificador CAN, el controlador CAN envía una interrupción.	0
9	TIE2	Habilita la interrupción de transmición por Buffer2. Cuando un mensaje se ha	

		transmitido con éxito por TXB2 o transmitido por Buffer2 esta accesible de nuevo, el controlador CAN envía una interrupción.	
10	TIE3	Habilita la interrupción de transmición por Buffer3. Cuando un mensaje se ha transmitido con éxito de TXB3 o transmitie por Buffer3 esta accesible de nuevo, el controlador CAN envía una interrupción.	
11:31	-	Reservado	NA

Fuente: Tabla 321 del LPC17xx manual del usuario

6. Inicialización del controlador CAN: Ve el registro CANMOD.

En la tabla 5-21 podemos ver cuales son los pines que utilizan:

Tabla 5-21: CAN Descripción de pines

Pin	Tipo	Detalle
RD1, RD2	Entrada	Recepción de datos
TD0, TD2	Salida	Transmisión de datos

Fuente: Tabla 311 del LPC17xx manual del usuario

Archivos del CMSIS

Los siguientes archivos de cabecera definen la interfaz de programación de aplicaciones (API) para la interfaz CAN:

Driver_CAN.h: Driver API para CAN bus de periféricos.

Funciones del CMSIS

int32_t **CAN_Initialize** (CAN_SignalUnitEvent_t cb_unit_event, CAN-_SignalObjectEvent_t cb_object_event)

Acción: Inicializa la interfaz CAN y registra la señal de funciones.

Devuelve: El código de error del estado.

Parámetros:

[in] cb_unit_event: Puntero a la función de devolución de llamada CAN_SignalUnitEvent

[in] cb_object_event: Puntero a la función de devolución de llamada CAN_SignalObjectEvent

La función realiza las siguientes operaciones:

Inicializa los recursos necesarios para la interfaz CAN, para la asignación de memoria dinámica, como por ejemplo, RTOS para la asignación de objetos, y posiblemente configuración de pines de hardware.

Registra la función de devolución de llamada *CAN_SignalUnitEvent*.

Registra la función de devolución de llamada *CAN_SignalObjectEvent*.

El parámetro *cb_unit_event* es un puntero a la función de devolución de llamada *CAN_SignalUnitEvent*; utilizar un puntero NULL cuando no se requieren señales de devolución de llamada.

El parámetro *cb_object_event* es un puntero a la función de devolución de llamada *CAN_SignalObjectEvent*; utilizar un puntero NULL cuando no se requieren señales de devolución de llamada.

int32_t **CAN_MessageSend** (uint32_t obj_idx, CAN_MSJ_INFO* msg_info, Const_unit8_t* data, unt8_t size)

Acción: La funcion CAN_MessageSend envía un mensaje CAN por el bus CAN, o setea el mensaje de datos que será devuelto automáticamente al recibir RTR (Remote Trnsmission Request) de un ID CAN.

Devuelve: valor> = 0 número de bytes de datos aceptados para enviar; valor <0 códigos de error del estado.

Parámetros:

[in] obj_idx: objeto de índice.

[in] msg_info: Puntero a la información del mensaje CAN.

[in] data: Puntero de datos de buffer de datos.

[in] size: Número de tamaño de bytes de datos para enviar.

Sólo un mensaje se puede enviar con una llamada a esta función (para CAN hasta 8 bytes; para la CAN FD hasta 64 bytes de datos). Una transmisión de mensajes se puede terminar con una llamada a la función con *CAN_Control* control = *CAN_ABORT_MESSAGE_SEND*.

El parámetro *obj_idx* especifica el índice de mensajes objeto.

El parámetro *msg_info* es un puntero a la estructura *CAN_MSG_INFO*, que contiene los siguientes campos de datos relevantes para enviar el mensaje:

id: Identificador del mensaje; bit 31 específica si se trata de un identificador de 11 bits o 29 bits.

rtr: Específica si la solicitud de transmisión a distancia se debe enviar (dlc se utiliza para el número de bytes solicitado), de lo contrario se enviará el mensaje de datos. Consulte Remote Frame para más detalles.

edl: Específica si se utiliza longitud de datos extendida; para CAN FD, el mensaje puede contener hasta 64 bytes de datos.

sa: Específica si la velocidad de bits de conmutación se va a utilizar; para CAN FD, la tasa de bits se puede aumentar durante la fase de datos.

dlc: Longitud de datos del código de bytes de datos solicitados al enviar solicitud de transmisión a distancia.

El parámetro *data* es un puntero al buffer de dato.

El parámetro *size* es el número de bytes de datos a enviar.

La función devuelve el número de bytes aceptados para ser enviados o si el hardware no está listo para aceptar un nuevo mensaje para su transmisión se utiliza la función *DRIVER_ERROR_BUSY*.

Una vez enviado el mensaje, la función devuelve la llamada a *CAN_SignalObjectEvent* y se llama a la señalización del objeto por *CAN_EVENT_SEND_COMPLETE.*

int32_t **CAN_MessageRead** (uint32_t obj_idx, CAN_MSJ_INFO* msg_info, Const_unit8_t* data, unt8_t size)

La función *CAN_MessageRead* lee el mensaje recibido en el bus CAN, si *obj_idx* fue configurado para la recepción o para la recepción de datos de mensajes automáticos utilizando RTR y la función realiza un callback a *CAN_SignalObjectEvent* que fue llamada como señalización a CAN_EVENT_RECEIVE. Si el mensaje fue invalidado por otro mensaje recibido, entonces la función de devolución llama a CAN_SignalObjectEvent donde se llamará a la señalización de CAN_EVENT_RECEIVE_OVERRUN.

Devuelve: valor > = 0 número de bytes de datos leidos; valor <0 Códigos de error de estado.

Parámetros:

[in] obj_idx: Objeto del índice.

[out] msg_info: Puntero para leer la información de mensajes CAN.

[out] data: Puntero de datos de buffer de datos para lectura.

[in] size: Número de tamaño de bytes de datos para leer.

La función se puede leer un máximo de 8 bytes de datos para la CAN y 64 bytes para CAN FD.

El parámetro *obj_idx* específica el índice del mensaje objeto.

El parámetro *msg_info* es un puntero a la estructura de información de la CAN.

El parámetro *data* es un puntero al buffer de dato atos para la lectura de datos.

El parámetro *size* es el tamaño del buffer de datos en bytes e indica el número máximo de bytes que se pueden leer.

La función devuelve el número de datos leídos en bytes o los códigos de error del estado.

Todos los campos de datos de la estructura CAN_MSG_INFO se actualizan como se describe a continuación:

id: Identificador del mensaje que se recibió, bit 31 específica si se trata de un identificador de 11 bits o identificador de 29 bits.

rtr: 1 = Solicitud Remote Frame se recibió (dlc es el número de bytes solicitado). 0 = mensaje de datos.

edl: 1 = CAN FD longitud de datos de mensajes extendidos fue recibido. 0 = Longitud de datos de mensajes no extendidos.

brs: 1 = CAN FD se utiliza para la velocidad de bits de conmutación de transferencia de mensajes. 0 = no se usa la velocidad de bits de conmutación.

esi: 1 = CAN FD Indicador del estado de error, está activo para el mensaje recibido. 0 = Indicador del estado de error no está activo.

dlc: Longitud de datos del código, es el número de bytes de datos en el mensaje o el número de bytes de datos recibidos solicitada por RTR.

5.1.c.3 SPI

SPI es una interfaz serial full duplex. Puede manejar múltiples maestros y esclavos siendo conectado a un bus dado. Sólo un maestro y un solo esclavo pueden comunicarse en la interfaz durante una transferencia de datos determinada. Durante una transferencia de datos el maestro siempre envía 8 a 16 bits de datos al esclavo y el esclavo siempre envía un byte de datos al maestro.

Características

• Compatible con la especificación Serial Peripheral Interface (SPI).

•Síncrono, Serie, Comunicación Full Duplex.

• SPI Maestro o esclavo.

• Tasa máxima de bits de datos de una octava parte de la velocidad del reloj.

• 8 a 16 bits por transferencia.

El SPI se configura con los siguientes registros:

1. Energía: En el registro PCONP, ajusta el bit PCSPI.

Registro PCONP – dirección 0x400FC0C4

Tabla 5-22: Bits del registro “Power Control for Peripherals register” (PCONP - address 0x400F C0C4)

Bit	Nombre	Descripción	RESET
●			
8	PCSPI	La interfaz power/clock SPI de control de bit.	1
●			

Fuente: Tabla 46 del LPC17xx manual del usuario

Observación: En el reset, el SPI está habilitado (PCSPI = 1).

2. Reloj: En el registro PCLKSEL0, setea el bit PCLK_SPI. En el modo maestro, el reloj debe ser un número par mayor o igual a 8.

Registro PCLKSEL0 – dirección 0x400F C1A8

Tabla 5-23: Descripción del registro “Peripheral Clock Selection register 0” (PCLKSEL0 - address 0x400F C1A8)

Bit	Nombre	Descripción	RESET
●			
17:16	PCLK_SPI	Selección de clock para el bus SPI.	00
●			

Fuente: Tabla 40 del LPC17xx manual del usuario

3. Pins: Los pines SPI se configuran utilizando tanto el PINSEL0 y el PINSEL1, como así el registro con PINMODE. PINSEL0 [31:30] se utiliza para configurar el pin SPI CLK. PINSEL1 [1: 0], PINSEL1 [3: 2] y PINSEL1 [5: 4] se utilizan para configurar los pines SSEL, MISO y MOSI, respectivamente.

4. Interrupciones: El flag de interrupción SPI está activado mediante el bit S0SPINT [0]. El flag de interrupción SPI debe estar habilitado en el NVIC.

En la tabla 5-24 podemos ver los pines que se utilizan:

Tabla 5-24: SPI Descripción de pines

Pin	Tipo	Detalle
SCK	Entrada / Salida	**Serial Clock**. Se utiliza para sincronizar la transferencia de datos a través de la interfaz SPI. El SPI es siempre impulsada por el maestro y recibida por el esclavo. El reloj se puede programar para ser en activa alta o activa bajo. El SPI sólo se activa durante una transferencia de datos.
SSEL	Entrada	**Selección del esclavo**. Activa por nivel bajo que indica que el esclavo se encuentra seleccionado para una transferencia de datos. Debe ser puesto en bajo antes de que comiencen las transAcciónes de datos y, se mantiene bajo durante la transacción. Si la señal de SSEL pasa a nivel alto la transferencia se aborta y el esclavo vuelve al estado de reposo, y los datos se desechan. Esta señal no es impulsada directamente por el maestro. Podría ser accionado por un pin de propósito general bajo control de software
MISO	Entrada / Salida	**Master in-Slave out**. Es una señal unidireccional utilizada para transferir datos en serie de un esclavo a un maestro.
MOSI	Entrada / Salida	**Master out-Slave in**. Es una señal unidireccional utilizada para transferir datos en serie de un maestro a un esclavo.

Fuente: Tabla 358 del LPC17xx manual de usuario.

Funciones del CMSIS

int32_t **SPI_Initialize**(SPI_SignalEvent_t cb_event)

Acción: La función SPI_Initialize inicializa la interfaz SPI.

Devuelve: El código de error del estado.

Parámetros:

[in] cb_event: Puntero a ARM_SPI_SignalEvent

El Parámetro *cb_event* es un puntero a la función de devolución de llamada *SPI_SignalEvent*; utilizar un puntero NULL cuando no se requieren señales de devolución de llamada.

La función se llama cuando el componente middleware inicia la operación y lleva a cabo las siguientes acciones:

• Inicializa los recursos necesarios para la interfaz SPI.

•Registra la función de devolución de la llamada de *SPI_SignalEvent.*

int32_t **SPI_Send**(Const_void* data, uint32_t num)

Acción: Esta función *SPI_Send* se utiliza para enviar datos al transmisor SPI.

Devuelve: El código del estado de error.

Parámetros:

[in] data: Puntero de datos para el buffer de datos a enviar al transmisor SPI.

[in] num: Número de elementos de datos para enviar.

El parámetro *data* específica el buffer de datos.

El parámetro *num* especifica el número de elementos a enviar.

El tamaño esta definida por el tipo de datos, que depende del número de bits de datos configurado.

El tipo de dato es:

uint8_t cuando se configura para 1..8 bits de datos.

uint16_t cuando se configura para 9..16 bits de datos.

uint32_t cuando se configura para 17..32 bits de datos.

El llamando a la función *SPI_Send* inicia la operación de envío. Cuando está en modo esclavo, la operación sólo se registra y comenzara cuando el maestro inicia la transferencia. La función es no bloqueante y regresa tan pronto como el driver ha comenzado la operación. Durante la operación no está permitido llamar a esta función o cualquier otra función de transferencia de datos de nuevo. También el buffer de datos debe permanecer asignado y el contenido de los datos enviados no debe ser modificado. Cuando se haya completado la operación de envío (número solicitado de elementos enviados), se genera el evento

SPI_EVENT_TRANSFER_COMPLETE. El progreso de la operación de envío también se puede controlar mediante la lectura del número de items que ya ha sido enviada por *SPI_GetDataCount*.

El estado del transmisor también se puede controlar mediante una llamada al *SPI_GetStatus* y comprobando el campo de datos *busy*, que indica si la transmisión está todavía en curso o pendiente.

Cuando en el modo maestro esta configurado para supervisar al esclavo y el esclavo se desactiva durante la transferencia, a continuación, los cambios en modo SPI se inactivan y se genera el evento *SPI_EVENT_MODE_FAULT.* (en lugar de *SPI_EVENT_TRANSFER_COMPLETE*).

Cuando el esclavo, esta en modo de enviar/recibir/transferencia de operación, sino se ha iniciado, los datos se envían/solicitado por el maestro, entonces se genera el evento *SPI_EVENT_DATA_LOST*.

Esta operación puede ser abortada llamando la función *SPI_Control* y con parámetro de control *SPI_ABORT_TRANSFER.*

int32_t **SPI_Receive**(Const_void* data, uint32_t num)

Acción: La función SPI_Receive se utiliza para recibir datos (transmite el valor predeterminado según lo especificado por *SPI_Control* y *SPI_SET_DEFAULT_TX_VALUE* como parámetro de control).

Devuelve: El código de error de estado.

Parámetros

[out] data: Puntero de datos para el buffer de recepción.

[in] num: Número de elementos de datos para recibir.

El parámetro *data* específica el buffer de datos.

El parámetro *num* específica el número de elementos a recibir.

El tamaño esta definido por el tipo de datos, que depende de la configuración del número de bits de datos.

Tipo de datos son:

uint8_t cuando se configura para 1..8 bits de datos.

uint16_t cuando se configura para 9..16 bits de datos.

uint32_t cuando se configura para 17..32 bits de datos.

Cuando se llama a la función SPI_Receive sólo se inicia la operación de recepción. La función es no bloqueante y regresa tan pronto como el conductor ha comenzado la operación. Cuando está en modo esclavo, la operación sólo se registra y comenzará cuando el maestro comience con la transferencia. Durante la operación no está permitido llamar a esta función o a cualquier otra función de transferencia de datos nuevamente. También el buffer de datos debe permanecer asignado. Al recibir la operación se completa, y genera el evento SPI_EVENT_TRANSFER_COMPLETE. El proceso de la operación de recepción también se puede controlar mediante la lectura del número de datos ya recibidos llamando a SPI_GetDataCount.

El estado del receptor también se puede controlar mediante una llamada al SPI_GetStatus y comprobando el campo de datos de ocupados, que indica si la recepción está aún en curso o pendientes.

Cuando el modo maestro esta configurado para supervisar al esclavo y el esclavo se desactiva durante la transferencia, a continuación, los cambios en modo SPI estan inactivo y el evento SPI_EVENT_MODE_FAULT se genera (en lugar de SPI_EVENT_TRANSFER_COMPLETE).

Cuando está en modo esclavo, espera enviar/recibir/transferir la operación no se ha inicia, hasta que el maestro solicita enviar los datos, entonces se genera el evento SPI_EVENT_DATA_LOST.

La operación de recepción se puede abortar llamando a SPI_Control con SPI_ABORT_TRANSFER como parámetro de control.

int32_t **SPI_Transfer**(Const_void* data_out, void* data_in, uint32_t num)

Acción: La función SPI_Transfer transfiere datos a través de la SPI. Sincrónicamente envía los datos al transmisor SPI y recibe datos desde el receptor SPI.

Devuelve: El código de error de estado.

Parámetros:

[in] data_out: Puntero al buffer de los datos para enviar al transmisor SPI.

[out] data_in: Puntero al buffer de datos que reciben del receptor SPI.

[in] num : Número de elementos de datos para transferir.

El parámetro *data_out* es un puntero al buffer de datos a enviar.

El parámetro *data_in* es un puntero al buffer que recibe los datos.

El parámetro *num* especifica el número de elementos a transferir.

El tamaño está definido por el tipo de datos que depende del número de bits de datos configurado.

int32_t **SPI_Control**(uint32_t control, uint32_t arg)

Acción: La función SPI_Control controla los ajustes de la interfaz SPI y ejecuta diversas operaciones.

Devuelve: El código de error de estado común y códigos de errores específico del controlador.

Parámetros:

[in] control: Control de operación.

[in] arg: El argumento de la operación (opcional).

El parámetro *control* es una máscara de bits que especifica varias operaciones.

- Diferentes controles de categorías ORed.
- Si se omite un control, se utiliza el valor por defecto de esa categoría.

• Varios controles no se pueden combinar.

SPI_STATUS **SPI_GetStatus**(void)

Acción: La función SPI_GetStatus devuelve el estado actual de la interfaz SPI.

Devuelve: El estado de la SPI, según la estructura SPI_STATUS.

La estructura SPI_STATUS se muestra en la siguiente tabla:

Tabla 5-25: Estructura SPI_STATUS.

Tipo	Campo
uint32_t	Busy: 1
uint32_t	Data_lost: 1
uint32_t	Mode_fault: 1

5.1.c.4 I2C0/1/2

Descripción

Una configuración típica del bus I2C se muestra en la siguiente figura 5-1. Dependiendo del estado de la dirección bit (R/W), son dos posibles tipos de transferencias de datos en el bus I2C.

• La transferencia de datos desde un transmisor a un receptor maestro esclavo. El primer byte transmitido por el maestro es la dirección del esclavo. Luego sigue un número de bytes de datos. El esclavo devuelve un bit de reconocimiento, después de cada byte recibido, a menos que el dispositivo esclavo no puede aceptar más datos.

• La transferencia de datos desde un transmisor esclavo de un receptor principal. El primer byte (la dirección del esclavo) se transmite por el maestro. El esclavo entonces devuelve un bit de reconocimiento. Luego siguen los bytes de datos transmitidos por el esclavo al maestro. El maestro devuelve un bit de reconocimiento, hasta que no sea el último byte recibido. Al final del último byte recibido, un "no reconoce" se devuelve. El dispositivo maestro genera todos los impulsos de reloj de serie y el inicio y las condiciones de parada. Una transferencia se terminó con una

condición de STOP o con una condición de START repetida. Desde una condición START repetida es también el comienzo de la próxima transferencia de serie, el bus I2C no se dará a conocer.

Las interfaces I2C del LPC17xx están orientadas bytes y tiene cuatro modos de funcionamiento: el modo transmisor master, el modo receptor master, el modo transmisor esclavo y el modo recptor esclavo.

Configuración básica

Las interfaces I2C0 /1/2 se configuran con los siguientes registros:

1. Energía: En el registro PCONP, setea los bits PCI2C0/1/2.

Registro PCONP – dirección 0x400F C0C4.

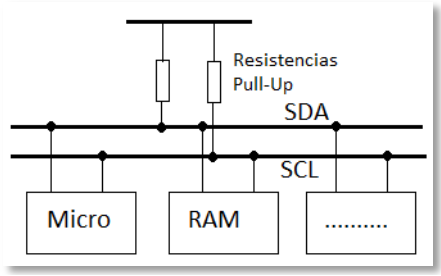


Figura 5-1: Conección típica de un bus I2C.

Tabla 5-25: Bits del registro “Power Control for Peripherals register” (PCONP - address 0x400F C0C4)

Bit	Nombre	Descripción	Reset
●			
7	PCI2C0	La interfaz power/ clock I2C0 para el control de bit.	1
●			
19	PCI2C1	La interfaz power/ clock I2C1 para el control de bit.	1
●			
26	PCI2C2	La interfaz power/ clock I2C2 para el control de bit.	1
●			

Fuente: Tabla 46 del manual del usuario LPC17xx.

Observación: Después de un reset, todas las interfaces I2C quedan habilitadas (PCI2C0/1/2 = 1).

2. Reloj: En PCLKSEL0 seleccione PCLK_I2C0; en PCLKSEL1 seleccione PCLK_I2C1 o PCLK_I2C2.

Registro PCLKSEL0 – dirección 0x400F C1A8

Tabla 5-26: Descripción del registro "Peripheral Clock Selection register 0" (PCLKSEL0 - address 0x400F C1A8)

Bit	Nombre	Descripción	Valor al RESET
●			
15:14	PCLK_I2C0	Selección de clock para el canal 0 del bus I2C.	00
●			

Fuente: Tabla 40 del manual del usuario LPC17xx

Registro PCLKSEL1 – dirección 0x400F C1AC

Tabla 5-27: Descripción de bits en registro "Peripheral Clock Selection register 1" (PCLKSEL1 - address 0x400F C1AC)

Bit	Nombre	Descripción	Valor al RESET
●			
7:6	PCLK_I2C1	Selección de clock para el canal 1 del bus I2C.	00
●			
21:20	PCLK_I2C2	Selección de clock para el canal 2 del bus I2C.	00
●			

Fuente: Tabla 41 del manual del usuario LPC17xx.

3. Pins: Seleccionar los pins I2C0, I2C1 o I2C2 a través del registro PINSEL. Seleccione los modos de los pines para las funciones de los pines del puerto I2C1 o I2C2 a través de los registros PINMODE (sin resistencias pull-up, pull-down) y los registros PINMODE_OD (open drain).

Observación: I2C0 no está disponible en el encapsulado de 80 pines.

Observación: Los pines I2C no utilizados se pueden configurar para un modo open-drain a través de los registros IOCON, y se puede

utilizar con el modo rápido (400 kHz) y el modo estándar (100 kHz). Estos pines no incluyen un filtro analógico para suprimir interferencias de línea, pero una función similar lleva a cabo el filtro digital en el propio bloque I2C. Estos pines deben configurarse como: sin pull-up, sin pull-down, modo open-drain.

4. Las interrupciones están habilitadas en el NVIC utilizando un apropiado seteo del registro.

5. Inicialización: Podrá enconcontrala en el manual de usuario para este microcontrolador.

Características

• Las interfaces de bus estándar I2C se pueden configurar como Maestro, Esclavo, Maestro/Esclavo.

• El arbitraje se maneja entre los maestros que transmiten simultáneamente, sin la perdida de datos en serie en el bus.

• Reloj programable, que permite el ajuste de las tasas de transferencia de I2C.

• La transferencia de datos es bidireccional entre maestros y esclavos.

• La sincronización de reloj permite a los dispositivos con diferentes velocidades comunicarse a través de un bus serie.

• La sincronización de reloj de serie se utiliza como mecanismo handshake para suspender y reanudar la transferencia serie.

• Soporta el modo Fast Plus (sólo I2C0).

• Reconocimiento opcional de hasta 4 direcciones de esclavos distintos.

• El modo de monitor permite la observación de todo el tráfico del bus I2C, independientemente de la dirección del esclavo, sin afectar al tráfico real.

• El bus I2C se puede utilizar para fines de test y diagnóstico.

Descripción de pines:

Tabla 5-28: I2C Descripción de Pines

Pin	Tipo	Descripcion
SDA0 (*)	Entrada/salida	I2C0 Serial Data
SCL0(*)	Entrada/salida	I2C0 Serial Clock
SDA1	Entrada/salida	I2C1 Serial Data
SCL1	Entrada/salida	I2C1 Serial Clock
SDA2	Entrada/salida	I2C2 Serial Data
SCL2	Entrada/salida	I2C2 Serial Clock

[1] I2C0 sólo está disponible en dispositivos LPC17xx de 100 pines. Los pines SDA0 y SCL0 son con tecnología open-drain para cumplir con las especificaciones de I2C. Se deben configurar en el registro I2CPADCFG para el modo Fast Plus. Fuente tabla 380 del manual de usuario LPC17xx.

Funciones del CMSIS

void **I2C_Init** (LPC_I2C_TypeDef *I2Cx, unit32_t clockrate)

Acción: Inicializa el periférico I2Cx.

Parámetros:

[in] I2Cx: Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

[in] clockrate: Valor de la velocidad del reloj para el periférico I2C inicializado (Hz)

void **I2C_Cmd** (LPC_I2C_TypeDef *I2Cx, FunctionalState NewState)

Acción: Habilita/Inhabilita un periférico I2C.

Parámetros:

[in] I2Cx: Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

Status **I2C_MasterTransferData** (LPC_I2C_TypeDef *I2Cx, I2C_M_SETUP_Type *TransferCfg, I2C_TRANSFER_OPT_Type Opt)

Acción: Transmite y recibe datos en modo maestro.

Devuelve: SUCCESS o ERROR

Parámetros:

[in] I2Cx: Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

[in] *TransferCfg:* Puntero a la estructura *I2C_M_SETUP_Type* que contiene información específica sobre la configuración para la transferencia de maestro.

[in] Opt: Un tipo I2C_TRANSFER_OPT_Type puede seleccionar la interrupción o el modo polling.

Notas:

• En caso de utilizar I2C para transmitir datos solamente, cuando se transmite la longitud se establece en 0, o cuando se transmiten datos el puntero se establece en NULL.

• En caso de utilizar I2C para recibir datos solamente, cuando se recibe la longitud se establece en 0, o cuando se reciben datos el puntero se establece en NULL.

• En caso de utilizar I2C para transmitir y recibir datos, transmite la longitud, transmite el puntero de datos, recibe la longitud y recibe el puntero de datos, se debe establecer los valores correspondientes.

Status **I2C_SlaveTransferData** (LPC_I2C_TypeDef *I2Cx, I2C_S_SETUP_Type *TransferCfg, I2C_TRANSFER_OPT_Type Opt)

Acción: Recibe y transmite datos en modo esclavo.

Devuelve: SUCCESS o ERROR

Parámetros:

[in] I2Cx: Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

[in]TransferCfg: Puntero a la estructura *I2C_M_SETUP_Type* que contiene información específica sobre la configuración para la transferencia del maestro.

[in] Opt: Un tipo I2C_TRANSFER_OPT_Type que ha seleccionado para el modo de interrupción o de modo polling.

Nota:

El modo de funcionamiento del esclavo depende del comando enviado desde maestro en el bus I2C. Si el maestro envía un comando SLA + W, esta sub-rutina recibe la longitud de datos y el puntero a los datos. Si el maestro envía un comando SLA + R, esta sub-rutina transmite la longitud de datos y el puntero a los datos. Si el maestro repite un comando de inicio o un comando de parada, el esclavo habilitará la condicción time out, durante la condición de time out, si no hay actividad en el bus I2C, el esclavo saldrá, de lo contrario (por ejemplo, el maestro envía un SLA + R / W), el esclavo cambia a un modo de operación relevante. El time out se debe utilizar porque el código de estado de retorno no puede mostrar la diferencia entre los comandos de parada e inico en el funcionamiento de esclavo. En caso de que la longitud de los datos de espera del maestro sea mayor, el esclavo puede soportar:

• En caso de operación de lectura (de maestro): El esclavo devolverá el valor I2C_I2DAT_IDLE_CHAR.

• En caso de operación de escritura (de maestro): El esclavo ignorará datos del maestro.

uint32_t **I2C_MasterTransferComplete** (LPC_I2C_TypeD_t ef* I2Cx)

Acción: Obtiene el estado de la transferencia del maestro.

Devuelve: El estado de la transferencia del maestro, podría ser:

• TRUE Transferencia del maestro completada.

• FALSE Transferencia del maestro no ha completado todavía.

Parámetros:

[in] I2Cx : Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

uint32_t **I2C_SlaveTransferComplete** (LPC_I2C_TypeDef *I2Cx)

Acción: Obtiene el estado de transferencia del esclavo.

Devuelve: Estado completado, puede ser: TRUE/FALSE.

Parámetros:

[in] I2Cx: Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

void **I2C_MonitorModeCmd** (LPC_I2C_TypeDef *I2Cx, FunctionalState NewState)

Acción: Habilita/Inhabilita el modo instructor del I2C.

Parámetros:

[in] I2Cx: Periférico seleccionado, puede ser: LPC_I2C0 / LPC_I2C1 / LPC_I2C2

[in] NewSate: Estado Nuevo de la función, debe ser: ENABLE: Habilita el modo instructor; DESACTIVAR: Inhabilita el modo instructor.

5.1.d. Timer 0/1/2/3

Descripción

- Temporizador de intervalo para el recuento de eventos internos.
- Demodulador de ancho de pulso a través de entradas de capturas.
- Temporizador libre.

El temporizador / contador está diseñado para contar ciclos de reloj del periférico (PCLK) o de un suministrado externo de reloj, y puede generar opcionalmente interrupciones o realizar otras acciónes en valores del temporizador especificados, basado en los cuatro registros. También incluye cuatro entradas de captura para atrapar el valor del temporizador cuando una transición de señal de entrada, genera opcionalmente una interrupción.

Descripción de pines:

Tabla 5-29: Descripción de pines Timer/Counter

Pin	Tipo	Descripcion
CAP0[1:0]	Entradas	Señal de captura - Una transición de captura de un pin puede ser configurado para cargar uno de los registros de captura con el valor en el contador del temporizador y, opcionalmente, generar una interrupción. Funcionalidad la captura se puede seleccionar con un número de pines. Cuando se selecciona más de un pin de entrada la captura en un solo canal TIMER0/1, se utiliza el pin con el número de puerto más bajo.
CAP1[1:0]		
CAP2[1:0]		
CAP3[1:0]		
MAT0[1:0]	Salidas	Salida de ajuste externa - Cuando un registro de ajuste (MR3:0) es igual al contador del temporizador (TC), esta salida se puede invertir, bajar, levantar, o no hacer nada. El registro de ajuste externo (EMR) controla la funcionalidad de esta salida. La funcionalidad de ajuste de salida puede seleccionar un número paralelo de pines.
MAT1[1:0]		
MAT2[1:0]		
MAT3[1:0]		

Fuente: Tabla 424 del LP17xx manual de usuario.

Funciones del CMSIS

uint32_t **Init_Timer** (uint8_t timer_num, uint32_t TimerInterval)

Acción: Inicializa el temporizador.

Parámetros:

[in] time_num: Número de temporizador en un rango entre 0 y 3.

[in] TimerInterval: Intervalo de tiempo.

Retorna: 1: True – exitoso; 0: False – error.

void **Enable_Timer**(uint8_t timer_num)

Acción: Habilita el temporizador.

Parámetro:

[in] time_num: Número de temporizador en un rango entre 0 y 3.

void **Disable_Timer**(uint8_t timer_num)

Acción: Inhabilita el temporizador.

Parámetro:

[in] time_num: Número de temporizador en un rango entre 0 y 3.

- ✓ void **TIMER0_Interrupt** (void)
- ✓ void **TIMER1_Interrupt** (void)
- ✓ void **TIMER2_Interrupt** (void)
- ✓ void **TIMER3_Interrupt** (void)

Temporizador/Contador por interrupción del handler.

5.1.e System Tick Timer - SysTick

Descripción

El System Tick Timer es una parte integral del Cortex- M3. El System Tick Timer esta destinado a generar una interrupción fija de 10 milisegundos para uso de un sistema operativo u otro software de gestión del sistema.

Desde que el System Tick Timer es una parte del Cortex-M3, facilita la portabilidad de software proporcionando un temporizador estándar que está disponible en los dispositivos basados en Cortex-M3 .

Características

• Intervalos de tiempos de 10 milisegundos.

• Vector de expensión dedicado.

• Puede ser registrado internamente por el reloj de la CPU o por una entrada de reloj desde un pin (STCLK).
El System Tick Timer se configura usando los siguientes registros:

Tabla 5-30: System Tick Timer registros asociados

Nombre	Descripción	Tipo de acceso	Reset	Dirección
STCTRL	Sistema de control del temporizador y registro de estado.	R/W	0x4	0xE000 E010
STRELOAD	El sistema recarga el valor del registro del temporizador.	R/W	0	0xE000 E014
STCURR	El sistema tiene el valor presente del registro del temporizador.	R/W	0	0xE000 E018
STCALIB	El sistema calibra el valor del registro del temporizador.	R/W	0x000 F 423F	0xE000 E01C

Fuente: Tabla 438 del manual del usuario LPC17xx.

1. Fuente de reloj: Seleccionar la CCLK interna o STCLK (P3.26) externa de la fuente de reloj en el registro STCTRL.
Registro STCTRL – dirección 0xE000 E010

Tabla 5-31: Descripción de bits "System Timer Control and status register"

Bit	Nombre	Descripción	Reset
0	ENABLE	Habilita el SysTick. (1 = habilitado; 0 = deshabilitado)	0
1	TICKINT	Habilita interrupción por SysTick. (Con 1 = habilitado; 0 = inhabilitado). Cuando está habilitado, se genera la interrupción cuando el contador SysTick llega a 0	0
2	CLKSOURCE	Selección del clock del SysTick. (Con 1 = CPU clock; 0 = reloj externo). Cuando es 0, se selecciona el pin de reloj externo (STCLK)	1
3:15	-	Reservado.	
16	COUNTFLAG	Este bit se pone en 1 cuando el contador SysTick llega a cero, se borra al leer el registro.	0
17:31	-	Reservado.	

Fuente: Tabla 439 del LPC17xx manual del usuario.

2. Pines: Si el STCLK (P3.26) fue seleccionado como fuente de reloj para habilitar el pin STCLK usando la función en el registro PINMODE.
3. Interrupción: El System Tick Timer habilitada la interrupción en el NVIC utilizando apropiadamente la interrupción del registro Set Enable.

Funciones del CMSIS

uint32_t **SysTick_Config**(uint32_t ticks)

Acción: Inicializa y lanza el temporizador del System Tick Timer y su interrupción. Después de esta llamada, el temporizador del SysTick crea interrupciones con el intervalo de tiempo especificado. El contador está en modo de funcionamiento libre para generar interrupciones periódicas.

Parámetros:

[in] ticks: Número de ticks entre dos interrupciones.

Retorna: 0 – Existoso; 1 – Fallido.

5.1.f. Pulse Width Modulator (PWM)

Descripción

El módulo PWM se basa en un bloque de temporizador estándar y heredando características, aunque sólo la función de los pines de salida del PWM se encuentran disponibles en la LPC17xx. El temporizador está diseñado para contar ciclos del reloj de periféricos (PCLK) y opcionalmente generar interrupciones o realizar otras Acciónes cuando se producen los valores de temporizador especificados, basado en siete registros. La función PWM suma estas características, y se basa en eventos de esos registros.

La capacidad de controlar por separado los flancos de subida y de bajada permite que el PWM se utiliza para más aplicaciones. Por ejemplo, el control de motor polifásico requiere normalmente tres

salidas PWM que no se superpongan con el control individual de las tres posiciones y anchos de pulso.

Dos registros se pueden utilizar para proporcionar una única salida PWM controlada por flancos. Un registro (PWMMR0) controla la velocidad de ciclo PWM, restableciendo el conteo. El otro registro controla la posición del flanco PWM. Adicionalmente las salidas PWM controladas requieren sólo un registro, ya que la tasa de repetición es la misma para todas las salidas PWM. Múltiples salidas PWM controladas por flanco tendrán un flanco ascendente en el inicio de cada ciclo PWM, cuando se produce una coincidencia PWMMR0.

Tres registros se pueden utilizar para proporcionar una salida PWM con ambos flancos controlados. Una vez más, el registro PWMMR0 controla la velocidad de ciclo PWM. Los otros registros controlan las dos posiciones de flancos PWM. Adicionalmente las salidas PWM controladas requieren sólo dos registros, ya que la tasa de repetición es el mismo para todas las salidas PWM.

Con salidas PWM controla, los registros específicos que controlan el flanco de subida y de bajada de la salida. Esto permite ambos pulsos PWM, de sentido positivo (cuando el flanco ascendente se produce antes del flanco descendente), y los pulsos PWM que van negativos (cuando el flanco de bajada se produce antes de la flanco ascendente).

Características

• Contador o temporizador.

• Siete registros que permiten controlar hasta 6 simples flancos o 3 flancos dobles de salidas PWM, o una mezcla de ambos tipos. Los registros de los partidos también permiten:

- Funcionamiento continúo con la opción de generación de interrupción.
- Temporizador de parada en coincidencia con la opción de generación de interrupción.
- Temporizador de restauración en coincidencia con la opción de generación de interrupción.

• Soporta un simple controlador de flancos y/o un doble controlador de flancos para las salidas PWM. El controlador simple de flancos para las salidas PWM, todos están en alta al comienzo de cada ciclo a menos que la salida sea una baja constante. El controlador doble de flancos para las salidas PWM controla cualquier posición dentro de un ciclo. Esto permite que ambos pulsos que van pasando positivos y negativos.

• El período y el ancho de pulso puede ser cualquier número del contador del temporizador. Esto permite una flexibilidad total en el equilibrio entre la resolución y la frecuencia de repetición. Todas las salidas PWM se producirán al mismo ritmo de repetición.

• El controlador doble de flancos para las salidas PWM pueden programar para ser impulsos positivos o negativos.

• Actualizaciones del registro se sincronizan con salidas de pulsos para evitar la generación de pulsos erróneos. El software debe "liberar" los nuevos valores antes de que puedan volverse efectivo.

• Se puede utilizar como un temporizador estándar si el modo PWM no está habilitado.

• Un temporizador/contador de 32 bits.

• Dos canales de captura de 32 bits toman un valor instantáneo del temporizador cuando hay transiciones de señal de entrada. Un evento de captura también puede generar opcionalmente una interrupción.

El PWM se configura con los siguientes registros:

1. Energía: En el registro PCONP, setea el bit PCPWM1.

2. Reloj del periférico: En el registro PCLKSEL0, selecciona PCLK_PWM1.

3. Pins: Seleccione los pines PWM a través de los registros PINSEL. Seleccione los modos de los pines para puerto y pines con las funciones PWM1 a través de los registros PINMODE.

4. Interrupciones: Ver los registros PWM1MCR y PWM1CCR para los eventos de captura. Las interrupciones se habilitan en el NVIC utilizando apropiadamente la interrupción del registro Set Enable.

Forma de onda con las reglas para el control simple y doble flanco

En la Figura 5-2 se muestra la salida lógica de los moduladores PWM

La lógica de control del PWM permite la selección de cualquier salida del PWM a través de multiplexores controlados por los bits del registro PWMSELn. Esta implementación soporta hasta N-1 salidas de flanco simple para salidas PWM o (N-1)/2 salidas de flanco dobles, donde N es el número de registros que se implementan. Diferentes tipos de PWM se pueden mezclar, si se desea.

Las selecciones de registro de coincidencia para diversas salidas PWM se muestran en la Tabla 5-32.

Las formas de onda del gráfico de la figura 5-1, muestran un solo ciclo de PWM y demuestra las salidas PWM bajo las siguientes condiciones:

El temporizador está configurado para el modo PWM (contador se restablece a 1).

Match 0 está configurado para restablecer el temporizador/contador cuando se produce un evento.

Los bits de control PWMSEL2 y PWMSEL4 estan seteados.

Los valores de los registros de ajuste son los siguientes:

MR0 = 100 (frecuencia PWM)

MR1 = 41, MR2 = 78 (salida PWM2)

MR3 = 53, MR4 = 27 (salida PWM4)

MR5 = 65 (salida PWM5)

Tabla 5-32: Set/Reset para entradas PWM

Canal PWM	PWM Simple flanco **(PWMSELn = 0)**		PWM Doble flanco **(PWMSELn = 1)**	
	Set	Reset	Set	Reset
1	Match 0	Match 1	Match 0	Match 1
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2	Match 3
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4	Match 5
6	Match 0	Match 6	Match 5	Match 6

Fuente: Tabla 443 del LPC17xx manual del usuario

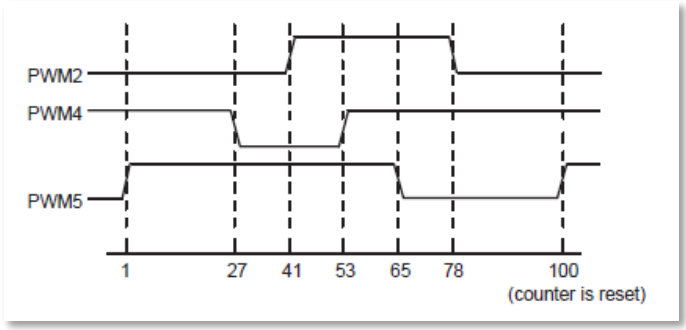


Figura 5-2: Salida lógica de los moduladores PWM.

Reglas para salidas PWM controlados por flanco simple

1. Todos los flancos simples controlan las salidas PWM que van al comienzo de un ciclo del PWM a menos que su valor sea igual a 0.

2. Cada salida PWM será bajo cuando se alcance su valor. Si no se produce ninguna coincidencia (es decir, el valor de coincidencia es mayor que la tasa PWM), la salida PWM se mantiene continuamente en alta.

Reglas para salidas PWM controlados por flancos dobles

Cinco reglas se utilizan para determinar el siguiente valor de una salida PWM cuando un nuevo ciclo está a punto de comenzar:

1. Los valores de concordancia para el siguiente ciclo de PWM se utilizan al final de un ciclo de PWM (un punto de tiempo que es coincidente con el comienzo del siguiente ciclo PWM), excepto como se indica en la regla 3.

2. Un valor igual a 0 o la tasa PWM actual (el mismo que el valor de ajuste de canal 0) tienen el mismo efecto, excepto como se indica en la regla 3. Por ejemplo, una solicitud de un flanco descendente en el comienzo del PWM ciclo tiene el mismo efecto que una solicitud de un flanco descendente al final de un ciclo del PWM.

3. Los valores están cambiando, si uno de los "viejos" valores es igual a la velocidad PWM, se utiliza de nuevo una vez si el que ninguno de los nuevos valores son iguales a 0 o la velocidad PWM, y no había valor antiguo a 0.

4. Si se solicitan tanto un seteo y un borrado una salida PWM, al mismo tiempo, el borrado tiene prioridad.

5. Si un valor partido está fuera de rango (es decir, mayor que el valor de la frecuencia PWM), en ningún caso se produce un evento y ese canal no tiene ningún efecto en la salida. Esto significa que la salida PWM se mantendrá siempre en un estado, permitiendo salidas siempre bajas, siempre altas, o "sin cambio".

La descripción de pines utilizados se muestra en la tabla 5-33.

Tabla 5-33: Descripción de pines

Pin	Tipo	Descripción
PWM1[1]	Salida	Salida PWM canal 1
PWM1[2]	Salida	Salida PWM canal 2
PWM1[3]	Salida	Salida PWM canal 3
PWM1[4]	Salida	Salida PWM canal 4
PWM1[5]	Salida	Salida PWM canal 5
PWM1[6]	Salida	Salida PWM canal 6
PCAP1[1-10]	Entrada	Entradas de captura. El PWM lleva a cabo 2 entradas de captura

Fuente: Tabla 444 del LPC17xx manual del usuario

Control de Motor con PWM

El control de Motor PWM (MCPWM) está optimizado para aplicaciones de control de motores de AC y DC de tres fases, pero puede ser utilizado en muchas otras aplicaciones que necesitan sincronización, cálculo, captura, y la comparación.

Descripción

El MCPWM contiene tres canales independientes, cada uno incluyendo:

- Un temporizador/contador de 32-bits (TC).
- Un registro límite de 32-bits (LIM).
- Un registro de ajuste de 32 bits (MAT).
- Un registro de tiempo muerto (DT) de 10-bits y un contador asociado al tiempo muerto de 10-bits.
- Un registro de captura de 32 bits (CAP).

• Dos salidas moduladas (MCOA y MCOB) con polaridades opuestas.

• Un período de interrupción, una interrupción de ancho de pulso, y una captura de interrupción.

Los pines de entrada MCI0-2 pueden desencadenar la captura del TC o incrementar el TC de un canal. Una entrada mundial abortada puede forzar a todos los canales a estar en estado "pasiva" y causar una interrupción.

Descripción de pines utilizados se muestra en la tabla 5-34.

Tabla 5-34: Descripción de pines

Pin	Tipo	Descripción
MCOA0, MCOA1, MCOA2	Salida	Salida “A” de los canales 0, 1, 2.
MCOB0, MCOB1, MCOB2	Salida	Salida “B” de los canales 0, 1, 2.
MCABORT	Entrada	Cancelación rápida (activo con nivel bajo).
MCI0, MCI1, MCI2	Entrada	Entrada de los canales 0, 1, 2.

Fuente: Tabla 453 del LPC17xx manual del usuario

Configuración de otros módulos para uso del MCPWM

Configure los siguientes registros en otros módulos antes de utilizar el PWM de control del motor:

1. Energía: en el registro PCONP, setea el bit PCMCPWM.

2. Reloj del periférico: en el registro PCLKSEL1 seleccionar PCLK_MCPWM.

3. Pins: seleccionar las funciones MCPWM a través de los registros PINSEL. Seleccione los modos para estos pines a través de los registros PNMODE.

4. Interrupciones: Ver en el manual de usuario en la sección 25.7.3 para PWM relacionados con las interrupciones de control de motores para mayor información. Las interrupciones se pueden activar en el NVIC utilizando apropiadamente la interrupción del registro Set Enable.

Funcionamiento general

El MCPWM incluye 3 canales, cada uno de los cuales controla un par de salidas que a su vez puede controlar algo fuera del chip, como un conjunto de bobinas de un motor. Cada canal incluye un registro para el temporizador/contador (TC) que se incrementa por el reloj del procesador (modo automático) o por un pin de entrada (modo contador).

Cada canal tiene un registro límite que se compara con el valor TC, y cuando se produce una coincidencia el TC se "guarda" en una de dos maneras. En "modo edge-aligned" el TC se pone a 0, mientras que en "modo centrado" cambia el TC a un estado en el que se decrementa en cada transición de reloj del procesador o la entrada del pin hasta que llega a 0, momento en que se inicia a contar de nuevo.

Cada canal también incluye un registro de ajuste que tiene un valor menor que el límite del registro. En el modo edge-aligned el canal de salida cambia cada vez que el TC coincide con cualquiera ajuste o el registro límite, mientras que en el modo center-aligned se encienden sólo cuando coincide con el registro de ajuste.

Así que el registro límite controla el período de las salidas, mientras que el registro de ajuste controla la cantidad de cada período de las salidas que pasan en cada estado. Tener un pequeño valor en el registro límite minimiza "ondulación" si la salida está integrada en un voltaje, y permite que el MCPWM controle los dispositivos que operan a alta velocidad.

El "inconveniente" de los pequeños valores en el registro límite es que reducen la resolución del ciclo de trabajo controlado por el registro de ajuste. Si usted tiene un 8 en el registro límite, el registro de ajuste sólo puede seleccionar el ciclo de trabajo entre 0%, 12,5%, 25%, ..., 87.5%, o 100%. En general, la resolución de cada paso en el valor 1 del ajuste es dividido por el valor límite.

Este compromiso entre la resolución y el período/frecuencia es inherente en el diseño de moduladores de ancho de pulso.

Funciones del CMSIS.

void **PWM_ChannelCmd** (LPC_PWM_TypeDef *PWMx, uint8_t PWMChannel, FunctionalState NewState)

Acción: Habilita/Inhabilita el canal de salida del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, puede ser: LPC_PWM1

[in] PWMChannel: Canal del PWM, puede ser un rango entre 1 a 6.

[in] NewState: Estado, puede ser: ENABLE: Habilita el canal de salida del PWM; DISABLE: Inhabilita el canal de salida del PWM.

void **PWM_ChannelConfig** (LPC_PWM_TypeDef *PWMx, uint8_t PWMChannel, uint8_t ModeOption)

Acción: Configura el modo del flanco para cada canal del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, puede ser: LPC_PWM1

[in] PWMChannel: Canal del PWM, puede ser un rango entre 2 a 6.

[in] ModeOption: El modo de optión PWM, puede ser:

PWM_CHANNEL_SINGLE_EDGE: Modo de flanco simple.

PWM_CHANNEL_DUAL_EDGE: Modo de Flanco dual.

Nota: El canal 2 no puede ser seleccionado para el modo de opción del PWM.

void **PWM_ClearIntPending** (LPC_PWM_TypeDef *PWMx, uint32_t IntFlag)

Acción: Borra la interrupción especifíca pendiente del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, puede ser: LPC_PWM1

[in] IntFlag: Interrupción de flags del PWM, puede ser:

PWM_INTSTAT_MR0: Interrupción del flag para el canal 0.

PWM_INTSTAT_MR1: Interrupción del flag para el canal 1.

PWM_INTSTAT_MR2: Interrupción del flag para el canal 2.

PWM_INTSTAT_MR3: Interrupción del flag para el canal 3.

PWM_INTSTAT_MR4: Interrupción del flag para el canal 4.

PWM_INTSTAT_MR5: Interrupción del flag para el canal 5.

PWM_INTSTAT_MR6: Interrupción del flag para el canal 6.

PWM_INTSTAT_CAP0: Int. del flag para la captura de entrada 0.

PWM_INTSTAT_CAP1: Int. del flag para la captura de entrada 1.

void **PWM_Cmd** (LPC_PWM_TypeDef * PWMx, FunctionalState NewState)

Acción: Habilita/Inhabilita el periférico del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, puede ser: LPC_PWM1

[in] NewState: Estado, puede ser: ENABLE: Habilita el periférico del PWM; DISABLE: Inhabilita el periférico del PWM.

void **PWM_ConfigCapture** (LPC_PWM_TypeDef *PWMx, PWM_CAPTURECFG_Type *PWM_CaptureConfigStruct)

Acción: Configura la captura de entrada del periférico para el PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, puede ser: LPC_PWM1

[in] PWM_CaptureConfigStruct: Puntero a la estructura PWM_CAPTURECFG_Type que contiene la información de la configuración especifíca para la captura de entrada del PWM.

void **PWM_ConfigMatch** (LPC_PWM_TypeDef *PWMx, PWM_MATCHCFG_Type *PWM_MatchConfigStruct)

Acción: Configura el periférico match del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, puede ser: LPC_PWM1

[in] PWM_MatchConfigStruct: Puntero a la estructura PWM_MATCHCFG_Type que contiene la información de la configuración especifíca para la función de ajuste del PWM.

void **PWM_ConfigStructInit** (uint8_t PWMTimerCounterMode, void *PWM_InitStruct)

Acción: Pone cada miembro del PWM_InitStruct member con su valor de default:

If PWMCounterMode = PWM_MODE_TIMER: + PrescaleOption = PWM_TIMER_PRESCALE_USVAL + PrescaleValue = 1

If PWMCounterMode = PWM_MODE_COUNTER: + CountInputSelect = PWM_COUNTER_PCAP1_0 + CounterOption = PWM_COUNTER_RISING.

Parámetros:

[in] PWMTimerCounterMode: Temporizador o contador, puede ser:

PWM_MODE_TIMER: Contador del periférico PWM esta en modo Timer.

PWM_MODE_COUNTER: Counter of PWM peripheral is in Counter mode Contador del periférico PWM esta en modo Counter.

[in] PWM_InitStruct: Puntero a la estructura (PWM_TIMERCFG_Type o PWM_COUNTERCFG_Type) podrá ser inicializada.

Nota: El puntero al PWM_InitStruct asignará a las dos correspondientes estructuras (PWM_TIMERCFG_Type o PWM_COUNTERCFG_Type) incluyendo PWMTimerCounterMode.

void **PWM_CounterCmd** (LPC_PWM_TypeDef *PWMx, FunctionalState NewState)

Acción: Habilita/Inhabilita el contador del periférico del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, pueder ser: LPC_PWM1.

[in] NewState: El estado de ésta función, puede ser:

ENABLE: Habilita el contador del periférico del PWM.

DISABLE: Desabilita el contador del periférico del PWM.

void **PWM_DeInit** (LPC_PWM_TypeDef *PWMx)

Acción: Inicializa los valores del registro del periférico del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico del PWM, puede ser: LPC_PWM1

uint32_t **PWM_GetCaptureValue**(LPC_PWM_TypeDef * PWMx, uint8_t CaptureChannel)

Acción: Captura el valor del registro para la lectura del periférico del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico del PWM, puede ser: LPC_PWM1

[in] CaptureChannel: Captura el número del canal, pueder ser de un rango de 0 a 1.

Retorna: El valor de la captura del registro.

IntStatus **PWM_GetIntStatus**(LPC_PWM_TypeDef *PWMx, uint32_t IntFlag)

Acción: Checkea los flags específicos de interrupción del PWM.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, pueder ser: LPC_PWM1.

[in] IntFlag: Interrupción de flags del PWM, puede ser:

PWM_INTSTAT_MR0: Interrupción del flag para el canal 0.

PWM_INTSTAT_MR1: Interrupción del flag para el canal 1.

PWM_INTSTAT_MR2: Interrupción del flag para el canal 2.

PWM_INTSTAT_MR3: Interrupción del flag para el canal 3.

PWM_INTSTAT_MR4: Interrupción del flag para el canal 4.

PWM_INTSTAT_MR5: Interrupción del flag para el canal 5.

PWM_INTSTAT_MR6: Interrupción del flag para el canal 6.

PWM_INTSTAT_CAP0: Interrupción del flag para la captura de entrada 0.

PWM_INTSTAT_CAP1: Interrupción del flag para la captura de entrada 1.

Retorna: El estado de la interrupción del flag del PWM (SET o RESET).

void **PWM_Init** (LPC_PWM_TypeDef * PWMx, uint32_t PWMTimerCounterMode, void *PWM_ConfigStruct)

Acción: Initializa el periférico correspondiente del PWMx especificando los parámetros en el PWM_ConfigStruct.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, pueder ser: LPC_PWM1.

[in] PWMTimerCounterMode: Temporizador o contador, puede ser:

PWM_MODE_TIMER: Contador del periférico PWM esta en modo Timer.

PWM_MODE_COUNTER: Counter of PWM peripheral is in Counter mode Contador del periférico PWM esta en modo Counter.

[in] PWM_ConfigStruct: Puntero a la estructura (PWM_TIMERCFG_Type or PWM_COUNTERCFG_Type) que será inicializada.

void **PWM_MatchUpdate** (LPC_PWM_TypeDef *PWMx, uint8_t MatchChannel, uint32_t MatchValue, uint8_t UpdateType)

Acción: Actualiza el valor de cada canal del PWM, actualizando el tipo de opción.

Parámetros:

[in] PWMx: Selecciona el periférico PWM, pueder ser: LPC_PWM1.

[in] MatchChannel: Canal de ajuste.

[in] MatchValue: Valor de ajuste.

[in] UpdateType: Actualización del tipo, puede ser:

PWM_MATCH_UPDATE_NOW: El valor se actualizará para este canal inmediatamente.

PWM_MATCH_UPDATE_NEXT_RST: El valor se actualizará para este canal en el siguiente reinicio producido por un evento en el PWM.

void **PWM_ResetCounter** (LPC_PWM_TypeDef *PWMx)

Acción: Resetea el contador del periférico del PWM.

Parámetro:

[in] PWMx: Selecciona el periférico PWM, pueder ser: LPC_PWM1.

5.1.g. Analog-to-Digital Converter (ADC)

La señal de reloj para el conversor A/D es provista por el clock APB. Un divisor programable que permite cambiar la frecuencia hasta un máximo de 13 MHz. El método de conversión es de aproximaciones sucesivas. El caso más desfavorable requiere de 65 pulsos de reloj.

Características

•Conversor de aproximaciones sucesivas de 12- bits.

• 8 entradas multiplexadas.

• Modo de ahorro de energía (power-down).

• Rango de entrada entre VREFN y VREFP (valor típico 3V).

• La tasa de conversión de 12-bits es de 200 kHz.

• El modo de ráfaga de conversión para las entradas individuales o múltiples.

• La conversión opcional de transición en el pin de entrada o la señal de ajuste de temporizador.

El ADC se configura con los siguientes registros:

1. Energía: En el registro PCONP, setea el bit PCADC.

2. Reloj: En el registro PCLKSEL0, seleccione PCLK_ADC. Para escalar el reloj para el ADC, consulte los bits del CLKDIV.

3. Pins: Habilitar los pines del ADC0 través de registros PINSEL. Seleccione los modos de pines para los pines del puerto con funciones del ADC0 a través de los registros PINMODE.

4. Interrupciones: Para habilitar las interrupciones en el ADC. Las interrupciones se habilitan en el NVIC utilizando apropiadamente la interrupción del registro Set Enable. Inhabilitar la interrupción ADC en el NVIC utilizando apropiadamente la interrupción del registro Set Enable.

5. DMA: Para las conexiones del sistema GPDMA.

Descripción de pines

Tabla 5-35: ADC descripción de pines

Pin	Tipo	Descripción
AD0.7 a AD0.0	Entrada	Entrada analógica. Precaución: el nivel de la señal de entrada no puede ser superior a Vdda.
Vrefp, Vrefn	Referencias	Voltajes de referencia.
Vdda, Vssa	Power	Fuente de alimentación y masa analógica. Valores típicos iguales a Vdd y Vss.

Fuente: Tabla 529 lpcxx manual del usuario

Funciones del CMSIS.

void **ADC_BurstCmd** (LPC_ADC_TypeDef * ADCx, FunctionalState NewState)

Acción: Modo configuración Burst del ADC.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] NewState: puede asumir los valores*: 1= S*eteo del modo Burst; 0= Reset del modo Burst.

void **ADC_ChannelCmd** (LPC_ADC_TypeDef* ADCx, uint8_t Channel, FunctionalState NewState)

Acción: Habilita/Inhabilita el número del canal del ADC.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] Channel: Número del canal.

[in] NewState: Habilitado o Inhabilitado.

uint16_t **ADC_ChannelGetData** (LPC_ADC_TypeDef* ADCx, uint8_t channel)

Acción: Captura el resultado del ADC.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] cannel: Número del canal, puede ser 0...7

Retorna: El dato convertido.

FlagStatus **ADC_ChannelGetStatus**(LPC_ADC_TypeDef * ADCx, uint8_t channel, uint32_t StatusType)

Acción: Captura el estado del canal del registro de dato del ADC.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] cannel: Número del canal, puede ser 0...7

[in] StatusType: puede asumir los valores: 0= Estado Burst; 1= Estado Done.

Retorna: SET / RESET

void **ADC_DeInit**(LPC_ADC_TypeDef * ADCx)

Acción: Borra el ADC.

Parámetros:

[in] *ADCx*: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

void **ADC_EdgeStartConfig**(LPC_ADC_TypeDef * ADCx, uint8_t EdgeOption)

Acción: Setea la configuración de inicio del flanco.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] EdgeOption: Puede asumir los valores

0= ADC_START_ON_RISING;

1= ADC_START_ON_FALLING

uint32_t **ADC_GetData**(uint32_t channel)

Acción: Captura el resultado de la conversión del registro de dato del A/D.

Parámetros:

[in] channel: Número que se desea leer de nuevo el resultado

Retorna: El resultado de la conversión.

uint32_t **ADC_GlobalGetData**(LPC_ADC_TypeDef * ADCx)

Acción: Captura el dato del registro global del ADC.

Parámetro:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

Retorna: El resultado de la conversion.

FlagStatus **ADC_GlobalGetStatus** (LPC_ADC_TypeDef * ADCx, uint32_t StatusType)

Acción: Captura el estado del canal del registro de dato global del ADC.

Parámetro:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in]StatusType: 0= Estado Burst; 1= Estado Done.

Retorna: SET/RESET

void **ADC_Init** (LPC_ADC_TypeDef * ADCx, uint32_t rate)

Acción: Inicializa el ADC + Setea el bit PCADC + Setea el reloj del ADC + Setea la frecuencia del reloj.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] rate: Taza de la conversión del ADC, puede ser <=200KHz

void **ADC_IntConfig** (LPC_ADC_TypeDef * ADCx, ADC_TYPE_INT_OPT IntType, FunctionalState NewState

Acción: Configuración de la interrupción del ADC.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] IntType: Tipo de interrupción, puede ser:

ADC_ADINTEN0: Interrupción del canal 0

ADC_ADINTEN1: Interrupción del canal 1 ...

ADC_ADINTEN7: Interrupción del canal 7

ADC_ADGINTEN: Canal individual/ flag global genera una interrupción.

[in] NewState: SET : Habilita la interrupción del ADC; RESET: Desabilita la interrupción del ADC.

void **ADC_PowerdownCmd** (LPC_ADC_TypeDef * ADCx, FunctionalState NewState)

Acción: Setea la conversión del ADC en modo power.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] NewState: 1: Conversión optional del ADC; 0: Conversión en modo power-down del ADC.

void ADC_StartCmd (LPC_ADC_TypeDef *ADCx, uint8_t start_mode)

Acción: Setea el inicio del modo del ADC.

Parámetros:

[in] ADCx: Puntero al LPC_ADC_TypeDef, puede ser: LPC_ADC

[in] start_mode: El modo inicial elije una definición de los tipos de modos 'ADC_START_OPT', que puede ser:

ADC_START_CONTINUOUS

ADC_START_NOW

ADC_START_ON_EINT0

ADC_START_ON_CAP01

ADC_START_ON_MAT01

ADC_START_ON_MAT03

ADC_START_ON_MAT10

ADC_START_ON_MAT11

5.1.h. Digital-to-Analog Converter (DAC)

Contador de operación del DMA

Cuando el contador habilita el bit CNT_ENA en DACCTRL se setea, un contador de 16 bits comenzará la cuenta regresiva, a la tasa seleccionada por PCLK_DAC, a partir del valor programado en el registro DACCNTVAL. El contador se decrementa cada vez que el contador llegue a cero, el contador se volverá a cargar con el valor de DACCNTVAL y la DMA requiere el bit INT_DMA_REQ seteará el hardware.

Tenga en cuenta que los contenidos de los registros DACCTRL y DACCNTVAL son accesibles para lectura y escritura, pero el propio temporizador no es accesible, para lectura o escritura.

Si el bit del DMA_ENA se setea en el registro DACCTRL, el requerimiento del DAC DMA se dirigirá a la GPDMA. Cuando se borre el bit del DMA_ENA del estado predeterminado después de un reinicio, el requerimiento del DAC DMA son bloqueado.

Doble buffering

El Double-buffering está habilitado sólo si ambos, el CNT_ENA y los bits DBLBUF_ENA se fijan en DACCTRL. En este caso, cualquier escritura al registro DACR sólo carga el pre-buffer, que comparte su dirección de registro con el registro DACR. El DACR sí se carga desde el pre-buffer siempre que el contador llegue a cero y el requerimiento de la DMA está establecido. Al mismo tiempo, el contador se vuelve a cargar con el valor del registro COUNTVAL.

Leyendo el registro DACR sólo devolverá el contenido del registro del DACR, en sí, no el contenido del registro de pre-buffer.

Si bien el bit del CNT_ENA o DBLBUF_ENA son 0, ninguna escritura en la dirección DACR ira directamente al registro DACR.

Características

- Conversor D/A de 10-bits.
- Método por cadena de resistencias.
- Salida con buffer.

• Modo Power-down.

• Velocidad seleccionable vs. Potencia.

• Tasa de actualización máximo de 1 MHz.

El DAC se configura con los siguientes registros:

1. Energía: El DAC está siempre conectado a V_{DDA}. El acceso al registro está determinado por la configuración por el PINSEL y PINMODE.

2. Reloj: En el registro PCLKSEL0, seleccione PCLK_DAC.

3. Pins: Habilitar el pin del DAC a través de los registros del PINSEL. Seleccione el modo de pin para puerto y pin del DAC a través de los registros PINMODE. Esto debe hacerse antes de acceder a cualquier registro del DAC.

4. DMA: El DAC se puede conectar al controlador GPDMA. Para las conexiones GPDMA.

Descripción de pines:

Tabla 5-36: DAC descripción de pines

Pin	Tipo	Descripción
AOUT	Salida	Salida analógica
Vrefp, Vrefn	Referencias	Voltajes de referencia
Vdda, Vssa	Power	Fuente de alimentación y masa analógica. Valores típicos iguales a Vdd y Vss

Fuente: Tabla 537 lpcxx manual del usuario

Función del CMSIS

```
void DAC_ConfigDAConverterControl ( LPC_DAC_TypeDef * DACx,
DAC_CONVERTER_CFG_Type * DAC_ConverterConfigStruct)
```

Habilita la operación y el control del temporizador de la DMA.

Parámetros:

[in] DACx: Puntero al LPC_DAC_TypeDef, puede ser: LPC_DAC

[in] DAC_ConverterConfigStruct: Puntero al DAC_CONVERTER_CFG_Type

DBLBUF_ENA : Habilita/Inhabilita el buffering doble del DACR.

CNT_ENA : Habilita/Inhabilita el contador de time out.

DMA_ENA : Habilita/Inhabilita el acceso a la DMA.

void **DAC_Init**(LPC_DAC_TypeDef * DACx)

Acción: Configuración inicial del DAC.

La corriente máxima es 700 µA.

Valor del AOUT es 0.

Parámetros:

[in] DACx: Puntero al LPC_DAC_TypeDef, puede ser: LPC_DAC

void **DAC_SetBias** (LPC_DAC_TypeDef * DACx, uint32_t bias)

Acción: Setea la corriente máxima del DAC.

Parámetros:

[in] DACx: Puntero al LPC_DAC_TypeDef, puede ser: LPC_DAC

[in] bias: 0= 700 µA; 1= 350 µA

void DAC_SetDMATimeOut (LPC_DAC_TypeDef * DACx, uint32_t time_out)

Acción: Setea el valor para el contador por interrupción/DMA.
Parámetros:

[in] DACx: Puntero al LPC_DAC_TypeDef, puede ser: LPC_DAC

[in] time_out: Contador por interrupción de time out/DMA.

void **DAC_UpdateValue** (LPC_DAC_TypeDef * DACx, uint32_t dac_value)

Acción: Actualiza el valor del DAC.

Parámetros:

[in] DACx: Puntero al LPC_DAC_TypeDef, puede ser: LPC_DAC

[in] dac_value: Valor de 10-bit convertido para la salida.

CAPITULO 6:

LPC1769 ARM Cortex-M3

6.1 INTRODUCCIÓN

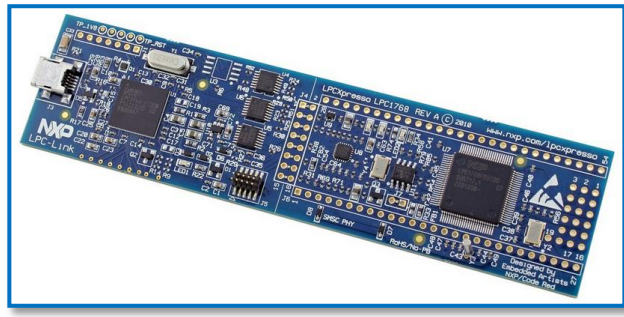


Figura 6-1: Placa LPC1769 NXP

La LPC1769 cuenta con microcontrolador ARM Cortex-M3 de NXP, el cual ha sido diseñado para hacer fácil la accesibilidad para iniciarse con microcontroladores Cortex-M3. Esta placa combina un Cortex-M3 con un depurador JTAG.

La LPC1769 tiene 64 kB SRAM, 512 kB Flash, 4xUART, 3xI2C, SPI, 2xSSP, 2xCAN, PWM, USB 2.0 Device/Host/OTG, RTC, Ethernet, I2S, etc.

Especificaciones

- Procesador: Microcontrolador NXP Cortex-M3 LPC1769 en empaque LQFP100.
- Flash: 512 kB.
- Memoria de Datos: 64 kB.
- Reloj Cristal de 12.000 para la CPU.

- Dimensiones: 35 x 140 mm.
- Alimentación: 3.15V-3.3V externa o USB via JTAG (LPC-LINK).
- Conectores: Todos los pines relevantes de la LPC1769 están disponibles en un conector de expansión (2x27 filas de pines, espaciamiento 100 mil, 900 mil entre filas).
- Otros:
    - Funcionalidad Embedded JTAG (LPC-LINK) via LPCXpresso toolchain.
    - La LPC-LINK puede ser conectada a procesadores externos después de hacer modificaciones a la placa LPCXpresso.
    - LED en PIO0_22.

6.2 ENTORNO DE DESARROLLO IDE LPCXPRESSO®

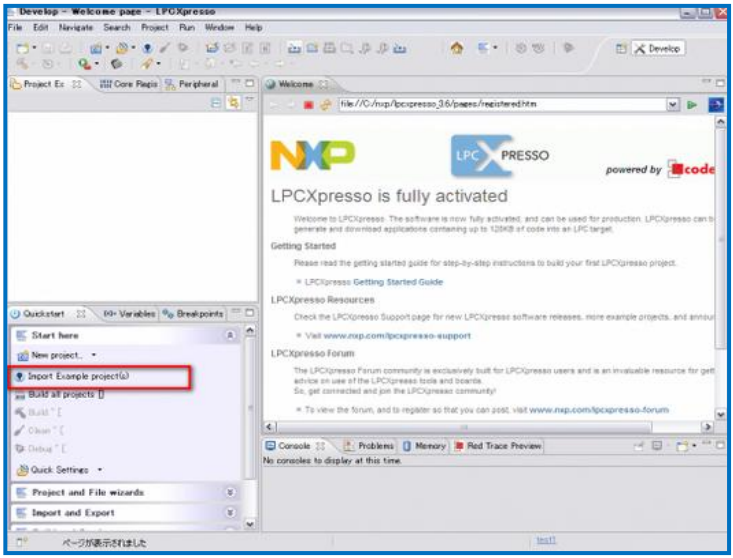


Figura 6-2: Entorno de desarrollo de software LPCXpresso

Es un entorno de desarrollo de software altamente integrado para microcontroladores LPC de NXP que incluye todas las herramientas necesarias para desarrollar soluciones de software de alta calidad de manera oportuna y rentable. LPCXpresso está basado en Eclipse y tiene muchas mejoras para simplificar el desarrollo con microcontroladores LPC de NXP.

Al momento de comenzar un proyecto nuevo es necesario realizar las configuraciones iniciales para trabajar en el IDE. En este caso el LCPXpresso sobre un micro NXP LPC1769.

Workspace

El workspace es el entorno de trabajo. Se refiere simplemente a una carpeta del directorio donde se alojarán todos los archivos de los distintos proyectos que se realizan.

Al iniciar el programa suele preguntar la ruta del workspace con el que se quiere trabajar.

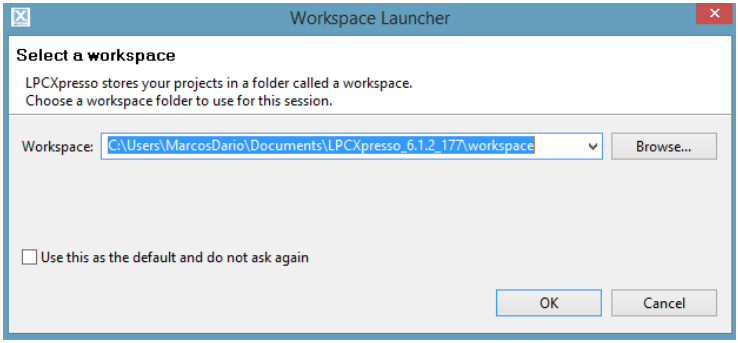


Figura: 6-3: Selección de la ruta del workspace

Una vez abierto se puede cambiar o crear uno nuevo.

Figura: 6-4: Selección del workspace

Además se puede elegir otro workspace de los que estén creados mediante **File>>Switch Workspace**.

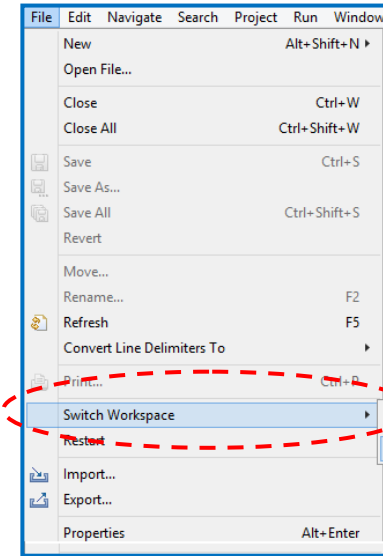


Figura 6-5: Selección para diferentes workspace

Importar bibliotecas

Una vez seleccionada la carpeta donde se alojará el proyecto (workspace), se deben cargar las bibliotecas del dispositivo que se utilizará. En este caso del LPC1769 y cómo se trabaja con CMSIS se utilizan las bibliotecas de CMSIS de este microcontrolador:

CMSIS_CORE _LPC17xx

Para incorporar cualquier biblioteca al workspace, las cuales se las puede buscar dentro de las carpetas de los programas de ejemplos de cada micro, seleccionar en el **Start here Import Project(s)>>Archive>>Browse….**

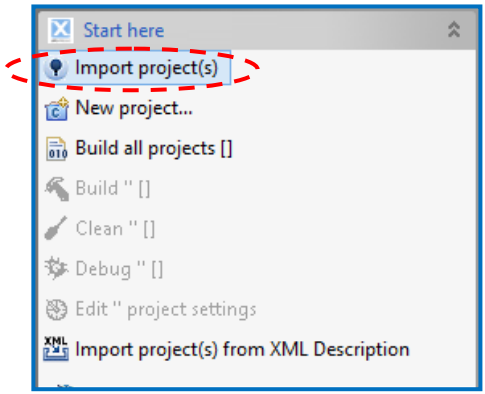


Figura 6-6: Importación de la librería CMSIS

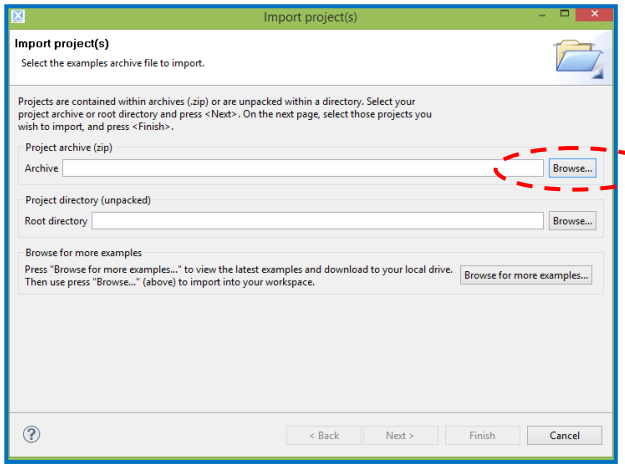


Figura 6-7: Busqueda de la librería CMSIS

Luego buscar la carpeta donde están guardadas las librerías, en este caso CMSIS_CORE_LPC17xx en:

C:\nxp\LPCXpresso_6.1.0_164\lpcxpresso\Examples\CMSIS_CORE \CMSIS_CORE_Latest.zip >>Abrir…

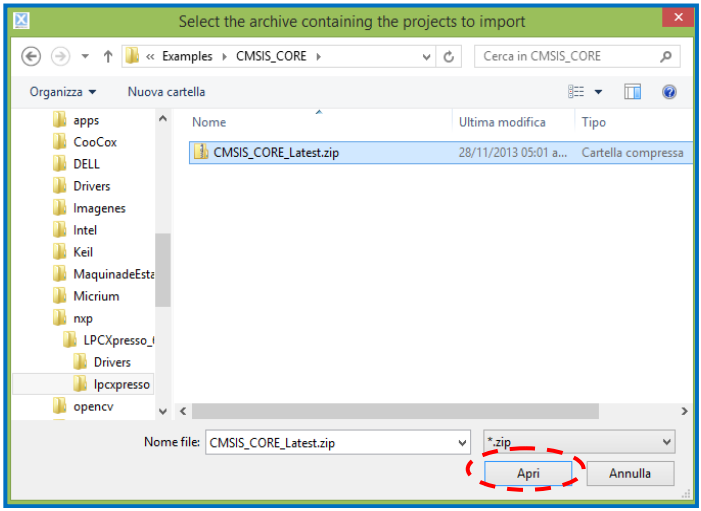


Figura 6-8: CMSIS_CORE_Lastet.zip

Abrir el archivo .zip para posteriormente solo seleccionar la CMSIS que se desea utilizar **Next >> Deselect All >> CMSIS_CORE_LPC17xx (…) >> Finish**

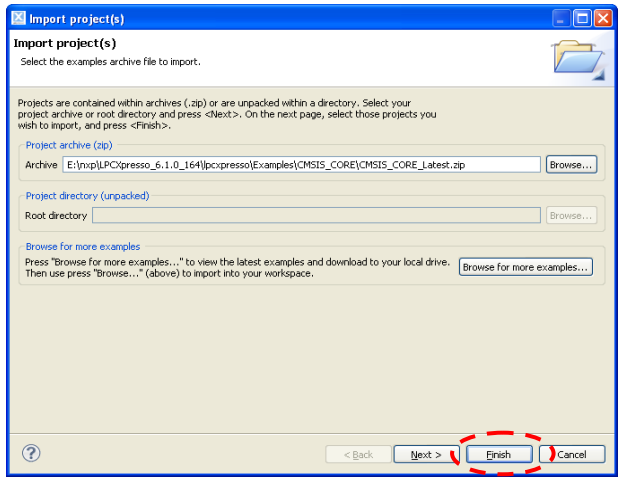


Figura 6-9: Selección de la CMSIS_CORE_Lastet.zip

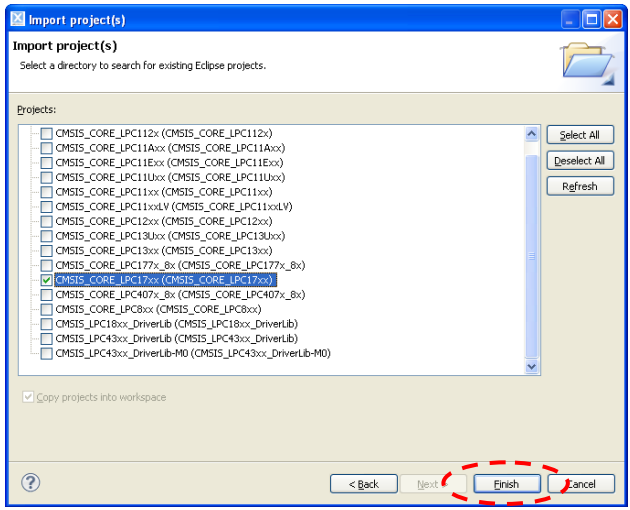


Figura 6-10: Selección de la CMSIS_CORE_LPC17xx

Si todo salió bien aparecerá un nuevo proyecto en el **Proyect Explorer**.

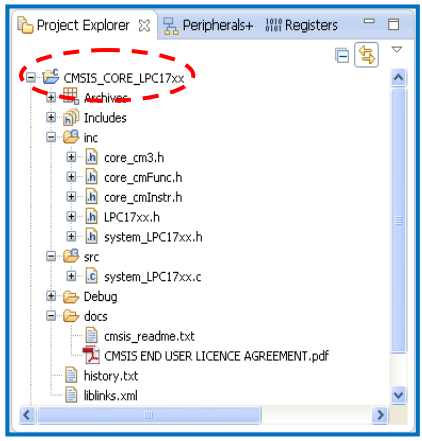


Figura 6-11: CMSIS_CORE_LPC17xx importado en el explorador del IDE

Crear un proyecto

Existen dos tipos de proyectos diferentes en este IDE:

- Las bibliotecas estáticas
- Los programas de aplicación

Las bibliotecas estáticas son todas las funciones y archivos .h que sirven para luego incluir en otros proyectos. De las bibliotecas no se saca código tipo HEX sino que solo se sacan código objeto (.o) con las funciones. En el caso de que el programa las utilice, el linker las va a buscar ahí.

La idea es que los programas pueden linkearse a las bibliotecas y así poder mantener organizado el código.

La biblioteca estática luego de compilar cada uno de sus archivos .c y generar salidas .o, junta todo en un archivo .a. Este archivo contiene el código objeto de todas las funciones declaradas en los diferentes .c. El proyecto compila el main.c, las funciones.c y genera

los .o. Luego el linker junta los .o y los .a y produce el .hex que es el que finalmente se le baja al microcontrolador.

La manera más fácil de crear algún proyecto es seleccionar en el **Start here -> New proyect** y seleccionar el tipo de proyecto. Para este caso LPCXpresso C Project. Luego se selecciona la familia de microcontrolador con la que se va a trabajar (NXP LCP175_x6x), el tipo de proyecto (C Project) y presionamos **->Next.**

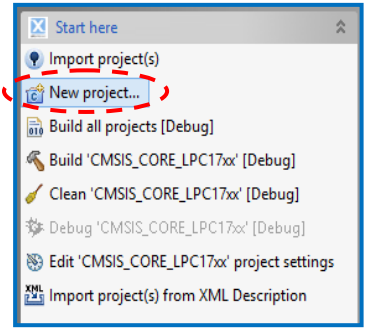


Figura 6-12: Creación de un nuevo proyecto

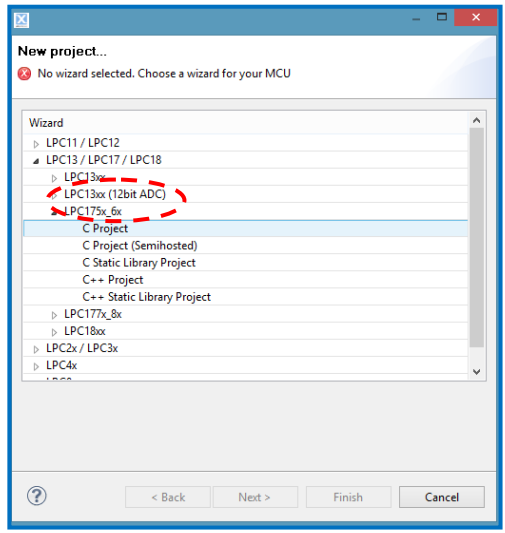


Figura 6-13: Selección de la LPC175x_6x

Colocamos el nombre de nuestro primer proyecto y seleccionamos el microcontrolador con el que trabajaremos (NXP LPC 1769) **->Next.**

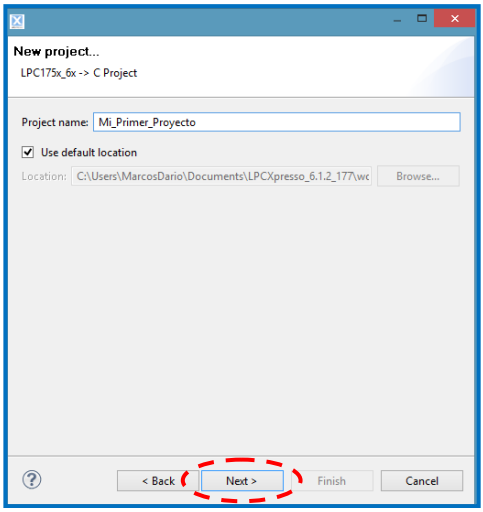


Figura 6-14: Nombre de nuestro proyecto

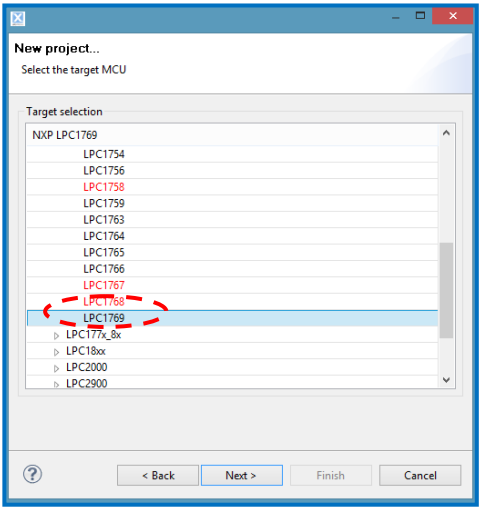


Figura 6-15: Selección para la target LPC1769

Luego linkeamos la Librería CMSIS Core que se habían agregado anteriormente con nuestro proyecto. Para eso en el desplegable **CMSIS Library to link project to**, se selecciona **CMSIS_CORE_LPC17xx**,

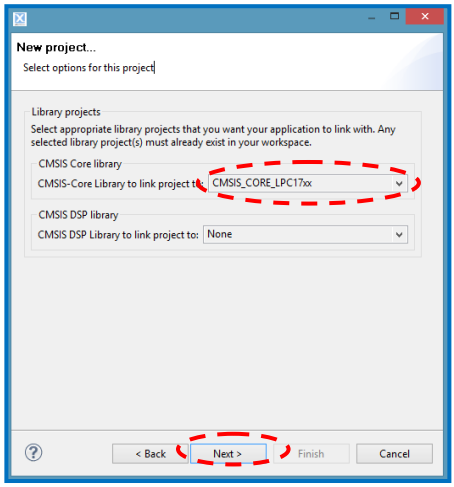


Figura 6-16: Linker nuestro proyecto con la CMSIS_CORE_LPC17xx

Por último creamos la carpeta **"inc"** la cual contendrá todos los *headers* del proyecto, presionamos el botón de **->Finish** (figura 6-17) y de esta manera habremos creado nuestro primer proyecto en el LPCXpresso.

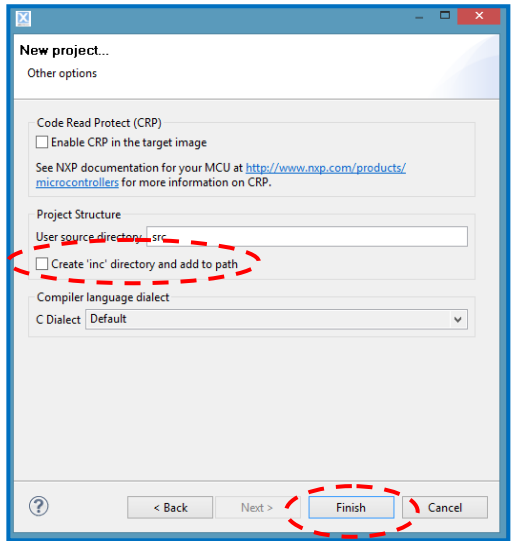


Figura 6-17: Creación de la carpeta "inc"

Como punto de control se puede realizar un **Built All Project**, para ver si todo funciona correctamente.

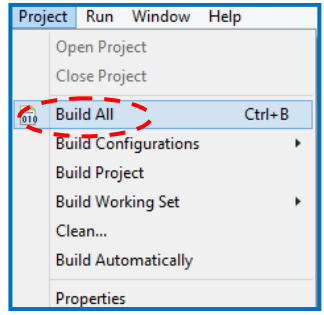


Figura 6-18: Compilación de todo el proyecto

Headers .h

Son archivos .h en los cuales se incluye los prototipos de funciones y definiciones. Para crearlos debemos realizar click con el botón derecho dentro de la carpeta **inc** y seleccionar **New-> Headers File.**

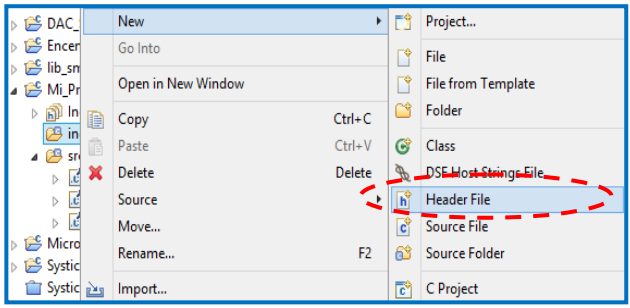


Figura 6-19: Creación de los headers files

Archivos . c

Son archivos complilables que incluyen las implementaciones de las funciones. Para crearlos debemos realizar click con el botón derecho dentro de la carpeta **src** y seleccionar **New-> Source File.**

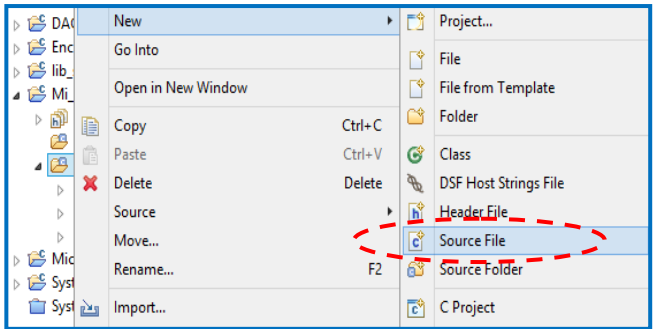


Figura 6-20: Creación de los souces files

6.3 EJERCICIOS SOBRE LPC1769

6.3.a Ejercicio 1 – Manejo de Entradas y Salidas

Se realizara una placa que contenga cuatro entradas (pulsadores) y cuatro salidas (leds) para ser conectada a la placa LPCXpresso.

Para ello es necesario conocer los pines disponibles en la placa y que podemos utilizar en nuestra expansión. La información la extremos de los esquemáticos de la placa LPCXpresso Rev.B

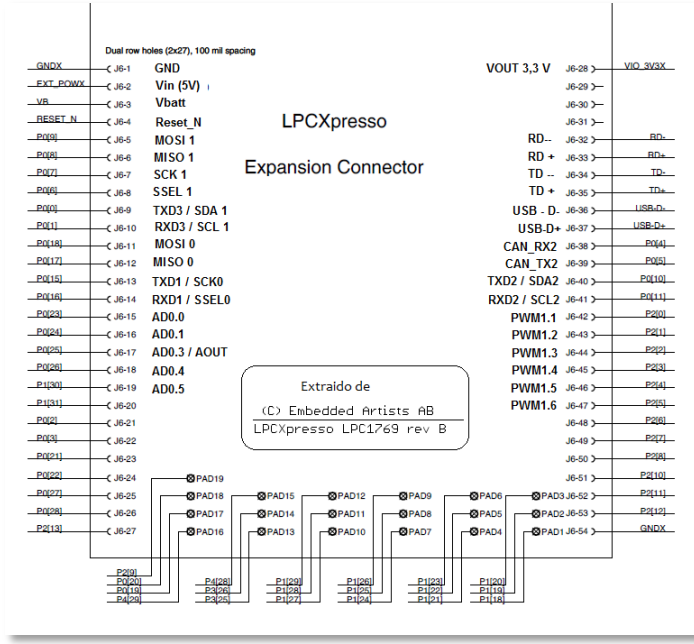
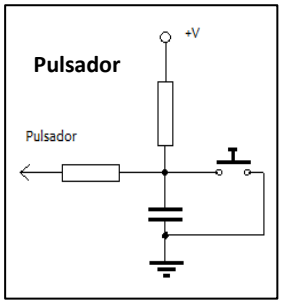
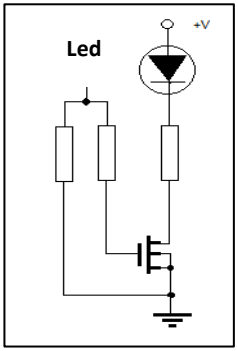


Figura 6-21: Pin-out de la placa LPCXpresso

Los elementos que queremos conectar son: cuatro leds, cuatro pulsadores y un display LCD, según los siguientes esquemas de conexión:





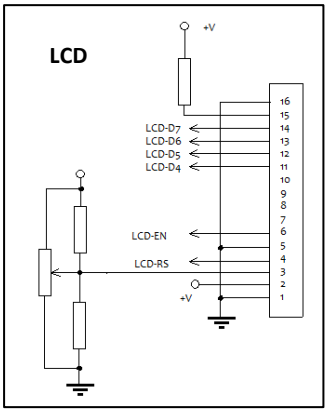


Figura 6-22: Esquemáticos de la conexión de periféricos (a) Pulsador, (b) Led y (c) Display LCD

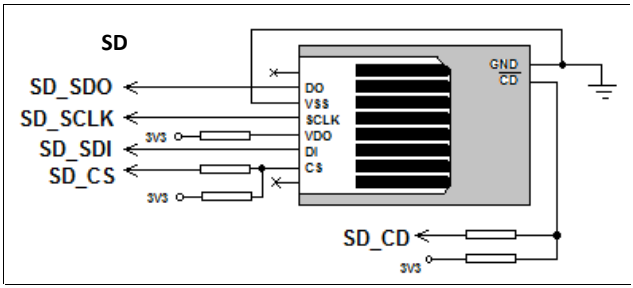


Figura 6-23: Conexión Lector/Grabador Tarjeta SD

Como vemos ya en el esquema se han indicado los puertos donde serán conectados

Los puntos de conexión se muestran en la siguiente tabla:

Tabla 6-1: Detalle de conexionado de los periféricos a la placa LPCXpresso

Señal	Conector	Pin	Detalle
SD_SDO	J6-12	Px[y]	Data output
SD_SCLK	J6-13	Px[y]	Clock
SD_SDI	J6-11	Px[y]	Data Input
SD_CS	J6-13	Px[y]	Chip Select
SD_CD	GND	Px[y]	Detección de tarjeta
Pulsador1	J6-49	Px[y]	"1" sin pulsar, "0" Pulsado
Pulsador2	J6-48	Px[y]	
Pulsador3	J6-47	Px[y]	
Pulsador4	J6-46	Px[y]	
Led1	J6-53	Px[y]	"1" enciende, "0" apaga
Led2	J6-52	Px[y]	
Led3	J6-51	Px[y]	
Led4	J6-50	Px[y]	
LCD-RS	J6-9	Px[y]	Register Select
LCD-EN	J6-10	Px[y]	Enable
LCD-D7	J6-8	Px[y]	Datos
LCD-D6	J6-7	Px[y]	
LCD-D5	J6-6	Px[y]	
LCD-D4	J6-5	Px[y]	

Nota: Solo describiremos el código de algunos de estos periféricos pero remitimos al lector a la página www.tecno.unca.edu.ar/lab_se/str_ejemplos.zip donde podrá encontrar los ejemplos completos.

Los drivers se han codificado en lenguaje C y a continuación se detalla el código correspondiente a los leds y pulsadores.

```
/*================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: leds.h
Autores: Matias Ferraro - Marcos Aranda
Descripción:    Prototipos    de    funciones    y    definiciones
correspondientes para los puertos de salidas para la LPC 1769
================================================================*/

#ifndef LEDS_H_
#define LEDS_H_

#define PIN_3    3
#define PIN_0    0
```

```
#define PIN_1    1
#define PIN_2    2
#define PIN_4    4
#define PIN_5    5
#define PIN_7    7
#define PIN_6    6
#define PIN_8    8
#define PIN_9    9
#define PIN_10   10
#define PIN_11   11
#define PIN_12   12
#define PIN_22   22
#define PIN_27   27
#define PIN_28   28
#define PIN_13   13
#define PTO_0    0
#define PTO_2    2

/*Inicializa un pin de un puerto como salida*/
void Init_Port_Salida(int puerto,int pin);

/*Pone en alto el nivel de salida de un pin*/
void Output_High(int puerto,int pin);

/*Pone en bajo el nivel de salida de un pin*/
void Output_Low(int puerto,int pin);

/*Invierte el nivel de salida de un pin*/
void Toggle(int puerto,int pin);

#endif


/*=================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: leds.c
Autores: Matias Ferraro - Marcos Aranda
Descripción: Implementaciones de las funciones correspondientes
para los puertos de salidas para la LPC 1769
=================================================================*/
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include "leds.h"

void Init_Port_Salida(int puerto, int pin)
{
    switch(puerto)
    {
          case  0: LPC_GPIO0->FIODIR |=(1<<pin);
                break;
          case  1: LPC_GPIO1->FIODIR |=(1<<pin);
                break;
          case  2: LPC_GPIO2->FIODIR |=(1<<pin);
                break;
          case  3: LPC_GPIO3->FIODIR |=(1<<pin);
                break;
          case  4: LPC_GPIO4->FIODIR |=(1<<pin);
                break;
```

```
            default:
                  break;
    }
}

void Output_High(int puerto,int pin)
{
     switch(puerto)
     {
           case  0: LPC_GPIO0->FIOSET |=(1<<pin);
                    break;
           case  1: LPC_GPIO1->FIOSET |=(1<<pin);
                    break;
           case  2: LPC_GPIO2->FIOSET |=(1<<pin);
                    break;
           case  3: LPC_GPIO3->FIOSET |=(1<<pin);
                    break;
           case  4: LPC_GPIO4->FIOSET |=(1<<pin);
                    break;
           default:
                    break;
    }
}
void Output_Low(int puerto,int pin)
{
     switch(puerto)
     {
           case  0: LPC_GPIO0->FIOCLR |= (1<<pin);
                    break;
           case  1: LPC_GPIO1->FIOCLR |= (1<<pin);
                    break;
           case  2: LPC_GPIO2->FIOCLR |= (1<<pin);
                    break;
           case  3: LPC_GPIO3->FIOCLR |= (1<<pin);
                    break;
           case  4: LPC_GPIO4->FIOCLR |= (1<<pin);
                    break;
           default:
                     break;
    }
}
void Toggle(int puerto,int pin)
{
    switch(puerto)
    {
            case  0: LPC_GPIO0->FIOPIN ^= (1<<pin);
                     break;
            case  1: LPC_GPIO1->FIOPIN ^= (1<<pin);
                     break;
            case  2: LPC_GPIO2->FIOPIN ^= (1<<pin);
                     break;
            case  3: LPC_GPIO3->FIOPIN ^= (1<<pin);
                     break;
            case  4: LPC_GPIO4->FIOPIN ^= (1<<pin);
                     break;
            default:
                     break;
    }
}
```

```
/*=================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: pulsadores.h
Autores: Matias Ferraro – Marcos Aranda
Descripción:    Prototipos    de    funciones    y    definiciones
correspondientes para los puertos de entradas para la LPC 1769
=================================================================*/

#ifndef PULSADORES_H_
#define PULSADORES_H_

#define PIN_3    3
#define PIN_0    0
#define PIN_1    1
#define PIN_2    2
#define PIN_4    4
#define PIN_5    5
#define PIN_7    7
#define PIN_6    6
#define PIN_8    8
#define PIN_9    9
#define PIN_10   10
#define PIN_11   11
#define PIN_12   12
#define PIN_22   22
#define PIN_27   27
#define PIN_28   28
#define PIN_13   13
#define PTO_0    0
#define PTO_2    2
#define ALTO     1
#define BAJO     0

/*Inicializa un pin de un puerto como entrada*/
void Init_Port_Entrada(int puerto,int pin);

/*Indaga si un pin de un puerto esta en alto*/
uint32_t Press_Pulsador(int puerto,int pin);

void Enviar_Dato_Puerto(int puerto,unsigned int dato);

#endif



/*=================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: pulsadores.c
Autores: Matias Ferraro – Marcos Aranda
Descripción: Implementaciones de las funciones correspondientes
para los puertos de entradas para la LPC 1769
=================================================================*/

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include "pulsadores.h"
```

```
void Init_Port_Entrada(int puerto, int pin)
{

    switch(puerto)
    {
         case  0: LPC_GPIO0->FIODIR &=~(1<<pin);
                 break;
         case  1: LPC_GPIO1->FIODIR &=~(1<<pin);
                  break;
         case  2: LPC_GPIO2->FIODIR &=~(1<<pin);
                  break;
         case  3: LPC_GPIO3->FIODIR &=~(1<<pin);
                  break;
         case  4: LPC_GPIO4->FIODIR &=~(1<<pin);
                  break;
         default:
                  break
    }

}

uint32_t Press_Pulsador(int puerto,int pin)
{
     switch(puerto)
     {
           case  0: return((LPC_GPIO0->FIOPIN & (0X1<<pin))== 0);
                  break;
           case  1: return((LPC_GPIO1->FIOPIN & (0X1<<pin))== 0);
                  break;
           case  2: return((LPC_GPIO2->FIOPIN & (0X1<<pin))== 0);
                 break;
           case  3: return((LPC_GPIO3->FIOPIN & (0X1<<pin))== 0);
                  break;
           case  4: return((LPC_GPIO4->FIOPIN & (0X1<<pin))== 0);
                 break;
           default:
                  break;
     }
}
void Enviar_Dato_Puerto(int puerto,unsigned int dato)
{
     switch(puerto)
     {
           case  0: LPC_GPIO0->FIOPIN = dato;
                  break;
           case  1: LPC_GPIO1->FIOPIN = dato;
                 break;
           case  2: LPC_GPIO2->FIOPIN = dato;
                 break;
           case  3: LPC_GPIO3->FIOPIN = dato;
                  break;
           case  4: LPC_GPIO4->FIOPIN = dato;
                  break;
           default:
                  break;
    }
}
```

```
/*============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: Proyecto Entrada_Salida.c
Autores: Matias Ferraro - Marcos Aranda
Descripción: Aplicación que utiliza los drivers de entrada/salida
de la LPC 1769
============================================================- */

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include "leds.h"
#include "pulsadores.h"

int main(void)
{
/*Inicializo el Puerto 2y los Pines 7,6,5 y 4 como entradas*/
    Init_Port_Entrada(PTO_2,PIN_7);
    Init_Port_Entrada(PTO_2,PIN_6);
    Init_Port_Entrada(PTO_2,PIN_5);
    Init_Port_Entrada(PTO_2,PIN_4);

/*Inicializo el Puerto 2 y los Pines 12,11,10 y 8 como salidas*/
    Init_Port_Salida(PTO_2,PIN_12);
    Init_Port_Salida(PTO_2,PIN_11);
    Init_Port_Salida(PTO_2,PIN_10);
    Init_Port_Salida(PTO_2,PIN_8);

    /*Ciclo Infinito*/
    while(1)
    {
          /*Controlo cuando los pulsadores estan presionados y
          enciendo o apago los leds*/
          if(Press_Pulsador(PTO_2,PIN_7)==ALTO) {
              Output_High(PTO_2,PIN_12);
          }
          else{
              Output_Low(PTO_2,PIN_12);
          }

          if(Press_Pulsador(PTO_2,PIN_6)==ALTO){
              Output_High(PTO_2,PIN_11);
          }
          else{
              Output_Low(PTO_2,PIN_11);
          }

          if(Press_Pulsador(PTO_2,PIN_5)==ALTO) {
              Output_High(PTO_2,PIN_10);
          }
          else {
              Output_Low(PTO_2,PIN_10);
          }

          if(Press_Pulsador(PTO_2,PIN_4)==ALTO){
              Output_High(PTO_2,PIN_8);
```

```
            }
            else {
                Output_Low(PTO_2,PIN_8);
            }
        }
        return 0;
}
```

6.3.b Ejercicio 2 – Comunicación por Bluetooth

Implementación de una comunicación bluetooth por interrupción entre el LPC1769 y una computadora en el cual el microcontrolador transmite y espera recibir carácter por carácter desde la computadora, para retransmitir lo que ha recibido nuevamente.

Modelado con VisualSTATE

Paso1. Identificar los eventos y las Acciónes

Existen los siguientes eventos externos:

- Reset
- eTransmite
- eRecibe
- eEspera

Las Acciónes que podemos definir serán:

- aTransmite: el microcontrolador transmite un carácter.
- aRecibe: el microcontrolador recibe un carácter.
- aEspera: el microcontrolador espera la comunicación.

Paso 2. Identificar los estados

Figura 6-24: Estados estables

Paso 3. Agrupar los estados por jerarquías

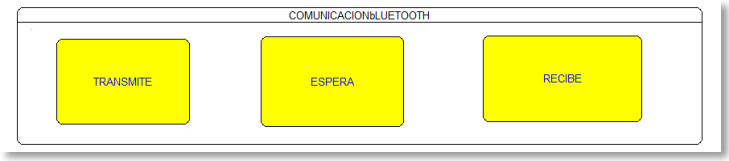


Figura 6-25: Agrupación de estados estables

Paso 4. Agrupar por concurrencia

En este ejemplo no existe concurrencia de tareas

Paso 5. Añadir las Acciónes y transiciones

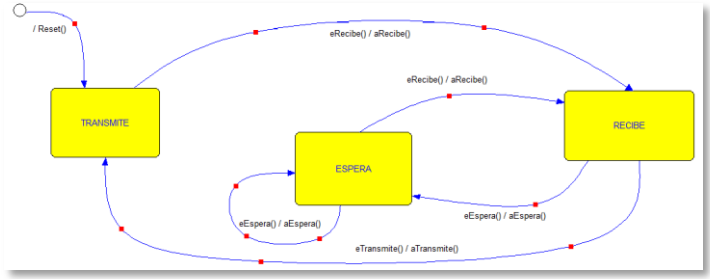


Figura 6-26: Transiciones entre estados

Paso 6. Añadir las sincronizaciones

No son necesarias

Paso 6. Codificación

```
/* ===============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: Proyecto Uart_Bluetooth.c
Autores: Marcos Aranda
Descripción: Comunicación UART a través de un módulo Bluetooth,
utilizando máquinas de estados
===============================================================*/

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#include "type.h"
#include "uart.h"
#include <string.h>
#endif

#include <cr_section_macros.h>

extern volatile uint32_t UART3Count;
extern volatile uint8_t UART3Buffer[BUFSIZE];

#define TRANSMITE        1
#define ESPERA           2
#define RECIBE           3

int main(void)
{
      int update = 1;
      int estado = TRANSMITE;
      const char* mensaje = "UART3 Tx y Rx: \r\n";

      /* Inicializo la UART3 a 9600 baudios */
      UARTInit(3, 9600);

      while(1)
      {
            switch(estado)
            {
                 case TRANSMITE:
                    if(update)
                    {
                   UARTSend(3,(uint8_t *)mensaje,strlen(mensaje));
                         update = 0;
                         estado = RECIBE;
                    }
                    else{
                         estado = RECIBE;
                    }

                    case ESPERA:
                         if ( UART3Count != 0 )
                         {
                             update = 1;
                             estado = RECIBE;
                         }
                         else
                         {
```

```
                        estado = ESPERA;
                    }
                case RECIBE:
                    if (update)
                    {
                UARTSend(3,(uint8_t *)UART3Buffer, UART3Count);
                        UART3Count = 0;
                        estado = TRANSMITE;
                        update = 0;
                    }
                    else {
                        estado = ESPERA;
                    }

                }
        }
        return 0 ;
}
```

CAPITULO 7:
EDU – CIAA – NXP

7.1 INTRODUCCIÓN

Figura 7-1: EDU - CIAA
Fuente http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp

La EDU-CIAA-NXP es una versión de bajo costo de la CIAA-NXP pensada para la enseñanza universitaria, terciaria y secundaria, está basada en la CIAA-NXP, por ser la primera versión de la CIAA que se encuentra disponible; por lo tanto su microcontrolador es de la marca NXP® y su designación comercial es LPC4337 (dual core ARM Cortex-M4F y Cortex-M0).

En la figura 7-2 se muestra el diagrama en bloques de la placa EDU-CIAA, en la misma se aprecian los bloques que la constituyen y son la base para la enseñanza de la programación sobre la EDU-CIAA. Para aplicaciones mas avanzadas se dispone de dos conectores de expansión (P1 y P2) donde se encuentran disponibles las señales correspondientes a los bloques ADC, DAC, SPI, I2C, Ethernet, Display LCD, teclado matricial, CAN y RS232.

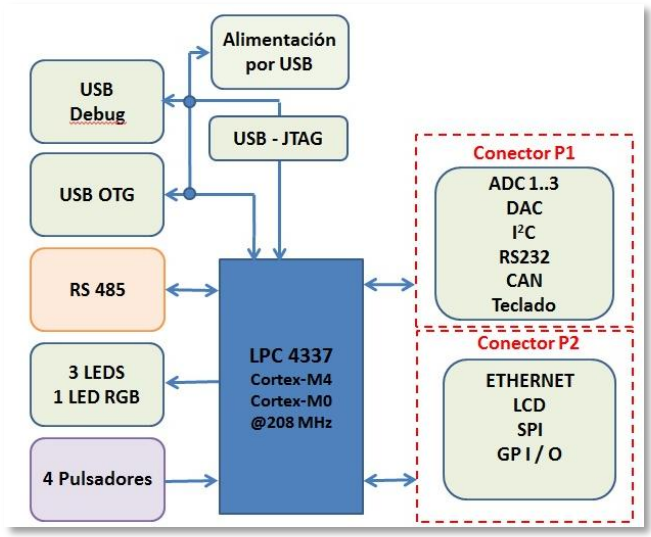


Figura 7-2: Diagrama de bloque de la EDU-CIAA.
Fuente http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp

Como se puede ver en la figura 7-3, el conector P1 dispone de conectividad para implementar.

- ✓ 3 entradas analógicas (ADC0, ADC1,ADC2),
- ✓ 1 salida analógica (DAC0),
- ✓ 1 conexión para un teclado de 3×4,
- ✓ 1 puerto I^2C,
- ✓ 1 puerto RS232,
- ✓ 1 puerto CAN,

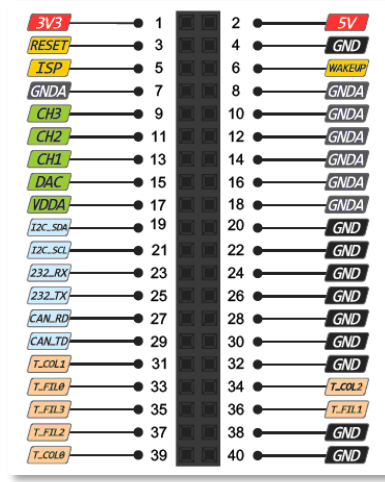


Figura 7-3: Conector P1.
Fuente Ing. Eric Pernia (2015)
http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp

La figura 7-4, muestra el conector P2 y las señales disponibles en el mismo

- ✓ 1 puerto Ethernet,
- ✓ 1 puerto SPI,
- ✓ Conección para display LCD,
- ✓ 9 pines genéricos de I/O.

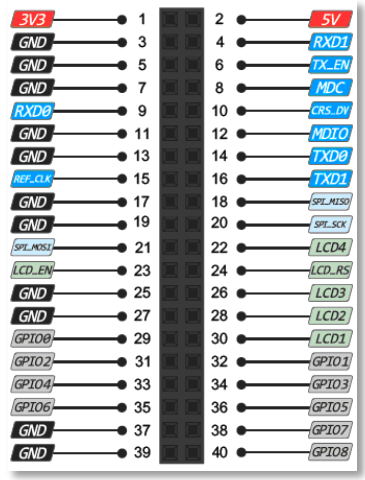


Figura 7-4: Conector P2.
Fuente Ing Eric. Pernia (2015)
http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp

7.2 LPC43XX - ARM CORTEX - M4/M0 MULTI-CORE

El LPC4337 es un procesador multinucleo basado en un Cortex-M4 y en un Cortex M0 (figura 7-5). Cortex M4 es otra arquitectura basada en ARMv7, por lo tanto en este apartado solo haremos referencia a algunas particularidades de esta arquitectura. El Cortex-M0 esta basado en una arquitectura ARMv6 y pose un set de instrucciones más limitado.

La inclusión de la letra F (Cortex-M4F) en la denominación se debe a la disponibilidad del bloque de operaciones con punto flotante, más un conjuntos de registros adicionales.

El procesador ARM Cortex-M4 se utiliza después del reset como el controlador del sistema de nivel superior. Tras el encendido o despertar del modo de apagado profundo, el núcleo M0 permanece en reset hasta que el reset se libera por el software que se ejecuta en el núcleo M4. A continuación, el M4 puede comunicarse con el

núcleo M0 a través del espacio de memoria compartida y las interrupciones.

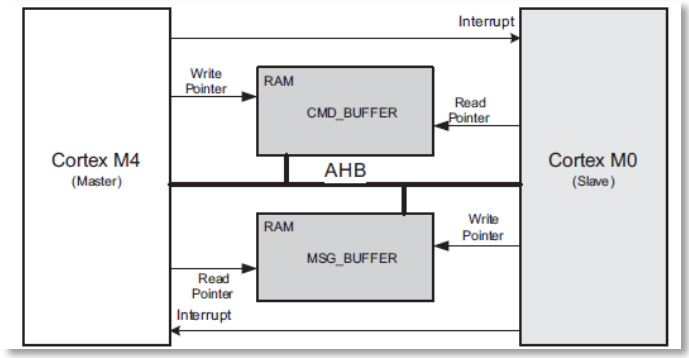


Figura 7-5: Dual Core diagrama en bloques
Fuente LPC43xx Manual del usuario

Ambos núcleos tienen completo acceso a la memoria, un resumen de sus características se muestra a continuación.

• Procesador Cortex-M4

- Procesador ARM Cortex-M4 (versión r0p1) corriendo hasta 204 MHz.
- Unidad de protección de memoria (MPU) interna con soporte de hasta 8 regiones.
- Nested Vectored Interrupt Controller (NVIC).
- Unidad de punto flotante por hardware.
- Entrada de interrupción no enmascarable (NMI).
- JTAG y Serial Wire Debug (SWD), serial trace, ocho breakpoints, and cuatro watch points.
- System tick timer.

• Procesador Cortex-M0

- Coprocesador ARM Cortex-M0 (versión r0p0).
- Corriendo hasta 204 MHz.
- JTAG.
- NVIC interno.

• Periféricos

- 83 Puertos de entrada salida digitales programables (GPIO) con pull-up / pull-down programables.
- Cuatro timers / contadores con capacidad de captura y match.

• Periféricos analógicos

- Un Conversor DA de 10 bits con soporte de DMA y frecuencia de conversión de 400kSamples/s.
- Dos conversores AD de 10 bits con soporte de DMA y frecuencia de conversión de 400 kSamples/s. Hasta ocho canales por ADC.

• Set de instrucciones

La figura 7-6 ilustra claramente la compatibilidad entre los set de instrucciones de la línea Cortex, un procesador M3 reconoce el código realizado para M0, un M4 reconoce el código de M3 y M0, etc.

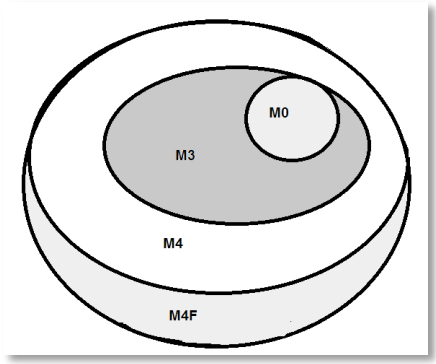


Figura 7-6: Compatibilidad en los Sets de instrucciones ARM.

7.3 CIAA-SOFTWARE-IDE

El CIAA-Software-IDE es el entorno de desarrollo del CIAA-Firmware. En su versión más completa, se ofrece un instalador llamado CIAA-IDE-Suite el cual puede descargarse desde:

https://github.com/ciaa/Software-IDE/releases

Con él podrán, de forma sencilla, instalar y configurar automáticamente la totalidad de las herramientas necesarias para trabajar con la EDU-CIAA. El paquete de instalación incluye:

- **Eclipse**: es el entorno base utilizado para desarrollo de aplicaciones, es decir, el softwareIDE.
- **PHP (Hypertext Pre-processor):** es un lenguaje de programación de uso general de código desde el lado del servidor, originalmente diseñado para el desarrollo de contenido dinámico. En este caso, se utiliza solamente en forma de scripts para poder generar algunos archivos del **Sistema Operativo OSEK**.
- **Cygwin**: es una consola que se ejecuta en Windows, de modo de emular la consola de comandos de Linux. Cuenta con todos los comandos, y el compilador GCC, propio del sistema operativo libre.

Además incluye otras herramientas, como se muestra en la siguiente tabla.

Tabla 7-1: Sotware para el desarrollo con EDU-CIAA

Tool	Tipo	Descripción	Instalación
Gcc-arm-none-eabi	Compiler	Compilador para ARM	Incluida
Gcc	Compiler	Compilador para Windows/Linux	Incluida
cygwin	Entorno Posix	Multiples Tools como perl – gcc – make – gdb – etc.	Incluida
Git	Sources	Control de versiones	Incluida
Eclipse CDT (C/C++)	IDE	Editor/Debugger grafico	Incluida
PHP	Script	Para la generacion de RTOS	Incluida
Firmware	Sources	Codigo fuente del firmware	Incluida
IDE4PLC	App	Entorno de programación como PLC	Incluida
OpenOCD	Debugger	On-Chip-Debugger	Incluida
Dirvers FTDI	Drivers	Driver JTAG-FT2232H-OpenOCD	Incluida(*)

Fuente: http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:software-ide

(*) La instalación de los drivers, puede que sea necesario hacerla manualmente.

Este instalador es compatible con las versiones de Windows XP, Windows Vista, Windows 7 y Windows 8, tanto en sus versiones de 32 como de 64 bits, para la instalación en Linux, al momento de cerrar esta publicación, todavía no se cuenta con un instalador unificado, por tal motivo va a ser necesario seguir los pasos indicados en la siguiente URL:

http://www.proyectociaa.com.ar/devwiki/doku.php?id=docu:fw:bm:ide:install_linux

Instalación en Windows®

El instalador provee todo el entorno necesario, con el cual se podrán instalar y configurar automáticamente de forma sencilla la gran mayoría de las herramientas necesarias para trabajar con la CIAA. Su uso es sumamente intuitivo, se recomienda no cambiar el directorio de instalación y, de ser necesario, elegir un nombre que NO contenga espacios.

Lanzado el asistente de instalación (figura 7-7) y aceptado el acuerdo de licencia (figura 7-8), comenzamos con la selección de componentes a instalar.

Figura 7-7: Asistente de instalación

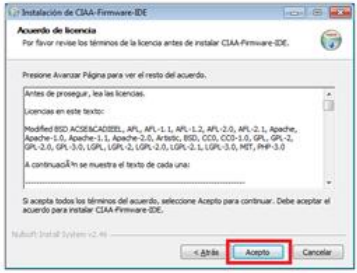


Figura 7-8: Acuerdo de licencia

En la siguiente ventana (figura 7-9), deberá elegir que componentes desea instalar. Si usted no posee una placa EDU-CIAA. No será necesario que instale los driver. Si en algún momento la adquiere, los controladores quedarán almacenados junto al CIAA-IDE, pudiendo hacer a mano la instalación de los mismos. De la misma

manera tiene la opción de elegir, que se descargue automáticamente la edición correspondiente del CIAA-Firmware.

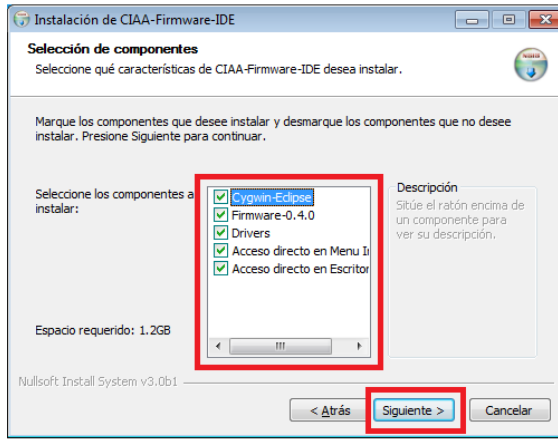


Figura 7-9: Selección de componentes

A continuación deberá colocar la ruta de instalación. Como se indicó anteriormente, si se desea cambiar la ruta de instalación, se debe tener la precaución, de no elegir una ruta donde los directorios posean espacios en su nombre.

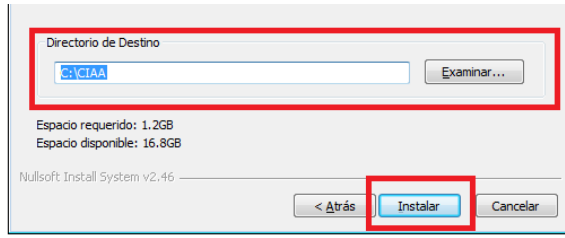
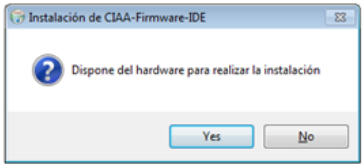


Figura 7-10: Ruta de instalación recomendada

Instalación de Drivers

Si en la ventana de instalación de componentes, usted dejó tildado el casillero de drivers, entonces se le harán un par de preguntas. Si no dispone del hardware, entonces simplemente conteste NO (figura 7-11).



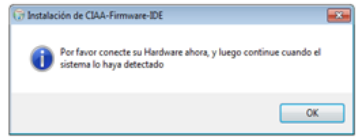


Figura 7-11: Disposición de hardware Figura 7-12: Conección del hardware

Si dispone de Hardware, entonces será necesario lo conecte al equipo (figura 7-12), y una vez reconocido el mismo por el Sistema Operativo continúe con la instalación. Haciendo click en *Extract*, se instalaran los controladores por defecto del fabricante FTDI para puerto virtual (figura 7-13).

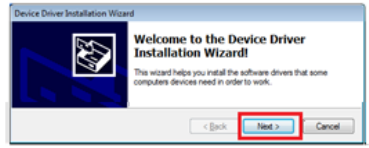


Figura7-13: Extracción del driver FTDI Figura7-14: Instalación del driver FTDI para Windows

Acepte la Licencia y presione Next > para poder finalizar con el proceso

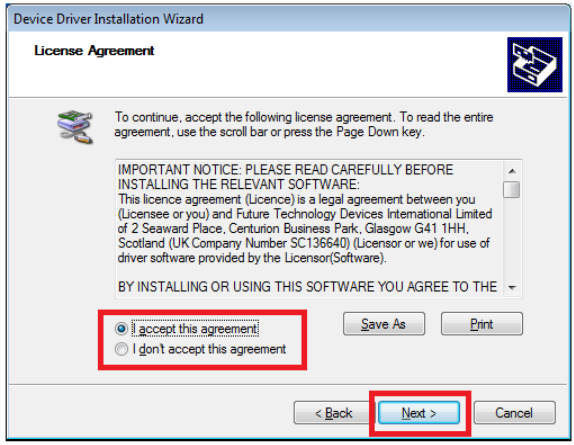


Figura 7-15: Aceptar la licencia de uso

Si la instalación fue correcta, podrá ver que los drivers FTDI fueron instalados correctamente (Figura 7-16). Si por algún motivo, la instalación de los controladores falló, aparecerá el mensaje de *falla en la instalación* que se muestra en la figura 7-17, indicando que puede llegar a ser necesaria la instalación manual de los mismos.

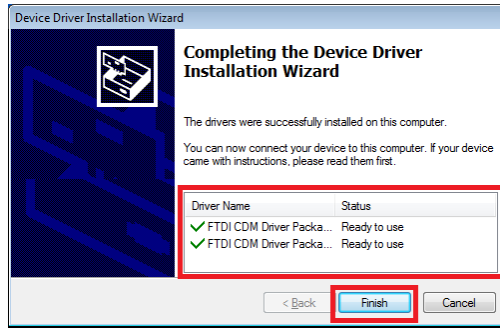


Figura 7-16: Instalación exitosa

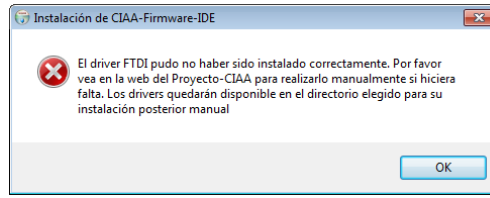


Figura 7-17: Mensaje por falla en la instalación

Si el proceso fue exitoso, solo resta terminar el proceso de instalación. Las últimas ventanas del instalador se muestran a continuación en las figuras 7-18 y 7-19.

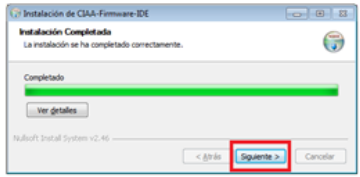


Figura 7-18: Finalizando la instalación

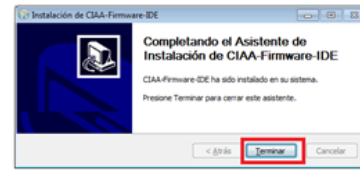


Figura 7-19: Cerrando el instalador

Corrección del driver FTDI

Un inconveniente que se nos puede presentar al momento de conectar nuestra placa con el sistema operativo, está vinculado con los drivers de la placa EDU-CIAA que se incluyen dentro del instalador descargado desde la página. El instalador incluye en la carpeta donde se instaló el software (**Por defecto, C:\CIAA**) un programa que configura el driver del controlador serie, emulado por la placa, para que uno de ellos pueda ser utilizado como interfaz JTAG. Dicho programa se llama **Zadig_Win_7_2_1_1.exe**.

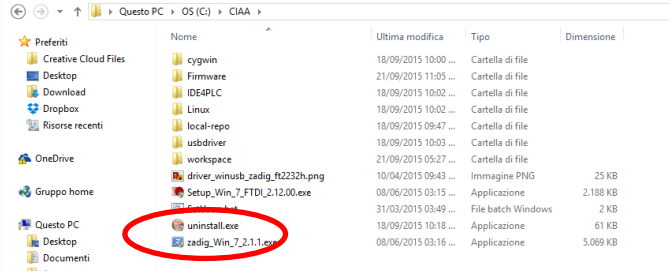


Figura 7-20: Instalador Zadig_Win_7_2_1_1.exe en la carpeta por defecto, C:\CIAA\

Para corregir los drivers, conectamos la placa a través del cable USB, hacemos **click en Options>> List All Devices**. Aparecerá una lista de dispositivos de comunicación relacionados al USB.

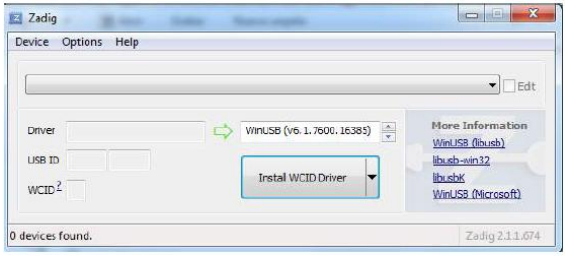


Figura 7-21: Zadig

Tenemos que buscar aquellos cuyos nombres tengan relación con el puerto serie (puede aparecer Dual RS232-HS, USB Serial Converter, o algo similar). Por lo general, aparecerán 2 con el, mismo nombre,

excepto que uno es Interface 0 y el otro Interface 1 (la lista de drivers que se muestra puede diferir, dependiendo de la computadora que se utilice).

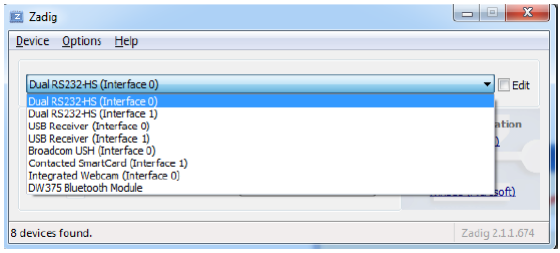


Figura 7-22: Opciones del puerto serie

Para configurar el driver se deberá:

a) Seleccionar la **Interfase 0** (figura 7-22)
b) Elegir el **"WinUSB v6.1"** (figura 7-23)
c) Hacer click en el botón **"Replace Driver"**. (figura 7-23)

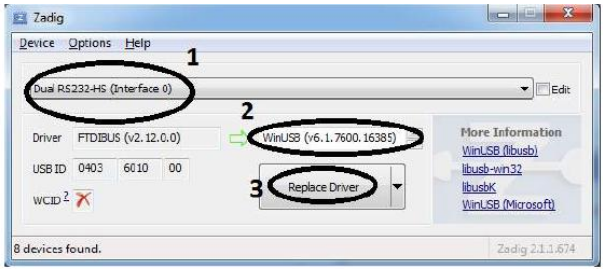


Figura 7-23: Configuración del driver FTDI

7.4 OPENOCD

El hardware de la CIAA cuenta con un puerto USB para poder realizar la programación y depuración del programa en el microcontrolador: esto está implementado en el chip **FTDI FT2232H.**

La herramienta de código abierto **OpenOCD** (On-Chip Debugger) es la encargada de manejar el chip FT2232H a través del USB y a la vez todo lo referido al JTAG. Con esto, el debugger (GDB) utilizado en el IDE-Eclipse puede hacer su tarea simplemente conectándose al

puerto 3333 (TCP) que el OpenOCD mantiene en escucha esperando la conexión.

Téngase en cuenta que el chip FT2232H posee 2 canales de comunicación independientes (A y B), pero ambos salen por el mismo puerto USB. Es decir, la computadora a la que está conectado verá 2 dispositivos distintos (en realidad uno compuesto): uno de estos dispositivos será el que conecta al JTAG manejado por OpenOCD, y el otro se verá como un puerto virtual COM: este último puede servir principalmente para debug. Cada uno de estos dos dispositivos tendrá un driver asociado.

Lo primero es instalar los drivers por defecto del fabricante FTDI para puerto virtual (VCP). En el Administrador de Dispositivos deberían aparecer 2 nuevos puertos COM.

7.5 CONFIGURACIÓN DEL ENTORNO CIAA-IDE

En este punto se explicaran los detalles de configuración del entorno de desarrollo CIAA Eclipse IDE. Comenzaremos importando el proyecto Firmware desde **File >> New >> Makefile Project with Existing Code**

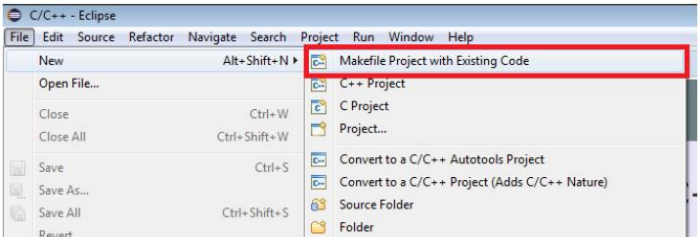


Figura 7-24: Creando un proyecto nuevo a partir de uno existente

Seguidamente aparecerá una ventana que se muestra en la figura 7-25, aqui se debe indicar el proyecto que se va a cargar: en este caso elegiremos la carpeta Firmware que, si usamos los directorios por defecto, se encuentra en la ubicación:

C:\CIAA\Firmware

En esta carpeta tenemos todos los archivos para trabajar, incluye librerías LPCOpen y Sistema operative OSEK.

En la misma ventana de la figura 7-25, encontramos opciones para **Toolchain for Indexer Settings,** seleccionaremos la opción **<none>,** lo cual dejará las opciones por defecto, configuradas en el Makefile.

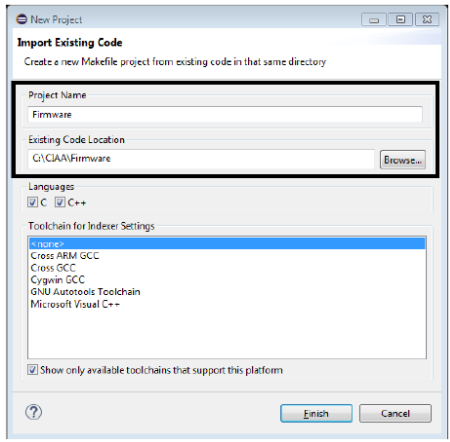


Figura 7-25: Carga de un proyecto existente

Una vez creado el proyecto, cerramos la pestaña **Welcome** con la que inicia Eclipse, y nos encontraremos con el entorno de desarrollo característico de Eclipse.

Los alcances de este libro no incluyen la modificación del firmware, por lo que nos limitamos a describir su estructura. En la carpeta **external** podremos encontrar librerías para casi todos los periféricos de la CIAA (figura 7-26).

Nuestros proyectos se deben crear en la carpeta **projects**. Dentro de ella se debe crear una carpeta llamada **drivers_bm** en la colocaremos los driver que escribe el programador (usuario), respetando una estructura de tres carpetas: **inc** donde van los archivos **.h**, **src** que contiene los **.c** y **mak** que contiene el **makefile** (Figura 7-27).

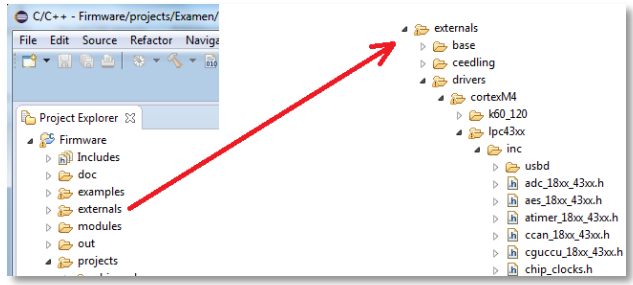


Figura 7-26: Estructura del Firmware de la EDU-CIAA

Crear un Proyecto

La forma más fácil de crear un proyecto nuevo, a partir del firmware recién mostrado, es ir al explorador de windows, dentro de la carpeta **projects**, crear nuestra propia carpeta para el proyecto y dentro de ella crear las tres estructura de carpeta **inc, src** y **mak**, luego en el CIAA Eclipse IDE presionar **[F5]** para refrescar, de esta manera tendríamos nuestra nueva estructura.

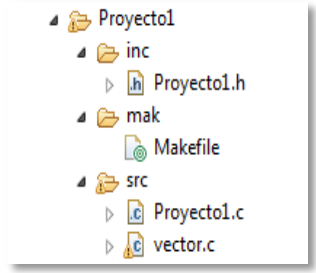


Figura 7-27: Estructura de carpetas de nuestro proyecto

En nuestro ejemplo “Proyecto1”, el archivo **Proyecto1.c** contiene el main y las funciones de la aplicación final, el archivo **vector.c** es un manejador de interrupciones y siempre debe estar ya que el reset actúa por interrupción, si una aplicación usa interrupciones externas o internas (como los timers) debemos retocar este archivo.

Ya escrita la aplicación, debemos compilar y para ello debemos indicarle al **makefile** que proyecto vamos a compilar, esto lo hacemos editando el archivo **makefile.mine**.

Si este archivo no existe debemos copiar el **makefile.config** en otro y renombrarlo como **makefile.mine**.

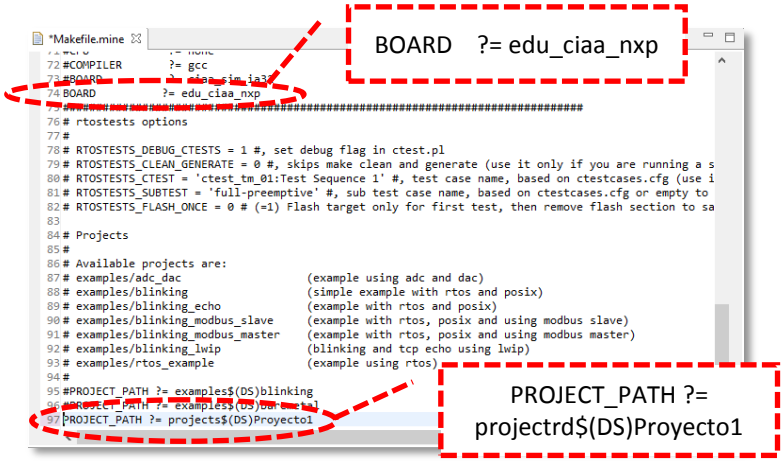


Figura 7-28: Modificaciones al archivo Makefile.mine

Modificamos, como lo muestra en la figura 7-28, en la línea 74 para indicarle el tipo de placa que vamos a usar y en la línea 97 para indicar que proyecto vamos a compilar. (El símbolo # indica que la línea es un comentario).

Para la compilación del proyecto, nos paramos sobre firmware y pulsamos botón derecho, sobre el botón desplegable seleccionamos **clean proyect** y luego **build proyect**.

Para los que prefieren utilizar la consola se puede ejecutar el simulador de Linux **Cygwin** (figuras 7-29), y desde la carpeta **/…../CIAA/Firmware** lanzar los siguientes comandos:

- make clean

  Que como vemos en la figura 7-29(a) elimina los archivos correspondiente a anteriores compilaciones.

- make

  Que lanza una nueva compilación. (Figura 7-29(b))

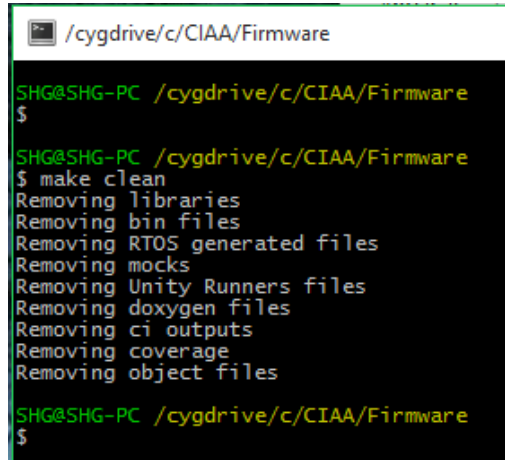


Figura 7-29 (a): Ejecución del comando make clean

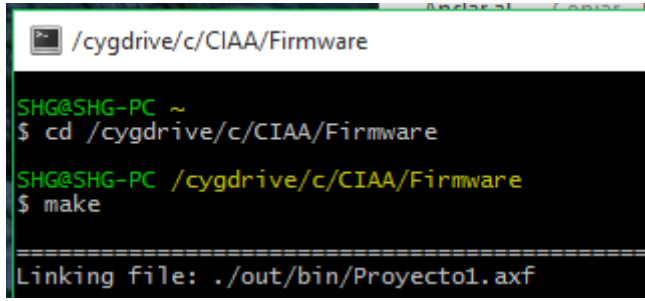


Figura 7-29 (b): Ejecución del comando make

Debugging

El debugging o **depuración de programas** es el proceso de identificar y corregir errores de programación. El nombre en ingles proviene de eliminación de bichos (delete bugs). Para este proceso existen facilidades integradas en los ambientes de desrrollo (IDEs).

Tanto el entorno CIAA-IDE como el Firmware, esta preparado para iniciar los primeros pasos sin necesidad de tener la placa EDU-CIAA, o con la posibilidad de hacer debug del software desarrollado, sin

ésta. En las siguientes páginas se explica el método de debugging con la placa EDU-CIAA conectada.

Las funciones de depuración están integradas en el procesador ARM Cortex-M4, la misma se apoya en una depuración JTAG estándar (mecanismo tradicional para las conexiones de depuración para ARM7). El procesador ARM Cortex-M4 está configurado para soportar hasta ocho puntos de interrupción y de cuatro puntos de observación.

Finalizada la compilación, conectamos la placa y podemos iniciar el debbug realizando, en primer lugar la configuración y para ello seleccionamos la carpeta Firmware y pulsamos botón derecho, seleccionamos **debug as** y luego **debug configuration.**

Se abrirá una ventana y sobre **GDB OpenOCD Debugging** oprimimos botón derecho del mouse y seleccionamos **new**, para crear el nombre de nuestro debug asociado a nuestro *Proyecto1*.

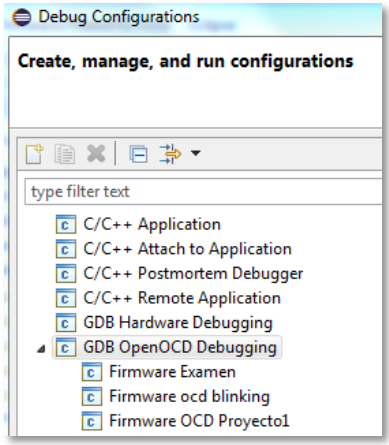


Figura 7-30: Selección de debuger GDB OpenOCD

Como se observa en la figura 7-31, en la solapa **Main** colocamos el nombre de nuestro proyecto a depurar y seleccionamos el archivo **.axf** que se generó en el proceso de compilación, cuya ruta:

C:\CIAA\Firmware\out\bin

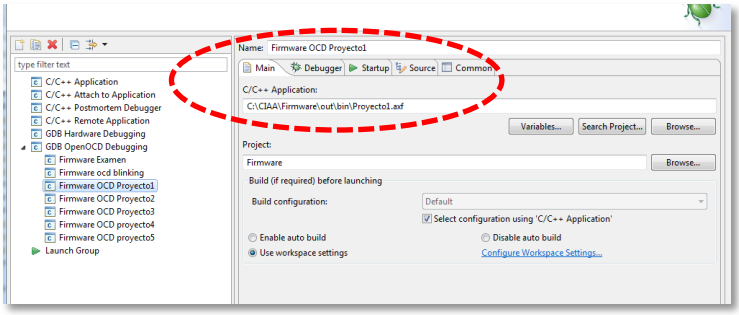


Figura 7-31: Modificaciones en la solapa Main del OCD de Proyecto1

Pasando a la solapa **Debugger**, quitamos el tilde a la opción de **Start OpenOCD locally** y finalmente seleccionamos el compilador en **GDB Client Setup>>Executable** que se encuentra en:

C:\CIAA\cygwin\usr\arm-none-eabi\bin\arm-none-eabi-gdb.exe

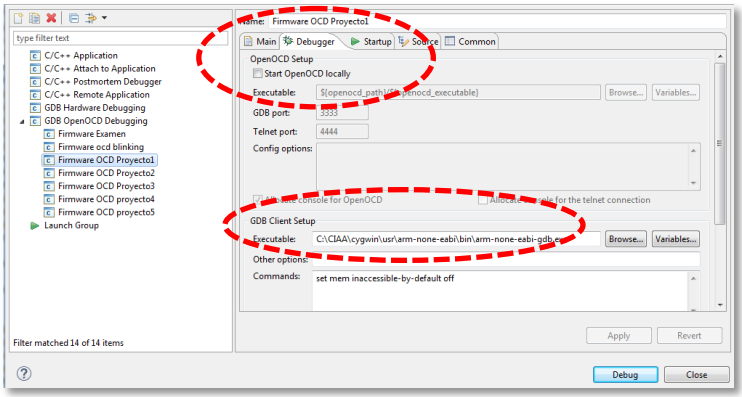


Figura 7-32: Configuración del OpenOCD Debugger

Oprimimos **Apply** y antes de presionar el botón de **Debug**, debemos lanzar por consola de **Cygwin** el **OpenOCD** desde adentro de carpeta **Firmware**, mediante el comando

- make openocd

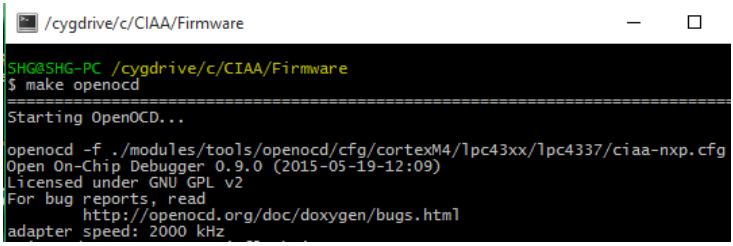


Figura 7-33: Ejecutando el comando make openocd

Hecho esto, las compilaciones sucesivas y los debbuger lo podemos hacer con las opciones de la barra de herramientas, que se muestran en la figura 7-34.

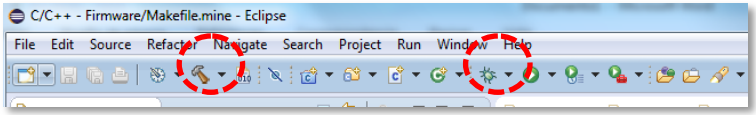


Figura 7-34: Ubicación de los iconos de compilación y debbug

Teniendo en cuenta estas indicaciones elementales, consideramos que la forma mas entretenida de comenzar a entender las capacidades de la EDU-CIAA en trabajando sobre ella. En el siguiente apartado iniciamos una serie de ejemplos elementales que creemos servirán de punto de partida para el dominio de la placa.

7.6 EJERCICIOS SOBRE EDU-CIAA

7.6.a Ejercicio 1 – Manejo de Entrada y Salida

7.6.a.1. Introducción a los puertos de salidas

Para trabajar con los puertos de salida disponibles en la placa, se puede utilizar la biblioteca **LPCOpen** para el acceso a los periféricos específicos del LPC4337. Esta biblioteca, se encuentra ya instalada en el Firmware de la CIAA, en:

CIAA\Firmware\externals\drivers\cortexM4\lpc43xx

Es destacable, que al cambiar de procesador, esta biblioteca se debe también cambiar por la biblioteca correspondiente, por ejemplo, para trabajar con la CIAA_FSL (CIAA Freescale), en:

CIAA\Firmware\externals\drivers\cortexM4\k60_120

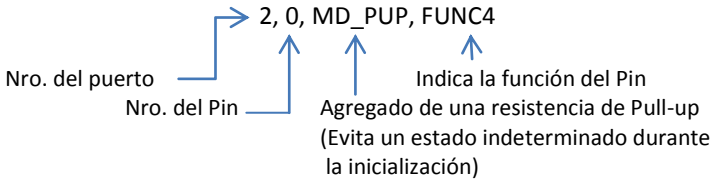
Para el manejo de puertos de entrada salida de uso general (GPIO), el archivo "*gpio_18xx_43xx.c*" (que forma parte de la LPCOpen), contiene todas las funciones para manejo de los mismos.

Para configurar un GPIO, lo primero que debemos hacer es inicializar el puerto, mediante el llamado a la función **Chip_GPIO_Init**(LPC_GPIO_T * pGPIO), a la hay que pasarle como parámetro la dirección base del periférico GPIO definida ya en "*chip_lpc 43xx.h*" como **LPC_GPIO_PORT**.

Luego hay que configurar la System Control Unit (SCU), para indicarle las características eléctricas de cada pin empleado y remapearlos como puertos GPIO. Hay que recordar que en este procesador, se puede elegir entre varias funciones disponibles para cada pin (ver tabla 7-1).

Por ejemplo, al llamar a la función **Chip_SCU_PinMux**(2,0,MD_PUP,FUNC4), remapea el P2_0 en GPIO5[0] correspondiente al LED0R de la placa EDU-CIAA y habilita las resistencias de pull up.

Antes de continuar veamos un poco en detalle los parámetros de la función.



En la siguiete tabla (extraida de la tabla 184. "LPC4350/30/20/10 pin description" del manual del usuario del LPC43xx) se puede ver el significado de incorporar FUNC4 entre los parámetros de la función Chip_SCU_PinMux

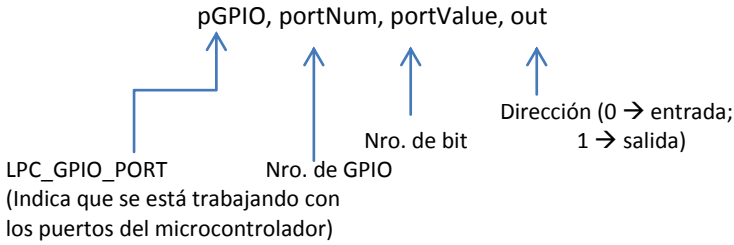
Tabla 7-1: Posibles roles que puede asumir el pin P2_0

P2_0	I/O	SGPIO4	General purpose digital pin	<--	FUNC0
	O	U0_TXD	Transmiter USART0	<--	FUNC1
	I/O	EMC_A13	External memory address line 13	<--	FUNC2
	O	USB0_PPWR	VBUS driver signal	<--	FUNC3
	I/O	GPIO5[0]	General purpose digital pin	<--	FUNC4
	-	Reservado		<--	FUNC5
	I	T3_CAP0	Capture input 0 of timer 3	<--	FUNC6
	O	ENET_MDC	Ethernet	<--	FUNC7

Finalizada esta aclaración, a continuación, se debe seleccionar el modo (entrada o salida) de cada pin con la función

Chip_GPIO_SetPinDir(LPC_GPIO_T *pGPIO, uint8_t portNum, uitn32_t portValue, uint8_t out);

Nuevamente nos detendremos para mirar un poco más en detalle los parámetros de la función.



Continuando, para setear y resetear los pines, existen numerosas funciones, entre ellas:

Chip_GPIO_ClearValue(LPC_GPIO_T *pGPIO, uint8_t portNum, uint32_t bitValue);

Chip_GPIO_SetValue(LPC_GPIO_T *pGPIO, uint8_t portNum, uint32_t bitValue);

Chip_GPIO_SetPinOutLow(LPC_GPIO_T *pGPIO, uint8_t port, uint8_t pin);

Chip_GPIO_SetPinOutHigh(LPC_GPIO_T *pGPIO, uint8_t port, uint8_t pin);

Chip_GPIO_SetPortOutHigh(LPC_GPIO_T *pGPIO, uint8_t port, uint32_t pins);

Chip_GPIO_SetPinToggle(LPC_GPIO_T *pGPIO, uint8_t port, uint8_t pin);

Chip_GPIO_SetPortToggle(LPC_GPIO_T *pGPIO, uint8_t port, uint32_t pins);

Como se puede apreciar, siempre hay que pasarles como parámetro la dirección base del periférico **LPC_GPIO_PORT**, el número **X** del puerto (GPIOX) y el bit **Y** a modificar (GPIOX[Y]).

En caso de las funciones que hacen referencia a un solo GPIO (contienen la palabra "Pin" en el nombre de la función) se le indica el número de bit del puerto que se desea modificar.

En caso de funciones que acceden a todo el puerto (Contienen la palabra "Port" en el nombre de la función), se le pasa una máscara del tipo *uint32_t* con los bits a modificar en 1.

Para resolver el ejercicio planteado, se ha desarrollado el código correspondiente a los drivers que nos permitirán utilizar los leds y los pulsadores de la plada EDU-CIAA. Respetando la estructura de carpetas del entorno de desarrollo creamos los archivos *leds.h* y *leds.c* en las carpetas .. *drivers_bm**inc* y .. *drivers_bm**src*, en el cual encontramos en el archivos *leds.h* las definiciones de constantes y los prototipos de las funciones que este driver pone a disposición del programador, en el archivo leds.c podemos ver el código completo de la implementación[2].

```
/*==================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: leds.h
Autor: Marcos Aranda
Descripción: Prototipos de funciones y definiciones
correspondientes para los puertos de salidas con leds para la EDU-
CIAA
==================================================================*/
#include "stdint.h"
```

[2] NOTA: quizás resulta excesiva la cantidad de código incorporado en las siguientes páginas, y después de discutir sobre los pros y los contras de tal incorporación nos decidimos por su incorporación a riesgo de ser un poco pesada su lectura. Pedimos disculpas al lector.

```
#include "ciaaPOSIX_stdbool.h"
#ifndef LEDS_H_
#define LEDS_H_
#define LED1   14
#define LED2   11
#define LED3   12
#define LEDR   0
#define LEDG   1
#define LEDB   2
#define SALIDA  1
#define PUERTO0 0
#define PUERTO1 1
#define PUERTO5 5

void Inicializar_Led(void);
void Encender_Led(uint8_t pin);
void Apagar_Led(uint8_t pin);
void Invertir_Led(uint8_t pin);
bool Leer_PuertoGPIO(uint32_t puerto, uint8_t pin);

#endif
```

Como se puede observar, se incluyen el archivo “stdint.h” los cuales posee definido tipos de datos de 8, 16 y 32bits (con y sin signo) y además el archivo “ciaaPOSIX_stdbool.h” el cual tiene definido el tipo de dato bool, cabe aclarar que solo será necesario incluirlos si el compilador con el que se encuentra trabajando lo requiere.

```
/*===================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: leds.c
Autor: Marcos Aranda
Descripción: Implementación de las funciones correspondientes para
los puertos de salidas con leds para la EDU-CIAA
===================================================================*/
#include "leds.h"
#include "stdint.h"
#include "chip.h"
#include "ciaaPOSIX_stdbool.h"

void Inicializar_Led(void)
{
        Chip_GPIO_Init(LPC_GPIO_PORT);
        /* remapea pines en GPIOx[y] y habilita el pull up*/
        Chip_SCU_PinMux(2,0,MD_PUP,FUNC4);
        Chip_SCU_PinMux(2,1,MD_PUP,FUNC4);
        Chip_SCU_PinMux(2,2,MD_PUP,FUNC4);
        Chip_SCU_PinMux(2,10,MD_PUP,FUNC0);
        Chip_SCU_PinMux(2,11,MD_PUP,FUNC0);
        Chip_SCU_PinMux(2,12,MD_PUP,FUNC0);

        /* Conf. los puertos donde están los leds como salida*/
        Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, PUERTO5, LEDR, SALIDA);
        Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, PUERTO5, LEDG ,SALIDA);
        Chip_GPIO_SetPinDIR(LPC_GPIO_PORT,PUERTO5, LEDB , SALIDA);
        Chip_GPIO_SetPinDIR(LPC_GPIO_PORT,PUERTO0, LED1 , SALIDA);
        Chip_GPIO_SetPinDIR(LPC_GPIO_PORT,PUERTO1, LED2 , SALIDA);
```

```
        Chip_GPIO_SetPinDIR(LPC_GPIO_PORT,PUERTO1, LED3 , SALIDA);
}


void Encender_Led(uint8_t pin)
{
        switch(pin)
        {
        case LEDR:
             Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, PUERTO5, pin);
             break;
        case LEDG:
             Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, PUERTO5, pin);
             break;
        case LEDB:
             Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, PUERTO5, pin);
             break;
        case LED1:
             Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, PUERTO0, pin);
             break;
        case LED2:
             Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, PUERTO1, pin);
             break;
        case LED3:
             Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, PUERTO1, pin);
             break;
        default:
             break;
        }
}

void Apagar_Led(uint8_t pin)
{
        switch(pin)
        {
        case LEDR:
             Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, PUERTO5,pin);
              break;
        case LEDG:
             Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, PUERTO5,pin);
             break;
        case LEDB:
             Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, PUERTO5, pin);
             break;
        case LED1:
             Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, PUERTO0, pin);
             break;
        case LED2:
             Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, PUERTO1,pin);
             break;
        case LED3:
             Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, PUERTO1, pin);
             break;
        default:
             break;
        }
}
void Invertir_Led(uint8_t pin)
{
        switch(pin)
```

```
        {
        case LEDR:
             Chip_GPIO_SetPinToggle(LPC_GPIO_PORT,PUERTO5, pin);
             break;
        case LEDG:
             Chip_GPIO_SetPinToggle(LPC_GPIO_PORT,PUERTO5, pin);
             break;
        case LEDB:
             Chip_GPIO_SetPinToggle(LPC_GPIO_PORT,PUERTO5, pin);
             break;
        case LED1:
             Chip_GPIO_SetPinToggle(LPC_GPIO_PORT,PUERTO0, pin);
             break;
        case LED2:
             Chip_GPIO_SetPinToggle(LPC_GPIO_PORT,PUERTO1, pin);
             break;
        case LED3:
             Chip_GPIO_SetPinToggle(LPC_GPIO_PORT,PUERTO1, pin);
             break;
        default:
             break;
        }
}

bool Leer_PuertoGPIO(uint32_t puerto, uint8_t pin)
{
      bool valor= 0;
      valor = Chip_GPIO_ReadPortBit(LPC_GPIO_PORT,  puerto, pin);
      return valor;
}
```

Como se puede observar, entre las inclusiones tenemos el archivo "chip.h", el cual incluye los archivos que alojan todas funciones necesarias para realizar las configuraciones de los GPIOs.

7.6.a.2. Introducción a los puertos de entradas

Para trabajar con los puertos de entrada disponibles en la placa, debemos configurar las entradas digitales utilizando, la función **Chip_SCU_PinMux**(1,0,MD_PUP|MD_EZI|MD_ZI,FUNC0), la cual, en este caso, remapea P1_0 en GPIO 0[4] para habilitar el SW1.

Luego debemos establecer estos pines como entrada utilizando la función **Chip_GPIO_SetDir**(LPC_GPIO_T * pGPIO, uint8_t portNum,uitn32_t portValue,uint8_t out)

Finalmente, para leer las señales, se pueden utilizar las funciones:

uint32_t **Chip_GPIO_ReadValue**(LPC_GPIO_T *pGPIO, uint8_t portNum) /* Devuelve el valor actual del puerto GPIO */

bool **Chip_GPIO_ReadPortBit**(LPC_GPIO_T *pGPIO, uint32_t port, uint8_t pin) /* devuelve verdadero si el GPIO esta en alto */

Para resolver el ejercicio de habilitar los pulsadores de la placa, se ha desarrollado el siguiente driver.

En el archivo *teclas.h,* al igual que para los leds, se definen constantes y se prototipan las funciones del driver.

```
/*=====================================================================
Dpto. de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: teclas.h
Autor: Marcos Aranda
Descripción: Implementación de las funciones correspondientes para
los puertos de entradas conectados a pulsadores, para la EDU-CIAA
=====================================================================*/
#ifndef TECLAS_H_
#define TECLAS_H_
#define ENTRADA 0
#define SW1  4
#define SW2  8
#define SW3  9
#define SW4  10

void Inicializar_Teclas(void);
bool Leer_Estado_Tecla(uint8_t tecla);

#endif
```

El código correspondiente al archivo *teclas.c* describe las funciones necesarias para inicializar las teclas (pulsadores) y para leer el estado de las mismas.

```
/*=====================================================================
Dpto. de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: teclas.c
Autor: Marcos Aranda
Descripción: Implementación de las funciones correspondientes para
los puertos de entradas conectados a pulsadores, para la EDU-CIAA
=====================================================================*/
#include "stdint.h"
#include "chip.h"
#include "teclas.h"
#include "leds.h"
#include "ciaaPOSIX_stdbool.h"

void Inicializar_Teclas(void )
{
     Chip_GPIO_Init(LPC_GPIO_PORT);
     /* remapea los pines donde estan conectados los SW, en sus
        correspondientes GPIO x[y], SW1 */
     Chip_SCU_PinMux(1,0,MD_PUP|MD_EZI|MD_ZI,FUNC0); /* SW1 */
     Chip_SCU_PinMux(1,1,MD_PUP|MD_EZI|MD_ZI,FUNC0); /* SW2 */
     Chip_SCU_PinMux(1,2,MD_PUP|MD_EZI|MD_ZI,FUNC0); /* SW3 */
```

```
    Chip_SCU_PinMux(1,6,MD_PUP|MD_EZI|MD_ZI,FUNC0); /* SW4 */
    /* Define a los pines donde estan los SW como entradas */
    Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, PUERTO0, SW1, ENTRADA);
    Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, PUERTO0, SW2, ENTRADA);
    Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, PUERTO0, SW3, ENTRADA);
    Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, PUERTO1, 9 , ENTRADA);
}

bool Leer_Estado_Tecla(uint8_t tecla)
{
    bool dato =0;
    switch(tecla)
    {
    case SW1:  /* si se quiere ver el estado del SW1 */
        dato=Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, PUERTO0,tecla);
        break;
    case SW2:
        dato=Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, PUERTO0,tecla);
        break;
    case SW3:
        dato=Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, PUERTO0,tecla);
        break;
    case SW4:
        dato=Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, PUERTO1,9);
        break;

    default:
        break;
    }
    return dato;  /* devuelve el estado del interruptor SWx */
}
```

Como cierre de este ejemplo, se muestra el uso de los drivers antes descriptos para el uso de entradas (teclas) y salidas (leds) de la EDU-CIAA. Nuevamente se crean los archivos .h y .c, en este caso los llamamos *EntradaSalida.h* y *EntradaSalida.c*, respectivamente y que se guardaran en la carpeta ...\projects\EntradaSalida

```
/*=============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: EntradaSalida.h
Autor: Marcos Aranda
=============================================================*/
#ifndef ENTRADASALIDA_H
#define ENTRADASALIDA_H
/*=================[inclusions]==============================*/
#include "stdint.h"
#include "leds.h"
#include "chip.h"
#include "teclas.h"
/*=================[macros]==================================*/
#define lpc4337            1
#define mk60fx512vlq15     2

#if (CPU == mk60fx512vlq15)
void Reset_Handler( void );

extern uint32_t __StackTop;
```

```
#elif (CPU == lpc4337)
extern void ResetISR(void);

extern void _vStackTop(void);

void RIT_IRQHandler(void);

#else
#endif
#endif
```

El desarrollo del código del ejemplo se muestra a continuación, el cual lee es estado de las teclas (SWx), y cual alguna de ellas se presione, se enciende o apaga el led asociado a ella (LEDx).

```
/*===================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: EntradaSalida.c
Autor: Marcos Aranda
Descripción: Aplicación que controla los driver de entrada y
salida de la EDU-CIAA
===================================================================*/
#include "EntradaSalida.h"

#ifndef CPU
#error CPU shall be defined
#endif
#if (lpc4337 == CPU)
#include "chip.h"
#elif (mk60fx512vlq15 == CPU)
#else
#endif

/* Decalaración de funciones utilizadas por la aplicación*/
static void Teclas(uint8_t pulsador);

int main(void)
{
   /* Inicialización de los LEDs */
   Inicializar_Led();
   /* Inicialización de los pulsadores */
   Inicializar_Teclas();
   /* Se apagan todos los LEDs */
   Apagar_Led(LEDR);
   Apagar_Led(LEDG);
   Apagar_Led(LEDB);
   Apagar_Led(LED1);
   Apagar_Led(LED2);
   Apagar_Led(LED3);

    /* Ciclo infinito */
    while(1)
    {
         /* Llamado a las funciones */
         Teclas(SW1);
         Teclas(SW2);
         Teclas(SW3);
         Teclas(SW4);
   }
```

```
   return 0;

}

/* Función que controla el estado del SW1, SW2 ,SW3 y SW4*/
static void Teclas(uint8_t pulsador)
{
      switch(pulsador)
      {
          case SW1:
                if(Leer_Estado_Tecla(SW1) ==0){
                        Encender_Led(LED1);
                }
                else{
                        Apagar_Led(LED1);
                }
                break;
          case SW2:
                if(Leer_Estado_Tecla(SW2) ==0){
                        Encender_Led(LED2);
                }
                else{
                        Apagar_Led(LED2);
                }
                break;
          case SW3:
                if(Leer_Estado_Tecla(SW3) ==0){
                        Encender_Led(LED3);
                }
                else{
                        Apagar_Led(LED3);
                }
                break;
          case SW4:
                if(Leer_Estado_Tecla(SW4) == 0)
                {
                        Encender_Led(LEDR);
                        Encender_Led(LEDG);
                        Encender_Led(LEDB);
                }
                else
                {
                        Apagar_Led(LEDR);
                        Apagar_Led(LEDG);
                        Apagar_Led(LEDB);
                }

          default:
                break;
        }
}
```

La función *Teclas* fue declarada como *static* para que la misma solo pueda ser ser accedida desde el programa principal y ningún otro archivo pueda verla.

7.6.b Ejercicio 2 – Uso de un LCD

Como siempre resulta útil disponer de una interfaz de salida para mostrar mensajes que nos permitan seguir/depurar un programa, proponemos este ejercicio para el manejo de un display LCD de 2x16 caracteres.

Para utilizar el display se agrega el driver del LCD con los archivos *lcd.h* y *lcd.c* en las carpetas .. *drivers_bm**inc* y .. *drivers_bm**src* respectivamente. El código perteneciente al ejercicio lo encontramos en la carpeta ...\projects\uso_display

En *lcd.h* (y en *lcd.c*) se ha colocado la definición de los pines que se utilizan para conectar el display (ver conector P2, figura 7-4) y las funciones que se utilizaran para manejar el display. En el código siguiente podemos ver estas definiciones:

```
/*====================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: lcd.h
Autor: Matias L. Ferraro
Descripción: Driver del display LCD de 2 lineas por 16 columnas
para la EDU-CIAA
====================================================================*/
#include "lpc_types.h"

typedef struct __LCD_DATA_Type {
        Bool D4;
        Bool D3;
        Bool D2;
        Bool D1;
        Bool RS;
        Bool EN;
} LCD_DATA_Type;

/*Donde esta el LCD*/
#define LCD_PORT 4
/*Pines del puerto*/
#define LCD4     10 /* conector EDU-CIAA P2-22 (port 4 pin 10)*/
#define LCD3     6  /* conector EDU-CIAA P2-26 (port 4 pin 6)*/
#define LCD2     5  /* conector EDU-CIAA P2-28 (port 4 pin 5)*/
#define LCD1     4  /* conector EDU-CIAA P2-30 (port 4 pin 4)*/
#define LCD_RS   8  /* conector EDU-CIAA P2-24 (port 4 pin 8)*/
#define LCD_EN   9  /* conector EDU-CIAA P2-23 (port 4 pin 9)*/

/* Funciones necesarias para manejar el display */
void lcd_init_port(void);
void lcd_init(void);
void lcd_gotoxy(int x, int y);
void lcd_putc(char c);
void enviar_lcd (LCD_DATA_Type data);
void printf_lcd(char * string);
void delay(void);
```

Como se puedo observar, se realizaron las definiciones correspondientes al puerto del LCD, como así también los pines. También se definieron los prototipos de todas las funciones necesarias creadas para el funcionamiento correcto del LCD.

```
/*=====================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: lcd.c
Autor: Matias L. Ferraro
Descripción: Driver del display LCD de 2 lineas por 16 columnas
para la EDU-CIAA
=====================================================================*/
#include "lcd.h"
#include "puertos_lcd.h"

LCD_DATA_Type data;

//Inicializa los puertos del LCD*/
void lcd_init_port(void)
{
     Init_Port_Pin(LCD_PORT,LCD_RS,MD_PLN,INIT_OUT);
     Init_Port_Pin(LCD_PORT,LCD_EN,MD_PLN,INIT_OUT);
     Init_Port_Pin(LCD_PORT,LCD4,MD_PLN,INIT_OUT);
     Init_Port_Pin(LCD_PORT,LCD3,MD_PLN,INIT_OUT);
     Init_Port_Pin(LCD_PORT,LCD2,MD_PLN,INIT_OUT);
     Init_Port_Pin(LCD_PORT,LCD1,MD_PLN,INIT_OUT);
}

/*Inicializa el LCD según el manual del dispositivo*/
void lcd_init(void)
{

//Paso 1)--------Enviar: D7,D6,D5,D4=0011---- RS=0 ----------
    delay();
    data.D1=1;
    data.D2=1;
    data.D3=0;
    data.D4=0;
    data.RS=0;
    enviar_lcd(data);
    delay();
//Paso 2)--------Enviar: D7,D6,D5,D4=0011---- RS=0 ----------
    delay();
    data.D1=1;
    data.D2=1;
    data.D3=0;
    data.D4=0;
    data.RS=0;
    enviar_lcd(data);
    delay();
//Paso 3)--------- Enviar: D7,D6,D5,D4=0011---- RS=0 ----------
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
//Paso 4)--------- Enviar: D7,D6,D5,D4=0010----- RS=0 ----------
    delay();
    . . . . . .   /* Asignar valores a los bits*/
```

```
    enviar_lcd(data);
    delay();
//Paso 5)---PRIMERO---- Enviar: D7,D6,D5,D4=0010----- RS=0 -----
//       ---DESPUES---- Enviar: D7,D6,D5,D4=1111------ RS=0 ----
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    data.RS=0;
    enviar_lcd(data);
//Paso 6)---PRIMERO----- Enviar: ---D7,D6,D5,D4=0000------ RS=0 --
//       ---DESPUES----- Enviar: ---D7,D6,D5,D4=1000------ RS=0 --
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
//Paso 7)---PRIMERO--------D7,D6,D5,D4=0000------------------
//       ---DESPUES--------D7,D6,D5,D4=0001------------------
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
//Paso 8)---PRIMERO--------D7,D6,D5,D4=0000------------------
//       ---DESPUES--------D7,D6,D5,D4=0110------------------

    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();

// FIN INICIALIZACION
//DISPLAY ON/OFF
///---PRIMERO--------D7,D6,D5,D4=0000---- RS = 0 --------
///----DESPUES-------D7,D6,D5,D4=1100---- RS = 0 --------
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
    . . . . . .   /* Asignar valores a los bits*/
    enviar_lcd(data);
    delay();
}

void lcd_gotoxy(int x, int y)
{

   int address;
   LCD_DATA_Type data;
   int nibble;
   switch(y)
```

```
   {
     case 1: address = 0;
     break;
     case 2: address = 64;
     break;
   }
/*Primero envio tres bits de direccion y d7=1, luego cuatro bits
más de dirección*/
   address =(address+(x-1));
   delay();
   data.D1=(!!(address & 16));
   data.D2=(!!(address & 32));
   data.D3=(!!(address & 64));
   data.D4=1;
   data.RS=0;
   enviar_lcd(data);
   delay();
   data.D1=(!!(address & 1));
   data.D2=(!!(address & 2));
   data.D3=(!!(address & 4));
   data.D4=(!!(address & 8));
   data.RS=0;
   enviar_lcd(data);
}

void lcd_putc(char C)
{
   LCD_DATA_Type data;
   int nibble;
   switch(C)
      {
       case '\f':
       break;
       case '\n':
       break;
       case '\b'://Borra pantalla
                delay();
                 data.D1=0;
                 data.D2=0;
                 data.D3=0;
                 data.D4=0;
                 data.RS=0;
                 enviar_lcd(data);
                 delay();
                 data.D1=1;
                 data.D2=0;
                 data.D3=0;
                 data.D4=0;
                 data.RS=0;
                 enviar_lcd(data);
                 delay();
       break;
       default:
                nibble= (int) C;
                data.D1=(!!(nibble & 16));
                data.D2=(!!(nibble & 32));
                data.D3=(!!(nibble & 64));
                data.D4=(!!(nibble & 128));
                data.RS=1;
                enviar_lcd(data);
```

```
                delay();
                data.D1=(!!(nibble & 1));
                data.D2=(!!(nibble & 2));
                data.D3=(!!(nibble & 4));
                data.D4=(!!(nibble & 8));
                data.RS=1;
                enviar_lcd(data);
                delay();
        break;
        }
}
void enviar_lcd (LCD_DATA_Type data)
{
   if (data.D1==1)Port_Pin_Alto_Bajo(LCD_PORT,LCD1,DISABLE, ALTO);
         else     Port_Pin_Alto_Bajo(LCD_PORT,LCD1,DISABLE,  BAJO);
   if (data.D2==1)Port_Pin_Alto_Bajo(LCD_PORT,LCD2,DISABLE,ALTO);
    else          Port_Pin_Alto_Bajo(LCD_PORT,LCD2,DISABLE, BAJO);
   if (data.D3==1)Port_Pin_Alto_Bajo(LCD_PORT,LCD3,DISABLE, ALTO);
         else     Port_Pin_Alto_Bajo(LCD_PORT,LCD3, DISABLE, BAJO);
   if (data.D4==1)Port_Pin_Alto_Bajo(LCD_PORT,LCD4,DISABLE, ALTO);
         else     Port_Pin_Alto_Bajo(LCD_PORT,LCD4, DISABLE, BAJO);
   if(data.RS==1)Port_Pin_Alto_Bajo(LCD_PORT,LCD_RS,DISABLE,ALTO);
         else     Port_Pin_Alto_Bajo(LCD_PORT,LCD_RS,DISABLE,BAJO);
   delay();
   Port_Pin_Alto_Bajo(LCD_PORT,LCD_EN, DISABLE, ALTO);
   delay();
   Port_Pin_Alto_Bajo(LCD_PORT,LCD_EN, DISABLE, BAJO);
   delay();
}
void printf_lcd(char * string)
{
   int c=0;
   while (string[c]!='\0')
   {
      lcd_putc(string[c]);
      c++;
   }
}
void delay(void)
{
   long int i=0,x=0;
   for(i=0; i<9999; i++){x++;}
}
```

Como podemos ver en las primeras líneas del código, se incorpora un archivo *puertos_lcd.h*, en este archivo se encuentran algunas funciones para el seteo y manejo de los puertos de uso general GPIO, que son necesarios para conectar el display.

```
/*================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: puertos_lcd.h
Autor: Marcos Aranda
Descripción: Prototipos y definiciones para la configuración de
pines para el LCD
================================================================*/
#include "chip.h"
```

```
#define ENTRADA  0
#define SALIDA   1
#define INIT_IN  0
#define INIT_OUT 1
#define ALTO     2
#define BAJO     3
#define STATE_IN 4
#define DISABLE  0

void Init_Port_Pin(uint8_t puerto,uint8_t pin,uint8_t config,
uint8_t modo);//Inicializa un pin de un puerto.

void Port_Pin_Alto_Bajo(uint8_t puerto, uint8_t pin, uint8_t
config ,uint8_t modo);

uint32_t Port_Pin_Estado(uint8_t puerto, uint8_t pin,uint8_t
modo);
```

Anteriormente se pudo ver como se definieron algunas constantes para poder ser utilizadas en la inicialización, en el seteo alto o bajo de un pin, como así también en la lectura del estado orrespondiente a los GPIO.

```
/*=================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: puertos_lcd.c
Autor: Marcos Aranda
=================================================================*/
#include "puertos_lcd.h"

void Init_Port_Pin(uint8_t puerto, uint8_t pin, uint8_t config
,uint8_t modo)
{
   /*El puerto 4 es utilizado por el LCD*/
   if( puerto == 4){
      if ((pin>=0)&&(pin<=6)){
         Chip_SCU_PinMux(4,pin,config,FUNC0);
            if (modo == 0){
              /*Inicializo el GPIO como entrada*/
              Chip_GPIO_SetPinDIR(LPC_GPIO_PORT,2,pin, INIT_IN);
            }
            else {
              /*Inicializo el GPIO como salida*/
              Chip_GPIO_SetPinDIR(LPC_GPIO_PORT,2,pin, INIT_OUT);
            }
       }
       if ((pin>=8)&&(pin<=10)){
        Chip_SCU_PinMux(4,pin,config,FUNC4);
          if (modo == 0){
            Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, 5,pin+4, INIT_IN);
           }
           else{
           Chip_GPIO_SetPinDIR(LPC_GPIO_PORT, 5,pin+4, INIT_OUT);
            }

        }
```

```
    }
}


void Port_Pin_Alto_Bajo (uint8_t puerto, uint8_t pin, uint8_t
config ,uint8_t modo){

   if (puerto == 4){
            if ((pin>=0)&&(pin<=6)){
            switch(modo)
                  {
            case ALTO:
                  Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, 2, pin);
                  break;
            case BAJO :
                  Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, 2, pin);
                  break;
            default:
                  break;
                  }

          }
           if ((pin>=8)&&(pin<=10)){
           switch(modo)
                  {

           case ALTO:
                  Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, 5, pin+4);
                  break;

           case BAJO :
                  Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, 5, pin+4);
                  break;
           default:
                  break;
                  }
         }
}

}

uint32_t Port_Pin_Estado (uint8_t puerto, uint8_t pin,uint8_t
modo){

         if (puerto == 4){
         if ((pin>=0)&&(pin<=6)){
         return Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, 2, pin);
         }
         if ((pin>=8)&&(pin<=10)){
         return Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, 5, pin+4);
         }
     }
}
```

Este driver contiene tres funciones utilizada por el LCD:

Init_Port_Pin(uint8_t puerto, uint8_t pin, uint8_t config ,uint8_t modo), utilizada para inicializar los GPIO del puerto 4 como entrada o salida.

Port_Pin_Alto_Bajo(uint8_t puerto, uint8_t pin, uint8_t config ,uint8_t modo), se utiliza para pone en alto o bajo el GPIO.

Port_Pin_Estado (uint8_t puerto, uint8_t pin,uint8_t modo), lee el estado del GPIO y lo devuele.

```
/*=============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uso_display.h
Autor: Matias L. Ferraro
=============================================================*/
#ifndef uso_display_H
#define uso_display_H

/*=================[inclusions]============================*/
#include "stdint.h"
#include "chip.h"
#include "puertos_lcd.h"
#include "lcd.h"
#include "leds.h"
#include "teclas.h"
#include "lpc_types.h"
/*=================[macros]================================*/
#define lpc4337            1
#define mk60fx512vlq15     2

#if (CPU == mk60fx512vlq15)
/* Reset Handler is defined in startup_MK60F15.S_CPP */
void Reset_Handler( void );

extern uint32_t __StackTop;
#elif (CPU == lpc4337)
extern void ResetISR(void);

extern void _vStackTop(void);

void RIT_IRQHandler(void);

#else
#endif
#endif
```

Finalmente para hacer uso del display, se ha creado una aplicación denominada "uso_display", consiste en un código que, tras la inicialización del display nos pide que presionemos una tecla, mostrando el nombre de la tecla pulsada en el display y

encendiendo un led. Las partes más significativas del código se muestran a continuación:

```
/*=================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uso_display.c
Autor: Matias L. Ferraro
Descripción: Driver del display LCD de 2 lineas por 16 columnas
para la EDU-CIAA
=================================================================*/
#include "uso_display.h"

#ifndef CPU
#error CPU shall be defined
#endif
#if (lpc4337 == CPU)
#include "chip.h"
#elif (mk60fx512vlq15 == CPU)
#else
#endif

int i=1,j=1,k=1,m=1;
char string_1[12]="Boton->UNO\0";
char string_2[12]="Boton->DOS\0";
char string_3[12]="Boton->TRES\0";
char string_4[16]="Boton->CUATRO\0";
char string_5[16]="Boton->ninguno\0";

int main(void)
{
    /*Inicialización de los Leds */
    Inicializar_Led();
    /*Inicialización de los GPIO para el LCD*/
    lcd_init_port();
    /*Inicialización de LCD*/
    lcd_init();
    /*Inicialización de las teclas*/
    Inicializar_Teclas();
    /*Posicionamos el cursor fila 1, columna 1*/
    lcd_gotoxy(1, 1);
    lcd_putc('P');      /*Mandamos a mostrar el carácter 'P' */
    lcd_putc('U');
    lcd_putc('L');
    lcd_putc('S');
    lcd_putc('A');
    lcd_putc('R');
    lcd_putc(':');

    while(1)
    {
           if (Leer_Estado_Tecla(SW4) == 0)
           {
                Encender_Led(LED3);
                lcd_gotoxy(1, 2);
                printf_lcd("                ");
                lcd_gotoxy(1, 2);
                printf_lcd(string_4);
           }else
           {
```

```
                Apagar_Led(LED3);
            }

            if (Leer_Estado_Tecla(SW3) == 0)
            {
                Encender_Led(LED2);
                lcd_gotoxy(1, 2);
                printf_lcd("                ");
                lcd_gotoxy(1, 2);
                printf_lcd(string_3);
            }else
            {
                Apagar_Led(LED2);
            }

            if (Leer_Estado_Tecla(SW2) == 0)
            {
                . . . . .
            }

            if (Leer_Estado_Tecla(SW1) == 0)
            {
               . . . . .
            }
        }
        return 0;
}
```

7.6.c Ejercicio 3 – Manejo del Reloj de Tiempo Real (RTC)

Continuando con ejemplos que podrán resultar de utilidad en muchas y variadas aplicaciones, proponemos ahora herramientas para el uso del RTC, y siguiendo con la misma metodología se agrega el driver del RTC con los archivos *rtc.h* y *rtc.c* en las carpetas .. *drivers_bm**inc* y .. *drivers_bm**src* respectivamente. El código perteneciente al ejercicio lo encontramos en la carpeta \...*projects**uso_reloj_tiempo_real*

En el código siguiente podemos ver las funciones que proveen el driver y el ejemplo de uso:

```
/*====================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: rtc.h
Autor: Matias L. Ferraro
Descripción: Driver del Reloj de Tiempo Real para la EDU-CIAA
====================================================================*/
#include "chip.h"
#include "stdint.h"
#include "lcd.h"
#include "rtc_18xx_43xx.h"

void rtc_init(void);
```

```
void rtc_set(int SECOND, int MINUTE, int HOUR, int DAYOFMONTH, int
DAYOFWEEK, int DAYOFYEAR, int MONTH, int YEAR);
uint32_t  get_rtc(RTC_TIMEINDEX_T time);
```

Como podemos ver, se incorpora la libreria "rtc_18xx_43xx.h" que nos brinda las herramientas elementales para el manejo del reloj incorporado en el micro. La funciones que nos prove este driver son para inicializar, setear y leer el reloj.

```
/*====================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: rtc.c
Autor: Matias L. Ferraro
Descripción: Driver del Reloj de Tiempo Real para la EDU-CIAA
====================================================================*/
#include "rtc.h"

void rtc_init(void)
{
    Chip_RTC_Init(LPC_RTC);
    Chip_RTC_Enable(LPC_RTC, ENABLE);
}
void rtc_set(int SECOND, int MINUTE, int HOUR, int DAYOFMONTH,
        int DAYOFWEEK, int DAYOFYEAR, int MONTH, int YEAR)
{
        RTC_TIME_T FullTime;
        FullTime.time[RTC_TIMETYPE_SECOND]     = SECOND;
        FullTime.time[RTC_TIMETYPE_MINUTE]     = MINUTE;
        FullTime.time[RTC_TIMETYPE_HOUR]       = HOUR;
        FullTime.time[RTC_TIMETYPE_DAYOFMONTH]= DAYOFMONTH;
        FullTime.time[RTC_TIMETYPE_DAYOFWEEK] = DAYOFWEEK;
        FullTime.time[RTC_TIMETYPE_DAYOFYEAR] = DAYOFYEAR;
        FullTime.time[RTC_TIMETYPE_MONTH]      = MONTH;
        FullTime.time[RTC_TIMETYPE_YEAR]       = YEAR;
       Chip_RTC_SetFullTime(LPC_RTC, &FullTime);
}


uint32_t get_rtc(RTC_TIMEINDEX_T time)
{
   RTC_TIME_T FullTime_;
   char aux[6];
   int anterior, dato_rtc;
   Chip_RTC_GetFullTime(LPC_RTC, &FullTime_);
   /*La función itoa() convierte de entero a string*/
   itoa (FullTime_.time[time],aux,10);
   if (FullTime_.time[time]!=anterior)
   {
     if (time==0){
       return anterior=0;
     }
     if (FullTime_.time[time]<=9){
       return (FullTime_.time[time]);
     }else
     {
       anterior=FullTime_.time[time];
       return anterior;
     }
  }
```

```
}
```

En el siguiente archivo se realizan las inclusiones de los headers necesarios para la compilación.

```
/*=============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: uso_reloj_tiempo_real.h
Autor: Matias L. Ferraro
=============================================================*/

#ifndef USO_RELOJ_TIEMPO_REAL_H
#define USO_RELOJ_TIEMPO_REAL _H

#include "chip.h"
#include "leds.h"
/*=================[macros]================================*/
#define lpc4337            1
#define mk60fx512vlq15     2

#if (CPU == mk60fx512vlq15)
/* Reset_Handler is defined in startup_MK60F15.S_CPP */
void Reset_Handler( void );

extern uint32_t __StackTop;
#elif (CPU == lpc4337)
extern void ResetISR(void);

extern void _vStackTop(void);

void RIT_IRQHandler(void);

#else
#endif

#endif
```

Para hacer uso de los drivers de RTC, se realizo una aplicación en la cual se muestra en el LCD, la hora y la fecha.

```
/*=============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: uso_reloj_tiempo_real.c
Autor: Matias L. Ferraro
Descripción: Ejemplo de uso del diver del RTC en la EDU-CIAA
=============================================================*/
#include "uso_reloj_tiempo_real.h"
#ifndef CPU
#error CPU shall be defined
#endif
#if (lpc4337 == CPU)
#include "chip.h"
#elif (mk60fx512vlq15 == CPU)
#else
#endif
#include "rtc.h"
#include "rtc_18xx_43xx.h"
#include "chip.h"
```

```
#include "stdint.h"
#include "lpc_types.h"

static void mostrar_rtc(int x, int y, uint_32_t dato);

int sec_anterior=0;
char auxuliar [6];
uint_32_t hora, minutos,segundos, dia, mes, anio;

int main(void)
{
   RTC_TIME_T FullTime;
   Inicializar_Led();
   lcd_init_port();
   lcd_init();

   rtc_set(0,0,14,16,6,16,1,2016);
   rtc_init();
   lcd_gotoxy(1, 1);   /* Damos formato a la pantalla */
   printf_lcd("Hora:\0");
   lcd_gotoxy(9, 1);
   lcd_putc(':');
   lcd_gotoxy(12, 1);
   lcd_putc(':');
   lcd_gotoxy(1, 2);
   printf_lcd("Fecha:\0");
   lcd_gotoxy(9, 2);
   lcd_putc('/');
   lcd_gotoxy(12, 2);
   lcd_putc('/');
   while(1)      /*Mostramos hora y fecha en forma indefinida */
   {
            hora = get_rtc(RTC_TIMETYPE_HOUR);
            mostrar_rtc(7,1,hora);
            minutos = get_rtc(RTC_TIMETYPE_MINUTE);
            mostrar_rtc(10,1,minutos);
            segundos = get_rtc(RTC_TIMETYPE_SECOND);
            mostrar_rtc(13,1,segundos);

            dia = get_rtc(RTC_TIMETYPE_DAYOFMONTH);
            mostrar_rtc(7,2, dia);
            mes = get_rtc(RTC_TIMETYPE_MONTH);
            mostrar_rtc(10,2);
            anio = get_rtc(RTC_TIMETYPE_YEAR);
            mostrar_rtc(13,2,anio);
   }
}

static void mostrar_rtc(int x, int y, uint_32_t dato){

         if (dato == 0){
              lcd_gotoxy(x, y);
              printf_lcd("  ");
         }
         if (dato <=9){
              lcd_gotoxy(x, y);
              printf_lcd("0");
         }
         else{
              /*La función itoa() convierte de entero a string*/
```

```
            itoa (dato,auxiliar,10);
            lcd_gotoxy(x, y);
            printf_lcd(strcat(auxiliar,"\0"));
        }
}
```

7.6.d Ejercicio 4 – Uso del Conversor A/D

En este ejercicio se hace uso del conversor A/D, para ello nos valemos de un sencillo hardware como el que se muestra en la figura 7-36, la señal analógica de entrada se simula utilizando un preset multivuelta actuando como divisor resistivo, este divisor resistivo entrega una tensión variable entre 0V y 3,3V.

El ejercicio hace uso de uno de los conversores analógico digital de la EDU-CIAA esta posee tres canales de entrada analógicas: ADC_CH1, ADC_CH2 o ADC_CH3- Se setea el conversor analógico digital para trabajar con 10 bits y con una tensión de referencia del conversor A/D de 3,3v, por lo tanto el rango de la entrada analógica es: 0 a $2^{10} - 1 = 1023$, correspondiendo un valor 0 para una entrada de 0V y 2013 para una entrada de 3,3v, el ejercicio hace uso del display (que ya hemos tratada en ejercicios anteriores), en donde queremos mostrar el mensaje < *Valor ADC: ####* > siendo #### el valor leído del conversor analógico digital.

```
/*============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: adc.c
Autor: Marcos Aranda
Descripción: Diver del ADC en la EDU-CIAA
============================================================*/
#include "adc.h"
#include "stdint.h"

void Configurar_Canal_ADC(uint8_t canal){

        switch(canal){
         case 1:
                Chip_SCU_ADC_Channel_Config(ADC0,canal);
                break;
         case 2:
                Chip_SCU_ADC_Channel_Config(ADC0,canal);
                break;
         case 3:
                Chip_SCU_ADC_Channel_Config(ADC0,canal);
                break;
        default:
                break;
        }
```

```
}

void Inicializar_ADC(uint8_t canal){

        ADC_CLOCK_SETUP_T ADC_CLKSetup;
        ADC_CLKSetup.adcRate = ADC_MAX_SAMPLE_RATE;
        ADC_CLKSetup.bitsAccuracy = ADC_10BITS;
        ADC_CLKSetup.burstMode = false;

 switch(canal){
        case 1:
                Chip_ADC_Init(LPC_ADC0, &ADC_CLKSetup);
                Chip_ADC_EnableChannel(LPC_ADC0,ADC_CH1 ,ENABLE);
                break;
        case 2:
                Chip_ADC_Init(LPC_ADC0, &ADC_CLKSetup);
                Chip_ADC_EnableChannel(LPC_ADC0, ADC_CH2,ENABLE);
                break;

        case 3:
                Chip_ADC_Init(LPC_ADC0, &ADC_CLKSetup);
                Chip_ADC_EnableChannel(LPC_ADC0, ADC_CH3,ENABLE);
                break;
        default:
                break;

        }
}

int LeerADC(uint8_t canal){
        uint16_t dato;

        switch(canal){
        case 1:
                Chip_ADC_SetStartMode(LPC_ADC0,ADC_START_NOW,ADC_
                TRIGGERMODE_RISING);

                while(Chip_ADC_ReadStatus(LPC_ADC0,ADC_CH1,ADC_DR
                _DONE_STAT) != SET);

                Chip_ADC_ReadValue(LPC_ADC0, ADC_CH1, &dato);
                break;
        case 2:
                Chip_ADC_SetStartMode(LPC_ADC0,ADC_START_NOW,ADC_
                TRIGGERMODE_RISING);

                while(Chip_ADC_ReadStatus(LPC_ADC0,ADC_CH2,ADC_DR
                _DONE_STAT) != SET);

                Chip_ADC_ReadValue(LPC_ADC0, ADC_CH2, &dato);
                break;
        case 3:
                Chip_ADC_SetStartMode(LPC_ADC0,ADC_START_NOW,ADC_
                TRIGGERMODE_RISING);

                while(Chip_ADC_ReadStatus(LPC_ADC0,ADC_CH3,ADC_DR
                _DONE_STAT) != SET);

                Chip_ADC_ReadValue(LPC_ADC0, ADC_CH3, &dato);
                break;
```

```
        default:
                break;
        }
  return dato;
}
```

El driver contiene tres funciones:

Configurar_Canal_ADC(uint8_t canal), permite configurar uno de los 3 canales disponibles.

Inicializar_ADC(uint8_t canal), inicializa y habilita uno de los 3 canales disponibles.

LeerADC(uint8_t canal), lee el valor convertido y lo devuelve.

```
/*============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: adc.h
Autor: Marcos Aranda
Descripción: Diver del ADC en la EDU-CIAA
============================================================*/

#include "stdint.h"
#include "ciaaPOSIX_stdbool.h"
#include "chip.h"

#ifndef ADC_H_
#define ADC_H_

#define ADC0 0
#define ADC1 1
#define ADC_MAXIMO_MUESTRA        400000
#define ADC_BITS10                        0
#define falso     0

void Configurar_Canal_ADC(uint8_t canal);
void Inicializar_ADC(uint8_t canal);
int LeerADC(uint8_t canal);
#endif
```

Al igual que los ejercicios anteriores, después de codificar el driver pasamos al ejemplo de uso de los mismos.

```
/*============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uso_adc.h
Autor: Marcos Aranda
Descripción: Ejemplo de uso del Diver del ADC en la EDU-CIAA
============================================================*/
#ifndef USO_ADC_H
#define USO_ADC_H

/*=================[inclusions]============================*/
#include "stdint.h"
```

```
#include "chip.h"
#include "puertos_lcd.h"
#include "lcd.h"
#include "adc.h"
#include "lpc_types.h"
/*==================[macros]================================*/
#define lpc4337            1
#define mk60fx512vlq15     2

#if (CPU == mk60fx512vlq15)
/* Reset_Handler is defined in startup_MK60F15.S_CPP */
void Reset_Handler( void );

extern uint32_t __StackTop;
#elif (CPU == lpc4337)
extern void ResetISR(void);

extern void _vStackTop(void);

void RIT_IRQHandler(void);

#else
#endif
#endif
```

En el código del ejercicio utilizaremos el canal 1, el cual se configura e inicializa, cuando se ingresa a un ciclo infinito mediante la función: *sprintf(cadena,"%d", LeerADC(1));* se lee el canal analógico y se guarda en una cadena de caracteres, luego se imprime este valor en el display, obteniendo un lectura en tiempo real del valor analógico presente.

```
/*==================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uso_adc.c
Autor: Marcos Aranda
Descripción: Implementación de uso del Diver del ADC en la EDU-
CIAA
==================================================================*/

#include "uso_adc.h"

#ifndef CPU
#error CPU shall be defined
#endif
#if (lpc4337 == CPU)
#include "chip.h"
#elif (mk60fx512vlq15 == CPU)
#else
#endif

int main(void)
{
```

```
    char cadena[6];
    /*Configuramos el canal 1 (ver hardware)*/
    Configurar_Canal_ADC(1);
    /*Inicializamos el canal 1*/
    Inicializar_ADC(1);
    lcd_gotoxy(1, 1);
    printf_lcd("Valor ADC:\0");
    while(1){
            lcd_gotoxy(11,1);
            /* Leemos y mostramos */
            sprintf(cadena,"%d", LeerADC(1));
            printf_lcd(cadena);
            printf_lcd("      /0");
     }
}
```

Para la implementacion de estos ejemplos que se han descripto, solo se requiere de una EDU-CIAA y un simple hardware que se muestra a continuación en la figura 7-35.

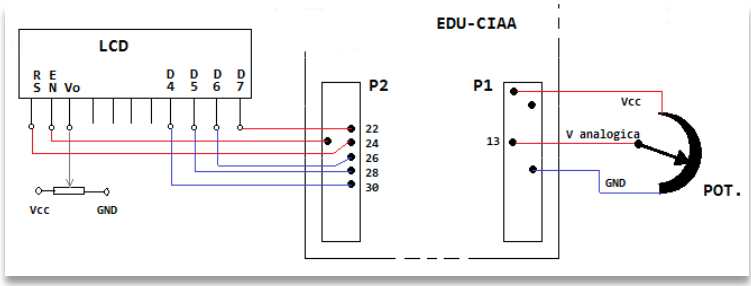


Figura 7-35: Conexión EDU-CIAA con LCD y potenciometro

7.7.e Ejercicio 5 – Uso del puerto serie (USART)

Para finalizar con el presente capitulo, incluimos un ejercicio que hace uso de la comunicación por el puerto serie (USART) utilizando un módulo HC-06 Bluetooth, como se muestra en la figura 7-36.

El ejercicio hace uso de la USART3 para realizar la comunicación entre la EDU-CIAA y una computadora a una velocidad de tranferencia de 9600 baudios.

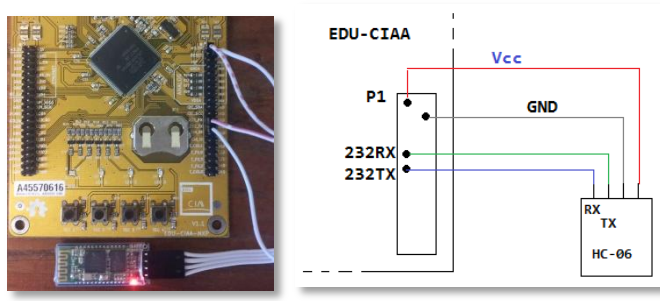


Figura 7-36: Conexión EDU-CIAA con HC-06

Notar que se ha creado el driver correspondiente y para ello se crearon los archivos *uart.h* y *uart.c*.

```
/*============================================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos - UNCa
Nombre: uart.c
Autor: Marcos Aranda
Descripción: Ejemplo de uso del Diver del puerto serie (USART2 y
USART3) en la EDU-CIAA
============================================================================*/
#include "uart.h"
#include "stdint.h"
#include "chip.h"

void Inicializar_USART2_USB(void)
{
   Chip_SCU_PinMux(7,1,MD_PDN, FUNC6); /* P7 1: UART2 TXD *
   Chip_SCU_PinMux(7,2, MD_EZI, FUNC6); /*P7 2:UART2 RXD */
   Chip_UART_Init(LPC_USART2);
   Chip_UART_SetBaud(LPC_USART2, 115200);
Chip_UART_SetupFIFOS(LPC_USART2,UART_FCR_FIFO_EN|UART_FCR_TRG_LEV)
;
   Chip_UART_TXEnable(LPC_USART2);
}

void Inicializar_USART3_BT(void)
{
   Chip_SCU_PinMux(2,3,MD_PDN, FUNC2); /* P2 3: UART3 TXD */
   Chip_SCU_PinMux(2,4, MD_EZI, FUNC2);/* P2 4:UART3 RXD */
   Chip_UART_Init(LPC_USART3);
   Chip_UART_SetBaud(LPC_USART3, 9600);
Chip_UART_SetupFIFOS(LPC_USART3,UART_FCR_FIFO_EN|UART_FCR_TRG_LEV0
);
   Chip_UART_TXEnable(LPC_USART3);
}

void Enviar_Dato_USART2_USB(char dato)
{
   Chip_UART_SendByte(LPC_USART2,dato);
}

void Enviar_Dato_USART3_BT(char dato){
```

```
   Chip_UART_SendByte(LPC_USART3,dato);
}
```

Este archivo incluye las funciones para inicializar y configuraciones necesarias de la USART2 por USB o la USART3 para la comunicación por Bluetooth, como así también la transmisión de un dato.

```
/*=============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uart.h
Autor: Marcos Aranda
Descripción: Ejemplo de uso del Diver del puerto serie (USART2 y
USART3) en la EDU-CIAA
=============================================================*/
#include "stdint.h"
#ifndef UART_H_
#define UART_H_

void Inicializar_USART2_USB(void);
void Enviar_Dato_USART2_USB(char dato);
void Inicializar_USART3_BT(void);
void Enviar_Dato_USART3_BT(char dato);

#endif
```

En el archivo anterior se definen los protopipos para la funciones que realizan la inicializaciones y envían datos, por el puerto serie, antes mencionadas.

El siguiente código muestra la implementación de un ejemplo de uso de los drivers del puerto serie (USART), en donde se inicializa la USART3 y luego se transmiten dos cadenas de datos desde la EDU-CIAA, hacia una computadora.

```
/*=============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uart_bluetooth.h
Autor: Marcos Aranda
=============================================================*/

#ifndef UART_BLUETOOTH_H
#define UART_BLUETOOTH _H
/*==================[inclusions]================================*/
#include "stdint.h"
#include "chip.h"
#include  "uart.h"
/*==================[macros]====================================*/
#define lpc4337            1
#define mk60fx512vlq15     2
/*==================[typedef]===================================*/
#if (CPU == mk60fx512vlq15)
/* Reset_Handler is defined in startup_MK60F15.S_CPP */
void Reset_Handler( void );
```

```
extern uint32_t __StackTop;
#elif (CPU == lpc4337)
extern void ResetISR(void);

extern void _vStackTop(void);

void RIT_IRQHandler(void);

#else
#endif

#endif
```

Como se puede apresiar en el código, por simplicidad del ejemplo se transmiten dos cadenas de caracteres, previamente definico como constantes.

```
/*============================================================
Dpto de Electrónica y Laboratorio de Sistemas Embebidos – UNCa
Nombre: uart_bluetooth.c
Autor: Marcos Aranda
Descripción: Implementación de uso del Diver del puerto serie en
la EDU-CIAA
============================================================*/
#include "uart_bluetooth.h"

#ifndef CPU
#error CPU shall be defined
#endif
#if (lpc4337 == CPU)
#include "chip.h"
#elif (mk60fx512vlq15 == CPU)
#else
#endif

int main(void)
{       int i = 0,j;
        uint8_t dato;
 const char mensaje[] = "MICROPROGRAMABLE\r\n";
 const char texto[] = "Comunicación USART por BT con la EDU-
CIAA\r\n";
        /*Inicializo la USART3*/
        Inicializar_USART3_BT();

        /*Transmito una cadena, hasta que el salto de línea \n */
        while(texto[i] != '\n'){
                Enviar_Dato_USART3_BT(texto[i]);
                for(j=0;j<10000;j++);
                i++;
        }
        /* Retardo al solo objeto de demorar la transmisión */
         for(j=0;j<10000000;j++);
  while(1){
          i = 0;
      /*Transmito una cadena, hasta que el salto de línea \n */
          while(mensaje[i] != '\n'){
              Enviar_Dato_USART3_BT(mensaje[i]);
              for(j=0;j<10000;j++);
```

```
            i++;
           }
           /* Retardo al solo objeto de demorar la transmisión */
          for(j=0;j<100000000;j++);
}
return 0;
}
```

Esperamos que el lector, a partir de estos simples ejemplos, pueda comenzar a desarrollar sus propias aplicaciones. El código completo de los ejemplos de este libro se puede descargar desde www.tecno.unca.edu.ar/lab_se/str_ejemplos.zip

BIBLIOGRAFÍA

- ARMv7-M (2010) *Architecture Reference Manual* Copyright © 2006-2010 ARM Limited.
- ARM (2009) *Cortex Microcontroller Software Interface Standard (CMSIS).* Version: 1.30 - 30. October 2009.
- Filomena, Eduardo. Reta, Juan M. (2015). *Programación de EDU-CIAA en lenguaje C*. 5ta Escuela de Sistemas Embebidos, Tucumán 2015.
- Alvarez, Raúl (2014) Ensamblador Versus C en Microcontroladores
- UM10360 (2010). *LPC 17xx User Manual*. NXP.
- Barrón Ruiz, Mariano. (2010) *Desarrollo de Software Basado en Modelos para Sistemas Embebidos*. SAAEI'10 – Bilbao.
- Canel, Susana Marta. (2007) *Microcontroladores ARM de 32 bits*. UTN, FRBA.
- Capella Hernández, Juan Vicente (2013) *Temporización mediante el temporizador del sistema SysTick en microcontroladores ARM Cortex-M.* Departamento de Informática de Sistemas y Computadores. Universidad Politecnica de Valencia.
- Caprile, Sergio. (2012) *Desarrollo con Microcontroladores ARM – Cortex M3*. Buenos Aires.
- CMSIS (2015) – *Cortex Microcontroller Software Interface Standard*. http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php
- Cruz, Juan Manuel. (2013) *Sistemas Embebidos de Tiempo Real*. Buenos Aires.
- IAR VisualSTATE. (2008) *Refence Guide*. November 2008.
- Ridolfi, Pablo. (2014) *Microcontroladores LPC 1769*. Tucumán – Escuela Sistemas Embebidos.

• Gonzales Vazquez, Jose A. (1992) *Introducción a los microcontroladores*. Editorial McGraw Hill

• Rodríguez Clemente, Arregui Olatz (1999) *Microprocesadores RISC Evolución y Tendencias*. Editorial RA-MA

• Martinez Dura, Grau, Solano. (2000) *Estructura de computadores y Periféricos*. Editorial RA-MA

• Angulo *Microprocesadores Fundamentos, diseño y aplicaciones*. Editorial Paraninfo

• Kingston Technology Company© (2002) *La Guía Completa De Memoria Kingston Technology*.

• Francucci, Leandro (2012) Diagramas *de Estado (Statecharts) - Los sistemas reactivos y la programación dirigida por eventos*. Simposio Argentino de Sistemas Embebidos – 2012.

• MISRA-C: 2004 *Guía para el uso en lenguaje en C para sistemas críticos*. Octubre 2004. www.misra-c.com