Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DESARROLLO DEL CURSO INTRODUCCIÓN AL DISEÑO DE SISTEMAS EMBEBIDOS, UTILIZANDO EL CONTROLADOR TM4C123GH6PM COMO ACTUALIZACIÓN DEL LABORATORIO DE MICROCONTROLADORES

David Josué Barrientos Rojas
Asesorado por el Ing. Iván René Morales Argueta

Guatemala, octubre de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DESARROLLO DEL CURSO INTRODUCCIÓN AL DISEÑO DE SISTEMAS EMBEBIDOS, UTILIZANDO EL CONTROLADOR TM4C123GH6PM COMO ACTUALIZACIÓN DEL LABORATORIO DE MICROCONTROLADORES

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

DAVID JOSUÉ BARRIENTOS ROJAS
ASESORADO POR EL ING. IVÁN RENÉ MORALES ARGUETA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, OCTUBRE DE 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Jurgen Andoni Ramírez Ramírez
VOCAL V	Br. Oscar Humberto Galicia Nuñez
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADORA	Inga. Ingrid Salomé Rodríguez de Loukota
EXAMINADORA	Inga. María Magdalena Puente Romero
EXAMINADOR	Ing. Marvin Marino Hernández Fernández
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DESARROLLO DEL CURSO INTRODUCCIÓN AL DISEÑO DE SISTEMAS EMBEBIDOS, UTILIZANDO EL CONTROLADOR TM4C123GH6PM COMO ACTUALIZACIÓN DEL LABORATORIO DE MICROCONTROLADORES

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 6 de junio de 2017.

David Josué Barrientos Rojas

Guatemala 10 de julio de 2,017

Ingeniero
Julio Cesar Solares Peñate
Coordinador del Área de Electrónica
Escuela de Mecánica Eléctrica
Facultad de Ingeniería, USAC

Estimado Ingeniero Solares.

Me permito dar aprobación al trabajo de graduación titular. **"Desarrollo del curso "Introducción al diseño de sistemas embebidos, utilizando el controlador TM4C123GH6PM" como actualización del laboratorio de microcontroladores"**, del señor David Josué Barrientos Rojas, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo de graduación y, yo, como su asesor, nos hacemos responsables por el contenido y conclusiones del mismo.

Sin otro particular, me es grato saludarle.

Atentamente.

Iván René Morales Argueta
Ingeniero Electrónico
Colegiado 12489

F._____
Ing. Iván Rene Morales Argueta
Colegiado 12,489
Asesor

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 44.2017.

21 de JULIO 2017.

Señor Director
Ing. Otto Fernando Andrino González
Escuela de Ingenierìa Mecànica Elèctrica
Facultad de Ingenierìa, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: DESARROLLO DEL CURSO INTRODUCCIÓN AL DISEÑO DE SISTEMAS EMBEBIDOS, UTILIZANDO EL CONTROLADOR TM4C123GH6PM COMO ACTUALIZACIÓN DEL LABORATORIO DE MICROCONTROLADORES, procede a la autorización del mismo. del estudiante David Josué Barrientos Rojas, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica

FACULTAD DE INGENIERIA - UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA

sro

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 32 . 2017.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; DAVID JOSUÉ BARRIENTOS ROJAS titulado: DESARROLLO DEL CURSO INTRODUCCIÓN AL DISEÑO DE SISTEMAS EMBEBIDOS, UTILIZANDO EL CONTROLADOR TM4C123GH6PM COMO ACTUALIZACIÓN DEL LABORATORIO DE MICROCONTROLADORES, procede a la autorización del mismo.

Ing. Otto Fernando Andrino González

FACULTAD DE INGENIERIA · UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA

GUATEMALA, 4 DE SEPTIEMBRE 2,017.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 457.2017

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DESARROLLO DEL CURSO DE INTRODUCCIÓN AL DISEÑO DE SISTEMAS EMBEBIDOS, UTILIZANDO EL CONTROLADOR TM4C123GH6PM COMO ACTUALIZACIÓN DEL LABORATORIO DE MICROCONTROLADORES,** presentado por el estudiante universitario: **David Josué Barrientos Rojas,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, octubre de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

/gdech

ACTO QUE DEDICO A:

Mi familia	Pedro de Jesús Barrientos Cruz y Delia Rojas de Barrientos, por impulsarme a alcanzar mis metas y brindarme su incondicional apoyo económico y moral.
Mis hermanas	Carol Elizabeth y Mariam Isabel Barrientos por brindarme su apoyo y cariño incondicional.
Mis amigos	Por hacer de mi carrera universitaria una experiencia inolvidable.
Los estudiantes de Ingeniería Electrónica	Para que siempre busquen el conocimiento y la excelencia en todo lo que realicen.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser la casa de estudios que permite a los ciudadanos formarse para generar un cambio en la nación.
Facultad de Ingeniería	Por ser el lugar donde pude formarme y expandir mis conocimientos.
IEEE	Por permitirme conocer un grupo de personas interesadas en el desarrollo de la tecnología para el beneficio de la humanidad
Laboratorio de Electrónica	Por brindarme trabajo, amistades y conocimientos durante la carrera.
Chantelle Cruz	Por su apoyo en la realización de este trabajo.
Víctor Carranza	Por su apoyo en la realización de este trabajo.
Ing. Iván Morales	Por ser un catedrático modelo, que inspira a nuevas generaciones de estudiantes a la excelencia.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A	Amperio
B	Ancho de banda
F_{max}	Frecuencia máxima
Hz	Hertz
$\int f \cdot dx$	Integral de f respecto a x
kb	Kilobit
kbps	Kilobit por segundo
KB	Kilobyte
KHz	Kilohertz
Mbps	Megabit por segundo
MB	Megabyte
MHz	Megahertz
µV	Microvoltio
mA	Miliamperio
ms	Milisegundo
mV	Milivoltio
mW	Miliwatt
ns	Nano segundo
&	Operación lógica AND
|	Operación lógica OR
%	Porcentaje
s	Segundo
V	Voltio

W	Watt

GLOSARIO

µDMA Micro-DMA (siglas en inglés para *direct memory access*), controlador encargado de transmitir datos entre un periférico a una memoria sin pasar por el procesador.

Algebra de Boole Sistema matemático aplicado en el diseño de sistemas digitales, para la resolución de preposiciones lógicas por medio de variables que toman el valor de 0 o 1 para representar falso o verdadero respectivamente.

ASCII Siglas en inglés para *american standard code for information interchange*, sistema encargado de la representación numérica de caracteres.

AND Compuerta lógica digital, genera una salida con estado lógico alto solamente cuando ambas entradas son estados lógicos altos.

Banderas Registro que cuenta con diferentes bits para indicar el estado de la máquina y el resultado de operaciones.

Big endian Formato que indica el orden en que se almacenan los bits, el bit más significativo se almacena primero.

Bit de paridad	Bit encargado de indicar si el número de bits con valor 1 en un conjunto de bits es par o impar.
CI	Circuito integrado, chip conformado de circuitos electrónicos.
Corriente eléctrica	Diferencial de carga eléctrica que atraviesa un punto en una unidad de tiempo.
DC	Siglas en inglés para *direct current* (corriente continua), flujo continuo de carga eléctrica entre dos puntos de distinto potencial que no varía con el tiempo.
Drive Strength	Valor de corriente encargado de variar el ángulo de inclinación que existe al momento de un cambio de estado lógico.
DSP	Siglas en inglés para *digital signal processor,* coprocesador especializado en la manipulación de señales digitales.
FIFO	Siglas en inglés para *first in first out*, estructura de datos donde el primer dato que entra es el primero en salir.
Flanco de subida	Cambio de estado lógico bajo a estado lógico alto en una señal digital.

Flanco de bajada	Cambio de estado lógico alto a estado lógico bajo en una señal digital.
FPGA	Chip de silicio reprogramable con bloques de lógica preconstruidos y recursos de ruteo programables para implementar funcionalidades personalizadas en hardware.
Linux	Sistema operativo de código fuente abierto.
Little endian	Formato que indica el orden en que se almacenan los bits, el bit menos significativo se almacena primero.
LSB	Siglas en inglés para *least significant bit*, posición de bit de un número binario que tiene el menor valor.
Memoria cache	Memoria que se sitúa entre la unidad de procesamiento y la RAM para reducir el tiempo de acceso a los datos.
Memoria flash	Memoria orientada al almacenamiento de grandes cantidades de datos en un espacio reducido, permitiendo la lectura y escritura de múltiples posiciones de memoria en la misma operación.
MI	Siglas en inglés para *maskable interrupt* (interrupción enmascarable), tipo de interrupción que puede ser ignorada.

MSB	Siglas en inglés para *most significant bit*, posición de bit de un número binario que tiene el mayor valor.
NAND	Compuerta lógica digital, genera una salida con estado lógico bajo solamente cuando ambas entradas son estados lógicos altos.
NMI	Siglas en inglés para *non-maskable interrupt* (interrupción no enmascarable), tipo de interrupción que no puede ser ignorada.
NOR	Compuerta lógica digital, genera una salida con estado lógico bajo cuando por lo menos una de las entradas tiene un estado lógico alto.
Open drain	Tipo de salida que se comporta como un interruptor. El transistor (MOSFET) se encontrará conectado a tierra cuando esté encendido y desconectado cuando se encuentre apagado (abierto).
OR	Compuerta lógica digital, genera una salida con estado lógico alto cuando por lo menos una de las entradas tiene un estado lógico alto.
Potencia eléctrica	Cantidad de energía entregada o absorbida por un elemento en un momento determinado.

Potencial eléctrico	Magnitud del campo eléctrico en dicho punto a través de la energía potencia electrostática que adquiriría una carga si se situase en ese punto.
QEI	Siglas en inglés para *quadrature encoder interface*, interfaz encargada de obtener datos de la posición mecánica y velocidad en sistemas con movimiento rotatorio.
RTOS	Siglas en inglés para *real time operating system*, sistema operativo con funcionamiento en tiempo real.
Resistencia *pull-up*	Resistencia conectada a voltaje, se utiliza para establecer un estado lógico alto en una entrada cuando esta se encuentra en reposo.
Resistencia *pull-down*	Resistencia conectada a tierra, se utiliza para establecer un estado lógico bajo en una entrada cuando esta se encuentra en reposo.
Servomotor	Motor que cuenta con un servomecanismo que permite un control preciso y estable de la posición angular dentro de su rango de operación.
TCM	Siglas en inglés para *tightly-coupled memory*, es una región pequeña de memoria dedicada, cercana al CPU, con la ventaja de poder ser accedida en un ciclo de reloj.

Tensión eléctrica	Magnitud que cuantifica la diferencia de potencial eléctrico entre dos puntos.
Tierra	Zona de referencia de potencial eléctrico, indica el cero absoluto del circuito.
USB	Siglas en inglés para universal serial bus, estándar industrial que define los cables, conectores y protocolos usados en un bus serial para conectar dos dispositivos.
VCO	Siglas en inglés para *voltage controlled oscillator*, circuito electrónico que genera una frecuencia de oscilación dependiente del voltaje de entrada.
World Wide Web	Combinación de recursos y usuarios en internet que utilizan el protocolo de transferencia de hipertexto.

RESUMEN

En el presente trabajo de graduación se desarrolla el material teórico y práctico para usar en el curso de diseño de sistemas embebidos. Este trabajo se enfoca en el aprendizaje del controlador TM4C123GH6PM, pero puede ser tomado como referencia para la enseñanza de cursos de microcontroladores con cualquier dispositivo similar.

En el primer capítulo se describen los conceptos fundamentales sobre señales, sistemas y números binarios como conocimiento base para la comprensión de los demás capítulos.

El segundo capítulo desarrolla el concepto, historia, estructura, características y tipos de sistemas embebidos.

En el tercer capítulo se desarrolla toda la base teórica en torno a los microcontroladores: historia, estructura y arquitectura.

En el cuarto capítulo se describe el controlador TM4C123GH6PM: características y arquitectura.

En el quinto capítulo se desarrollan los conceptos de programas informáticos y su transición hasta cargarse a un microcontrolador.

En el sexto capítulo se desarrolla el funcionamiento, características y configuración de los periféricos básicos.

En el séptimo capítulo se presentan prácticas de configuración y uso de los periféricos del controlador TM4C123GH6PM en lenguaje C.

OBJETIVOS

General

Desarrollar los temas necesarios para que sirvan de soporte a los estudiantes de la carrera Ingeniería Electrónica de modo que puedan desarrollar aplicaciones que involucren el diseño de sistemas embebidos.

Específicos

1. Dar una introducción al uso del controlador TM4C123GH6PM para el diseño de sistemas embebidos.

2. Realizar ejercicios prácticos, para que el estudiante comprenden la teoría detrás de la implementación práctica.

3. Proveer el material teórico necesario en español para la actualización del laboratorio de sistemas embebidos en la Facultad de Ingeniería.

INTRODUCCIÓN

El diseño de sistemas embebidos es, sin duda, una de las áreas más importantes en la electrónica como se conoce actualmente, al punto de convertirse en una parte fundamental de nuestra vida. Por el nombre poco convencional suele pensarse se refiere a computadoras; sin embargo, difieren ya que los sistemas embebidos se encargan de la realización de tareas específicas y no cumplir un amplio rango de tareas.

Debido a la masiva cantidad de controladores distintos, es necesario conocer los requerimientos y las características que optimizan el diseño de un sistema embebido, así como los diferentes periféricos que provee un controlador para cumplir las tareas específicas. Actualmente, no se cuenta con documentación debidamente estructurada y en español que contemple el contenido de un curso de sistemas embebidos con un microcontrolador vigente.

Por lo anterior, este documento pretende servir de enlace entre el estudiante y el conocimiento sistematizado por etapas que permitan una introducción al tema de interés en términos teóricos y prácticos adecuados al nivel universitario de pregrado. El desarrollo de esta estructuración se apoya en la experiencia del autor en instrucción de laboratorios afines al diseño de sistemas embebidos y en la asesoría de personas conocedoras del sector estudiantil.

1. SEÑALES, SISTEMAS Y NÚMEROS BINARIOS

El ser humano recibe y procesa estímulos obtenidos de su entorno (aromas, texturas, sonidos, luz, entre otros) por medio de sus sentidos. Todo en la naturaleza emite constantemente señales de diferentes tipos que los sentidos permiten al cuerpo humano interpretar. Estas señales son interpretadas por el cerebro, que a su vez emite señales hacia distintas partes del cuerpo en respuesta a estímulos externos. Muchas de estas señales naturales pueden ser modeladas matemáticamente y transformadas de forma que sean útiles para el diseño de sistemas embebidos.

1.1. Señales

Tabla I. **Definición de señal**

Una señal es una variación en la amplitud de una magnitud, encargada de transmitir información en el tiempo.

Fuente: elaboración propia.

Las señales dependen de dos parámetros importantes: su amplitud y su relación con el tiempo. La magnitud de su amplitud indica cómo interactúa con la naturaleza, en el caso de una señal eléctrica o hidráulica. Su relación con el tiempo indica cómo se comporta en cada intervalo del tiempo, se puede comportar de manera continua o discreta.

1.1.1. Señal analógica

Una señal analógica es aquella variación que permite transmitir información continua en amplitud y en intervalos de tiempo.

1.1.2. Señal digital

Una señal digital es aquella variación que permite transmitir información discreta en amplitud y en intervalos de tiempo.

Las señales analógicas son fundamentales para la toma de decisiones de un sistema que debe interactuar con el mundo real, análogo por naturaleza; las señales digitales son importantes dado que son fáciles de procesar por medios digitales, además de proveer otras ventajas que incluyen:

- Reproducibilidad de resultados (no varía en proporción a la temperatura, voltaje de alimentación, antigüedad y otros factores).
- Facilidad de diseño y modelado.
- Flexibilidad, velocidad y facilidad de manipulación y procesamiento.
- Programabilidad.

1.2. Sistemas

Tabla II. **Definición de sistema**

Conjunto de elementos que se encuentran interrelacionados e interactúan entre sí.

Fuente: elaboración propia.

Un sistema se encuentra definido por el observador, quien define los elementos que los relacionan y cómo interactúan entre sí, lo que para una persona es un sistema puede no serlo para otra.

Un sistema digital es cualquier dispositivo destinado a la generación, transmisión, procesamiento o almacenamiento de señales digitales.

Los sistemas digitales se dividen en dos categorías según el álgebra de Boole:

- Sistemas digitales combinacionales

Sistemas cuyas salidas solo dependen del estado de sus entradas en un momento dado, no dependen de los estados previos de las entradas.

- Sistemas digitales secuenciales

Sistemas cuyas salidas dependen además del estado de sus entradas en un momento dado, de estados previos.

Los sistemas digitales se encuentran conformados por dos partes:

- Parte lógica: conformada por el software y el firmware.
    - Software: secuencia ordenada de cada instrucción específica almacenada en memoria definiendo exactamente que tarea se realizará.

- Firmware: es un bloque de instrucciones de programa para propósitos específicos, grabado en una memoria de tipo no volátil, que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.

- Parte física: conformada por el hardware

  - Hardware: parte tangible de un sistema digital, conformada por partes como sus componentes eléctricos, electrónicos, electromecánicos y mecánicos.

Un sistema que procesa señales digitales no es capaz de comprender valores analógicos de forma directa; de hecho, un sistema digital trabaja completamente con valores codificados de manera binaria. Los circuitos internos de un sistema digital solamente son capaces de comprender la existencia de voltaje o su falta. Estos dos estados pueden ser representados por medio del sistema binario; cada combinación se hace por medio de los números binarios.

1.3. Números binarios

Cada dígito en un número decimal cuenta con un lugar y un valor. El lugar es una potencia de 10 y el valor es un número entre 0 y 9. Un número decimal se encuentra formado por la combinación de sus dígitos multiplicado por potencias de 10.

De manera similar, cada dígito de un número binario cuenta con un lugar y un valor, con la diferencia que el lugar es una potencia de 2 y el valor solamente es un número entre 0 y 1.

Tabla III. **Definición de número binario**

Un número binario es una combinación entre sus dígitos multiplicado por potencias de 2.

Fuente: elaboración propia.

- Un dígito binario es conocido como bit
- El arreglo de 4 bits se le conoce como nibble
- El arreglo de 8 bits se le conoce como byte
- El arreglo de 16 bits se le conoce como media palabra
- El arreglo de 32 bits se le conoce como palabra

Por medio de arreglos de dígitos binarios se puede representar cualquier número del sistema digital. Si se desea representar un número con signo, un bit extra se agrega al comienzo del arreglo el cual estará encargado de definir si es positivo o negativo.

1.3.1. Punto flotante

Cuando se habla de punto flotante, se refiere a la notación científica que utilizan los sistemas digitales para representar números racionales extremadamente grandes y pequeños de manera compacta y eficiente que permiten la realización de aritméticas.

Para esta representación se utiliza el estándar IEEE 754 establecido en 1985 por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) que divide el valor en cuatro partes:

- Signo del número: conformado por un bit que indica si es un número positivo o negativo.

- Signo del exponente: conformado por un bit que indica si será un exponente positivo o negativo.

- Exponente: indica la cantidad de posiciones que se desea mover el punto decimal.

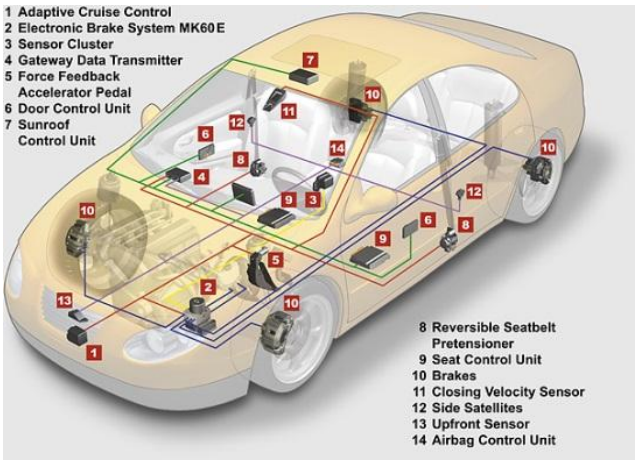
- Mantisa: número a representar previo al exponente.

Si se desea expresar el número $7{,}14 \times 10^{-50}$ en estándar de punto flotante, el exponente sería 50, la mantisa 7,14, el signo del número sería 0 (positivo), y el signo del exponente 1 (negativo).

2. SISTEMAS EMBEBIDOS

Es importante tener claro este concepto, pues, a pesar de que se vive rodeados de estos, raramente son mencionados o notados. La mayoría de los dispositivos eléctricos con los que el ser humano interactúa en su diario vivir son sistemas embebidos o están basados en uno. De hecho, es difícil encontrar algún dispositivo que no se encuentre basado en un sistema embebido. Se encuentran ejemplos de sistemas embebidos en electrodomésticos simples (hornos microondas, refrigeradores, estufas, hornos tostadores), dispositivos de uso diario (relojes digitales, celulares, calculadoras, radios, reproductores de música, televisiones) e incluso en dispositivos más complejos (elevadores, carros y aviones).

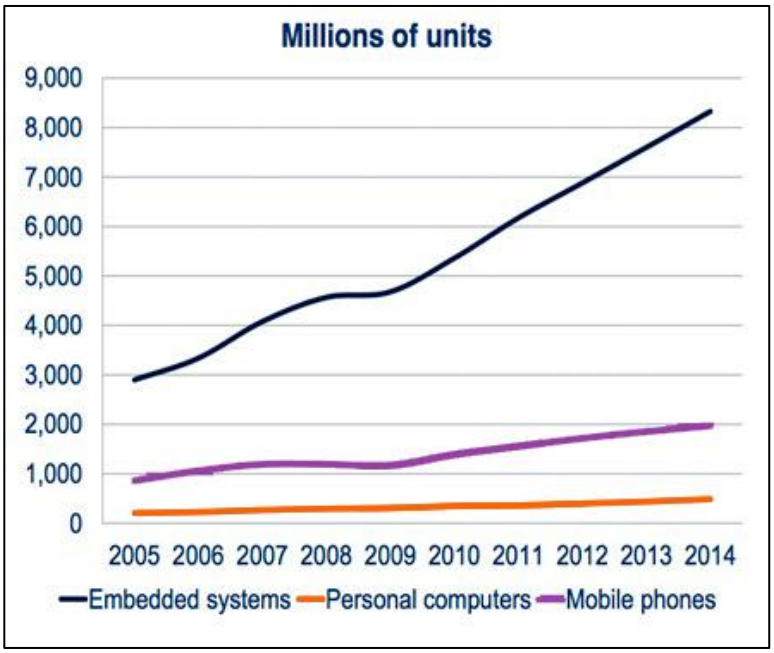
Figura 1. **Sistemas embebidos de un automóvil**



Fuente: *AA1 Car.* http://www.aa1car.com/library/bywire2.jpg. Consulta: 13 de junio de 2017.

El mercado de electrónicos es altamente dominado por los sistemas embebidos (por encima de los teléfonos móviles y computadoras personales), impulsados por el crecimiento constante, en producción y ventas, de dispositivos electrónicos para el consumidor y el incremento en la inversión en la automatización de la industria.

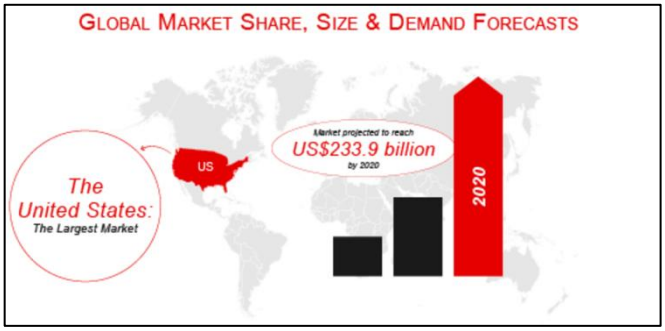
Figura 2. **Mercado de electrónicos en millones de unidades**



Fuente: *Sistemas embebidos: innovando hacia los sistemas inteligentes.* http://www.semanticwebbuilder.org.mx/es_mx/swb/Sistemas_Embebidos_Innovando_hacia_los_Sistemas_Inteligentes_. Consulta: 13 de junio de 2017.

En el 2013 se estimó que el mercado de sistemas embebidos fue de US$ 140 mil millones con muchos analistas proyectando que para el año 2020 superaría los US$ 234 billones.

Figura 3. **Tamaño del mercado global de sistemas embebidos**



Fuente: *Embedded systems market trends*. http://www.strategyr.com/MarketResearch/Embedded_Systems_Market_Trends.asp. Consulta: 13 de junio de 2017.

Tras establecer su importancia, se procederá a definir el concepto de sistema embebido.

2.1. Definición

Tabla IV. **Definición de sistema embebido**

Sistema de computación diseñado para realizar una o pocas funciones dedicadas e integrada en un dispositivo.

Fuente: elaboración propia.

Al hablar de sistemas embebidos, se hace referencia específicamente a un sistema de microcomputador embebido; se procederá a considerar cada una de las palabras por separado. La palabra embebido implica que se encuentra adjunto o ligado dentro de algo. Un sistema se refiere a un arreglo en el que todas las unidades trabajan en conjunto de acuerdo a un plan. Finalmente, el

término micro significa pequeño y computadora implica que tiene una unidad de procesamiento (procesador), memoria y los medios necesarios para el intercambio de datos con el mundo externo.

Físicamente, esto determina qué es un sistema embebido; sin embargo, es la funcionalidad la que realmente lo define. Esto es porque los sistemas embebidos son creados para una o muy pocas funciones específicas. Si se hace la comparación con una computadora personal, aun cuando esta cumple físicamente con los requerimientos de un sistema embebido, se notará que su funcionalidad difiere, pues una computadora personal debe ser flexible: suelen brindar al usuario la capacidad de trabajar en diferentes tareas en paralelo con diferentes usos, incluso abriendo la posibilidad de poder realizar una función diferente cada día.

Un sistema embebido debe cumplir y dedicarse por completo a la función para la que fue creado, similar a lo que haría un ASIC (circuito integrado de aplicación específica) con la posibilidad de variar sus parámetros solamente cambiando algunas líneas de código en su programación.

Como punto final, se debe tener en cuenta que un sistema embebido se encuentra integrado en un dispositivo, entiéndase dentro del producto final (como un horno microondas o un refrigerador, por ejemplo); es posible encontrarlo como componente de un sistema mucho más grande. Automóviles modernos pueden contar con más de veinte sistemas embebidos, cada uno dedicado a una función específica.

El diseño de sistemas es una sinergia productiva entre diseño de software y hardware, que escoge la combinación correcta de componentes para lograr las metas en termino de velocidad y eficiencia.

2.2. Historia

La historia y evolución de los sistemas embebidos se encuentra estrechamente ligada a sus aplicaciones y a la evolución de los dispositivos electrónicos y sistemas de computación.

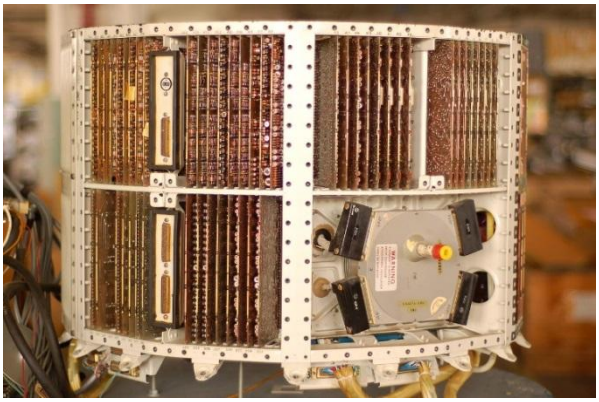
El primer sistema embebido reconocido fue desarrollado en el año 1960 por C.S Draper en el laboratorio de instrumentación del Instituto de Tecnología de Massachusetts (MIT, por sus siglas en inglés). Este consistía en un sistema de guía para las misiones Apollo en su viaje hacia la luna. Su función consistía en manejar el sistema de guía inercial de los módulos de excursión lunar. Este sistema fue el primero en utilizar circuitos integrados para la reducción de tamaño y peso, así como en contar con una memoria RAM magnética. El programa fue escrito en el lenguaje ensamblador propio de la arquitectura del sistema.

Figura 4. **Sistema de guía del Apollo**

Fuente: *MIT Museum.* http://museum.mit.edu/nom150/entries/518.
Consulta: 13 de junio de 2017.

En 1961 se comienza a producir el primer sistema embebido en masa: The Autonetics D-17 guidance computer. Este fue el sistema encargado de la guía del misil norteamericano Minuteman II, construido con lógica de transistores en circuitos integrados que contaban con la capacidad de reprogramar los algoritmos de guía.

Figura 5. **Autonetics D-17**



Fuente: *Minute Man Missile*. http://www.minutemanmissile.com/images/AutoneticsD17BGuidanceComputerB.jpg Consulta: 22 de junio de 2017.

En 1965 se crea la minicomputadora DEC 12 bit PDP-8, la primera computadora embebida en la instrumentación comercial y la computadora más vendida para el año 1973.

En 1969 se crea el primer sistema embebido en un carro, el sistema de inyección de gasolina para el Volkswagen 1600.

En 1971 se crea el primer microprocesador de 4 bits: el Intel 4004, diseñado específicamente para el uso en calculadoras electrónicas.

En 1974 Intel diseña el microprocesador 8008 mientras que Motorola introduce el MC6800, ambos los primeros microprocesadores de 8 bits.

En 1975 se crea la primera computadora personal por MITS: la Altair 8800.

Figura 6. **Altair 8800**

Fuente: *Old Computers*. http://oldcomputers.net/altair-8800.html. Consulta: 22 de junio de 2017.

En 1980 se crean los primeros microcontroladores, los cuales a diferencia de los microprocesadores se crearon para optimizar su tamaño físico y reducir el consumo de potencia.

Para este punto, el mercado de los sistemas embebidos ya se encontraba consolidado y comenzando a crecer a gran escala.

En 1984 Xilinx crea las FPGA las cuales vendrían a cambiar el diseño de sistemas por medio de circuitos integrados reprogramables.

En 1987 Wind River introduce el sistema operativo para sistemas embebidos: VxWorks; el cual terminaría siendo utilizado para el robot explorador de Marte en 1997.

En 1990 los sistemas embebidos dan un gran salto en su evolución con la consolidación de la World Wide Web, de modo que los diseños avanzaron utilizando una tecnología de computación distribuida por medio de la red como nuevo recurso, por ejemplo, Google.

En 1992 los sistemas embebidos se vuelven inalámbricos con más de 10 millones de teléfonos móviles en uso.

Tras esto, los sistemas embebidos ya habían madurado completamente con pocas mejoras significativas en su evolución.

En 1999 Linux debuta en el mercado por medio de programas embebidos de Linux, naciendo así los primeros sistemas embebidos de Linux.

En el año 2005 Google compra Android Inc., con lo cual en el 2008 el código Android se declara código abierto; Android es actualmente una de las plataformas más utilizadas para el diseño de sistemas embebidos.

Actualmente, más del 95 % de los chips electrónicos producidos son destinados para el diseño de sistemas embebidos.

2.3. Estructura y características de un sistema embebido

Un sistema embebido se encuentra, en muchas ocasiones, conformado principalmente por una unidad de procesamiento y un programa que se ejecuta

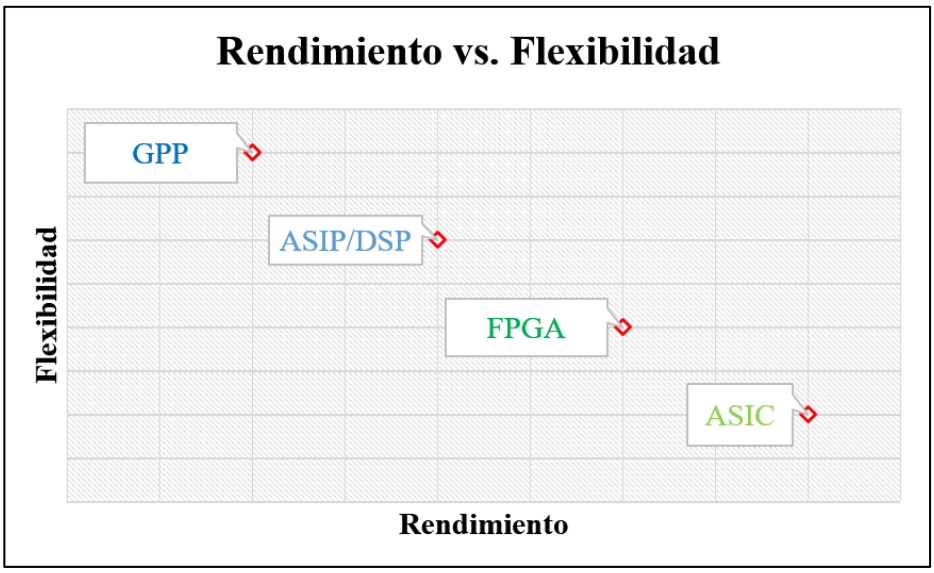
sobre este (existen casos en los que se trata de un conjunto o arreglo de transistores configurados para sintetizar funciones de hardware, como en el caso de las FPGA). Dado que el programa necesita un lugar donde almacenarse, todo sistema embebido necesitará de cierta cantidad de memoria. Finalmente, por las aplicaciones para las que se utiliza, un sistema embebido debe contar con alguna forma de comunicación con el mundo exterior, presentándose en forma de entradas y salidas. Estos tres componentes interconectados entre sí forman la estructura de todo sistema embebido. Existen más partes que se pueden enumerar en su disposición, pero estos son dependientes de aplicaciones específicas.

La elección de una unidad de procesamiento suele ser un esfuerzo largo pues se cuenta con una gran cantidad de dispositivos que cumplen los requisitos. Para el diseño de sistemas embebidos se optará por el uso de procesadores embebidos, pues estos traen incorporados los circuitos necesarios para su funcionamiento, sin depender de circuitos externos. De manera general se pueden dividir los procesadores embebidos en cuatro grandes categorías de la siguiente manera:

- Procesadores de uso general (GPP).
- Procesadores con sets de instrucciones para aplicaciones específicas (ASIP).
- Hardware reconfigurable (FPGA).
- Circuitos integrados de aplicación específica (ASIC).

Cualquiera puede utilizarse para el diseño de un sistema embebido, pero se deben tomar en cuenta sus ventajas específicas ya sea en flexibilidad o en rendimiento.

Figura 7. **Comparación entre unidades de procesamiento**



Fuente: elaboración propia, utilizando Excel.

2.4. Características

Un sistema embebido debe cumplir con ciertas características para poder ser considerado como tal:

- Ejecutar una o muy pocas tareas

En el momento que un sistema se encuentra ejecutando muchas tareas, pierde su objetivo de ser dedicado lo que no permite la optimización de recursos y tamaño del dispositivo final. Existen casos excepcionales en que ciertos sistemas especiales (como *real time operating systems* o RTOS) realizan varias tareas y son aún considerados sistemas embebidos por cumplir con muchas otras características de esta categoría.

- Procesamiento en tiempo real

Definir el tiempo real puede ser complejo, pues no existe una constante que indique a partir de qué momento cuenta como tal. Se considera que un sistema realiza procesamiento en tiempo real cuando es capaz de terminar procesos más rápido de lo que suceden eventos externos que modifiquen el sistema de modo que su reacción sea aparentemente instantánea. Respuestas que llegan muy tarde o muy temprano son incorrectas.

- Bajo costo

El sistema embebido es concebido para ser producido en miles o millones de unidades y comercializado. Por tanto, el costo por unidad es un aspecto importante a tener en cuenta durante su diseño. La elección de una unidad de procesamiento adecuada es clave para esto.

- Bajo consumo de potencia

Un sistema con un alto consumo de potencia difícilmente es bien aceptado por el público, puesto que eso implica un mayor gasto a largo plazo. Parte de la ingeniería del diseño de sistemas embebidos se enfoca en la optimización del consumo energético del sistema. Debe considerarse también que muchos dispositivos funcionarán alimentados por una batería, por lo que un mayor consumo implica una batería de mayor tamaño y, por tanto, un incremento total en el tamaño físico total del producto final afectando.

- Mínimo uso de memoria

El tamaño del código debe ser el mínimo posible, al igual que el uso de memoria que responde siempre a mantener dimensiones físicas pequeñas y costos bajos. Así mismo, el sistema embebido cuenta con la posibilidad de evolucionar con el tiempo, por lo que revisiones, mejoras o implementaciones nuevas del código pueden generarse y de no haber sido optimizado el código desde un comienzo podría provocar que el sistema no cuente con los recursos necesarios para trabajar con la nueva versión del programa.

- Dimensiones físicas pequeñas

Si se desea reducir el costo y consumo energético, se debe tener la capacidad de administrar bien los recursos del sistema utilizando solo el mínimo necesario. Al momento de diseñar un sistema embebido, se debe considerar que su unidad de procesamiento sea de dimensiones físicas pequeñas para no afectar los costos de producción. En sistemas más estrictos el tamaño puede llegar a ser crucial pues se cuenta con un espacio reducido donde se debe instalar uno o más sistemas embebidos.

- Capaz de interactuar con el mundo físico

Un sistema embebido interactuará con su entorno por medio de sensores y actuadores, por lo que periféricos que permitan esta interacción con el mundo físico son obligatorios.

2.5. Tipos de sistemas embebidos

Ya se ha definido que una de las características de un sistema embebido es trabajar en tiempo real. Todas las aplicaciones en tiempo real se pueden clasificar en dos:

2.5.1. *Hard real time*

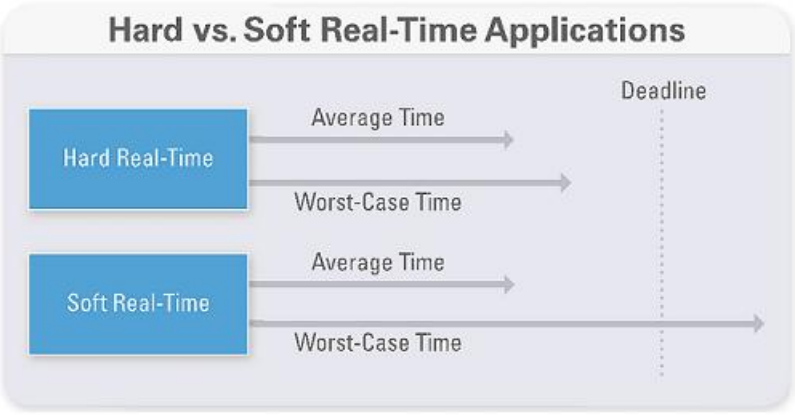
Un sistema *hard real time* es aquel cuya tarea será realizada antes que su tiempo límite; y en caso suceda algo inesperado que pueda prolongar la tarea, el tiempo del peor escenario será aún menor que el tiempo límite.

Estas aplicaciones son utilizadas generalmente en sistemas en los que un error podría costar una vida humana. Por ejemplo: el sistema que acciona la bolsa de aire de un carro. Hágase la suposición de que el tiempo para accionar la bolsa de aire de un carro debe ser como máximo 15 ms. Al momento de diseñar el sistema embebido para esta situación, la tarea debe cumplirse menos tiempo, asúmanse 5 ms. En caso algo retrasara el sistema, se debería buscar que, en el peor escenario, este completara la tarea entre 12 y 13 ms como máximo.

2.5.2. *Soft real time*

Un sistema *soft real time* es aquel que realizara su tarea antes del tiempo límite, sin embargo, si sucediera un contratiempo, no existiría un problema en pasar el tiempo límite pues el resultado no sería desastroso. Se puede tomar como ejemplo un reproductor de música, el cual realiza la tarea en 20 ms. Si existiera un contratiempo y se supera el tiempo límite de 30 ms esto no afectaría mucho pues el usuario difícilmente notaria que existió un problema.

Figura 8. **Tipos de sistemas embebidos**



Fuente: *National Instruments.* http://www.ni.com/white-paper/14238/en/.
Consulta: 22 de junio de 2017.

2.6. Diseño de sistemas embebidos

El diseño de sistemas embebidos es un arte que involucra la escritura eficiente de software y el uso óptimo del hardware. No obstante, pocas empresas cuentan con una metodología para el desarrollo de estos productos, que afecta directamente la calidad, costos de producción y, por tanto, un aumento en el costo final.

Para optimizar el diseño de sistemas embebidos es necesario comprender los requerimientos del producto para diseñar una plataforma adecuada, sobre la cual se ejecutará el software embebido que gobernará el sistema.

2.6.1. Procesos de desarrollo de sistemas embebidos

Se dividen los procesos necesarios para el desarrollo de sistemas embebidos en siete diferentes:

- Prueba de concepto: en esta etapa se busca valorar el concepto del producto previo a comenzar su desarrollo, con el propósito de verificar que el producto en cuestión puede ser explotado de manera útil. Entre las ventajas que ofrece se encuentran el incremento de la eficiencia del diseño, mejora su colaboración y asegura la exploración exhaustiva de conceptos. Una parte importante de este proceso es detallar el enfoque del producto, así como sus vulnerabilidades, limitaciones, funciones y tareas.

- Análisis y diseño: en esta etapa se analiza la viabilidad del producto, obteniendo los requerimientos funcionales y no funcionales del sistema. El proceso debe partir desde la información esencial hasta el detalle de la implementación. Se debe realizar un análisis técnico, económico, y se deben establecer las restricciones de presupuestos y planificación.

- Desarrollo del hardware: esta etapa implica una investigación exhaustiva del procesador embebido a utilizar, que verifican que cumpla con los requerimientos del sistema. Es importante recordar que un sistema con mayores características generalmente tendrá un mayor consumo de potencia, por lo que es necesario saber decidir en el modelo que cumpla las necesidades a cabalidad. Esta etapa, también, involucra todo el hardware del sistema fuera del procesador embebido.

- Desarrollo del software: esta etapa implica la escritura eficiente del código y el uso mínimo de memoria, que toman en cuenta la posibilidad de una revisión de versión hecha posteriormente. Para esto el diseñador debe dominar la configuración óptima de periféricos, la cual se abordará en el capítulo 6.

- Integración y pruebas: es usual que un sistema embebido no sea desarrollado solamente por un ingeniero. El proceso de integración entre las diferentes partes desarrolladas por cada área es un proceso que se debe verificar minuciosamente. Existen múltiples plataformas que permiten llevar un historial de versiones. El proceso de prueba se realiza generalmente por medio de una tarjeta de desarrollo para crear así el primer prototipo funcional.

- Administración del producto: finalmente, esta etapa busca la administración de la producción en masa del equipo. Se realizan pruebas de errores y vulnerabilidades que culminan con la comercialización del producto.

3. MICROCONTROLADORES

La elección de un procesador embebido es un proceso que no debe ser tomado a la ligera. Una mala elección podría afectar el desempeño final del sistema que obliga a descartar meses de trabajo. Existen cientos de dispositivos diferentes que pueden cumplir el rol, lo que puede volver tedioso este proceso. Para realizar la elección, como primer paso, se deben conocer las necesidades del sistema, detallar su función y definir su público objetivo, por mencionar algunos factores. Seguido se deben verificar sus restricciones para conseguir el rendimiento y la flexibilidad que se busca, escogiendo una de las cuatro familias de procesadores embebidos existentes. Finalmente, una vez ya se ha elegido una, se optará por verificar cada una de las características de los dispositivos de esa familia para determinar cuál cumple los requisitos.

Existe una familia dentro de los procesadores de propósito general que destaca sobre todos los demás debido a que fue creada con un propósito específico: ser el corazón de sistemas embebidos. Estos dispositivos reciben el nombre de microcontroladores y suelen ser confundidos con microprocesadores por su aparente parecido. Los términos microprocesador y microcontrolador suelen confundirse e intercambiarse como si se refirieran al mismo concepto. Ambos tienen ventajas y desventajas dependiendo de la aplicación o solución que se está desarrollando.

3.1. Características

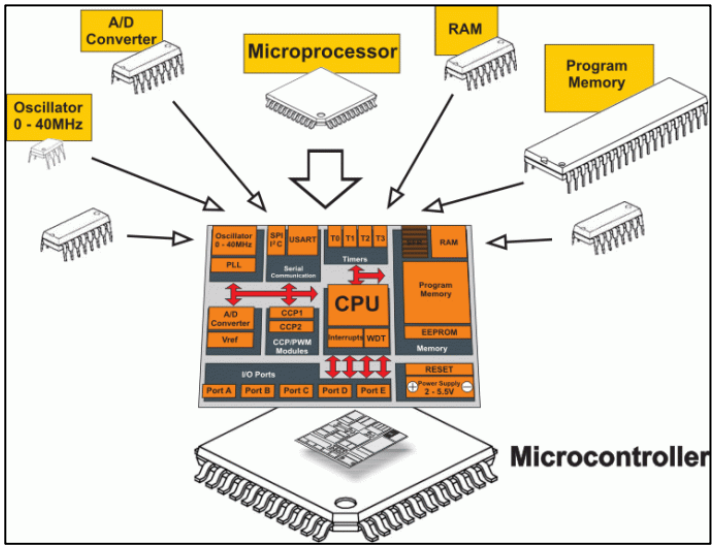
Tabla V. **Definición de microcontrolador**

Circuito integrado programable, compuesto por una unidad central de procesamiento, memoria y periféricos de entrada y salida dentro de un chip. Ejecuta instrucciones grabadas en su memoria para realizar una tarea específica.

Fuente: elaboración propia.

Un microcontrolador se compone por una unidad central de procesamiento (CPU) que funciona como el cerebro central del sistema, un módulo de memoria con diferentes tipos de memoria ya sea para el almacenamiento de datos o de programa (RAM, ROM); finalmente, por periféricos, entre los cuales se distinguen entradas, salidas, temporizadores, conversores, entre otros.

Figura 9. **Microcontrolador**



Fuente*: Microcontrolador vs. Microprocesador*. http://maxembedded.com/2011/06/mcu-vs-mpu/. Consulta: 22 de junio de 2017.

Se estima que más de la mitad de los dispositivos de procesamiento de información vendidos actualmente son microcontroladores. Esto se debe a que los microcontroladores ofrecen una amplia gama de aplicaciones aun cuando solo algunas se exploran usualmente. Existen varios fabricantes de microcontroladores: Texas Instruments, Motorola, Atmel, Intel, Microchip, Toshiba y National Instruments, por mencionar algunos. Todos ofrecen microcontroladores con características similares, con modelos que varían en cuanto a memoria, velocidad, periféricos, etc.

3.2. Historia

La historia de los microcontroladores comienza en 1970. En aquella época, Intel se encontraba trabajando en el desarrollo de un microprocesador. Paralelamente, los ingenieros de Texas Instruments, Gary Boone y Michael Cochran, comenzaban a trabajar en una idea similar. Texas Instruments llevaba ya un tiempo pensando en desarrollar calculadoras de bolsillo y Boone desarrolló un circuito integrado en chip con casi todo lo necesario para esto. Este dispositivo recibió el nombre de TMS1802NC y contaba con cinco mil transistores, tres mil bits de memoria para programa, 128 bits de memoria de acceso aleatorio y la posibilidad de ser reprogramado.

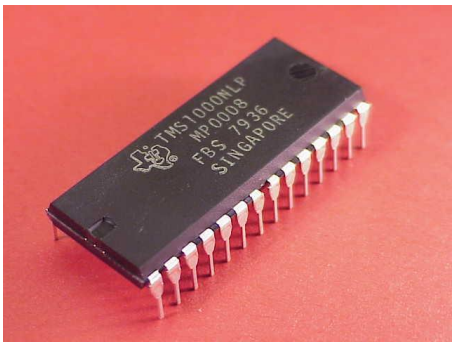
Figura 10. **TMS 1802**

Fuente: *Microcontrolador.* http://ethw.org/Microcontroller. Consulta: 22 de junio de 2017.

A pesar de haber sido creado para calculadoras, rápidamente se notó que, con un poco de programación, podría ser utilizado para una amplia variedad de aplicaciones. El microprocesador, que era la base para computadoras más poderosas, contaba con mayor capacidad de procesamiento, pero requería de memorias, periféricos y tarjetas externas para funcionar. El microcontrolador era capaz de desarrollar funciones de manera independiente sin necesidad de dispositivos externos o con la ayuda mínima de otros circuitos y módulos.

En 1974, Texas Instruments finalmente desarrolló el primer microcontrolador de propósito general, el TMS 1000, que era un dispositivo de 4 bits enfocado en el diseño de sistemas embebidos combinando una memoria solo de lectura, una memoria de lectura y escritura, un procesador y un reloj; todo en el mismo empaquetado.

Figura 11. **TMS 1000**



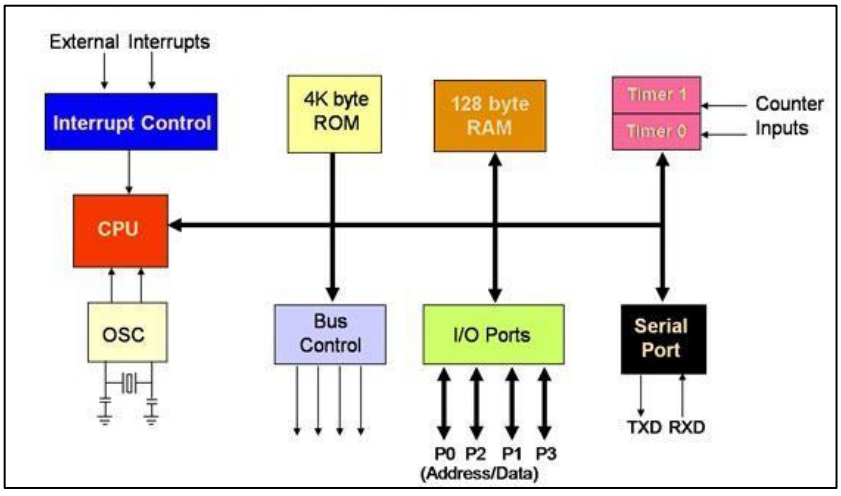
Fuente: *TI TMS 1000 microcontroller.* http://www.computermuseum.li/Testpage/Chip-TexasInstrumentsTMS1000.htm Consulta: 23 de junio de 2017.

En 1976, Intel y Mostek introdujeron nuevos microcontroladores con arquitecturas de 8 bits: El MK3870 por parte de Mostek y la familia MCS-48 de Intel. Ambos fueron desarrollados para cumplir con las aplicaciones más

demandantes de la época: automóviles y periféricos de computadoras. La versión 8048 de Intel contaba con una memoria EPROM. Esta nueva característica brindaba a los desarrolladores la capacidad de realizar prototipos de forma más ágil debido a la facilidad de borrado eléctrico que implicaba.

En 1980 el mercado de los microcontroladores había crecido tanto que existían arquitecturas propias en Europa, Japón y Estados Unidos, capaces de servir a una gran cantidad de aplicaciones. De estos el más importante fue el Intel 8051, microcontrolador de 8 bits que marcó la segunda generación de microcontroladores. Su arquitectura definió la tendencia de la época utilizando características que aún se encuentran presentes en dispositivos modernos. Utilizaban una arquitectura conocida como Harvard modificada y contaba con 128 bytes de RAM, 4K byte de ROM, temporizadores, puerto serial y cuatro puertos de 8 bits.

Figura 12. **Intel 8051**



Fuente: *8051 Microcontroller.*
https://www.tutorialspoint.com/embedded_systems/es_microcontroller.htm.
Consulta: 23 de junio de 2017.

A mediados de la década de los ochentas, Motorola hace su aparición en el mercado con el microcontrolador 68HC05. Este microcontrolador de alto rendimiento implementaba la arquitectura Von-Neuman en que las instrucciones, datos, entradas/salidas y temporizadores ocupan un mismo espacio de memoria. El puntero de pila tenía un ancho de palabra de 5 bits, lo que limitaba la pila a 32 posiciones. Entre sus periféricos destacados se encontraban: conversor A/D, sintetizador PLL y entradas y salidas en serie.

General Instrument's Microelectronics Division comenzó a trabajar en microcontroladores desde 1976, sin embargo, fue hasta 1985 cuando vendió su división de micro electrónicos. Los nuevos propietarios decidieron cancelar todos los proyectos vigentes en ese momento debido a que se encontraban obsoletos, excepto por el microcontrolador PIC, que fue mejorado con la inclusión de una memoria EPROM. El PIC ha vendido más de doce mil millones de unidades desde entonces; es tan popular por ser el primer microcontrolador en implementar un set de instrucciones RISC. Esto implica que la simplicidad de diseño permite añadir más características a bajo precio.

Las arquitecturas de 16-bits comenzaron como modelos híbridos. Los microcontroladores de 8 bits comenzaron a agregar registros de 16 bits dentro de sus arquitecturas. El primer modelo establecido de 16 bits fue el 68HC11 de Freescale, introducido en 1985.

En 1989 Motorola anuncia la llegada de su microcontrolador 68332, el primer microcontrolador de 32 bits.

Las arquitecturas de 16 bits nunca vieron mucha luz en el mercado, pues rápidamente fueron desplazadas por las arquitecturas de 32 bits. En la pelea del mercado, las arquitecturas de 8 bits dominaban por ser de muy bajo costo,

mientras las de 32 bits proveían un alto rendimiento. En el momento mismo las arquitecturas de 8 bits empezaron a mejorar su rendimiento y las de 32 bits empezaron a bajar su precio, las arquitecturas de 16 bits fueron en declive.

3.3. Estructura de un microcontrolador

Ya se ha hablado de que un microcontrolador se encuentra compuesto por una unidad central de procesamiento, memoria y periféricos de entrada y salida integrados en un solo empaquetado. Para comprender su estructura se debe analizar cada uno de sus módulos.

3.3.1. Unidad central de procesamiento (CPU)

Tabla VI. **Definición de CPU**

Dispositivo encargado de interpretar las instrucciones de un programa mediante la realización de operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

Fuente: elaboración propia.

Este se divide en tres áreas principales:

- Unidad lógica aritmética (ALU): circuito digital combinacional encargado de realizar operaciones aritméticas y lógicas de números binarios enteros.

- Unidad de control (CU): encargada de decodificar las instrucciones y controlar cada uno de los componentes internos de la CPU para que esta funcione.

- Registros: Pequeñas memorias de rápido acceso. Su tamaño es dependiente de la arquitectura de la CPU, van desde 4 hasta 64 bits. Se dividen en:

    - Registros de propósito general
    - Registros de datos
    - Registros de memoria
    - Registros de punto flotante
    - Registros constantes
    - Registros de propósito específico

3.3.2. Unidad de memoria

Tabla VII. **Definición de memoria**

Colección de elementos de hardware encargados del almacenamiento de información.

Fuente: elaboración propia.

Se dividen de la siguiente forma:

- Por su tipo

    - Volátiles: almacenan datos mientras estén alimentadas
    - No volátiles: almacenan datos aun sin estar alimentadas

- Por tecnología empleada

    - Ópticos

- Magnéticos
- Semiconductores

3.3.2.1. Memoria de acceso aleatorio (RAM)

Memoria volátil. Es la encargada de almacenar el sistema operativo, los programas, los datos y el código en funcionamiento. Recibe el nombre de acceso aleatorio porque el tiempo de respuesta tras realizar una lectura o escritura no depende de la posición de memoria con la que trabaja. Esto implica que estas memorias no deben seguir un orden secuencial para acceder a la información de la manera más rápida posible.

Existen dos tipos de memoria RAM:

- SRAM (*static random access memory*): es de acceso rápido y simple, pues no necesita ser actualizada periódicamente. Sin embargo, son más caras y de menor capacidad que las de otros tipos.

- DRAM (*dynamic random access memory*): necesitan ser refrescadas periódicamente para no perder la información. Tienen mayor capacidad y suelen ser más baratas que las SRAM.

Las memorias SRAM son empleadas generalmente en la memoria caché, y las DRAM en la memoria principal del sistema.

3.3.2.2. Memoria de solo lectura (ROM)

Estas son memorias no volátiles. Se usan en los sistemas para almacenar datos aun en ausencia de una fuente de alimentación. Se llaman memorias de

solo lectura pues su contenido no permite escritura, solamente lectura. Actualmente, la variante de la memoria ROM más utilizada es la EEPROM (siglas en inglés de *electrically-erasable programable,* ROM) las cuales permiten borrar su contenido de una manera simple por medio de un aumento de potencial eléctrico. La memoria flash de muchos sistemas actuales es de tipo EEPROM, con una velocidad de lectura aceptable aun siendo más lentas que las memorias SRAM o DRAM.

3.3.3. Periféricos

Los periféricos se dividen en dos categorías:

- Entradas/salidas
- Circuitos especializados

3.3.3.1. Entradas y salidas

Debido a que un microcontrolador está diseñado para interactuar con su entorno físico, comunicarse con su exterior es imprescindible los microcontroladores cuentan con una serie de pines de propósito general que funcionan como entradas y salidas digitales del sistema que brindan información por medio de señales eléctricas.

3.3.3.2. Circuitos especializados

Dentro de estos se habla de cualquier circuito electrónico integrado dentro del microcontrolador encargado de realizar una función específica. Algunos periféricos, como el reloj del sistema, son necesarios para el funcionamiento del microcontrolador. Existe una gran variedad de circuitos especializados y cada

uno varía entre modelos diferentes. Es necesario analizar un modelo específico para poder detallar cada uno.

3.3.4. Buses de comunicación

Componentes de interconexión dentro del sistema, encargados de la transmisión de datos entre componentes internos. Existen tres buses dentro de un microcontrolador:

- Bus de control: encargado de la transmisión de ordenes entre módulos para el uso y acceso a las líneas de datos y direcciones.

- Bus de datos: encargado de transmitir datos entre módulos internos.

- Bus de direccionamiento: encargado de transportar la dirección de lectura o escritura entre unidades.

3.4. Arquitectura de microprocesadores

Tabla VIII. **Definición de ISA**

Se denomina ISA (siglas en inglés de *instruction set architecture*) al conjunto de instrucciones, tipos de datos, estados y semántica que el procesador podrá ejecutar.

Fuente: elaboración propia.

Existen diferentes tipos de ISA, sin embargo, los dos predominantes en el mercado son:

- CISC (siglas en inglés de *complex instruction set computer*): este tipo de arquitectura trabaja ejecutando muchas operaciones de bajo nivel. También, permite realizar operaciones de muchos pasos o modos de direccionamiento con una o pocas instrucciones. En general, una arquitectura CISC realizara una tarea en muy pocas instrucciones, pues cada instrucción realiza más acciones y consume más energía.

- RISC (siglas en inglés de *reduced instruction set computer*): este tipo de arquitectura utiliza instrucciones simples que pueden distribuirse para realizar operaciones de bajo nivel dentro de un solo ciclo de reloj. Se necesitan más instrucciones para realizar una tarea, pero su consumo energético es menor.

Para el diseño de sistemas embebidos se le dará preferencia a la arquitectura RISC. Esto debido a que generalmente un sistema embebido no se compone por algoritmos complejos, pero si suele ser alimentado con fuentes de energía limitadas como baterías, de modo que el ahorro energético prevalece sobre el tiempo de computación.

3.5. Arquitecturas de memoria

En la actualidad existen dos arquitecturas de memoria. Estas han ido evolucionando desde su aparición, pero conservan el nombre e idea originales.

3.5.1. Arquitectura Von-Neumann

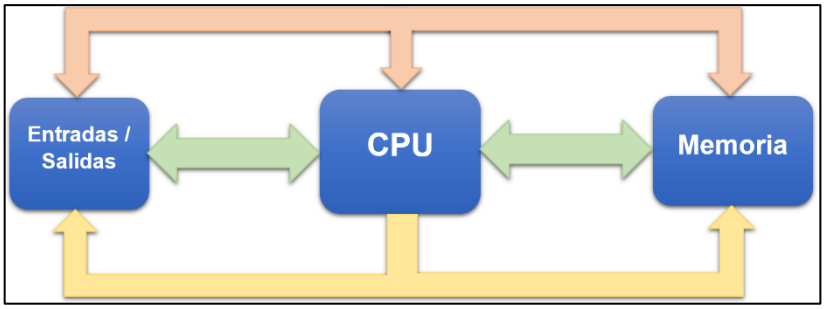
Los primeros sistemas computacionales surgieron de la arquitectura descrita por el matemático John Von-Neumann en 1945 quien expuso que para

construir un sistema computacional eran necesario cuatro componentes básicos:

- Unidad de procesamiento central
- Unidad de memoria
- Entradas y salidas
- Buses de comunicación

La importancia de esta arquitectura es que opera con una sola unidad de memoria; por lo que los datos y el programa se graban en la misma memoria y comparten los mismos buses de comunicación.

Figura 13. **Diagrama de la arquitectura Von-Neumann**



Fuente: elaboración propia, utilizando Microsoft Excel.

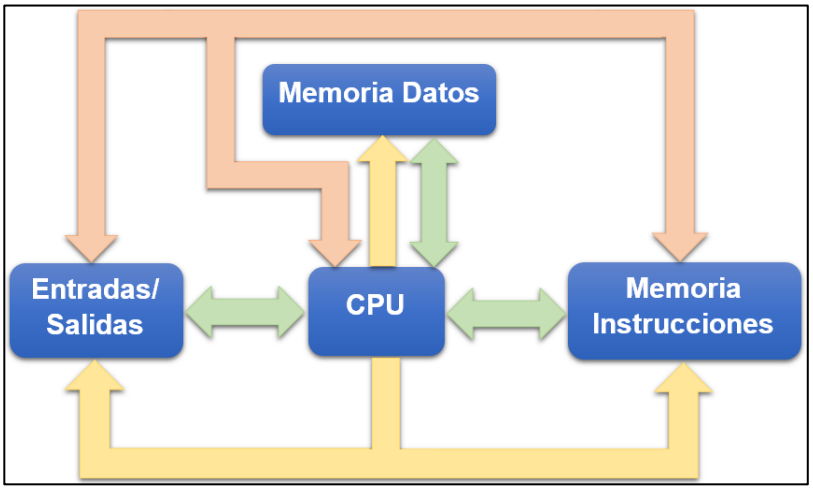
3.5.2. Arquitectura Harvard

Fue desarrollada de manera casi paralela a la de Von-Neumann por estudiantes de la Universidad de Harvard, quienes trabajaban en el sistema computacional llamado Harvard Mark I. Esta arquitectura funciona con los siguientes componentes básicos:

- Unidad de procesamiento central
- Unidad de memoria de instrucciones
- Unidad de memoria de datos
- Entradas y salidas
- Buses de comunicación

Este sistema planteaba una arquitectura similar a la de Von-Neumann con la diferencia de que incorporaba una memoria para instrucciones y una memoria para datos, cada una con un bus diferente, lo que permitía una mayor velocidad de procesamiento.

Figura 14. **Diagrama de la arquitectura Harvard**



Fuente: elaboración propia, utilizando Microsoft Excel.

La figura 14 muestra el diagrama de bloques de esta arquitectura, en rojo se encuentra el bus de control, en verde el bus de datos y en amarillo el bus de direccionamiento.

3.5.2.1. Arquitectura Harvard modificada

Es una variación de la arquitectura Harvard; permite el acceso al contenido de la memoria de instrucciones como si fueran datos. De esta manera se libera la estricta separación entre las instrucciones y los datos. La mayoría de arquitecturas Harvard actualmente son de hecho Harvard modificada, pues proveen las ventajas de una arquitectura Harvard junto a la simplicidad de una arquitectura Von-Neuman.

4. MICROCONTROLADOR TM4C123GH6PM

El TM4C123GH6PM de Texas Instruments es un microcontrolador de alto rendimiento de 32-bits con una arquitectura ARM Córtex-M y una amplia cantidad de herramientas para el desarrollo de aplicaciones.

4.1. Características

Tabla IX. **Especificaciones del TM4C123GH6PM**

Características	Detalles
Rendimiento	
Núcleo	ARM Cortex-M4F
Rendimiento	80-MHz; 100 DMIPS
Flash	256 KB de ciclo único
SRAM	32 KB de ciclo único
EEPROM	2KB
ROM Interna	Interna con Tivaware cargado
Interfaces de comunicación	
Universal asynchronous Receivers/transmitter (UART)	8 módulos
Synchronous serial interface (SSI)	4 módulos
Inter-integrated circuit (I2C)	4 módulos con cuatro velocidades de transmisión incluido modo de alta velocidad
Controller area network (CAN)	2 controladores CAN 2.0 A/B
Universal serial bus (USB)	USB 2.0 OTG/Host/Device
Integración del Sistema	
Micro direct memory access (µDMA)	32 canales
General-purpose timer (GPTM)	Seis contadores de 16/32 bits y seis contadores de 32/64 bits
Watchdog timer (WDT)	2 módulos
Hibernation module (HIB)	Módulo de bajo consumo
General-purpose input/output (GPIO)	6 bloques físicos

Continuación de la tabla IX.

Control avanzado de movimiento	
Pulse width modulator (PWM)	2 módulos, cada uno con 4 generadores, haciendo un total de 16 salidas PWM
Quadrature encoder interface (QEI)	2 módulos
Soporte analógico	
Analog-to-digital converter (ADC)	2 módulos de 12 bits de resolución con una tasa de conversión de 1 millón de muestras por segundo cada uno
Comparador analógico	2 comparadores analógicos independientes integrados
Comparador digital	16 comparadores
JTAG *and serial wire debug* (SWD)	Un módulo JTAG con ARM *serial wire debug* integrado

Fuente: *Texas Instruments.* http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. Consulta: 23 de junio de 2017.

El hecho de que este microcontrolador se encuentre diseñado sobre una arquitectura ARM Córtex-M implica que el procesador proveerá alto rendimiento, bajo costo, mínimo uso de memoria, uso reducido de pines y bajo consumo de potencia, mientras brinda un alto rendimiento computacional y una respuesta excepcional ante interrupciones.

Entre sus características principales están:

4.1.1. Procesador ARM Cortex-M4F

4.1.1.1. Procesador ARM Cortex-M4F

ARM (siglas en inglés de *advanced RISC machine*) es una arquitectura RISC de 32 y 64 bits. Fue desarrollada por la compañía ARM Holdings que solamente se limita a licenciar sus diseños.

La característica principal de los procesadores ARM, es su bajo consumo de potencia, lo cual los hace perfectos para el uso en dispositivos portátiles. Esta simplicidad se debe a la falta de microcódigo y su ISA basado en RISC que utiliza una menor cantidad de transistores en comparación con otras arquitecturas.

4.1.1.2. Historia

En 1980, la compañía británica Acorn Computers, dedicada a la fabricación de computadoras, empezó a desarrollar la arquitectura Acorn RISC Machine (ARM). Los primeros productos ARM eran coprocesadores para las series de computadoras BBC Micro. Dado que esta computadora fue un éxito, Acorn Computers consideró modificar el diseño de sus coprocesadores para competir en el mercado de procesadores dominado por IBM. Tras intentar trabajar junto a procesadores de otras marcas como Motorola y National Instruments y no lograr los requisitos propuestos, Acorn Computers decidió que era necesaria una nueva arquitectura.

Inspirados por los documentos técnicos publicados por el proyecto Berkeley RISC, Acorn Computers comenzó a diseñar su propio procesador. Este proyecto recibiría el nombre de Acorn RISC Machine Project que comenzó en el año 1983.

El objetivo era lograr manejar interrupciones, entradas y salidas con baja latencia. Los primeros diseños de ARM trabajaron exitosamente en las pruebas hechas en 1985. En el año 1986 se introdujo al mercado la primera versión utilizada comercialmente, bajo el nombre de ARM2. El éxito del ARM2 llevó a la compañía Apple Computer a proponer trabajar en conjunto con Acorn Computers para desarrollar nuevas versiones del núcleo ARM. Sin embargo,

Acorn era un competidor directo de Apple Computer, por lo que notó que una compañía que fabrica computadoras y también fabrica procesadores provocaría que los clientes se retiraran, así que era necesario crear una nueva compañía. Para el año 1990 se crea oficialmente la nueva compañía bajo el nombre de Advanced RISC Machines Ltd. conformada por Acorn Computers, Apple Computers y VLSI Technologies. De esta unión nace el ARM6, coprocesador que Apple utilizaría para sus productos. El núcleo mantuvo su simplicidad a pesar de los cambios. La idea era que el usuario final combinara el núcleo del ARM con un número opcional de periféricos integrados y otros elementos, pudiendo crear un procesador completo a la medida de sus necesidades.

El diseño de ARM tuvo su mayor alcance al patentarse, logró que compañías como Freescale, IBM, Infineon Technologies, OKI, Texas Instruments, Nintendo, Philips, VLSI, Atmel, Sharp, Samsung y STMicroelectronics licenciaran el diseño básico del ARM.

Actualmente 98 % de los dispositivos móviles y 75 % de los procesadores de 32 bits utilizan un núcleo con arquitectura ARM.

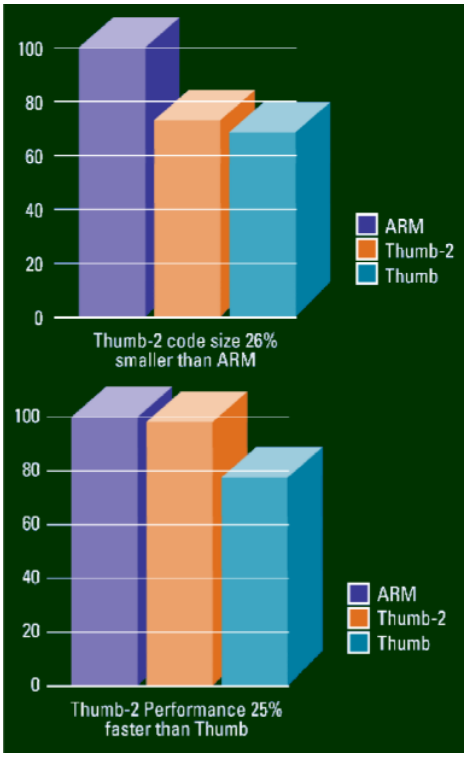
4.1.1.3. Set de instrucciones

Las arquitecturas ARM implementan dos sets de instrucciones: ARM y Thumb; su diferencia es su codificación. ARM es un set de instrucciones de 32 bits de largo, mientras que las instrucciones de Thumb son de 16-bits. ARM tiene una codificación de 4 bytes, esto implica que, sus instrucciones de 32-bits se dividen en cuatro bytes mientras que Thumb cuenta con una codificación de 2 bytes. El desarrollador decide qué tipo de codificación utilizará, de modo que el procesador decodificará las instrucciones como tipo Thumb o ARM. Las instrucciones ARM son más rápidas y potentes (hay instrucciones que solo

están en este modo) pero implican mayor consumo de memoria y de electricidad. Thumb por el contrario es más limitado, con instrucciones que ocupan 2 bytes, pero con menor consumo de potencia.

En las versiones ARMv-7 se comienza a implementar un nuevo set de instrucciones conocido como Thumb-2. Las instrucciones Thumb-2 son una mezcla entre ARM y Thumb, de modo que las instrucciones pueden ser de 16 o 32 bits. Tiene un código 26 % menor que ARM y 25 % en mejora de rendimiento con respecto a Thumb.

Figura 15. **Comparación entre set de instrucciones**



Fuente: ARM. *The ARM Architecture with a focus on v7A and Cortex-A8.* https://www.coursehero.com/file/13338101/ARM-Arch-A8/. Consulta: 25 de junio de 2017.

4.1.1.4. Familias

En la actualidad, ARM ofrece un amplio rango de núcleos para microprocesadores. Cada uno busca suplir necesidades de nichos específicos, ya sea en rendimiento, consumo de potencia o costo para todas las aplicaciones en el mercado. Estas se dividen en familias:

- Córtex-A (*application*): procesadores diseñados para aplicaciones de alto rendimiento con sistemas operativos generales. Se encuentran disponibles en un solo núcleo o hasta cuatro núcleos de computación con la habilidad de integrar bloques de procesamiento multimedia y operaciones avanzadas de punto flotante. Están diseñados para trabajar con sistemas operativos y aplicaciones de terceros.

- Cortex-R (*real time*): procesadores embebidos para sistemas de tiempo real críticos con alto rendimiento. Diseñados para aplicaciones que requieren bajo consumo de potencia, un buen manejo de las interrupciones y alta compatibilidad con las plataformas existentes.

- Cortex-M (*microcontroller*): procesadores orientados para el diseño de microcontroladores y sistemas embebidos. Se caracterizan por contar con una alta velocidad de respuesta, manejo determinístico de interrupciones y bajo consumo de potencia.

- SecurCore: procesadores de bajo consumo de potencia, bajo costo y buen rendimiento. Se especializan en aplicaciones de seguridad y cuentan con una gran cantidad de herramientas para ello.

4.1.1.4.1. Cortex-M

La familia de los Cortex-M fue diseñada específicamente para el mercado de los microcontroladores. Los procesadores Cortex-M cuentan con un alto rendimiento comparado con otro procesador típico de la mayoría de microcontroladores, destacan por su bajo consumo de potencia. Han llegado a ser tan bien aceptados que dominan el mercado de los microcontroladores; es la arquitectura más empleada de 32-bits en la historia. Entre sus características principales se encuentran:

- Trabajar con un set de instrucciones Thumb-2
- Alto rendimiento y eficiencia
- Compatibilidad con lenguajes C y ASM
- Herramientas de fácil aprendizaje para desarrollo
- Soporte para sistemas operativos
- Fácil depuración
- Manejo de fallas
- El manejo de interrupciones se hace por medio del NVIC

Los procesadores Cortex-M tienen 5 miembros diferentes dentro de su familia; apuntan cada uno a una necesidad diferente ya sea de alto rendimiento o de ultrabajo consumo de potencia. Los procesadores Cortex-M son:

- Cortex-M0: un procesador muy pequeño (de 12 mil compuertas), de bajo costo, ultrabajo consumo de potencia y sistemas embebidos reducidos. Utiliza una versión ARMv6-M con arquitectura de memoria Von Neumann.

- Cortex-M0+: el procesador más eficiente en consumo de potencia de la familia. Diseñado para sistemas embebidos pequeños, cuenta con un tamaño similar al Cortex-M0, pero con algunas características extras. Utiliza una versión ARMv6-M con arquitectura de memoria Von Neumann.

- Cortex-M1: procesador pequeño, con un diseño optimizado para su implementación en diseños de FPGA. Instrucciones y arquitectura iguales a las del Cortex-M0.

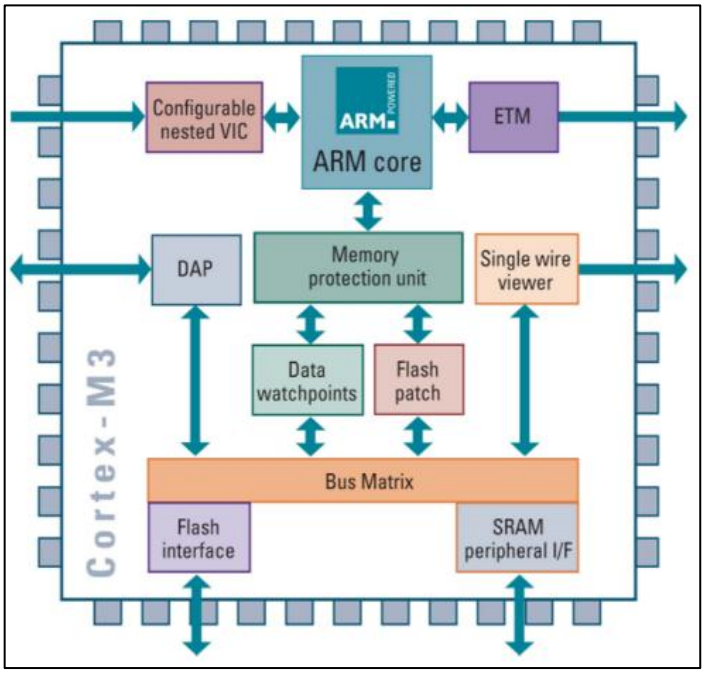
- Cortex-M3: procesador pequeño, pero de alto rendimiento para microcontroladores de bajo consumo de potencia con un set de instrucciones alto (Thumb-2) que le permite realizar tareas complicadas de manera rápida. Cuenta con un divisor de hardware, una unidad MAC (siglas en inglés de *multiply-accumulate*), una unidad para depuración y diferentes herramientas para desarrollo. Utiliza una versión ARMv7-M con arquitectura de memoria Harvard.

- Cortex-M4: contiene todas las características de un Cortex-M3, con instrucciones adicionales para el procesamiento digital de señales, esto debido a que incluye una unidad DSP incorporada. También, cuenta con instrucciones SIMD (siglas en inglés para *single instruction multiple data*) y operaciones de un ciclo MAC. Algunas versiones incluyen un coprocesador de punto flotante con soporte para el estándar IEEE-754. Estos últimos reciben el nombre de Cortex-M4F.

- Cortex-M7: procesadores de alto rendimiento para microcontroladores de alto nivel con aplicaciones de procesos intensivos. Contiene todas las características de un Cortex-M4 con soporte adicional para unidad de

punto flotante de doble precisión y algunas características de memoria como caché y TCM (siglas en inglés para *tightly coupled memory*).

Figura 16. **Cortex M3**



Fuente: ARM. *The ARM Architecture with a focus on v7A and Cortex-A8.* https://www.coursehero.com/file/13338101/ARM-Arch-A8/. Consulta: 25 de junio de 2017.

4.1.2. Temporizador del sistema (SysTick)

Los procesadores ARM Cortex-M4F incluyen un temporizador llamado SysTick. Este se incluye en todos los diseños de la familia Cortex-M. Su importancia radica en que los periféricos incluidos dentro de un microcontrolador dependen del fabricante; sin embargo, el temporizador del sistema siempre estará presente en todos los diseños ARM, por lo que su uso

no depende del mismo. Es un temporizador simple de 24-bits decreciente que se puede usar de diferentes formas.

4.1.3. *Nested vectored interrupt controller* (NVIC)

El microcontrolador TM4C123GH6PM incluye un dispositivo llamado *nested vectored interrupt controller* (NVIC). Este se encarga de manejar todas las interrupciones. Más información del mismo se tratará en el capítulo 6.4.

4.1.4. *System control block* (SCB)

El SCB, es un bloque encargado del control del sistema, configuración, control y reporte de las excepciones o fallas del mismo.

4.1.5. *Memory protection unit* (MPU)

Sistema de protección de memoria para ARM7. Utiliza el modelo PMSA (siglas en inglés para *protected memory system architecture*). Es el encargado de proteger regiones de la memoria, evitar traslape de regiones, los permisos de acceso a memoria y la exportación de atributos al sistema.

4.1.6. *Floating point unit* (FPU)

Coprocesador del sistema encargado de dar soporte a operaciones de precisión simple, como suma, resta, multiplicación, división, multiplicación, acumulación y raíz cuadrada. También, permite la conversión entre datos e instrucciones de enteros y de punto flotante. Cuenta con las siguientes características:

- Instrucciones de precisión simple de 32 bits para operación de datos.

- Combinación entre instrucciones de multiplicación y acumulación para mejora de precisión.

- Soporte en hardware para conversión, suma, resta multiplicación con opción para acumulación, división y raíz cuadrada.

- Registros de 32 bits dedicados.

4.1.7. Memoria integrada en chip

El microcontrolador trae integrado dentro del chip las siguientes memorias:

- 32 KB de memoria SRAM de ciclo único
- 256 KB de memoria flash
- 2KB de memoria EEPROM
- ROM interna con Tivaware cargado para desarrollo en lenguaje C

4.1.8. Periféricos

Cuenta con diferentes periféricos encargados de las interfaces de comunicación, integración del sistema, control avanzado de movimiento y soporte analógico. Estos se tratarán detalladamente en el capítulo 6.

4.1.9. Módulo de hibernación

El módulo de hibernación provee la lógica para apagar el procesador principal y sus periféricos y volver a activarlo en eventos basados en tiempo. Entre sus características se destacan:

- Control de la potencia de sistema con regulador discreto externo
- Control de la potencia del sistema por medio de registros de control
- Pin dedicado para despertar el sistema por medio de una señal externa
- Detección de batería baja con opción a despertar cuando esto suceda
- Retención de estados en los pines durante hibernación

4.1.10. *Watchdog timer*

Watchdog timer es un control incluido en el sistema, encargado de reaccionar de una manera definida en caso de fallo del sistema o un error en el programa. El TM4C123GH6PM puede generar una interrupción enmascarable, no enmascarable o un reinicio cuando se alcanza un valor de tiempo muerto. El TM4C123GH6PM cuenta con dos módulos de *watchdog timer:*

- *Watchdog timer* 0: usa el reloj del sistema como temporizador
- *Watchdog timer* 1: usa el oscilador principal como temporizador

4.1.11. JTAG y ARM *serial wire debug*

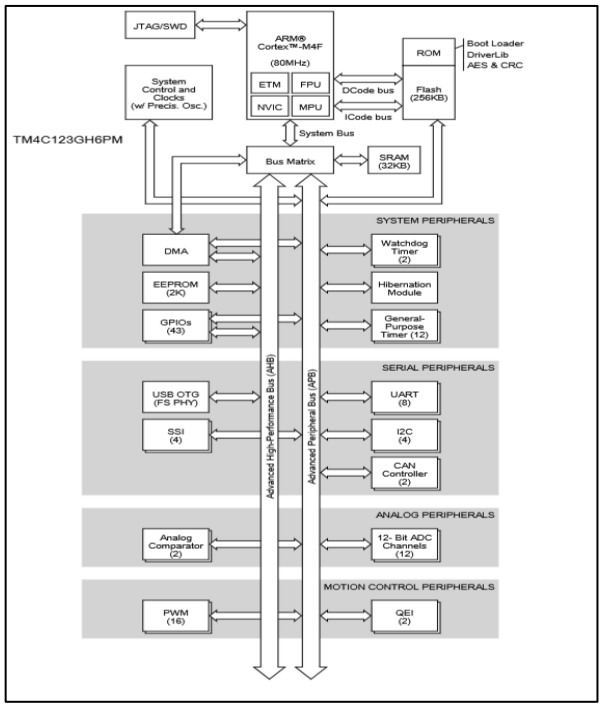
JTAG (siglas en inglés de *joint test action group*) es un estándar IEEE que define un puerto de acceso de pruebas y un escaneo en la arquitectura por medio de una interfaz, lo que proporciona información de los circuitos y

componentes para realización de ensayos, observación, control y depuración del código.

Texas Instruments reemplaza el ARM SW-DP y el JTAG-DP con el ARM *serial wire debug port* (SWJ-DP). Este puerto combina el SWD y JTAG para depuración en un solo módulo; provee todas las funcionalidades de depuración del JTAG en tiempo real, así como acceso a la memoria del sistema sin necesidad de detener el programa o el núcleo donde se encuentra el código.

4.2. Diagrama de bloques del TM4C123GH6PM

Figura 17. **TM4C123GH6PM**



Fuente: Texas Instruments. *TM4C123GH6PM Datasheet.* http://www.alldatasheet.com/view.jsp?Searchword=Tm4c123gh6pm%20datasheet. Consulta: 27 de junio 2017.

5. PROGRAMA INFORMÁTICO

Se ha definido que un sistema embebido cuenta con un programa informático que corre en la unidad de procesamiento o describe el hardware necesario para su funcionamiento. Conocer a detalle cómo este funciona es esencial en el diseño de sistemas embebidos pues, al momento de diseñar uno, se presentan diferentes opciones: lenguajes de programación, entornos de desarrollo, librerías, entre otros.

5.1. Definición

Tabla X. **Programa Informático**

Un programa informático es una secuencia de instrucciones que se almacenan en algún lugar de la memoria con el objetivo de realizar una tarea específica.

Fuente: elaboración propia.

Si se pudiera observar un programa dentro de un sistema embebido, solamente se podría notar una larga sucesión de bits (código máquina) pues, a nivel de hardware, es lo único que comprende la unidad de procesamiento. Al escribir un programa, no se hace utilizando sucesiones de bits, puesto que esto implicaría demasiado tiempo y es muy propenso a errores. Es por esto que los primeros operadores de computadoras decidieron reemplazar estas cadenas de bits por palabras y letras de fácil entendimiento para el ser humano, lo cual dio origen al lenguaje ensamblador.

Al leer un archivo escrito en lenguaje ensamblador se sustituye cada una de las palabras y parámetros clave por código de operación correspondiente en sistema binario. Es por esto que recibe el nombre de lenguaje de bajo nivel o de nivel próximo a máquina.

A finales de los años 50, surge un nuevo tipo de lenguajes de programación que tenían como objetivo que el usuario común pudiese escribir código de una manera más fácil y rápida. Estos lenguajes tienen una estructura semántica similar al lenguaje utilizado por los seres humanos por lo que reciben el nombre de lenguajes de alto nivel o de tercera generación. Entre sus ventajas se encuentran las siguientes:

- Código sencillo y comprensible
- Reducción del tamaño del código
- Permite el uso de paradigmas de programación
- Independientes de un dispositivo en particular
- Fácil mantenimiento

Estos lenguajes traen consigo también algunas desventajas, pues generan códigos ineficientes, especialmente cuando se desconocen los detalles de la arquitectura del procesador del sistema. Un código será ineficiente siempre que no se tenga conocimiento de lo que sucede detrás de cada instrucción de alto nivel al implementarse en lenguaje máquina. Al momento de trabajar sistemas embebidos, es de suma importancia que el desarrollador domine un lenguaje de alto nivel y lenguaje ensamblador.

5.1.1. Lenguaje C

El lenguaje C, desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell, es un lenguaje de alto nivel que posee muchas características importantes: programación estructurada, métodos de llamadas a funciones, traspaso de parámetros, estructuras de control, entre otras.

Este lenguaje es de gran utilidad debido a su gran flexibilidad de programación; permite la habilidad de combinar comandos simples de bajo nivel en complicadas funciones de alto nivel. Provee acceso a la memoria de bajo nivel y construcciones del lenguaje que representan al código de la forma más cercana a código de máquina, a tal punto que algunos llegan a considerarlo actualmente un lenguaje de bajo nivel, motivo por el cual es el lenguaje más popular para escribir programas en microcontroladores.

La conversión de un lenguaje de alto nivel a uno de bajo y posteriormente a lenguaje de máquina, respectivamente, se hace por medio de un entorno de desarrollo integrado.

5.2. Entorno de desarrollo integrado (IDE)

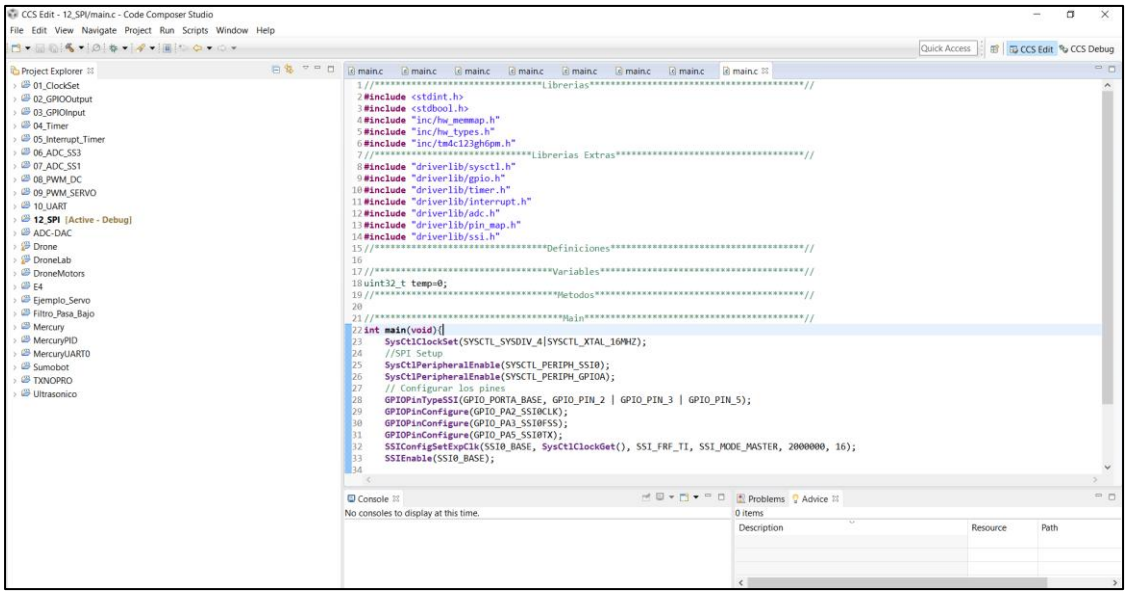
Un IDE es un entorno de programación que ha sido empaquetado como una aplicación. Está diseñado para maximizar la productividad del programador ofreciendo las herramientas para la creación, modificación, compilación, interpretación y depuración de código. Su objetivo es reducir el tiempo de desarrollo, pues aprender el uso de un IDE es más rápido que aprender e integrar manualmente todas las herramientas por separado.

Para el controlador TM4C123GH6PM se cuenta con diferentes entornos de desarrollo integrado:

- *Mentor embedded*
- *IAR systems*
- *ARM keil*
- *Code composer studio*
- *Energy*

Este se encuentra compuesto por un compilador, un enlazador, un cargador de programa y en algunos casos por un módulo de depuración.

Figura 18. ***Code composer studio***



Fuente: elaboración propia

5.2.1. Compilador

Esta herramienta es la encargada de convertir el lenguaje de alto nivel a lenguaje de máquina. Sus tareas se dividen en tres fases diferentes:

- *The-front-end:* encargada del análisis y generación del código para ser procesado por la fase *middle-end.*

- Análisis de código: se realiza en diferentes pasos:

    - Análisis léxico: encargado de leer el programa de izquierda a derecha y agruparlo en componentes léxicos (secuencias de caracteres con significado). Elimina espacios en blanco, líneas en blanco, comentarios y toda información innecesaria del programa. Verifica que los símbolos del lenguaje como palabras clave u operadores se hayan escrito correctamente.

    - Análisis sintáctico: en esta fase los componentes léxicos y caracteres se agrupan en frases gramaticales que el compilador utiliza para sintetizar la salida.

    - Análisis semántico: verifica tipos y operadores, encontrando errores semánticos en el programa. Reúne toda la información para la generación del código posterior.

    - Fase de síntesis: genera un programa para una máquina abstracta, debe ser fácil de producir y traducir al programa objeto.

- *The-middle-end:* realiza la optimización del código recién generado. Esta fase consiste en mejorar este código de modo que se obtenga un código de máquina que se ejecute más rápido. Esto lo logra removiendo código inservible, descubriendo y propagando valores constantes, reubicación de procesos a lugares con menor ejecución. Genera otro código intermedio para ser procesado por la fase *back-end*.

- *The-back-end:* genera el código en lenguaje ensamblador, realiza la ubicación de los procesos en los registros y optimiza el código para el uso del hardware asociado.

5.2.2. Enlazador

Esta herramienta, también conocida como *linker*, es la encargada de tomar uno o más archivos de objetos generados por el compilador y combinarlos en un solo archivo ejecutable. Esto se hace debido a que un programa generalmente tendrá más de un solo archivo de objetos, como librerías, vectores de interrupciones u otros archivos externos declarados dentro del mismo código.

5.2.3. Cargador de programa

Esta herramienta, también conocida como *loader*, es la encargada de cargar a memoria el archivo ejecutable generado por el enlazador. También, es la encargada de inicializar los registros.

Una vez el código ha pasado por todos estos pasos finalmente se tendrá un programa informático ejecutándose dentro del procesador.

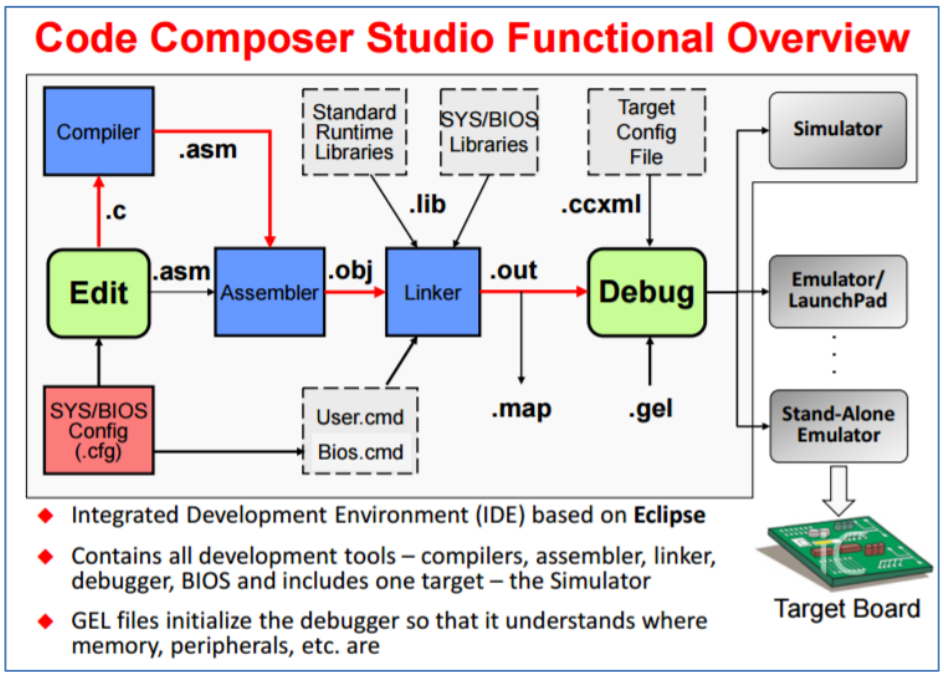
5.3. Depurador

Este proceso, conocido también como *debugging,* es utilizado para probar y depurar (eliminar) errores en los programas. Funciona deteniendo la ejecución del programa a depurar cuando se cumplen ciertas condiciones que permiten examinar la situación al desarrollador.

El depurador permite detener el programa en un punto determinado por una condición de ruptura. Durante esa interrupción, el usuario puede:

- Examinar y modificar la memoria y las variables del programa.
- Examinar el contenido de los registros del procesador.
- Examinar la pila de llamadas que han desembocado en la situación actual.
- Cambiar el punto de ejecución, de manera que el programa continúe su ejecución en un punto diferente a en el que fue detenido.
- Ejecutar de modo instrucción a instrucción.
- Ejecutar partes determinadas del código, como el interior de una función, o el resto de código antes de salir de una función.

Figura 19. **Diagrama del funcionamiento de Code Composer Studio**



Fuente: *TM4C123G LaunchPad Workshop.* https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf. Consulta: 27 de junio de 2017.

6. PERIFÉRICOS

Se entiende por periféricos a todos aquellos circuitos especializados auxiliares e independientes a la unidad central de procesamiento. El controlador TM4C123GH6PM cuenta con una gran cantidad de periféricos, como se especificó en sus características. En este apartado se tratarán solamente las esenciales para el diseño de sistemas embebidos.

6.1. Señal de reloj (*clock*)

Tabla XI. **Definición de señal de reloj**

La señal de reloj es una señal digital cuadrada encargada de la sincronización de los componentes de un sistema digital.

Fuente: elaboración propia.

Esta señal oscila entre estados altos y bajos con un ciclo de trabajo del 50 %, esto implica que los primeros duran lo mismo que los otros. Se mide utilizando las unidades Hertz, que representan cantidad de ciclos por segundo.

La señal de reloj es de gran importancia en cualquier sistema digital, dado que dicta la ejecución de las instrucciones dentro de la unidad de procesamiento y la sincronización de los componentes internos. La unidad de procesamiento necesita un número establecido de ciclos de reloj para ejecutar una instrucción, de modo que, a una mayor velocidad de reloj, mayor cantidad de instrucciones se ejecutarán en menor tiempo.

A primera vista se podría considerar que una mayor velocidad de reloj se traduce en una mayor cantidad de tareas realizadas, pero no es necesariamente el caso. No se puede comparar el rendimiento de un sistema digital dependiendo de su velocidad de reloj únicamente. Un CPU podría realizar una multiplicación en 20 ciclos de reloj, mientras que otro podría ejecutar la misma operación en un solo ciclo de reloj. El cálculo del rendimiento debe considerar también la arquitectura del sistema.

La sincronización de los componentes internos por medio del reloj funciona como un metrónomo, cada flanco de subida o bajada (según se especifique), cada componente realizará una acción, de modo que los estados cambiarán todos al mismo tiempo y mantendrá el funcionamiento del sistema de manera ordenada, evitando congestionamiento dentro del mismo y corrigiendo la diferencia de tiempos de ejecución entre componentes.

Al iniciar el diseño de un sistema embebido, debe configurarse inicialmente la velocidad de reloj y su fuente para el sistema. Es importante también recordar que cada periférico necesita estar sincronizado con la unidad de procesamiento por lo que es imprescindible la configuración y habilitación de una señal de reloj para cada periférico a utilizar.

6.1.1. Fuentes de reloj

Los sistemas digitales cuentan con dos formas de generación de una señal de reloj:

- Circuito oscilador: por medio de circuitos tanques RC o LC es posible generar una señal cuadrada. La desventaja de estos circuitos es que pueden ser inestables por lo que su proceso de fabricación conlleva un

mayor esfuerzo al tener que seleccionar los componentes adecuados, lo que no siempre resuelve el problema. Esta desventaja genera errores cuando existen variaciones en la temperatura, variaciones en la carga o cambios en el voltaje de alimentación que afecta el rendimiento en conjunto del sistema. La mayoría de los sistemas digitales cuentan con uno de estos circuitos integrados para generar su señal de reloj.

- Cristales de cuarzo: pequeños cristales que funcionan por medio del efecto piezoeléctrico. Este efecto es una propiedad del cristal en que se convierte la tensión mecánica en electricidad y la electricidad en vibraciones mecánicas. Si el cristal no se encuentra bajo ninguna tensión mecánica externa, las cargas se desplazan uniformemente en las moléculas del cristal. Cuando el cristal se estira o contrae, el orden de los átomos varía ligeramente. Este cambio causa que las cargas negativas se acumulen en un sector y las positivas en el lado opuesto, generando un gradiente de cargas que se traduce en una diferencia de potencial eléctrico. De forma complementaria, aplicar una señal de corriente eléctrica al cristal, cambia su forma.

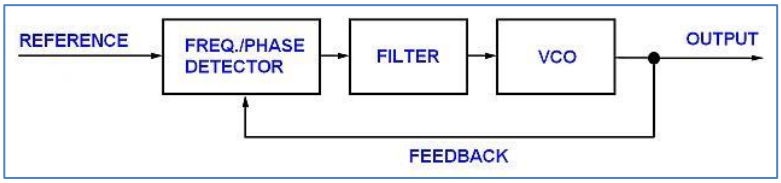
Acoplando estos cristales con un circuito adecuado, se puede generar una señal de reloj muy precisa. Aplicando voltaje al cristal, se genera tensión mecánica, que a su vez originará una señal de voltaje a una frecuencia proporcional.

En sistemas digitales los cristales son la opción por defecto a utilizar, debido a que se mantienen estables ante las variaciones de temperatura, carga o cambios en la fuente de alimentación. Su única desventaja es que ocupan mayor espacio físico, por lo que raramente se presentan integrados al sistema.

6.1.2. Bucle de enganche de fase (PLL)

Tras haber seleccionado la fuente, la mayoría de sistemas digitales cuentan con un circuito especializado llamado PLL. Este cumple la función de un sintetizador de frecuencias, es decir, genera a la salida una frecuencia que es múltiplo de la frecuencia de entrada.

Figura 20. **Diagrama de bloques de un PLL**



Fuente: PLL Seminar. http://www.changpuak.ch/electronics/PLL_Seminar_1.php. Consulta: 27 de junio de 2017.

El circuito cuenta con un primer bloque encargado de detectar la fase de dos señales: la de entrada y la de salida. Realiza una comparación de fase entre estas y genera una salida equivalente a la diferencia de fase entre ambas. Esta diferencia de fases pasa por un filtro pasa bajos, que generan un voltaje directo donde, a mayor sea la diferencia de fase, mayor será el voltaje y viceversa. Una señal de voltaje directo alimenta al VCO, que genera una frecuencia proporcional a dicha señal. Esta salida de frecuencia retroalimenta al detector de fase, formando un circuito de lazo cerrado. El proceso se repite hasta obtener una señal en la salida exactamente igual a la de la entrada.

Si se agrega un divisor en el bloque de retroalimentación, la comparación de fase obtendrá siempre un dividendo de la señal de salida, por lo que la salida del circuito deberá generar un múltiplo para generar una señal igual. De esta

manera, agregando divisores a la retroalimentación, es posible generar múltiplos de la señal de entrada.

Tabla XII. **Definición de divisor de frecuencia**

Circuito formado por contadores digitales encargados de dividir la frecuencia de entrada en una relación entera o racional.

Fuente: elaboración propia.

6.1.3. Señal de reloj en el TM4C123GH6PM

Para configurar la señal de reloj del TM4C123GH6PM, se lleva a cabo el siguiente procedimiento:

- Selección de fuente de reloj

Las fuentes de reloj que se pueden utilizar son:

  - Oscilador interno de precisión (PIOSC)
    - Frecuencia de 16 MHz ± 3 %
  - Oscilador principal (MIOSC)
    - Circuito oscilador externo
    - Cristal externo

- Oscilador interno
    - Frecuencia de 30 KHz ± 50 %, para uso en modo de ahorro de batería.
- Reloj para modo de hibernación
    - Uso con un cristal de 32,768 Hz, para aplicaciones en tiempo real.

- Uso del circuito PLL de fuente

Debe seleccionarse a continuación el circuito PLL del sistema para la generación de la señal de reloj. Este permite convertir la señal de entrada de la fuente de reloj en una señal con frecuencia de 400 MHz. La salida del circuito PLL, en este caso, cuenta con un divisor de 2; por lo que los cálculos pertinentes deben hacerse tomando como base una frecuencia de 200 MHz para la señal de reloj.

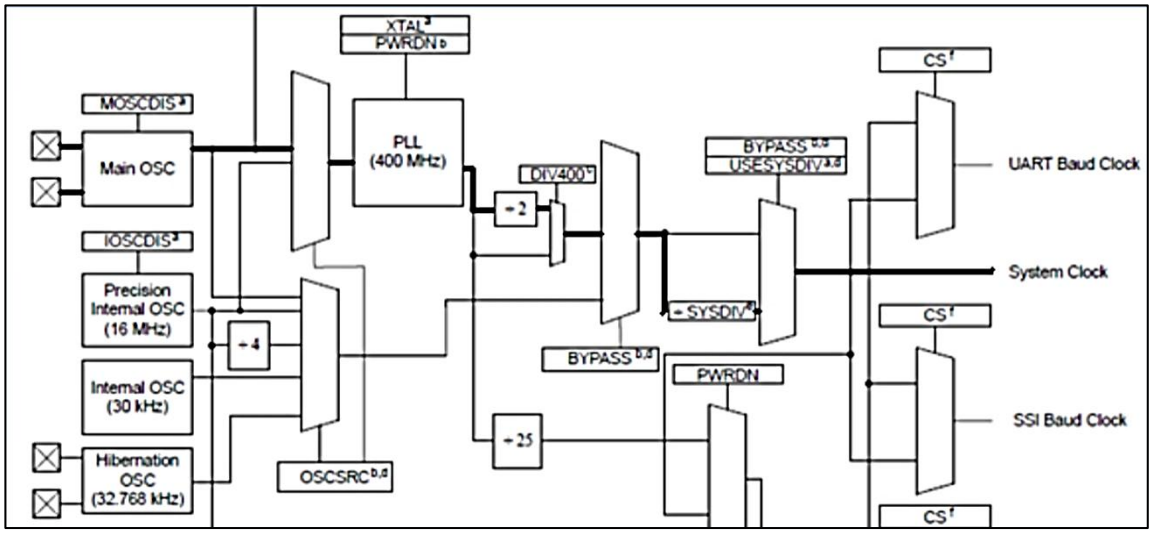
- Divisor definido por el usuario

Verificando las especificaciones del controlador en el capítulo 4, se observa que la velocidad máxima a la que es posible utilizarlo es 80 MHz. Por esto que se debe utilizar un divisor definido por el usuario siempre que se utiliza el circuito PLL.

Las opciones disponibles para la definición de divisores son las definidas en la tabla I del apéndice 1.

La arquitectura interna para el periférico se encuentra distribuida de la siguiente forma:

Figura 21. **Diagrama interno para generación de CLOCK**



Fuente: *Getting Started with the Tiva™ TM4C123G LaunchPad Workshop.* https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf. Consulta: 27 de junio 2017.

Por los puntos previamente expuestos es recomendable utilizar un cristal externo en conjunto con el PLL. Si se utiliza la tarjeta de desarrollo Tiva C, esta incluye un cristal de 16 MHz para utilizar como fuente de señal de reloj.

Es posible resumir todas estas opciones, con la siguiente fórmula:

$$Frequency = \frac{200\ [MHz]}{SysDiv}$$

6.1.4. Código de configuración del CLOCK

Se presenta el código en lenguaje C para configurar la señal de reloj en el controlador TM4C123GH6PM.

- SysCtlClockSet(SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ | SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL); //Configuración a 40MHz, utilizando PLL, cristal de 16MHz y divisor de 5

6.2. Entradas y salidas de propósito general (GPIO)

Las entradas y salidas de propósito general GPIO (siglas en inglés para *general purpose inputs and outputs*) son pines sin algún propósito específico definido cuyo comportamiento puede ser controlado por el usuario. Un puerto GPIO es un grupo de pines GPIO ordenados y controlados como grupo, generalmente de hasta 8 pines.

Si se configuran como entradas, su propósito será introducir datos externos al sistema digital para su procesamiento. Si son configurados como salidas, su propósito será recibir la información procesada por el sistema digital y reproducirla.

6.2.1. Familia lógica

Las familias lógicas indican la tecnología utilizada dentro de los circuitos integrados al sistema, expresando sus niveles lógicos y características. Las características principales de la familia lógica incluyen:

- Velocidad: determinada por el tiempo entre la aplicación de una señal de entrada y el cambio en su salida.

- *Fan-in*: determina la cantidad de entradas que una compuerta puede manejar.

- *Fan-out*: Determina el número de circuitos que una puerta puede controlar.

- Inmunidad al ruido: Cantidad máxima de ruido que el circuito puede soportar sin que afecte su salida.

- Disipación de potencia: cantidad de potencia disipada entre cambios de estados.

Existen tres diferentes familias lógicas:

- TTL (*transistor-transistor logic*): sus compuertas y circuitos están compuestas por transistores BJT (siglas en inglés para *bipolar junction transistor*).

- CMOS (*complementary metal oxide semiconductor*): sus compuertas y circuitos están compuestas por transistores CMOS.

- ECL (*emitter coupled logic*): sus compuertas y circuitos están compuestos por transistores BJT diferenciales. Funciona con niveles de voltaje negativos.

Tabla XIII. **Comparación entre algunas características de las familias lógicas**

Specification	TTL	ECL	CMOS
Basic Gate	NAND	OR/NOR	NAND/NOR
Fan-out	10	25	>50
Power per gate (mW)	1-22	4-55	1 at 1MHz
Noise Inmunity	Very Good	Good	Excellent

Fuente: *RF Wireless World.* http://www.rfwireless-world.com/Terminology/Difference-between-TTL-ECL-CMOS.html. Consulta: 28 de junio de 2017.

De estas tres familias, ECL es la más rápida; sin embargo, su uso de voltajes negativos la hace una familia raramente utilizada. El mercado se encuentra dominado por las tecnologías TTL y CMOS, siendo la segunda la que tiene una mayor presencia por sus ventajas en comparación con la primera.

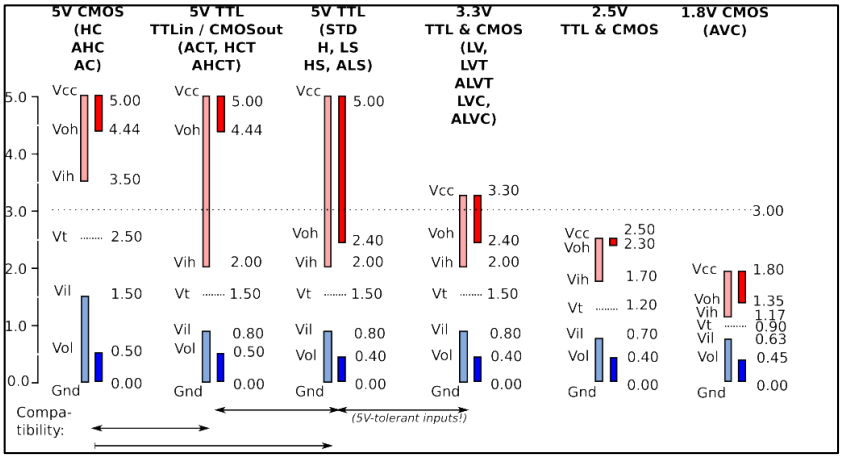
Los componentes TTL son más baratos a los componentes CMOS; sin embargo, la tecnología CMOS tiende a ser más economía en una escala mayor puesto que los circuitos son más pequeños y requieren menor regulación. Los componentes CMOS no consumen potencia durante estados estáticos, pero su consumo de potencia incrementa en proporción a la frecuencia de la señal de reloj. TTL cuenta con un consumo de potencia constante. CMOS también cuenta con un requerimiento bajo de corriente, por lo que los circuitos son más baratos y fáciles de diseñar para bajo consumo de potencia, haciéndolos ideales para microcontroladores.

6.2.1.1. Niveles lógicos de voltaje

Un nivel lógico es uno de muchos estados en que una señal digital puede encontrarse. En general, se definen dos estados diferentes en un circuito digital: lógico alto y lógico bajo.

Los niveles lógicos de voltaje expresan los valores de voltaje DC que representan estados altos y bajos, pues estos varían de familia a familia.

Figura 22. **Niveles lógicos de voltaje**



Fuente: *Logic Voltage Levels.* http://www.jsykora.info/2014/05/logic-voltage-levels/. Consulta: 28 de junio de 2017.

El controlador TM4C123GH6PM utiliza el estándar LVLCMOS3.3, lo cual implica que sus circuitos utilizan tecnología CMOS y un nivel de voltaje de 3,3 V, donde un estado lógico alto de entrada varía entre 2,0 V y 3,3 V, un estado lógico bajo de entrada varía entre 0 V y 1,3 V.

6.2.2. GPIOS en controlador TM4C123GH6PM

El módulo GPIO provee control sobre 8 pines diferentes e independientes de cada puerto. Se cuenta con 6 puertos diferentes nombrados alfabéticamente desde A hasta F. Cada pin puede ser configurado con cada uno de los siguientes parámetros:

- Configuración como entrada o salida: la configuración por defecto es de entrada.

- En modo entrada, pueden generar interrupciones en niveles altos, bajos, flancos de subida, flancos de bajada o ambos.

- En modo de salida pueden ser configurados para un *drive strength* de 2, 4 u 8 mA.

- Opción para resistencias *pull-up* y *pull-down*, por defecto están inactivas.

- Opción para operación *open-drain*.

6.2.3. Configuración de los GPIO en el TM4C123GH6PM

Los GPIOs como entradas funcionan para la lectura de valores y los GPIOS como salidas funcionan para escribir valores. Su configuración se realiza en el siguiente orden:

- Habilitar reloj para el periférico
- Debe habilitarse el puerto a utilizar
- Escoger los pines que funcionarán como entradas o salidas
- Si es entrada, configurar resistencias *pull-up* o *pull-down*
- Si es salida, configurar *drive strength*

6.2.4. Escritura y lectura de los GPIO

Un puerto cuenta con 8 pines. Para cada puerto se tiene asignado un registro de 8 bits, de modo que, si se desea escribir un estado sobre el pin 0 de

un puerto en específico, se debe cambiar el valor del bit 0. Sobre este bit se escribirá un 1 para un estado en alto o un 0 para un estado bajo. Si se desea escribir un estado sobre el pin 1, se debe cambiar el valor del bit 1 y de la misma forma hacerse con cada pin. Es posible cambiar diferentes estados al mismo tiempo cambiando todos los bits de interés a la vez. La escritura se hace en el sistema binario, también es posible realizarlo en sistema decimal o hexadecimal, según se facilite.

La lectura de datos es un proceso similar, la diferencia radica en que se interpretarán los datos solamente. Leer un estado lógico alto sobre el bit 0 implicaría que el pin 0 se encuentra en alto.

Al momento de hacer una lectura o una escritura no se trabajará sobre un bit específico, sino sobre todo el registro.

6.2.5. Máscaras

En programación, cuando se refiere a máscaras, se está definiendo un proceso de manipulación de registros. El uso más frecuente es la máscara de bits, este permite la extracción del valor de bits particulares o su escritura según sea la operación que se aplique en conjunto a la máscara.

- Operación lógica OR: la operación lógica OR permite escribir un estado lógico en alto, donde los bits de la máscara presenten estados lógicos altos.

    - Supóngase un registro de valor: 0b 0000 0000.
    - A este registro se le aplica una máscara y una operación lógica OR, generando un resultado.

- Si se desea escribir un lógico alto en el último bit solamente, se aplica una máscara: 0b 0000 0001. Esto genera como resultado: 0b 0000 0001, donde su ultimo bit es 1.

- Si se desea escribir un lógico alto en el primer bit solamente, se aplica una máscara: 0b 1000 0000. Esto genera como resultado: 0b 1000 0000, donde su ultimo bit es 1.

- Operación lógica AND: esta operación permite la lectura de los bits que la máscara presente en estado lógico alto.

  - Suponer un registro de valor: 0b 1101 0110.
  - A este registro se le aplica una máscara y una operación lógica AND, generando un resultado.
  - El resultado varía dependiendo de la máscara de la siguiente forma:

    - Si se desea leer el valor que tiene su ultimo bit solamente, se aplica una máscara: 0b 0000 0001. Esto genera como resultado: 0b 0000 0000, donde su ultimo bit es 0.

    - Si se desea leer el valor de su penúltimo bit solamente, se aplica una máscara 0b 0000 0010, generando como resultado 0b 0000 0010, retornando el valor del penúltimo bit.

6.2.6. Código de configuración de GPIOS

Se presenta el código en lenguaje C para configurar entradas y salidas en el controlador TM4C123GH6PM.

- SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //Habilitar periférico.
- GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, //Definir como salida los pines. GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
- GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4); //Definir como entrada los pines.
- GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD); //Configurar resistencias *pull-up*.

6.3. Temporizador de propósito general (GPTM)

El temporizador de propósito general GPTM (siglas en inglés para *general purpose timer module*) es el módulo encargado de llevar conteos o realizar eventos controlados por tiempo. Estos funcionan por medio del aumento o decremento en un registro de tamaño específico, de modo que, si es un contador ascendente, contará desde el valor indicado hasta que exista un desbordamiento. Si es un contador descendente este contará desde el valor hasta cero.

Cuando un temporizador llega a su valor final, es posible configurarlo para realizar un aviso, haciéndolo un módulo muy confiable para llevar a cabo cualquier tarea dependiente del tiempo de manera precisa y exacta.

6.3.1. Modos de funcionamiento del temporizador

- Disparo único: el temporizador solamente realizará el conteo una vez, luego de esto dejará de funcionar.

- Periódico: el temporizador realizará su recorrido desde el valor especificado hasta el final, una vez este haya terminado se reiniciará el contador y comenzará el recorrido de nuevo.

- RTC (*real time clock*): en conjunto con un cristal de 32 768 Hz, este temporizador se utiliza para aplicaciones con un reloj de tiempo real. Esto es posible dado que 32768 es un valor de potencia 2, (2^{15}). Por lo que se puede obtener un valor preciso de un segundo usando un contador de 15 bits.

- Contador de flancos: este modo funciona aumentando el valor del contador cada vez que ocurre uno de tres eventos: flancos de subida, flancos de bajada o ambos.

6.3.2. Temporizador en el TM4C123GH6PM

El módulo de temporizador provee dos contadores de media amplitud que pueden ser configurados para operar independientemente o combinados para trabajar como contador de amplitud completa. Los dos contadores de media amplitud se les refieren como TimerA y TimerB. Dependiendo del módulo que se utilice, se puede configurar temporizadores de 16 bits de media amplitud y un temporizador de 32 bits de amplitud completa. También, se puede trabajar con dos temporizadores de media amplitud de 32 bits y uno de amplitud completa de 64 bits.

6.3.3. Configuración del temporizador en el TM4C123GH6PM

Cada uno de los módulos Timer0 y Timer1 cuenta con la opción para TimerA y TimerB. Su configuración se realiza de la siguiente manera:

- Habilitar el periférico del temporizador a utilizar (Timer0 o Timer1)
- Escoger el modo de funcionamiento del temporizador
- Cargar el valor al temporizador y seleccionar su amplitud
- Habilitar el temporizador

6.3.4. Código de configuración del temporizador

Se presenta el código en lenguaje C para configurar el temporizador en el controlador TM4C123GH6PM.

- SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); //Habilitar periférico
- TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); //Definir modo de funcionamiento
- TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());//Seleccionar amplitud y valor del temporizador.
- TimerEnable(TIMER0_BASE, TIMER_A); //Habilitar el temporizador.

6.4. Controlador de interrupciones (NVIC)

El controlador de interrupciones anidadas NVIC (siglas en inglés para *nested vectored interrupt controller*) es el periférico encargado de manejar las interrupciones del sistema.

Tabla XIV. **Definición de interrupción**

Una interrupción, IR (siglas en inglés para *interrupt request*) es una señal recibida por la unidad de procesamiento para indicarle que debe interrumpir el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Fuente: elaboración propia.

Las interrupciones, como su nombre indica, se encargan de detener el curso actual del sistema, dando paso a la ejecución de un código para reaccionar a la situación actual del sistema. Este código se encuentra definido en la memoria, solamente esperando a ser ejecutado cuando la unidad de procesamiento lo requiera. La unidad de procesamiento sabrá que debe ejecutarlo cuando reciba una petición de la misma por parte de los periféricos. Todos los periféricos vienen equipados con la capacidad de generar señales de interrupción cambiando el estado de un registro específico. Este registro se comunica utilizando el bus de control.

6.4.1. Manejo de una IR

Los pasos que realiza el procesador para manejar una interrupción son los siguientes:

- Terminar la ejecución de la instrucción máquina en curso.

- Guardar el estado del procesador (valores de registros y banderas) y el valor del contador de programa en la pila, de manera que, en la CPU, al terminar el proceso de interrupción, pueda seguir ejecutando el programa a partir de la última instrucción.

- El procesador salta a la dirección donde está almacenada la rutina de servicio de interrupción (*interrupt service routine*, o abreviado ISR) y ejecuta esa rutina que tiene como objetivo atender al dispositivo que generó la interrupción.

- Una vez que la rutina de la interrupción termina, el procesador restaura el estado que había guardado en la pila en el paso b y retorna al programa que se estaba usando anteriormente.

Se puede dividir los pasos para manejar una interrupción:

- PUSH: toma los pasos a, b y c hasta previo a ejecutar la rutina
- ISR: ejecución de la rutina de interrupción
- POP: paso d

6.4.2. Características del NVIC en el TM4C123GH6PM

El NVIC es parte de la arquitectura Cortex-M, por lo cual no es exclusivo solamente de este controlador. Todas las arquitecturas ARM Cortex-M lo incluyen. Sin embargo, varía de controlador a controlador pues, aunque su funcionamiento es similar, la cantidad de periféricos cambia. De sus características se destacan:

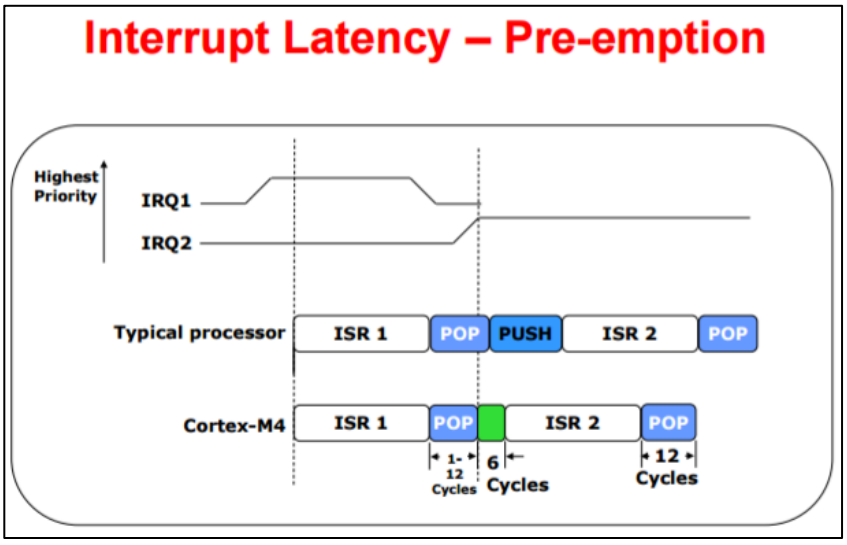
- Manejo interrupciones y excepciones
- 8 niveles diferentes de prioridad entre interrupciones
- 7 excepciones y 71 interrupciones
- Grabado y restauración de estados automático
- Lectura automática de la tabla de entrada del vector
- Anticipación de interrupciones

- Manejo de llegada tarde en interrupciones
- Encadenamiento de interrupciones
- Determinístico: Siempre toma 12 ciclos o 6 con encadenamiento

6.4.2.1. Anticipación de interrupciones

Si existen dos interrupciones, IR1 e IR2, siendo IR1 la de mayor prioridad. Cuando se activa la interrupción IR1, se genera una señal PUSH para IR1, seguido de su ISR y finalizando con un POP; si durante el POP se activa IR2, entonces el NVIC no generará un nuevo PUSH, sino que irá directamente al ISR y POP de la segunda interrupción. De esta forma, ahorra tiempo evitando un PUSH en comparación con otros procesadores.

Figura 23. **Anticipación de interrupciones**

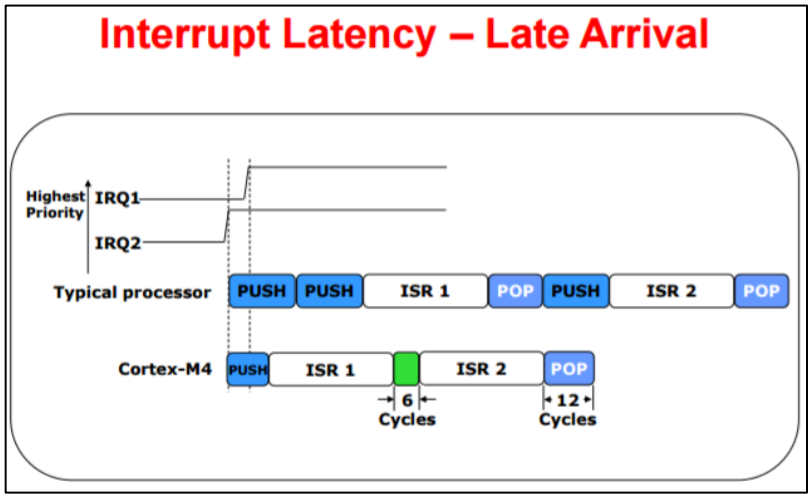


Fuente: *Getting Started with the Tiva™ TM4C123G LaunchPad Workshop.* https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf. Consulta: 28 de junio de 2017.

6.4.2.2. Llegada tarde de interrupciones

En este caso, igualmente, si existen las dos interrupciones mencionadas en el inciso anterior. Se activa IR2, generando un PUSH para IR2, si durante este tiempo se activa IR1, en lugar de ejecutar ISR2 directamente, el procesador ejecutará ISR1; luego, dejará pasar 6 ciclos y ejecutará ISR2 finalizando con un PUSH, ahorrando PUSH de IR1, POP de IR1 y PUSH de IR2.

Figura 24. **Llegada tarde de interrupciones**



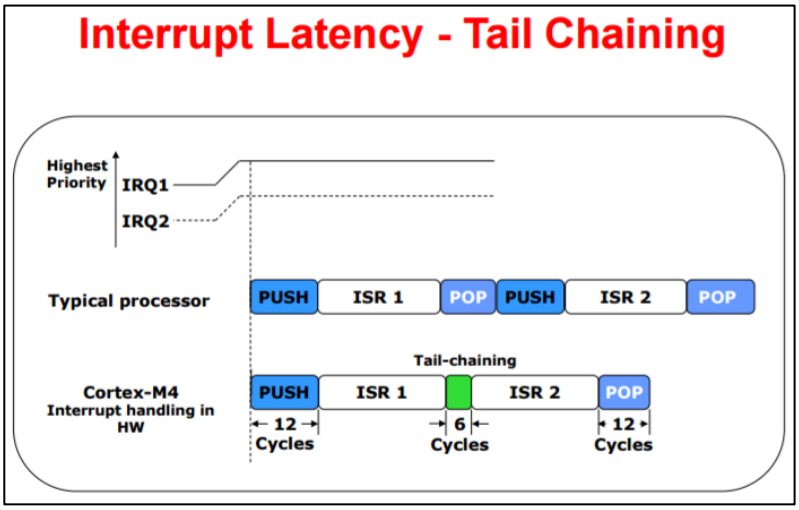
Fuente: *Getting Started with the Tiva™ TM4C123G LaunchPad Workshop.* https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf. Consulta: 28 de junio de 2017.

6.4.2.3. Encadenamiento de interrupciones

Si se generan dos interrupciones al mismo tiempo, IR1 e IR2, donde IR1 tiene mayor prioridad. La anticipación de las interrupciones funciona solamente

generando un PUSH y un POP para ambas interrupciones, dejando solamente 6 ciclos entre ambos ISR. Otros procesadores generarían un PUSH y un POP para cada interrupción.

Figura 25. **Encadenamiento de interrupciones**



Fuente: *Getting Started with the Tiva™ TM4C123G LaunchPad Workshop.* https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf. Consulta: 28 de junio de 2017.

6.4.3. Configuración del NVIC en el TM4C123GH6PM

A continuación, se describen los pasos necesarios para habilitar una IR:

- Habilitar las interrupciones globales
- Habilitar la interrupción del periférico a usar
- Establecer el nivel de prioridad de la interrupción
- Generar la rutina de interrupción
- Agregar la rutina al vector de interrupciones

6.4.4. Código de configuración de interrupciones

Se presenta el código en lenguaje C para configurar interrupciones en el controlador TM4C123GH6PM.

- IntMasterEnable(); //Habilitar interrupciones globales
- IntEnable(INT_TIMER0A); //Habilitar interrupción del periférico
- IntPrioritySet(INT_TIMER0A, 0); //Definir la prioridad

6.5. Conversor analógico digital (ADC)

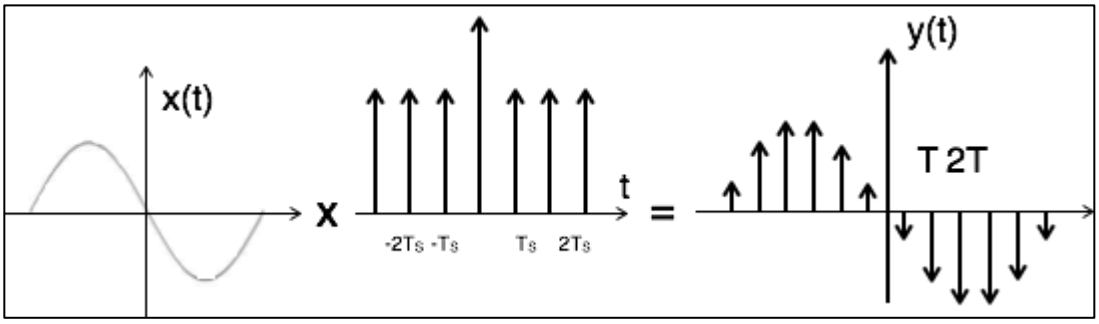
El conversor analógico digital ADC (siglas en inglés para *analog to digital converter)* es el periférico encargado de llevar a cabo la conversión de señales analógicas (obtenidas de medios continuos) a digitales (manejadas en tiempo y amplitud discreta). Al momento de diseñar un sistema embebido, trabajando con un sistema completamente digital, este es incapaz de comprender cualquier señal que no sea digital. Uno de los requerimientos de un sistema embebido es que este sea capaz de interactuar con el mundo exterior, por tanto, un conversor analógico a digital es de gran importancia. El proceso de conversión se divide en tres pasos principales:

6.5.1. Muestreo

El muestreo, conocido también como *sampling*, consiste en la toma de muestras de una señal analógica. Esto se logra al tomar valores discretos de voltaje a intervalos regulares en diferentes puntos de la señal.

Esto se expresa matemáticamente con la multiplicación de una señal analógica por un tren de impulsos.

Figura 26. **Muestreo por tren de impulsos**



Fuente: *Signals sampling techniques.* https://www.tutorialspoint.com /signals_and_systems/signals_sampling_techniques.htm. Consulta: 29 de junio de 2017.

De esta manera se obtiene el primer paso de la conversión, que es convertir los valores de tiempo continuo en valores de tiempo discreto. La selección del tiempo de muestreo es una parte esencial de este paso pues, de ser elegida de manera incorrecta, la información que lleva la señal se perdería.

6.5.1.1. Teorema de muestreo de Nyquist-Shannon

El teorema demuestra que la reconstrucción exacta de una señal periódica continua en banda base a partir de sus muestras, es matemáticamente posible si la señal está limitada en banda y la tasa de muestreo es superior al doble de su ancho de banda.

$$F_s > 2F_{max} = 2B$$

6.5.2. Cuantización

El proceso de cuantización (*quantization*), es el proceso de convertir los valores continuos de amplitud de la señal, a valores discretos de amplitud.

Esto se logra dividiendo el rango de amplitud en diferentes niveles y aproximando cada valor de amplitud tomado durante el muestreo al nivel más cercano; ninguna muestra puede estar fuera de estos niveles requeridos. A mayor cantidad de niveles, menor será la pérdida de información durante la aproximación de los valores, por tanto, elegir un dispositivo con una alta resolución es de suma importancia. Tras la cuantización se cuenta ya con una señal de tiempo y amplitud discreta.

6.5.2.1. Resolución de un ADC

Se refiere a la cantidad de niveles con los que trabaja el periférico. Esta se mide en cantidad de bits, donde cada bit indica la cantidad de niveles según la fórmula:

$$s = 2^N$$

Donde N indica la cantidad de bits y s la cantidad de niveles.

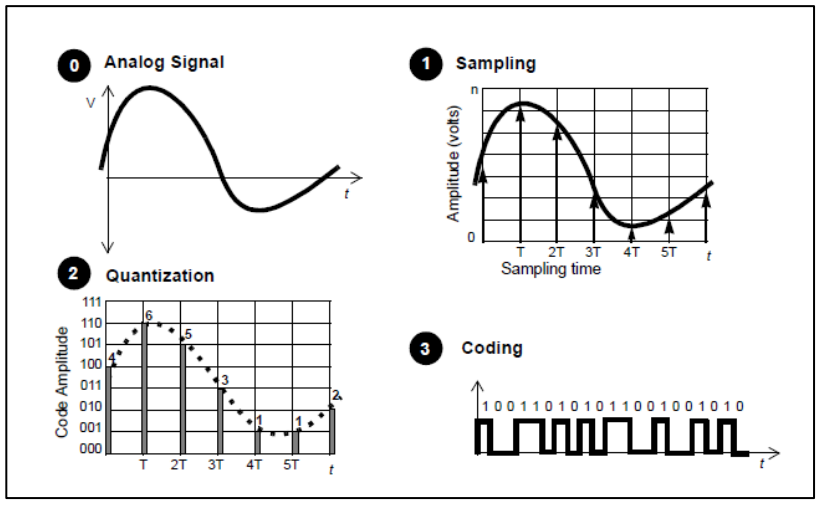
Suponer que se tiene un ADC de 8 bits, entonces se tendrán un total de 256 niveles diferentes, tomando niveles desde 0 hasta 255.

6.5.3. Codificación

El último paso en este proceso de conversión es el de codificación. La codificación se encarga de representar los valores tomados por la cuantización de forma numérica utilizando códigos ya establecidos. El código más utilizado es el código binario, donde a cada nivel se asigna un valor en binario desde 0 hasta 2^n-1.

Tener los valores de forma binaria implica que ya se cuenta con una señal completamente digital, por lo que es posible almacenarla en memoria y proceder a procesarla según sea necesario.

Figura 27. **Pasos de conversión por un ADC**



Fuente: Pulse Code Modulation. http://www.mapyourtech.com/entries/general/rejuvenating-pcm-pulse-code-modulation consulta: 29 de junio de 2017.

6.5.4. Características del ADC en el TM4C123GH6PM

El controlador TM4C123GH6PM cuenta con dos módulos ADC incorporados, dando así la posibilidad de poder realizar dos conversiones analógicas a digitales en paralelo. Esta es una gran ventaja sobre otros controladores pues la mayoría solamente cuenta con un módulo ADC. Entre las características de estos módulos se tienen:

- Canales: cuenta con 12 diferentes canales y un sensor de temperatura interno, que permiten configurar 12 pines diferentes como entradas analógicas.

- Resolución: 12 bits de resolución, que permiten 4 096 niveles de cuantización. Los niveles de voltaje permitidos están entre 0 y 3,3 V, logrando así detectar cambios de 805 µV.

- Frecuencia de muestreo: 1 millón de muestras por segundo por módulo. Si se trabaja con ambos módulos y estos se desfasan, es posible muestrear una misma señal a un total de 2 MHz.

- Memoria: 4 secuenciadores encargados de almacenar los datos con tamaños de buffer variables:

    - Secuenciador 0: Almacena ocho muestras
    - Secuenciador 1: Almacena cuatro muestras
    - Secuenciador 2: Almacena cuatro muestras
    - Secuenciador 3: Almacena una muestra

- *Trigger*: control de activación flexible que permiten configurar la activación de toma de muestras por diferentes accionadores, siendo estos los siguientes:

    - Software
    - Temporizadores
    - Comparadores
    - PWM
    - GPIO

- Promedio por hardware: permite obtener el promedio de muestras tomadas por medio de hardware para obtener una mejor precisión en los datos y no comprometer el tiempo de muestreo.

6.5.5. Configuración del ADC en el TM4C123GH6PM

Para configurar el funcionamiento del ADC en el controlador se deben seguir los siguientes pasos:

- Habilitar el periférico ADC y escoger el módulo
- Seleccionar el secuenciador a utilizar
- Seleccionar el tipo de *trigger* a utilizar
- Definir el nivel de prioridad
- Definir el canal a utilizar, si es un pin, configurarlo como tipo ADC
- Definir el orden de las muestras a tomar según el secuenciador
- Habilitar el secuenciador
- Tomar muestras

6.5.6. Código de configuración de ADC

Se presenta el código en lenguaje C para configurar el periférico ADC en el controlador TM4C123GH6PM

- SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); //Habilitar puerto.
- GPIOPinTypeADC(GPIO_PORTB_BASE, GPIO_PIN_5); //Definir pin como tipo ADC.
- SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //Habilitar periférico ADC.
- ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0); //Definir secuenciador, *trigger* y prioridad.
- ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH11 | ADC_CTL_IE | ADC_CTL_END); //Definir canal y orden de muestras.
- ADCSequenceEnable(ADC0_BASE, 3); //Habilitar secuenciador.

6.6. Modulación por ancho de pulso (PWM)

La modulación por ancho de pulso PWM (siglas en inglés para *pulse width modulation*) es una técnica empleada para la codificar un mensaje por medio de una señal de pulsos.

La modulación de ancho de pulso funciona modificando el ciclo de trabajo de una señal. Esto lo logra por medio del ciclo de trabajo de una señal cuadrada.

$$\overline{\mathrm{V}} = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\cdot dt$$

$$\overline{\mathrm{V}} = \frac{1}{T_t}\left[\int_0^{T_a} Vcc \cdot dt + \int_{T_a}^{T_b} 0 \cdot dt\right]$$

$$\overline{\mathrm{V}} = \frac{Vcc}{T_t}\left[\int_0^{T_a} dt\right]$$

$$\overline{\mathrm{V}} = \frac{Vcc \cdot t}{T_t}\Big|_0^{T_a}$$

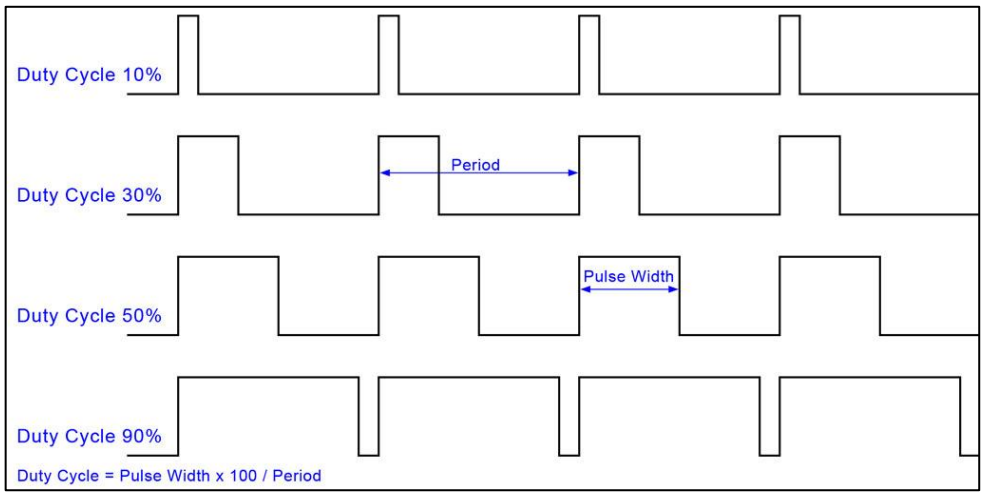
$$\overline{\mathrm{V}} = \frac{Vcc \cdot T_a}{T_t}, \qquad D = \frac{T_a}{T_t}$$

A la relación entre el tiempo en alto y el tiempo total de una señal cuadrada se le conoce como el ciclo de trabajo *D*. Se observa que el voltaje promedio entregado por una señal cuadrada es dependiente solamente del tiempo en alto que este pase, pues el tiempo total no cambiará la frecuencia ni la amplitud. Este principio es utilizado para darle diferentes aplicaciones fuera de la modulación solamente, entre estas destacan:

- Conmutación de fuentes de poder
- Control de motores
- Posicionamiento de servos
- Control lumínico

La señal PWM es digital cuyo valor de voltaje promedio a entregar varía, pero en ningún momento actúa como una señal analógica. En términos de aplicación, dado que la amplitud de la señal no cambia, se hace referencia solamente al ciclo de trabajo para denominar la señal en forma de porcentaje del voltaje total aplicado.

Figura 28. **Ciclo de trabajo de una señal periódica**



Fuente: *Pulse Width Modulation.* https://protostack.com.au/2011/06 /atmega168a-pulse-width-modulation-pwm/. Consulta: 29 de junio de 2017.

6.6.1. Generación de señales PWM con el TM4C123GH6PM

El periférico generador de PWM en el controlador TM4C123GH6PM cuenta con dos módulos PWM incluidos: PWM0 y PWM1, respectivamente.

Cada módulo provee cuatro bloques generadores, donde cada uno de estos bloques generadores puede producir dos señales PWM: independiente o sincronizada. Cada bloque generador también cuenta con opción de *trigger* y configuración de interrupciones. Las características del periférico son las siguientes:

- Hasta cuatro bloques generadores, donde cada uno incluye:
    - Un contador de 16-bit ascendente o descendente

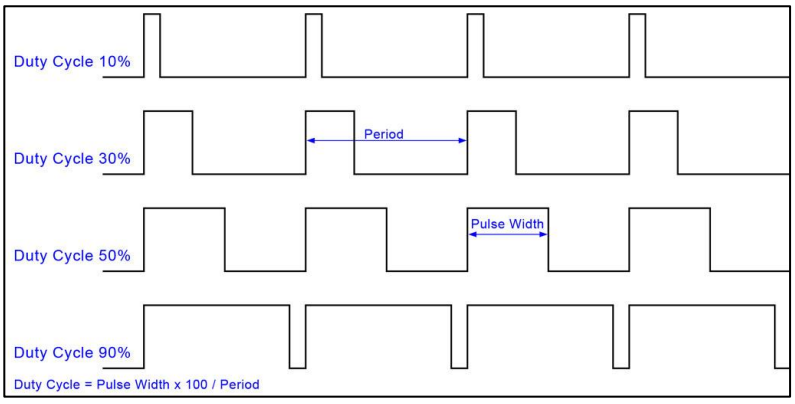
- Dos comparadores
- Generación de señal PWM
- Generación de banda muerta

- Bloque de control encargado de:

  - Habilitar o deshabilitar las salidas PWM
  - Controlar la polaridad de la señal
  - Sincronización de las señales
  - Manejo de errores
  - Manejo de interrupciones

En total es posible generar 16 señales PWM diferentes en paralelo con este controlador.

Figura 29. **Diagrama de bloques de generador PWM**



Fuente: *Getting Started with the Tiva™ TM4C123G LaunchPad Workshop*. https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf. Consulta: 29 de junio de 2017.

6.6.2. Configuración del módulo generador PWM en el TM4C123GH6PM

A continuación, se describe el proceso de configuración del módulo generador PWM:

- Habilitar el periférico correspondiente (PWM0, PWM1).
- Configurar los pines de salida como tipo PWM.
- Definir el canal.
- Configurar el generador a utilizar.
- Configurar la polaridad de la señal.
- Configurar el valor de carga para la señal, usado para determinar el periodo de la señal.
- Habilitar la salida del bloque generador a utilizar.
- Definir el ciclo de trabajo.

6.6.3. Código de configuración de PWM

Se presenta el código en lenguaje C para configurar el periférico PWM en el controlador TM4C123GH6PM

- SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); //Habilitar el periférico.
- GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6); //Configurar pin como PWM.
- GPIOPinConfigure(GPIO_PB6_M0PWM0); //Definir el canal.
- PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN); //Configurar generador y polaridad.

- PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, LoadDC); //Definir el valor de carga para el periodo de la señal.
- PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true); //Habilitar la salida.
- PWMGenEnable(PWM0_BASE, PWM_GEN_0); //Habilitar el bloque generador.
- PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, LOAD*5/100); //Definir el ciclo de trabajo.

6.7. Transmisor-receptor universal asíncrono

El transmisor-receptor universal asíncrono UART (siglas en inglés para *universal asynchronous receiver transmitter*) es un protocolo serial de comunicación entre dispositivos. Un protocolo de comunicación serial implica que la transmisión se hace utilizando la mutiplexación de la división temporal. Esto implica que se usa un mismo canal y se envían los datos en el tiempo, dándole una duración constante a cada bit (un bit detrás de otro).

UART utiliza una sola línea de transmisión y una de recepción, por donde se envían generalmente 8 bits (esto se puede modificar) que utilizan niveles lógicos. Recibe el nombre de comunicación asíncrona pues no depende una señal de reloj que indique el inicio o final de cada bit. La forma es de la siguiente manera:

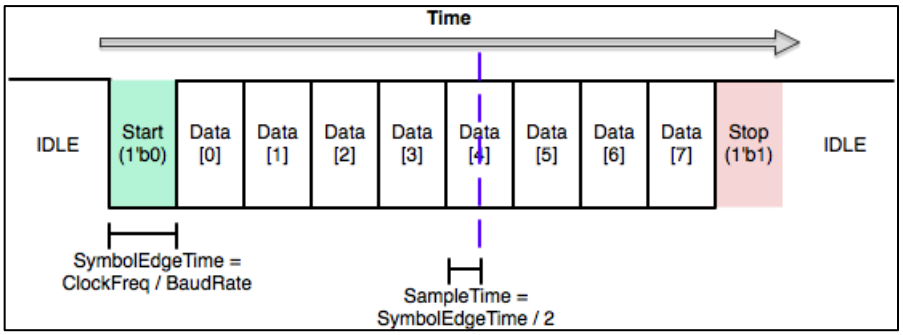
- Se mantiene la línea de transmisión en alto mientras no exista envío de datos.

- Se empieza la transmisión con un bit de inicio, el cual se encarga de cambiar el estado de la línea de transmisión de alto a bajo.

- Se envía bit tras bit, desde el bit 0 hasta el bit N.

- Se envía un bit de parada, encargado de regresar la línea de transmisión a su estado en alto sin importar el último valor que se haya enviado.

Figura 30. **Diagrama temporal de protocolo UART**



Fuente: UART. https://github.com/CourseReps/ECEN489-Fall2014/wiki/Arduino-Serial,-UART consulta: 30 de junio de 2017.

Para mantener una comunicación por medio del protocolo UART, deben mantenerse constantes la cantidad de bits a enviar y el tiempo de duración de cada bit.

6.7.1. *Bitrate y baudrate*

El *bitrate* es el número de bits que se transmiten por unidad de tiempo a través de un sistema de transmisión digital, con este dato es posible conocer la duración de cada bit, obviando por completo la necesidad de un reloj para la sincronización.

La tasa de baudios, (en inglés *baudrate*) también conocida como baudaje, es el número de unidades de señal por segundo. Un baudio puede contener varios bits. Una forma simple de entender esto, es tomando la cantidad de caracteres que se envía por segundo y no la cantidad de bits, de modo que, si se envían 1 000 bits en un segundo, pero cada carácter cuenta de 10 bits (incluyendo bit de inicio y de parada), el *baudrate* sería de 100, pues se envían solamente 100 caracteres en 1 000 bits. La entidad que definió el protocolo es la que determina la cantidad de bits que contiene un baudio en él. Para el caso de UART, un baudio equivale a un bit, por lo que el *bitrate* y *baudrate* son iguales, no es el caso para otros protocolos.

Tabla XV. **Valores de *baudrate* comunes en UART**

Rate No.	*M*	Baud Rate$_M$ (bps)	*BT* (CClk cycles)	*W* (CClk cycles)
1	1	921,600	32	14
2	2	460,800	64	24
3	3	307,200	96	32
4	4	230,400	128	40
5	6	153,600	192	64
6	8	115,200	256	86
7	12	76,800	384	130
8	16	57,600	512	174
9	24	38,400	768	260
10	32	28,800	1,024	350
11	48	19,200	1,536	524
12	64	14,400	2,048	702
13	96	9,600	3,072	1,098
14	128	7,200	4,096	1,356
15	192	4,800	6,144	2,098
16	256	3,600	8,192	2,814
17	384	2,400	12,288	4,914
18	768	1,200	24,576	9,832

Fuente: UART. http://umassamherstm5.org/tech-tutorials/pic32-tutorials/pic32mx220-tutorials/uart-to-serial-terminal. Consulta: 30 de junio de 2017.

6.7.2. Características del periférico UART en el TM4C123GH6PM

El periférico UART en el controlador cuenta con 8 módulos diferentes de UART, entre sus características principales se tienen:

- 16 memorias FIFO de 8 bits
- Generador de *baudrate* programable
- Detección de ruptura del canal
- Cantidad de bits de datos programable entre 5,6,7 y 8 bits
- Bits de paridad para par, impar o sin paridad
- Uno o dos bits de parada

6.7.3. Configuración del periférico UART en el TM4C123GH6PM

Se debe considerar que el periférico funciona por medio de interrupciones, por lo que una rutina de interrupción debe ser creada y configurar sus interrupciones respectivamente.

- Habilitar el periférico UART a utilizar
- Habilitar el puerto a utilizar
- Definir los pines como tipo UART
- Definir cuál será el transmisor y el receptor
- Definir el *baudrate*
- Definir la cantidad de bits a enviar
- Definir la cantidad de bits de parada a utilizar
- Definir si se usará bit de paridad

6.7.4. Código de configuración de UART

Se presenta el código en lenguaje C para configurar el periférico UART en el controlador TM4C123GH6PM

- SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //Habilitar periférico.
- GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //Definir pines como tipo UART.
- GPIOPinConfigure(GPIO_PA0_U0RX); //Definir pin transmisor.
- GPIOPinConfigure(GPIO_PA1_U0TX); //Definir pin receptor.
- UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE)); //Definir baudrate, bits de datos, paridad y parada.

6.8. Interfaz serial síncrona (SSI)

La interfaz serial síncrona SSI (siglas en inglés para *synchronous serial interface*) es un protocolo de comunicación entre dos dispositivos serial. Es síncrona pues cuenta con un reloj de sincronización entre ambos dispositivos.

Fue presentado por primera vez por Motorola, con el primer microcontrolador derivado de la arquitectura Motorola 68000 anunciado en 1979.

Su principal ventaja radica en que remueve la limitante de comunicación uno a uno, y permite mantener una comunicación serial entre uno y varios dispositivos, donde uno es el encargado de dictar el orden (maestro) y los otros esperan indicaciones solamente (esclavos). Su principal desventaja es que no

depende solamente de una línea de transmisión y una de recepción, sino de cuatro líneas de comunicación como mínimo. La cantidad de líneas de transmisión está dada por la cantidad de dispositivos interconectados, siendo esta una cantidad de M líneas de transmisión donde M es:

$$M= N + 3$$

Y N es la cantidad de dispositivos interconectados. Sus líneas de transmisión son:

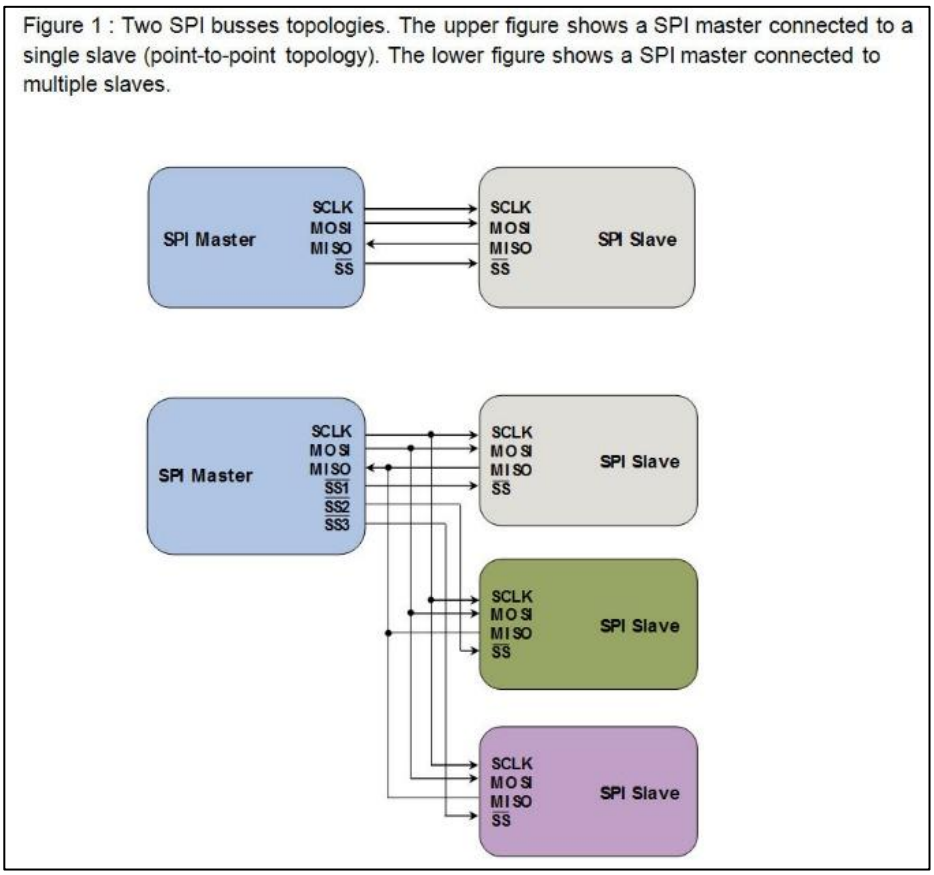
- SCLK: *serial clock*, encargado de transmitir la señal de reloj que sincronizará la transmisión de datos entre ambos dispositivos.

- MOSI: *master output slave input*, línea de transmisión de datos desde el maestro hacia el esclavo.

- MISO: *master input slave output*, línea de transmisión de datos desde el esclavo hacia el maestro.

- SS: *slave select*, encargado de habilitar el dispositivo con el que el maestro mantendrá la comunicación, debe existir uno por cada dispositivo conectado.

Figura 31. **Maestro comunicado por SSI con tres esclavos**



Fuente: *Introduction to SPI Protocol.* http://www.byteparadigm.com/applications /introduction-to-i2c-and-spi-protocols/. Consulta: 30 de junio de 2017.

6.8.1. Modos de transmisión SSI

Existen un total de cuatro modos de trasmisión SSI, dependiente de dos variables diferentes, la polaridad y la fase. En el protocolo, cuando no existe una trasmisión de datos, este estado se conoce como espera o idle. El término polaridad hace referencia al estado de la señal de reloj mientras se está en idle. Si un estado idle se realiza manteniendo la línea de señal de reloj en un estado

lógico bajo, la polaridad será 0. En cambio, si durante el estado idle, la línea de reloj se mantiene en un estado lógico alto, la polaridad será 1.

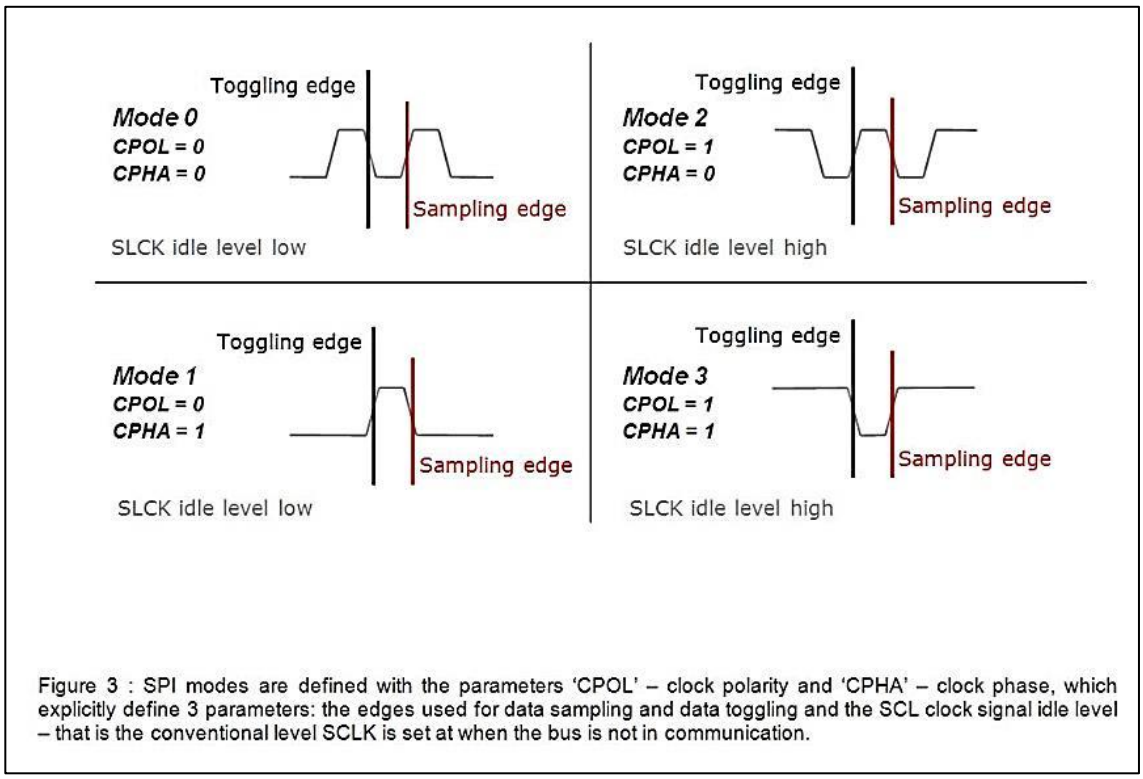
La fase depende del estado idle también. Al momento de comenzar una trasmisión se debe realizar un cambio de estado idle, si este se encuentra en estado lógico alto, debe cambiar a un estado lógico bajo, y si se encuentra en un estado lógico bajo debe cambiar a un estado lógico alto. Es posible configurar el sistema para que realice la toma de datos en flanco de subida o flanco de bajada.

Si la polaridad anterior era de 0, y se desea realizar la toma de datos en un flanco de subida, es necesario esperar un ciclo de reloj mientras se realiza el cambio de estado para poder comenzar la toma de datos, generando un desfase de un ciclo de reloj.

De la misma manera sucede si la polaridad es 1 y se desea realizar la toma de datos en un flanco de bajada. Cuando este desfase sucede, la fase es 1. Si tiene una polaridad 0, y se desea realizar una toma de datos en flanco de bajada se puede hacer inmediatamente sin esperar un desfase, de la misma manera que con polaridad 1 y flanco de subida, en este caso no existirá desfase y la variable fase será 0. Los cuatro modos de transmisión son:

- Modo 0: polaridad 0 y fase 0
- Modo 1: polaridad 1 y fase 0
- Modo 2: polaridad 0 y fase 1
- Modo 3: polaridad 1 y fase 1

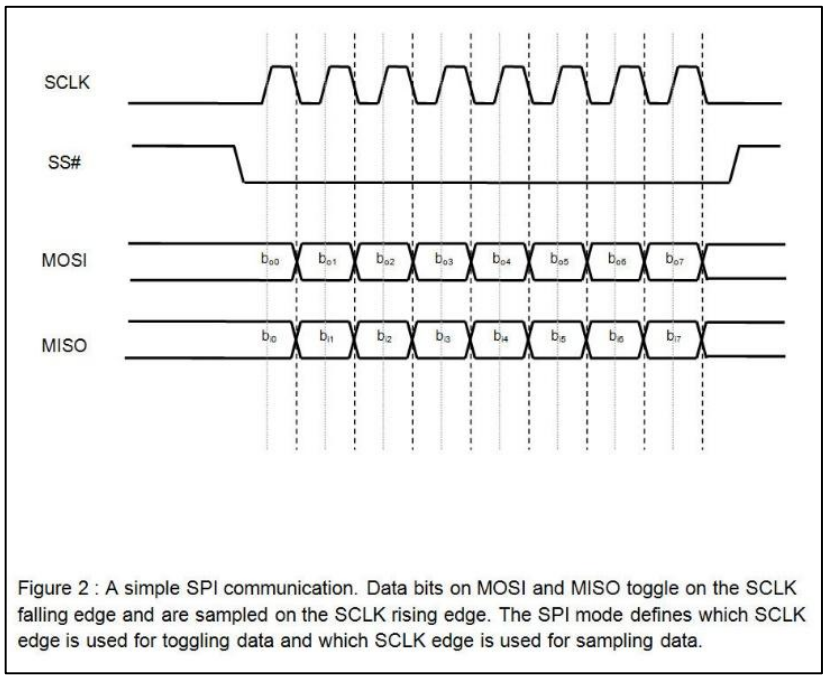
Figura 32. **Modos de transmisión SSI**



Fuente: *Introduction to SPI Protocol.* http://www.byteparadigm.com/applications /introduction-to-i2c-and-spi-protocols/ consulta: 30 de junio de 2017.

El maestro es el encargado de iniciar todas las comunicaciones con los esclavos. Cuando el maestro desea enviar o solicitar datos de un esclavo, debe seleccionar el esclavo con quien desea iniciar la comunicación por medio del pin SS. Seguidamente, debe activar la señal de reloj, los datos se toman sincronizados con el reloj.

Figura 33. **Diagrama temporal protocolo SSI**



Fuente: *Introduction to SPI Protocol.* http://www.byteparadigm.com/applications /introduction-to-i2c-and-spi-protocols/ consulta: 30 de junio de 2017.

6.8.2. Características del periférico SSI en el TM4C123GH6PM

El periférico SSI cuenta con cuatro módulos diferentes: SSI0, SSI1, SSI2 y SSI3. Cada módulo contiene:

- Memoria FIFO interna capaz de almacenar hasta 16 datos
- Configuración de bits de datos a enviar, entre 4 a 16 bits
- Conversión serial-paralelo y viceversa automática
- Opción a ser utilizado como maestro o esclavo
- Generación de reloj dependiente del reloj del sistema
- Opción de acceso directo a memoria

6.8.3. Configuración del periférico SSI en el TM4C123GH6PM

A continuación, se describe el proceso de configuración.

- Habilitar el periférico SSI a utilizar
- Habilitar el puerto a utilizar
- Configurar los pines del tipo SSI
- Establecer los pines para SCLK, CS, MISO y MOSI
- Definir el modo de trasmisión
- Definir como maestro o esclavo
- Definir la frecuencia de SCLK
- Definir la cantidad de bits de datos a utilizar

6.8.4. Código de configuración de SSI

Se presenta el código en lenguaje C para configurar el periférico SSI en el controlador TM4C123GH6PM

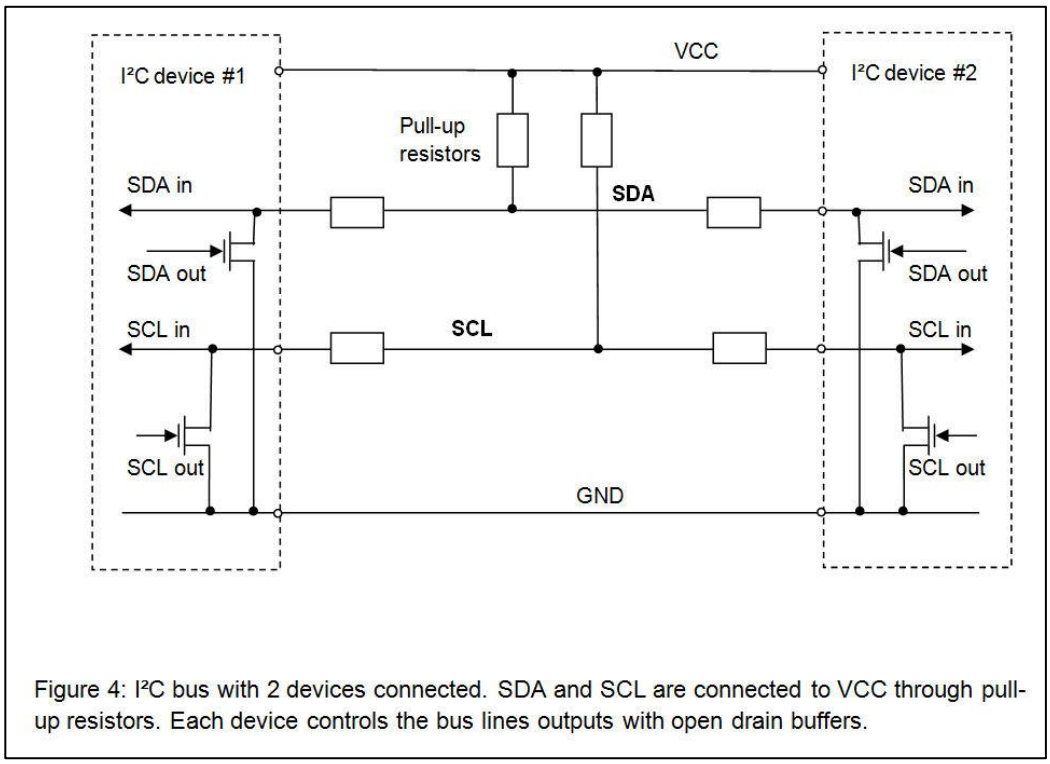
- SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0); //Habilitar periférico.
- GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_5); //Definir pines como tipo SSI.
- GPIOPinConfigure(GPIO_PA2_SSI0CLK); //Definir SCLK.
- GPIOPinConfigure(GPIO_PA3_SSI0FSS); //Definir SS.
- GPIOPinConfigure(GPIO_PA5_SSI0TX); //Definir MOSI.
- SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_TI, SSI_MODE_MASTER, 2000000, 16); //Definir modo de transmission, modo maestro, frecuencia de reloj y cantidad de bits.
- SSIEnable(SSI0_BASE); //Habilitar el funcionamiento.

6.9. Protocolo circuito inter-integrado (I2C)

El bus I2C (*inter-integrated circuit*) fue desarrollado en 1982 por Philips; su propósito original era el de proveer una forma fácil de conectar el CPU a periféricos en un set de televisión. El protocolo I2C es un protocolo serial síncrono, requiere solamente dos señales para conectar todos los periféricos al microcontrolador: SDA y SCL. La especificación original definió la velocidad de bus a 100 kbps (kilobits por segundo). Esta especificación fue revisada muchas veces, hasta introducir la velocidad de 400 kbps en 1995 y desde 1998, 3,4 Mbps para periféricos más rápidos.

La comunicación suele ser un poco más elaborada que los otros protocolos, pero esta integra la ventaja de pocas líneas de trasmisión y comunicación entre muchos dispositivos. Cuando el maestro desea comunicarse con el esclavo comienza enviando la secuencia de inicio del protocolo. Cuando se desea terminar la comunicación se envía la secuencia de parada. Se diferencian las secuencias de inicio y parada ya que son los únicos lugares donde SDA (línea de dato) se le permite cambiar mientras SCL (línea *clock*) se encuentra en alto. Un esclavo no puede iniciar la transferencia a través del bus I2C, solamente el maestro.

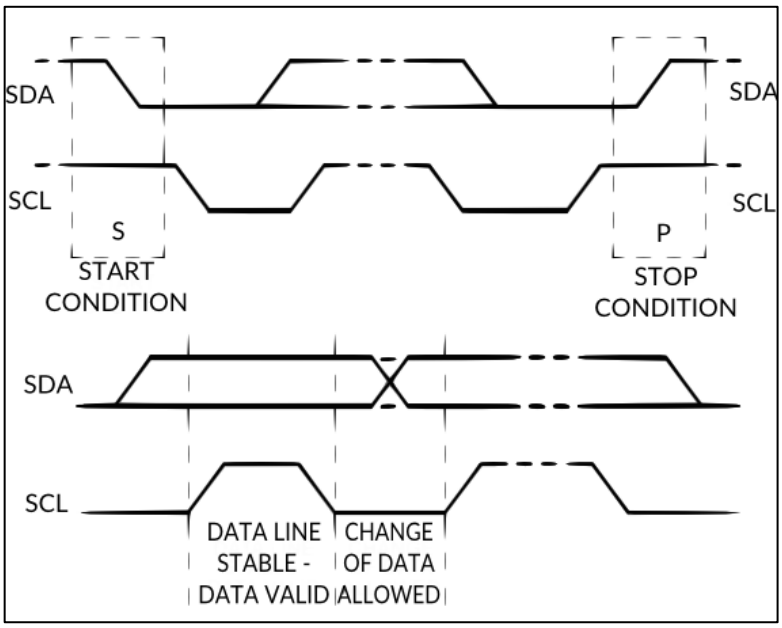
Figura 34. **Conexión del bus I2C**



Fuente: *Introduction to I2C Protocol.* http://www.byteparadigm.com/applications /introduction-to-i2c-and-spi-protocols/ consulta: 30 de junio de 2017.

Los datos son transferidos en secuencias de 8 bits. Los bits se envían por la línea SDA en formato *big endian*. Por cada 8 bits transferidos, el dispositivo recibiendo los datos envía de regreso un bit de ACK (aceptación), por lo que se requieren 9 cambios del reloj para transferir 8 bits. Si el dispositivo receptor envía un bit ACK en bajo, significa que ha recibido la información y está listo para aceptar otro byte. Si envía un bit de ACK en alto, entonces, indica que no puede aceptar más datos en ese momento y el maestro deberá enviar una secuencia de parada.

Figura 35. **Secuencias del bus I2C**



Fuente*: Introduction to I2C Protocol*. http://www.byteparadigm.com/applications /introduction-to-i2c-and-spi-protocols/ consulta: 30 de junio de 2017.

6.9.1. Direccionamiento de dispositivos

Todas las direcciones son de 7 o 10 bits. En general, se utilizan 7 bits. Las direcciones se utilizan para diferenciar el dispositivo con el que se establecerá comunicación, esto implica que se puede tener hasta 128 dispositivos en el bus. Junto a la dirección se envía un bit para informar al esclavo si el maestro estará leyendo o escribiendo. Si el bit es 0 el maestro escribirá al esclavo; si es 1 el maestro leerá del esclavo.

6.9.2. Características del periférico I2C en el TM4C123GH6PM

Cuenta con módulos I2C maestro y esclavo con la habilidad de comunicarse con otros dispositivos. El bus está definido para trabajar con dispositivos que soportan tanto lectura como escritura de datos. El módulo I2C puede funcionar a diferentes velocidades:

- *Standard* (100 kbps)
- *Fast* (400 kbps)
- *Fast plus* (1 Mbps)
- *High speed* (3,33 Mbps)

Tanto el maestro como el esclavo pueden generar interrupciones cuando la operación es completada o abortada debido a un error. El esclavo genera interrupciones cuando se envían o requieren datos de parte del maestro.

6.9.3. Configuración del periférico I2C en el TM4C123GH6PM

Para realizar la configuración se debe definir primero el modo a utilizar, estos pueden ser:

- Modo maestro
    - Habilitar el periférico I2C a utilizar
    - Habilitar el puerto
    - Configurar los pines como tipo I2C
    - Definir el pin SCL y SDA
    - Definir la dirección del esclavo

- Modo esclavo

    - Habilitar el periférico I2C a utilizar
    - Habilitar el puerto
    - Configurar los pines como tipo I2C
    - Definir el pin SCL y SDA
    - Definir la dirección del esclavo
    - Verificar si el maestro requiere datos

6.9.4. Código de configuración de I2C

Se presenta el código en lenguaje C para configurar el periférico I2C modo maestro en el controlador TM4C123GH6PM

- SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1); //Habilitar periférico.
- GPIOPinConfigure(GPIO_PA7_I2C1SDA); //Definir pin de datos.
- GPIOPinConfigure(GPIO_PA6_I2C1SCL); //Definir pin de clock.
- I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(),true); //Definir clock del periferico.
- I2CMasterSlaveAddrSet(I2C1_BASE, 0x3B, false); //Definir dirección.

7. PRÁCTICAS EN LENGUAJE C PARA USO DE PERIFÉRICOS

Se presentan las prácticas propuestas de configuración de periféricos para el uso en los laboratorios de sistemas embebidos.

7.1. Práctica 1: configuración señal de RELOJ

Código con una configuración del reloj a 80MHz, con el uso del depurador para observar el valor del reloj en tiempo real.

7.2. Práctica 2: configuración de GPIO entrada

Código con una configuración de GPIO para uso de un interruptor como entrada.

7.3. Práctica 3: configuración de GPIO salida

Código con una configuración de GPIO para uso de pines como salidas, se aprovechan los leds de la tarjeta de desarrollo

7.4. Práctica 4: configuración de temporizador

Código con una configuración de un temporizador con valor de carga de un segundo. Se utiliza el depurador para visualizar su valor.

7.5. Práctica 5: configuración de interrupción por temporizador

Código con una configuración de interrupciones generadas por un temporizador para generar un led que parpadea.

7.6. Práctica 6: configuración de ADC SS3

Código con una configuración de toma de muestra de una señal analógica en un pin con el secuenciador 3, el *trigger* se hace por medio de un temporizador.

7.7. Práctica 7: configuración de ADC SS1

Código con una configuración de toma de muestra del sensor de temperatura interno de la tarjeta de desarrollo usando el secuenciador 1, el *trigger* se hace por medio de software.

7.8. Práctica 8: configuración de PWM DC

Código con una configuración del periférico PWM para generación de una señal de frecuencia 1 KHz. Se varía el ciclo de trabajo por medio de pulsadores.

7.9. Práctica 9: configuración de PWM servo

Código con una configuración del periférico PWM para generación de una señal de frecuencia 50 Hz para control de servomotores. Se varía el ángulo del motor por medio de pulsadores.

7.10. Práctica 10: configuración de UART

Código con una configuración del periférico UART para uso del protocolo a 115200 baudios. Se reciben códigos ASCII y se varía el color de led dependiendo de la entrada.

7.11. Práctica 11: configuración de SSI

Código con una configuración del periférico SSI para transmisión de datos a un TLV5616.

7.12. Práctica 12: configuración de I2C

Código con una configuración del periférico I2C para comunicación con un sensor BH1750FVI.

Los códigos se encuentran disponibles en el siguiente enlace:

- https://github.com/dbr2907/Introduccion_Sistemas_Embebidos_TM4C123GH6PM.

CONCLUSIONES

1. Se desarrollaron los temas necesarios para brindar soporte a los estudiantes de la carrera de Ingeniería en Electrónica y afines de modo que puedan desarrollar aplicaciones que involucren el diseño de sistemas embebidos.

2. Se introdujo el controlador TM4C123GH6PM como procesador embebido para el diseño de sistemas embebidos.

3. Se realizaron ejercicios prácticos y se subieron a una plataforma en línea de fácil acceso para que el estudiante pueda comprender la teoría detrás de la implementación práctica.

4. Se desarrolló el material teórico necesario en español para la actualización del laboratorio de microcontroladores en la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala.

RECOMENDACIONES

1. Seis pines en el controlador TM4C123GH6PM se encuentran bloqueados contra programación accidental. Esto se debe a que cumplen un propósito alterno. PC3, PC2, PC1 y PC0 funcionan para el JTAG y SWD. PD7 y PF0 funcionan para interrupciones no enmascarables. Para su uso se deben desbloquear modificando el registro GPIOLOCK.

2. Si el sistema embebido a desarrollar es dependiente de muchos sensores, existen un conjunto de librerías para el TM4C123GH6PM dedicas al uso de sensores bajo el nombre de Sensorlib.

3. Al momento de realizar un sistema embebido, se debe comenzar creando un prototipo, para esto se recomienda el uso de la tarjeta de desarrollo Tiva C.

4. En este documento se tratan solamente los periféricos más comunes. Por lo amplio del contenido que representa el diseño de sistemas embebidos, se debe proponer a las autoridades encargadas de la Escuela de Ingeniería de Mecánica Eléctrica implementar un curso opcional que pueda dar continuidad al mismo.

5. El auxiliar de laboratorio que tome como base el presente trabajo, debe considerar que el laboratorio de microcontroladores es parte del curso Electrónica 3, por lo que no existe un curso que presente la teoría necesaria para el diseño de sistemas embebidos. Por tal razón, realizar

un doble esfuerzo, impartiendo el doble de clases, una para cubrir el contenido teórico y otra para el práctico.

6. La decisión del procesador embebido a utilizar debe hacerse por medio de una investigación exhaustiva y no por gustos personales o popularidad. La mala selección de un procesador embebido generará aumento de costos y puede finalizar en un sistema ineficiente.

BIBLIOGRAFÍA

1. *ARM Architecture*. [en línea]. <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0014q/I83164.html>. [Consulta: 30 de junio de 2017].

2. *ARM Developer Cortex M-4.* [en línea]. <https://developer.arm.com/products/processors/cortex-m/cortex-m4>. [Consulta: 18 de junio de 2017].

3. *Difference between RISC and CISC architecture*. [en línea]. <http://www.firmcodes.com/difference-risc-sics-architecture/>. [Consulta: 18 de junio de 2017].

4. *Getting Started with the Tiva™ TM4C123G LaunchPad Workshop.* [en línea].<https://www.cse.iitb.ac.in/~erts/html_pages/Resources/Tiva/TM4C123G_LaunchPad_Workshop_Workbook.pdf>. [Consulta: 15 de junio de 2017].

5. GIPPER, Jerry. *ARMed and ready*. [en línea]. <http://vita.mil-embedded.com/articles/armed-ready/>. [Consulta: 19 de junio de 2017].

6. *IDE de Programación*. [en línea]. <https://www.ecured.cu/IDE_de_Programaci%C3%B3n>. [Consulta: 21 de junio de 2017].

7. RAMIREZ, Oscar. *Introducción a los sistemas físico-cibernéticos como actualización en las prácticas de laboratorio del curso de sistemas de control.* Trabajo de graduación de Ing. Electrónica. Universidad de San Carlos de Guatemala, Facultad de Ingeniería, 2016. 586 p.

8. SHIMPI, Anand. *ARM's Cortex M: Even smaller and Lower Power CPU Cores.* [en línea]. <http://www.anandtech.com/show/8400/arms-cortex-m-even-smaller-and-lower-power-cpu-cores>. [Consulta: 20 de junio de 2017].

9. *The ARM Architecture With a focus on v7A and Cortex-A8*. [en línea]. <https://www.arm.com/files/pdf/ARM_Arch_A8.pdf>. [Consulta: 15 de junio de 2017].

10. *Tiva™ TM4C123GH6PM Microcontroller.* [en línea]. <http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf>. [Consulta: 15 de junio de 2017].

11. *TivaWare™ Peripheral Driver Library USER'S GUIDE.* [en línea]. <http://www.ti.com/lit/ug/spmu298d/spmu298d.pdf>. [Consulta: 15 de junio de 2017].

12. WADDA, Ken. *System Ticks.* [en línea]. <http://www.embedded.com/electronics-blogs/embedded-round-table/4406452/System-ticks>. [Consulta: 20 de junio de 2017].

13. *Which ARM Cortex Core Is Right for Your Application: A, R or M?* [en línea]. <https://www.silabs.com/documents/public/white-papers/Which-ARM-Cortex-Core-Is-Right-for-Your- Application. pdf >. [Consulta: 18 de junio de 2017].

14. YIU, Joseph. *ARM® Cortex®-M for Beginners.* [en línea]. <https://community.arm.com/cfs-file/__key/telligent-evolutioncomponents-attachments/01-2142-00-00-00-00-52-96/White- _2D00_M-for-Beginners-_2D00_-2016-_2800_final-v3_2900_.pdf>. [Consulta : 18 de junio de 2017].

APÉNDICE

Apéndice 1. **Divisor de frecuencia y frecuencia de la señal de reloj**

Divisor	Frecuencia (MHz)
2,5	80,00
3,0	66,67
3,5	57,14
4,0	50,00
4,5	44,44
5,0	40,00
5,5	36,36
6,0	33,33
6,5	30,77
7,0	28,57
7,5	26,67
8,0	25,00
8,5	23,53
9,0	22,22
9,5	21,05
10,0	20,00
10,5	19,05
11,0	18,18
11,5	17,39
12,0	16,67
12,5	16,00
13,0	15,38
13,5	14,81
14,0	14,29
14,5	13,79
15,0	13,33
15,5	12,90
16,0	12,50
16,5	12,12
17,0	11,76
17,5	11,43
18,0	11,11
18,5	10,81
19,0	10,53

Continuación del apéndice 1.

19,5	10,26
20,0	10,00
20,5	9,76
21,0	9,52
21,5	9,30
22,0	9,09
22,5	8,89
23,0	8,70
23,5	8,51
24,0	8,33
24,5	8,16
25,0	8,00
25,5	7,84
26,0	7,69
26,5	7,55
27,0	7,41
27,5	7,27
15,0	13,33
15,5	12,90
16,0	12,50
16,5	12,12
17,0	11,76
17,5	11,43
18,0	11,11
18,5	10,81
19,0	10,53
19,5	10,26
20,0	10,00
20,5	9,76
21,0	9,52
21,5	9,30
22,0	9,09
22,5	8,89
23,0	8,70
23,5	8,51
24,0	8,33
24,5	8,16
25,0	8,00
25,5	7,84
26,0	7,69
26,5	7,55
27,0	7,41

Continuación del apéndice 1.

27,5	7,27
15,0	13,33
15,5	12,90
16,0	12,50
16,5	12,12
17,0	11,76
17,5	11,43
18,0	11,11
18,5	10,81
19,0	10,53
19,5	10,26
20,0	10,00
20,5	9,76
21,0	9,52
21,5	9,30
22,0	9,09
22,5	8,89
23,0	8,70
23,5	8,51
24,0	8,33
24,5	8,16
25,0	8,00
25,5	7,84
26,0	7,69
26,5	7,55
27,0	7,41
27,5	7,27
28,0	7,14
28,5	7,02
29,0	6,90
29,5	6,78
30,0	6,67
30,5	6,56
31,0	6,45
31,5	6,35
32,0	6,25
32,5	6,15
33,0	6,06
33,5	5,97
34,0	5,88
34,5	5,80
35,0	5,71

Continuación del apéndice 1.

35,5	5,63
36,0	5,56
36,5	5,48
37,0	5,41
37,5	5,33
38,0	5,26
38,5	5,19
39,0	5,13
39,5	5,06
40,0	5,00
40,5	4,94
33,0	6,06
33,5	5,97
34,0	5,88
34,5	5,80
35,0	5,71
35,5	5,63
36,0	5,56
36,5	5,48
37,0	5,41
37,5	5,33
38,0	5,26
38,5	5,19
39,0	5,13
39,5	5,06
40,0	5,00
40,5	4,94
41,0	4,88
41,5	4,82
42,0	4,76
42,5	4,71
43,0	4,65
43,5	4,60
44,0	4,55
44,5	4,49
45,0	4,44
45,5	4,40
46,0	4,35
46,5	4,30
47,0	4,26
47,5	4,21
48,0	4,17

Continuación del apéndice 1.

48,5	4,12
49,0	4,08
49,5	4,04
50,0	4,00
50,5	3,96
51,0	3,92
51,5	3,88
52,0	3,85
52,5	3,81
53,0	3,77
53,5	3,74
54,0	3,70
54,5	3,67
55,0	3,64
55,5	3,60
56,0	3,57
56,5	3,54
57,0	3,51
57,5	3,48
58,0	3,45
58,5	3,42
59,0	3,39
59,5	3,36
60,0	3,33
60,5	3,31
61,0	3,28
61,5	3,25
62,0	3,23
62,5	3,20
63,0	3,17
63,5	3,15
64,0	3,13

Fuente: elaboración propia