Todo sobre sistemas embebidos

Arquitectura, programación y diseño de aplicaciones prácticas con el PIC18F

Sergio Salas Arriarán

Todo sobre sistemas embebidos

Arquitectura, programación y diseño de aplicaciones prácticas con el PIC18F

Sergio Salas Arriarán

Lima, setiembre de 2015
Universidad Peruana de Ciencias Aplicadas

© Universidad Peruana de Ciencias Aplicadas (UPC)

Primera publicación: setiembre de 2015

Edición: Diana Félix
Corrección de estilo: Luigi Battistolo
Diseño de cubierta: Christian Castañeda
Diagramación: Diana Patrón Miñán / Christian Castañeda

Editor del proyecto editorial

Universidad Peruana de Ciencias Aplicadas S. A. C.

Av. Alonso de Molina 1611, Lima 33 (Perú) Teléf:

313-3333

www.upc.edu.pe

Primera edición: setiembre de 2015

Versión ebook 2015

Digitalizado y Distribuido por Saxo.com Perú S.A.C.

saxo

www.saxo.com/es

yopublico.saxo.com

Tel: 51-1-221-9998

Dirección: Av. 2 de Mayo 534 Of. 304, Miraflores

Lima-Perú

Universidad Peruana de Ciencias Aplicadas (UPC)
Centro de Información

Salas Arriarán, Sergio. Todo sobre sistemas embebidos. Arquitectura, programación y diseño de aplicaciones prácticas con el PIC18F

Lima: Universidad Peruana de Ciencias Aplicadas (UPC), 2015

ISBN de la versión impresa: 978-612-318-033-1

ISBN de la versión en pdf: xxxxx

ISBN de la versión epub y mobi: xxxxx

Conceptos básicos - Arquitectura del microcontrolador PIC18F - El compilador para PIC18F. El MPLAB X IDE - El lenguaje ensamblador del PIC18F - Rutinas típicas en lenguaje ensamblador - Interrupciones - Los puertos de entrada y salida - El lenguaje ANSI C - Los módulos de temporización - El convertidor analógico digital - El módulo mejorado de comunicación serial síncrona asíncrona (EUSART) - La interfaz serial periférica - La interfaz serial I2C - El módulo USB.

006.22 SALA

Todos los derechos reservados. Esta publicación no puede ser reproducida, ni en todo ni en parte, ni registrada en o transmitida por un sistema de recuperación de información, en ninguna forma ni por ningún medio, sea mecánico, fotoquímico, electrónico, magnético, electroóptico, por fotocopia o cualquier otro, sin el permiso previo, por escrito, de la editorial.

El contenido de este libro es responsabilidad del autor y no refleja necesariamente la opinión de los editores.

A mis alumnos, quienes con su interés, a través de constantes preguntas e inquietudes, me animaron al desarrollo de esta obra.

A mis excolegas del INICTEL-UNI, por su compañerismo y las experiencias compartidas.

A mis padres, por las enseñanzas a lo largo de toda la vida.

Contenido

Prólogo	11
Introducción	13
Capítulo 1. Conceptos básicos	17
1.1 Definiciones básicas	17
1.2 Sistemas embebidos	38
1.3 Lenguajes de programación	39
1.4 La familia PIC18F4X de Microchip	42
1.5 Herramientas de desarrollo para Microchip	44
1.6 Características específicas del PIC18F4550	48
Capítulo 2. Arquitectura del microcontrolador PIC18F	51
2.1 La Unidad central de proceso (CPU)	53
2.2 Las unidades de memoria	59
2.3 Almacenamiento de las instrucciones en la memoria de programa	71
2.4 Las interfaces de entrada y salida (E/S)	78
2.5 El Reset	81
2.6 Temporización y opciones de reloj	84
2.7 Modos de ahorro de energía	88
2.8 El Perro guardián	90
Capítulo 3. El compilador para PIC18F. El MPLAB X IDE	93
3.1 IDE	93
3.2 Creación de un nuevo proyecto en MPLAB X usando el compilador MPASM	94
3.3 Elaboración de un programa sencillo. Partes del código y reglas básicas	98
3.4. Simulación de un programa en lenguaje ensamblador	106
3.5 Los archivos de configuración	111
3.6 El compilador MPLAB XC8	113

3.7 Creación de un nuevo proyecto en MPLAB usando el compilador MPLAB XC8	115
3.8 Proceso de grabación del microcontrolador PIC18F4550	118
Capítulo 4. El lenguaje ensamblador del PIC18F	123
4.1 El conjunto de instrucciones	124
4.2 Modos de direccionamiento	165
4.3 El Puntero de programa	173
4.4 La pila de direcciones	175
4.5 Instrucciones para el manejo de tablas en la memoria de programa	180
4.6 Acceso a la memoria EEPROM de datos	185
4.7 Directivas para el lenguaje ensamblador	187
Capítulo 5. Rutinas típicas en lenguaje ensamblador	217
5.1 Diseño de un diagrama de flujo	217
5.2 Rutinas básicas de retardo de tiempo	221
5.3 Conversión de bases	225
5.4 Operaciones matemáticas	237
5.5 Detección de teclas pulsadas en un teclado matricial	248
Capítulo 6. Interrupciones	261
6.1 El concepto de interrupción	261
6.2 El vector de interrupción	265
6.3 La prioridad de una interrupción	267
6.4 Latencia de una interrupción	270
6.5 Ejemplo de configuración de una interrupción externa	271
Capítulo 7. Los puertos de entrada y salida	277
7.1 Los puertos de entrada y salida del PIC18F4550	277
7.2 Ejemplo de control de un teclado matricial mediante la interrupción de cambio de estado del Puerto B	282
7.3 Manejo de un módulo LCD alfanumérico basado en el controlador HD44780	286

Capítulo 8. El lenguaje ANSI C	311
8.1 Historia del lenguaje ANSI C	311
8.2 La estructura de un programa en lenguaje C	315
8.3 Los tipos de datos	317
8.4 Los operadores en ANSI C	322
8.5 Sentencias condicionales e iterativas	328
8.6 Funciones y librerías	338
8.7 Arreglos y punteros	346
8.8 Estructuras y uniones	350
8.9 Tipos de variables compuestos	358
8.10 Uso del lenguaje ensamblador dentro de ANSI C	360
8.11 Manejo de interrupciones en MPLAB XC8	362
8.12 Las directivas del preprocesador	365
8.13 Ejemplos de aplicaciones con lenguaje ANSI C	371
 Capítulo 9. Los módulos de temporización	 401
9.1 Los temporizadores del PIC18F4550	401
9.2 Diseño de un semáforo sincronizado con el <i>Timer 0</i>	414
9.3 Diseño de un reloj en tiempo real con el <i>Timer 1</i>	420
9.4 El módulo de entrada de captura	423
9.5. El modo de comparación de salida	435
9.6 El modo de modulación por ancho de pulso (PWM)	444
 Capítulo 10. El convertidor analógico digital	 451
10.1 Fundamentos de un sistema de adquisición de datos	451
10.2 La conversión analógica digital	454
10.3 El teorema del muestreo	459
10.4 El convertidor analógico digital del PIC18F4550	460
10.5 Ejemplos de aplicaciones con el convertidor analógico digital del PIC18F4550	468

Capítulo 11. El módulo mejorado de comunicación serial síncrona asíncrona (EUSART)	479
11.1 Nociones básicas de la comunicación serial	479
11.2 La tasa de bit y el formato NRZ	482
11.3 El estándar RS232	484
11.4 El módulo EUSART del PIC18F4550	486
11.5 Ejemplos de aplicación con el EUSART del PIC18F4550	494
 Capítulo 12. La interfaz serial periférica	 515
12.1 Fundamentos de la comunicación serial síncrona SPI	515
12.2 Pines del SPI	516
12.3 Formatos de transmisión del bus SPI	517
12.4 El módulo SPI del PIC18F4550	519
12.5 Ejemplos de aplicación con el bus SPI del PIC18F4550	524
 Capítulo 13. La interfaz serial I²C	 551
13.1 Características del protocolo I ² C	551
13.2 Transferencia de datos en el protocolo I ² C	553
13.3 Arbitrariedad	555
13.4 Direccionamiento de periféricos	555
13.5 Formato de transferencia de datos	556
13.6 Registros de configuración del PIC18F4550	557
13.7 Programación del módulo MSSP en modo I ² C	561
13.8 Ejemplos de aplicación con el módulo I ² C del PIC18F4550	564
 Capítulo 14. El módulo USB	 575
14.1 Fundamentos del bus USB	576
14.2 Modos de transferencia	581
14.3 El proceso de enumeración	584
14.4 Las clases de descriptores	585
14.5 El módulo USB del PIC18F4550	587
14.6 La librería MLA (<i>Microchip Library Application</i>) para dispositivos USB de Microchip	590
14.7 Ejemplo de comunicación entre el PIC18F4550 y un computador a través del puerto USB	593
 Bibliografía	 603

Prólogo

Escribir un libro académico es dejar una huella para siempre. Decidir hacerlo no solamente conlleva capacidad, conocimientos, experiencia, dedicación y sacrificio, sino también la intención de que quienes lo lean y lo utilicen puedan aprender su contenido de una manera más fácil y comprensible. Ello constituye un gran desafío, que únicamente puede tener éxito cuando el autor domina el tema y cuando el estilo utilizado para transmitir el conocimiento es muy motivador, utilizando adecuadamente las herramientas audiovisuales disponibles.

En el presente libro, titulado *Todo sobre sistemas embebidos*, Sergio Salas, su autor, ha logrado de manera efectiva compendiar, en sus catorce capítulos, todos los elementos necesarios para la cabal comprensión de dichos sistemas y de sus aplicaciones, a través de ejemplos prácticos. La ruta propuesta para ello comienza por tratar los conceptos básicos acerca de los sistemas de numeración utilizados en la electrónica digital, lo que le sirve de soporte para poder explicar la arquitectura del microcontrolador PIC18F, el cual ha sido elegido en el texto por ser uno de los más populares en la actualidad. Contando con dicha base, procede a explicar el compilador MPLAB X IDE, así como el lenguaje ensamblador con sus rutinas típicas. Introduce luego el concepto de interrupciones en un microcontrolador, y a continuación describe en forma detallada los puertos de entrada y salida del microcontrolador PIC18F4550. Luego justifica la utilidad del lenguaje ANSI C, y lo explica mostrando en todo momento aplicaciones prácticas actuales. Cumplido lo anterior, trata el importante tema de la temporización y sus elementos. A esta altura del camino, el punto siguiente sobre la conversión analógica digital permite apreciar cómo adquirir datos a partir de un sensor y cómo convertirlas de su formato analógico original a uno digital, lo cual hace posible el desarrollo de soluciones para aplicaciones específicas que se puedan necesitar de acuerdo con la identificación de un problema real. Como el microcontrolador debe intercambiar datos con el mundo exterior, el texto completa su esquema desarrollando el módulo mejorado de comunicación serial asíncrona (EUSART), el protocolo de comunicación serial síncrono half dúplex SPI (Serial Peripheral Interface), y, por último, el módulo USB (Universal Serial Bus). A lo largo de cada capítulo, es de destacar la exposición clara y pedagógica, apoyada en la utilización de videos y en preguntas de repaso que, sin duda, distinguen este libro de otros disponibles en la literatura, y que además posibilita y pone en práctica uno de los paradigmas del modelo educativo de la Universidad Peruana de Ciencias Aplicadas (UPC): el de adquirir nuevos conocimientos haciendo las cosas por uno mismo.

El libro está dirigido a estudiantes de pregrado, y es directamente aplicable al curso Sistemas Embebidos de la UPC, ya sea de las carreras de Ingeniería Electrónica, Ingeniería de Telecomunicaciones y Redes, e Ingeniería Mecatrónica. Pero también genera interés en carreras afines, así como en ingenieros que deseen introducirse o actualizarse en este importante tema que, dicho sea de paso, nos encamina al fascinante mundo de internet de las cosas, donde tendremos cada vez más aplicaciones y soluciones para todo tipo de problemas en todos los escenarios que podamos imaginar.

Sin duda alguna, esta obra constituye un valioso aporte para el diseño y la implementación de aplicaciones innovadoras en todos los proyectos de ingeniería que incluyan sistemas embebidos, y contribuye de manera eficaz al desarrollo tecnológico propio que tanto se requiere para la mejora del bienestar y de la competitividad del país.

Carlos R. Valdez Velásquez-López, Dr. Eng.
Director de la Escuela de Ingeniería Electrónica
Universidad Peruana de Ciencias Aplicadas (UPC)

Introducción

En la enseñanza de sistemas embebidos, tanto para estudiantes universitarios como para alumnos a nivel técnico, no se encuentran materiales bibliográficos que involucren temas de arquitectura de sistemas embebidos, lenguajes de programación en ensamblador y ANSI C, que expliquen el funcionamiento y las características de los módulos periféricos de un microcontrolador y que muestren aplicaciones reales basadas en estos dispositivos. Todos estos temas se encuentran de forma separada y en su mayoría, la bibliografía está en idioma inglés. Por esta razón, se decidió tomar todo el material y la experiencia utilizada para la enseñanza de un curso sobre sistemas embebidos y volcarlos en un libro de texto que presente esta información de forma ordenada y simple.

Este libro se ha desarrollado pensando en estudiantes que se introducen al mundo de los sistemas embebidos por primera vez, o que cuentan también con cierta experiencia y desean profundizar conceptos para conocer nuevos ejemplos de aplicaciones con microcontroladores. De la misma manera, el libro está orientado a ingenieros que tengan cierta experiencia en el desarrollo de *hardware* digital o a docentes en el campo de sistemas embebidos que deseen ingresar al mundo de los microcontroladores de la familia PIC18F de Microchip.

Se ha tomado como base para el desarrollo del libro el curso de microcontroladores que se ofrece en las carreras de Ingeniería Electrónica, Ingeniería de Telecomunicaciones y Redes e Ingeniería Mecánica de la Facultad de Ingeniería de la Universidad Peruana de Ciencias Aplicadas de Lima, Perú.

Para mostrar ejemplos aplicados se utiliza el PIC18F4550, ya que es uno de los microcontroladores más populares de la empresa Microchip, se encuentra fácilmente en el mercado nacional a un costo bastante accesible y permite desarrollar un sinnúmero de aplicaciones prácticas para soluciones de problemas reales, ya sea en la industria electrónica de consumo o electrónica médica, entre otras. Todos los ejemplos mostrados en el libro se basan en el microcontrolador propuesto. Sin embargo, la gran mayoría de estos se pueden migrar fácilmente a cualquier otro modelo de microcontrolador de la misma familia, ya que los lenguajes de programación utilizados y la arquitectura son compatibles.

Adicionalmente, el PIC18F4550 es uno de los pocos microcontroladores de la familia PIC18F que cuenta con un módulo USB interno, el cual permite que sea conectado al computador a través de este protocolo a velocidades de 1.5Mbps y 12Mbps, lo que abre las puertas a una serie de aplicaciones relacionadas a interfaces de comunicación con el computador. Es por esta razón que el libro contiene un capítulo dedicado a este tema tan interesante, además de un ejemplo real.

El presente libro pretende ser lo más claro y explicativo posible, a fin de que el lector pueda obtener el máximo provecho. De esta manera, en el aula los alumnos tendrán un mejor entendimiento sobre los temas a tratar en el curso sobre sistemas embebidos, y podrán convertir una sesión de clase explicativa en un debate entre el profesor y los estudiantes, lo cual enriquecerá el nivel de los proyectos propuestos que se puedan producir como resultado de la asignatura.

Cada capítulo del libro trata un tema diferente, pero a la vez necesario para poder entender los capítulos siguientes. Los ejemplos que presentan están listos para que el lector los pueda utilizar directamente en la programación del sistema embebido. Todos estos programas han sido probados en módulos de desarrollo y su funcionamiento se encuentra completamente verificado. El contenido de cada capítulo ha sido dividido de tal manera que el lector primero aprenda a programar en lenguaje ensamblador, entendiendo la arquitectura del microcontrolador y sus modos de acceso a memoria y uso de periféricos de entrada y salida. La segunda parte explica el lenguaje ANSI C, con lo cual algunos ejemplos suben en complejidad, en virtud de la facilidad que conlleva el uso de un lenguaje de alto nivel.

En el primer capítulo se presentan aspectos básicos de los sistemas de numeración binaria, conversión entre bases, las operaciones aritméticas y lógicas en el sistema binario y la representación de cifras decimales, BCD y ASCII. Luego, se explica claramente el concepto de sistema embebido y las herramientas que ofrece Microchip para poder programar y depurar los microcontroladores de la familia PIC18F y otros más. Finalmente, se analizan los elementos más importantes de esta familia y se detallan las características particulares del microcontrolador modelo de este libro: El PIC18F4550.

Posteriormente, el segundo capítulo muestra los detalles de la arquitectura del PIC18F. Se presentan las características del microprocesador, sus registros internos y unidad aritmética lógica. Luego, se explican las distintas unidades de memoria que existen en la familia de microcontroladores PIC18F y las interfaces de entrada y salida, las opciones de Reset, los módulos de temporización, las características del funcionamiento del reloj y los modos de bajo consumo.

En el tercer capítulo se muestra la herramienta de desarrollo: El MPLAB X IDE. Se explica con ejemplos cómo crear un nuevo proyecto, editar un programa, simularlo y grabarlo en la memoria FLASH del microcontrolador. Primero se realiza la demostración con el lenguaje ensamblador y luego se hace lo propio con el lenguaje ANSI C. Esta metodología deberá ser usada por el lector para la verificación del resto de ejemplos presentados en el libro.

El cuarto capítulo detalla las instrucciones del lenguaje ensamblador del PIC18F. Cada instrucción se presenta con un ejemplo que ayuda a entender su forma de operación. A continuación se presentan las directivas del lenguaje ensamblador, las cuales también han sido ejemplificadas, con el objetivo de mostrar claramente su función.

En el quinto capítulo se explica la metodología para plantear un programa en lenguaje ensamblador. Se plantean rutinas básicas que son útiles para la mayoría de aplicaciones, como retardo de tiempo, conversión de bases y operaciones matemáticas. Finalmente, se muestra una técnica para detectar las teclas pulsadas de un teclado matricial conectado al microcontrolador a través de uno de sus puertos.

Seguidamente, el sexto capítulo aborda el tema de las interrupciones. Se explica el concepto y luego se analiza la estructura de las interrupciones ofrecidas por el PIC18F. Luego se muestra un ejemplo sencillo en el que se configurará la interrupción externa 0 del PIC18F4550 para detectar flancos generados por un pulsador.

En el séptimo capítulo se presentan los puertos de entrada y salida del PIC18F4550. Mediante estos conceptos, se muestran ejemplos de configuración y control de un teclado matricial y un módulo LCD alfanumérico.

Luego, en el octavo capítulo se aborda la sintaxis completa del lenguaje ANSI C. Se explican sus instrucciones con ejemplos sencillos y se muestra la forma de creación de librerías haciendo uso del compilador XC8. Se mencionan temas más específicos, como el uso de ensamblador dentro del ANSI C, el manejo de interrupciones y el uso de directivas del preprocesador. Finalmente, se muestran programas de ejemplo, aplicados haciendo uso del lenguaje C y de los periféricos del microcontrolador.

En el noveno capítulo se emprende el tema de los módulos de temporización del microcontrolador. Aquí se presenta la teoría y el mecanismo de funcionamiento de un temporizador, además de diversos ejemplos de funcionamiento. Posteriormente, se explican los modos de Entrada de captura, comparación de salida y generador de PWM, los cuales se ejemplificarán con aplicaciones reales que requieren el uso de servomotores y sensores infrarrojos para su puesta a prueba.

El décimo capítulo toca el tema del convertidor analógico digital del PIC18F. Se presenta la teoría básica de digitalización de señales analógicas y algunos aspectos importantes, como el teorema del muestreo. Luego, se explica el módulo ADC del PIC18F4550 y el método de configuración. Finalmente, se muestran dos ejemplos aplicados para medir el voltaje de forma digital y la temperatura ambiente haciendo uso de un sensor LM35.

Más adelante, en el decimoprimer capítulo se aborda el tema de la comunicación serial asíncrona y el estándar RS232. Se analiza el módulo EUSART del PIC18F y su forma de configuración. Luego, se muestran ejemplos aplicados para obtener una comunicación entre el PIC18F4550 y un computador a través del puerto RS232 y otra aplicación médica muy interesante: el diseño de un oxímetro de pulso para medir el nivel de saturación de oxígeno y pulso cardíaco de un paciente.

En el decimosegundo capítulo se presenta la interfaz serial periférica (SPI), la cual corresponde a uno de los módulos de comunicación síncrona del microcontrolador bastante utilizada por muchos dispositivos periféricos en estos días. Se explica el módulo MSSP del microcontrolador y los registros de configuración. Luego, se muestran dos ejemplos de aplicación: un reloj en tiempo real haciendo uso del circuito integrado DS1305 y el diseño de una matriz de LED basada en el circuito integrado MAX6952.

Posteriormente, el decimotercer capítulo trata de la interfaz serial síncrona I²C. Se explican las características del protocolo y los métodos de intercambio de información entre dos o más dispositivos conectados mediante esta interfaz. Luego, se explica el módulo MSSP del microcontrolador configurado en modo I²C y el método de configuración. Se analiza la librería “i2c” del MPLAB XC8 y se muestran dos ejemplos aplicados: uno para obtener la temperatura de un sensor digital como el DS1621 y el acceso de lectura y escritura sobre una memoria EEPROM modelo 24LC08.

El último capítulo toca el tema de la interfaz USB. Se explican de forma general los fundamentos del bus, los métodos de transferencia, el proceso de enumeración y las clases de dispositivo. Luego, se analizan las características más relevantes del módulo USB del PIC18F4550 y se presenta la librería MLA de Microchip, que permite manipular el módulo USB del microcontrolador a través de funciones amigables que manejan los detalles de la configuración, facilitando la elaboración de aplicaciones. Al finalizar el capítulo, se muestra un ejemplo de comunicación entre el PIC18F4550 y el computador a través del bus USB usando la clase CDC.

Capítulo 1. Conceptos básicos

Para la correcta comprensión del contenido del presente capítulo y de los capítulos siguientes, es necesario tener en claro ciertos conceptos básicos acerca de los sistemas de numeración utilizados en la electrónica digital (básicamente, el sistema binario y el hexadecimal), los tipos de operaciones matemáticas relacionadas con estas bases numéricas y las diversas interpretaciones que se pueden obtener de las cifras binarias.

1.1 Definiciones básicas

A continuación se presenta una serie de conceptos importantes sobre electrónica digital; los tipos de base numérica binaria, decimal y hexadecimal; los métodos aritméticos para obtener los cambios de base, operaciones lógicas y aritméticas en estos sistemas numéricos; y los formatos de numeración BCD y ASCII, que son sumamente utilizados en las aplicaciones con sistemas embebidos.

1.1.1 El sistema de numeración binario y hexadecimal

En el mundo de los dispositivos lógicos digitales, las operaciones realizadas siempre involucran patrones numéricos en formato binario. La unidad mínima de este formato es el bit. Un bit se representa por dos niveles o estados lógicos: '1' o '0'. Normalmente, un nivel '1' lógico viene a estar físicamente representado por una magnitud en voltios, y el nivel '0', en otra magnitud de voltaje. Por ejemplo, en la tecnología TTL¹, el nivel lógico '1' se encuentra determinado en el rango entre 2 V y 5.5 V, mientras que el nivel '0' está entre 0 V y 0.8 V. Cualquier nivel de voltaje fuera de este rango (entre 0.8 y 2 V) es considerado como un estado indeterminado o 'X', con el cual será imposible realizar operaciones lógicas a nivel digital.

Las cifras binarias, por lo general, no se componen de un solo bit. Normalmente, están compuestas por una cantidad múltiplo de ocho bits. En el gráfico 1.1 se muestran los tipos de arreglos binarios más utilizados en la electrónica digital.

1 *Transistor Transistor Logic*. Tecnología digital basada en transistores bipolares.

Gráfico 1.1. Arreglos binarios más comunes usados en la electrónica digital

Tamaño	Valor mínimo y máximo	Denominación
1 bit	0 – 1	Bit
4 bits	0000 – 1111	Nibble
8 bits	0000 0000 – 1111 1111	Byte
16 bits	0000 0000 0000 0000 – 1111 1111 1111 1111	Word
32 bits	0000 0000 0000 0000 0000 0000 0000 0000 – 1111 1111 1111 1111 1111 1111 1111 1111	Double Word

Elaboración propia.

Una cifra binaria posee la siguiente nomenclatura: (código binario)₂, donde el subíndice 2 indica la base. Por ejemplo, la cifra 01010000₂ es un byte con un valor equivalente al número 80 en base decimal (base 10). La nomenclatura para la base hexadecimal se representa antecediendo un 0x sobre el código hexadecimal. Por ejemplo, la cifra mostrada se puede representar como 0x50 en la base hexadecimal. El bit ubicado en el extremo derecho de la cifra en base binaria es conocido como el bit LSB² y representa el valor menos significativo. De la misma manera, el bit del extremo izquierdo es considerado el bit MSB³ o más significativo. Al igual que en el sistema de numeración decimal, una cifra ubicada en la posición de las centenas o millares (izquierda) presenta mayor significancia que una cifra ubicada en la posición de las decenas o unidades (derecha).

En el sistema binario, la base numérica utilizada es 2. Esto significa que cada bit puede estar representado únicamente por dos cifras: '0' o '1'. La posición del bit dentro de la cifra especifica un valor formado por la cifra 2 elevado a una potencia y multiplicado por el valor del bit ('0' o '1'). El valor de dicha potencia es la posición del bit en la cifra. Por ejemplo, si se observa el gráfico 1.2, en el byte de la figura, el bit LSB tiene un aporte de 2⁰ (ya que 0 es la posición menos significativa), mientras que el bit MSB aporta con 2⁷. Esto implica que el valor de la cifra es el siguiente:

$$valor = 2^7 \times 0 + 2^6 \times 1 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0$$

Como se puede observar, el valor obtenido es 64+16 = 80, cifra equivalente en base decimal. Conforme se agreguen más dígitos binarios a esta cifra, el exponente de la base binaria se incrementará en uno. Por ejemplo, si una cifra binaria posee 11 bits, el bit MSB tendrá un peso equivalente de 2¹⁰.

Al igual que en el caso de las cifras decimales, donde existen prefijos como kilo (10³), Mega (10⁶) o Giga (10⁹), entre otros que son usados para abreviar magnitudes elevadas, existen también los mismos prefijos para definir grandes magnitudes en cifras binarias formadas por longitudes con base en 8 bits (1 byte). Es así que para el sistema binario:

2 El bit menos significativo. Proviene de las siglas del idioma inglés *Least Significant Bit*.

3 El bit más significativo. Proviene de las siglas del idioma inglés *Most Significant Bit*.

- 1 kB (Se pronuncia un kilo byte) = 2^{10} bytes o 1024 bytes
- 1 MB (Se pronuncia un mega byte) = 2^{10} kB o 1024 kB
- 1 GB (Se pronuncia un giga byte) = 2^{10} MB o 1024 MB

Gráfico 1.2. Ejemplo sobre los pesos de cada bit de una cifra binaria

Elaboración propia.

Al igual que en el sistema de numeración binario, se emplean otros sistemas numéricos para representar cifras en otras bases. Uno de los más usados en la programación de sistemas embebidos es la base hexadecimal⁴. En ella cada cifra tiene un peso, dependiendo de una potencia en base 16. En este sistema existen 16 cifras diferentes (valores de 0 a 15) para representar una magnitud. Por ejemplo, para las cifras del 0 al 9 se utilizan los mismos dígitos que en el sistema decimal. Para completar los 6 dígitos restantes se utilizan letras del alfabeto, como el siguiente caso:

0xA	=	10	=	1010_2
0xB	=	11	=	1011_2
0xC	=	12	=	1100_2
0xD	=	13	=	1101_2
0xE	=	14	=	1110_2
0xF	=	15	=	1111_2

En el número hexadecimal mostrado en el gráfico 1.3, cada cifra representa una *nibble*, dado que se tienen 16 posibles valores diferentes y esto equivale a una combinación de 4 bits (2^4 estados). Cada dígito representa una potencia de 16, según su posición en la cifra. En el ejemplo, el valor decimal equivalente del número hexadecimal mostrado es el siguiente:

⁴ Observar que en esta publicación para la notación hexadecimal se utiliza el símbolo «0x» (cero equis). Es distinto del símbolo «por» utilizado la multiplicación.

$$valor = 16^3 \times 10 + 16^2 \times 3 + 16^1 \times 4 + 16^0 \times 12$$

De esta manera, el valor decimal equivalente es $40\,960 + 768 + 64 + 12 = 41\,804$. Como se puede observar, esta cifra pudo haber sido representada en el sistema binario con el siguiente código: 1010001101001100_2 , lo cual habría resultado en una cifra más larga, al estar conformada por 16 dígitos. En cambio, en el sistema hexadecimal se utilizan únicamente 4 dígitos para representar la misma cifra numérica, lo cual permite abreviar valores de magnitudes muy extensas.

El sistema de numeración hexadecimal es el preferido cuando se quiere representar magnitudes superiores a 2^8 , y, al mismo tiempo, se desea mantener la visualización del código binario. Esto permite obtener una mejor comprensión del orden de la cifra sin perder de vista su forma a nivel binario (ya que es fácilmente visible el código binario a partir de cada *nibble* o dígito hexadecimal).

Gráfico 1.3. Ejemplo sobre los pesos de cada cifra hexadecimal

Elaboración propia.

1.1.2 Conversiones entre bases

La base decimal es el formato más utilizado por los seres humanos para interpretar las cifras numéricas que operan diariamente para sus tareas cotidianas. Las personas comunes y corrientes han aprendido a realizar operaciones aritméticas en base a este sistema de numeración. Entonces, la pregunta que cae por su propio peso es la siguiente: «¿por qué el programador debe conocer otros sistemas de numeración, como el binario y el hexadecimal?» La respuesta a esta pregunta es obvia: porque al programar un sistema embebido no se debe pensar como seres humanos sino como un computador, y el sistema de numeración utilizado por estas máquinas es, por naturaleza, el binario.

Durante la programación de un sistema embebido no es necesario que el programador realice las conversiones de bases en el código ingresado. Para el software de programación de sistemas embebidos, el uso de una base u otra es indiferente. La conversión de una base es un proceso automáticamente efectuado por el compilador en el momento de generar el código de salida del programa. Sin embargo, el programador debe conocer las diferentes bases para poder interpretar las cifras mentalmente y así

agilizar la implementación de su algoritmo. La base decimal es por excelencia la más sencilla de interpretar por cualquier ser humano. Por tal motivo, se toma como referencia en las conversiones de base.

Convertir una cifra de base binaria a decimal es bastante simple y ya se explicó en el subcapítulo 1.1.1. Solo basta con multiplicar cada dígito binario ('0' o '1') por su correspondiente peso (base dos elevado a la posición del dígito) y sumar todos los dígitos resultantes. Por ejemplo, si se tiene la cifra binaria 10101010_2 , la transformación a la base decimal se haría de la siguiente manera:

$$\text{valor decimal} = 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 1 + 2^4 \times 0 + 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0$$

De tal modo, el valor decimal equivalente obtenido es $128 + 32 + 8 + 2 = 170$.

Para realizar la conversión de una cifra binaria a hexadecimal, el procedimiento también resulta sumamente sencillo. Si se toma como ejemplo el valor binario 010111000011_2 , primero se deberá separar la cifra en el número total de *nibbles* que lo componen. Para este caso son tres: 0101_2 , 1100_2 y 0011_2 . Cada *nibble* posee su equivalente en cifra hexadecimal. Por ejemplo, 0101_2 es $0x5$, 1100_2 es $0xC$ (doce) y 0011_2 es $0x3$. Por lo tanto, la cifra hexadecimal equivalente es $0x5C3$.

El proceso de conversión de decimal a binario es ligeramente más complejo que en los casos anteriores. Consiste en realizar divisiones sucesivas entre 2 hasta lograr que el cociente final de la división sea 1. Si se desea convertir el valor decimal 17 a código binario, se deben realizar las siguientes operaciones sucesivas:

17	÷	2	= 8,	Residuo	= 1 (bit LSB, primer dígito)
8	÷	2	= 4,	Residuo	= 0 (segundo dígito)
4	÷	2	= 2,	Residuo	= 0 (tercer dígito)
2	÷	2	= 1 (bit MSB, quinto dígito)	Residuo	= 0 (bit LSB, cuarto dígito)

La cifra binaria equivalente se forma al tomar el último cociente como bit MSB y los residuos restantes como los dígitos ordenados de izquierda a derecha. La cifra binaria, entonces, estará compuesta de 5 dígitos y será el valor 10001_2 .

Si se repite el ejercicio con una cifra decimal par, se podrá notar que el bit LSB siempre será '0', ya que la primera división entre 2 de un número par dará como residuo 0. El dígito MSB siempre será '1' para cualquier cifra. Por ejemplo, para convertir el número 18 a base decimal se tiene la siguiente operación:

18	÷	2	= 9,	Residuo	= 0 (bit LSB, primer dígito)
9	÷	2	= 4,	Residuo	= 1 (segundo dígito)
4	÷	2	= 2,	Residuo	= 0 (tercer dígito)
2	÷	2	= 1 (bit MSB, quinto dígito)	Residuo	= 0 (bit LSB, cuarto dígito)

Como resultado se obtiene el código binario 10010_2 , del cual se puede observar que el bit LSB es '0'.

La conversión de base hexadecimal a decimal es muy parecida a la de binario a decimal, con la diferencia de que cada elemento tiene asociada una potencia en base 16. Por ejemplo, se tiene un número hexadecimal cuyo valor es 0xFA28. Para convertirlo a su equivalente en base decimal, se debe observar que el número en base 16 estará compuesto por 4 cifras:

0xF = 15 (*nibble* más significativo),
0xA = 10,
0x2 = 2 y
0x8 = 8 (*nibble* menos significativo)

La cifra 0xF es la de mayor significancia (MS⁵), por lo cual le corresponde un peso de 16³, mientras que la cifra 0x8 es la menos significativa (LS⁶) y, por lo tanto, le corresponde el menor peso, de 16⁰. Posteriormente, se multiplica cada cifra por su respectiva potencia de 16 según el orden:

$$valor = 16^3 \times 15 + 16^2 \times 10 + 16^1 \times 2 + 16^0 \times 8$$

De esta manera, el valor decimal da como resultado 61 440 + 2560 + 32 + 8 = 64 040. Cada cifra es múltiplo de 16; sin embargo, es posible expresar cualquier magnitud entera en base hexadecimal.

Para pasar de una base decimal a hexadecimal el procedimiento también resulta ser similar al caso de la conversión decimal a binaria. La diferencia radica en que las divisiones sucesivas se realizan sobre un divisor de 16. Suponiendo que se tiene el número 39 481 en base decimal y se desea transformar a base hexadecimal, el procedimiento sería el siguiente:

39 481	÷	16	= 2467.	Residuo	= 9	(cifra LS, primer dígito)
2467	÷	16	= 154.	Residuo	= 3	(segundo dígito)
154	÷	16	= 9 (cifra MS, cuarto dígito)	Residuo	= 10	(tercer dígito)

El resultado de la conversión de 39 481 a hexadecimal es, entonces, 0x9A39. La división sucesiva deberá detenerse cuando el último cociente obtenido sea un valor menor a 15 (0xF).

Un proceso mucho más sencillo y que no requiere de cálculos aritméticos es la conversión binaria a hexadecimal. Como ejemplo, se cuenta con el número binario 1110011011000010₂ y se desea transformar a base hexadecimal. El primer paso de este proceso es dividir el número binario en grupos de cuatro bits o *nibbles*. Si no se cuenta con un número de bits múltiplo de cuatro, se deberán agregar varios '0' a la izquierda para completar los dígitos. Entonces, el número binario se divide de la siguiente manera:

Nibble MS: 1110₂ = 0xE
Segundo *nibble*: 0110₂ = 0x6
Tercer *nibble*: 1100₂ = 0xC
Nibble LS: 0010₂ = 0x2

5 *Most Significant* (más significativo).
6 *Least Significant* (menos significativo).

Por tanto, la cifra hexadecimal equivalente es 0xE6C2.

La conversión hexadecimal a binario es un proceso muy similar al anterior. Simplemente se deberá tomar cada cifra del número hexadecimal, para convertirla al *nibble* correspondiente en código binario. Por ejemplo, se tiene el número hexadecimal 0xABCD2 y se desea convertirlo a binario. El procedimiento es como se muestra:

<i>Nibble</i> MS:	0xA = 1010 ₂
Segundo <i>Nibble</i> :	0xB = 1011 ₂
Tercer <i>Nibble</i> :	0xC = 1100 ₂
Cuarto <i>Nibble</i> :	0xD = 1101 ₂
<i>Nibble</i> LS:	0x2 = 0010 ₂

De esta manera, el resultado de la conversión es 10101011110011010010₂. Se puede observar que no es conveniente expresar una cifra de esta magnitud en base binaria, ya que resulta muy larga y difícil de interpretar. Durante la programación de sistemas embebidos, el programador es libre de utilizar cualquiera de las tres bases vistas (base binaria, decimal y hexadecimal). Sin embargo, muchos programadores optan por utilizar la combinación de las tres bases en sus programas, ya que, para ciertas circunstancias, algunas bases se adecúan mejor que otras. Por lo general, un programador opta por la base decimal cuando realiza operaciones matemáticas. La base hexadecimal se utiliza para manejar direcciones de memoria (que usualmente son magnitudes elevadas) o cargar códigos binarios a registros, y la base binaria suele ser utilizada para la asignación de estados a los puertos de entrada y salida (E/S) del sistema embebido.

Si bien es cierto que existen otras bases, como la octal (base 8), estas son poco utilizadas, debido a que no son versátiles y no resultan atractivas para la mayoría de programadores (al parecer, conocer tres bases numéricas ya es suficiente). No obstante, la mayoría de compiladores siguen aceptando este tipo de base.

1.1.3 Operaciones lógicas

En el mundo digital existen dos maneras de operar las cifras binarias (o en la base en que se interpretan, pues finalmente todo número es manejado de forma binaria en un sistema embebido). Cada cifra puede ser operada en modo aritmético o en modo lógico. El modo aritmético se presentará en el siguiente subcapítulo. A continuación se muestra el modo lógico.

El modo lógico se basa en las operaciones lógicas. Estas se obtienen mediante la manipulación de los patrones binarios en base a las reglas creadas por George Boole⁷ a mediados del siglo XIX, las cuales constituyen el álgebra de Boole. El álgebra de Boole opera sobre variables que pueden ser verdaderas o falsas. Las reglas de esta álgebra son actualmente utilizadas en las operaciones lógicas binarias en los sistemas digitales, ya que cada cifra binaria puede ser tratada como verdadero (nivel binario '1') y falso (nivel binario '0')⁸.

7 Filósofo y matemático inglés, padre de la lógica booleana [1815-1864].

8 Cfr. Sánchez y Canton 2007.

Las operaciones lógicas típicas más empleadas en los sistemas embebidos son las siguientes:

- NOT
- OR
- AND
- XOR
- SWAP
- SHIFT

Cabe señalar que las operaciones lógicas son fundamentales en la programación de sistemas embebidos, ya que permiten la implementación de una serie de operaciones vitales para el funcionamiento del programa. Por ejemplo, invertir el estado de un pin de entrada y salida de un sistema embebido, configurar un registro interno, realizar operaciones de conversión binario BCD o implementar el proceso de división, entre otras. Estos son ejemplos en los cuales se requiere hacer uso intensivo de las operaciones lógicas.

La operación lógica más simple de todas es la NOT (negación). Esta operación simplemente invierte un estado verdadero a falso y viceversa. En el caso binario, negar un bit significa cambiar su estado de '1' a '0' o de '0' a '1'. Por ejemplo, se tiene una variable de nombre A que representa un bit. Por lo tanto, A puede tomar dos valores: '0' o '1'. Si se niega el valor de A, entonces $\bar{A}$ (se denomina a esta variable 'A negado'. En otras literaturas se considera A' también como 'A negado') denota la negación. El símbolo para la compuerta lógica que realiza la negación se muestra en el gráfico 1.4, junto con su tabla de verdad.

Gráfico 1.4. a) Tabla de verdad de la operación lógica NOT. b) Símbolo de la compuerta NOT

Elaboración propia.

También es posible realizar la operación lógica NOT sobre un conjunto de bits, que podrían ser un *nibble* o un byte. Si la variable A equivale al siguiente valor:

$$A = 10110011_2$$

Entonces:

$$\bar{A} = 01001100_2$$

La operación lógica OR, también conocida como OR inclusiva (IOR), es otra de las operaciones básicas del álgebra de Boole. Su lógica es muy similar a la suma, excepto cuando ambos operandos son '1'⁹. En una operación OR deben existir al menos dos operandos, y basta que uno de ellos sea '1' para que el resultado de la operación sea '1'. En el gráfico 1.5 se observa la tabla de verdad y el símbolo de la compuerta lógica OR.

La operación OR se realiza normalmente en conjuntos binarios. Por ejemplo, en el caso de que la variable $A = 10000011_2$ y la variable $B = 00011111_2$, el resultado de la operación OR sería el siguiente:

$$A \text{ OR } B = A + B = 10000011_2 + 00011111_2 = 10011111_2$$

También es posible combinar las operaciones. Por ejemplo, se puede negar la operación OR entre A y B, lo cual se conoce como la operación NOR. Esto daría el siguiente resultado para los valores dados de A y B:

$$\overline{A \text{ OR } B} = \overline{A + B} = \overline{10000011_2 + 00011111_2} = \overline{10011111_2} = 01100000_2$$

Gráfico 1.5. a) Tabla de verdad de la operación lógica OR. b) Símbolo de la compuerta OR

Elaboración propia.

En muchos sistemas embebidos, la operación NOR no forma parte del conjunto de instrucciones. Por lo cual, para su implementación, se deben ejecutar las operaciones OR y NOT de manera combinada.

La operación lógica AND toma dos variables binarias y realiza un proceso muy similar al de la multiplicación, sin tomar en cuenta el acarreo. En esta operación basta que uno de los operandos sea '0' para que todo el resultado de la operación dé '0'. En el gráfico 1.6 se muestra la tabla de verdad y el símbolo de la compuerta lógica AND.

9 Si se tiene la operación $1_2 \text{ OR } 1_2$, el resultado en una suma debería ser 10_2 . Sin embargo, para la operación lógica el resultado es 1_2 .

Gráfico 1.6. a) Tabla de verdad de la operación lógica AND. b) Símbolo de la compuerta AND

Elaboración propia.

En el caso de que la variable $A = 11001100_2$ y la variable $B = 01001011_2$, el resultado de la operación AND será el siguiente:

$$A \text{ AND } B = A.B = 11001100_2.01001011_2 = 01001000_2$$

En el resultado se puede observar que solo aquellos bits que poseen la cifra ‘1’ en la misma posición mantienen su valor ‘1’. El resto son ‘0’. También es posible agregar una tercera variable, de nombre C, a la operación. Si se tiene que $C = 00001111_2$, entonces la operación AND de las tres variables generará el siguiente resultado:

$$A \text{ AND } B \text{ AND } C = A.B.C = 11001100_2.01001011_2.00001111_2 = 00001000_2$$

También se puede combinar la operación lógica AND con la NOT para obtener un operador NAND. Este tipo de operación tampoco es muy común en el conjunto de instrucciones de sistemas embebidos con arquitectura tipo RISC¹⁰. Por ejemplo, la operación NAND de las variables A y B se muestra a continuación:

$$\overline{A \text{ AND } B} = \overline{A.B} = \overline{11001100_2.01001011_2} = \overline{01001000_2} = 10110111_2$$

Hasta el momento se han analizado las tres operaciones lógicas fundamentales. Si se combinan estos tres operadores, se logrará obtener una nueva operación lógica que se encuentra disponible en la arquitectura de la gran mayoría de sistemas embebidos: el operador XOR (OR exclusivo). Este operador genera un resultado verdadero solo si una de las entradas es verdadera y la otra es falsa. Pero si ambas entradas a la operación XOR son iguales, el resultado será falso.

10 Conjunto de instrucciones reducido, de las siglas en inglés *Reduced Instruction Set Computing*. Es un tipo de arquitectura de sistemas embebidos que se explicará en el segundo capítulo.

Si A y B son dos variables binarias, entonces la operación XOR se denota de la siguiente manera:

$$A \text{ XOR } B = A \oplus B = A.\bar{B} + \bar{A}.B$$

En el gráfico 1.7 se muestra la tabla de verdad de la operación lógica XOR y el símbolo de su compuerta.

La operación XOR es muy utilizada para verificar si dos números son iguales. Por ejemplo, la variable $A = 10001110_2$ y la variable $B = 10001110_2$. Ambas variables tienen asignados el mismo código binario. Entonces, la operación XOR dará el siguiente resultado:

$$A \oplus B = 10001110_2 \oplus 10001110_2 = 00000000_2$$

Gráfico 1.7. a) Tabla de verdad de la operación lógica AND. b) Símbolo de la compuerta XOR

a)			b)	
A	B	$A \oplus B$		
0	0	0
0	1	1		
1	0	1		
1	1	0		

Elaboración propia.

Como se puede observar, al ser todos los bits iguales en cada posición, el resultado de la operación XOR da 00000000_2 . Este resultado será el mismo cada vez que los operandos sean iguales. Si, en cambio, se altera un bit de la variable B y su nuevo valor fuera $B = 10011110_2$, entonces el resultado de la operación XOR sería el siguiente:

$$A \oplus B = 10001110_2 \oplus 10011110_2 = 00010000_2$$

El valor obtenido es diferente de 00000000_2 , lo cual será siempre así si es que ambos operandos no son iguales.

La operación SWAP no posee un símbolo lógico. Si bien es cierto, no es una operación lógica propiamente dicha sino el resultado de la combinación de operaciones básicas, muchos sistemas embebidos utilizan una instrucción de nombre *swap* para alterar el contenido de un byte de información.

En la operación *swap*, los *nibbles* de un byte son intercambiados de posición. Con una variable $A = 11000101_2$, se observa que el *nibble* más significativo es 1100_2 , mientras que el *nibble* menos significativo es 0101_2 . Si se realiza la operación *swap* sobre A, el nuevo resultado será $A = 01011100_2$. Como se puede constatar, simplemente se han intercambiado de posición los *nibbles* del byte.

Otra de las operaciones clásicas realizadas por todo sistema embebido es la operación *shift* o de desplazamiento binario. En una cifra binaria, cada bit posee una posición. Con el operador *shift* es posible realizar un cambio de posición de cada bit, ya sea para la derecha o para la izquierda.

Nuevamente, la operación *shift* no es una compuerta, sino un conjunto de operaciones que involucran el movimiento de los bits en un registro. Esta operación está incluida en el conjunto de instrucciones de todo sistema embebido y resulta muy útil para una gama de aplicaciones en las cuales se requiere multiplicar, dividir o convertir de binario a BCD, entre otras.

En la operación *shift* se utilizan los símbolos $\gg$ o $\ll$, para denotar un corrimiento binario hacia la derecha o hacia la izquierda, respectivamente. Por ejemplo, si se tiene que la variable $A = 10011000_2$, entonces, al realizar la operación $A \gg 3$, los bits de la variable A se desplazarán tres posiciones a la derecha, quedando de la siguiente manera: $A = 00010011_2$. Al realizar esta operación han aparecido tres bits de valor cero ($b'000$) al lado izquierdo del bit que solía ser el MSB de la cifra. Estas tres posiciones en realidad son reemplazadas por bits fuera de las ocho posiciones del byte. Por defecto, cuando se realiza el corrimiento, estos nuevos valores son siempre ceros.

Ahora, en el caso de que la variable $A = 00010011_2$ sufre un corrimiento a la izquierda con la siguiente operación: $A \ll 4$, el resultado quedaría de la siguiente forma: $A = 0011000_2$. En este caso se puede observar que uno de los bits '1' de la variable A ha desaparecido (el bit en la posición 4). Esto en realidad no ha ocurrido, sino que este bit ha tomado la posición 8 o se ha convertido en el noveno bit, el cual no se muestra, ya que el valor a tratar es un byte y solo tiene 8 posiciones (la novena posición, verdaderamente, solo estará en la mente del programador).

1.1.4 Operaciones aritméticas

Otra de las formas que existen para manipular los números binarios a nivel digital es a través de operaciones aritméticas. Las operaciones aritméticas típicas que involucran a los sistemas embebidos de mediano rango son las siguientes:

- Suma
- Resta
- Multiplicación

Las reglas de la aritmética que se conocen para el sistema decimal son muy similares al sistema binario. La diferencia es que el sistema binario es más simple y las reglas que hay que aprender para efectuar una suma, por ejemplo, son más sencillas que en el caso de la base decimal. En general, para llevar a cabo una sumatoria en el sistema binario se deben tener en cuenta únicamente tres reglas:

$$\left. \begin{array}{l} 0_2 + 0_2 = 0_2 \\ 0_2 + 1_2 = 1_2 \\ 1_2 + 0_2 = 1_2 \\ 1_2 + 1_2 = 10_2 \text{ (0 y un bit de acarreo)} \end{array} \right\} \text{ La suma aritmética es conmutativa.}$$

El acarreo es una parte fundamental en la suma. Por ejemplo, en la suma decimal es común que se produzca un acarreo cuando la sumatoria supera el valor de nueve. Se empieza sumando las cifras menos

significativas y se termina cuando la última suma involucra a las dos cifras más significativas de los operandos (o a solo uno de ellas). Si se analiza la siguiente suma decimal: $137 + 26$, sucederá lo siguiente:

1. $7 + 6 = 3$ (acarreo1 = 1)
2. $(\text{acarreo1}) + 3 + 2 = 6$ (acarreo2 = 0)
3. $(\text{acarreo2}) + 1 = 1$

Resultado = 163

En general, para la base decimal el acarreo es un número entre 0 y 1. Para el caso de la base binaria, el acarreo también será una cifra entre 0_2 y 1_2 , ya que la suma de dos dígitos binarios solo puede dar '0', '1' o 10_2 . En el caso de que se quiera realizar la suma de los siguientes dígitos binarios: 01101100_2 y 10011010_2 , primero, se comienza sumando las cifras menos significativas, de la siguiente manera:

1. $'0' + '0' = '0'$ (acarreo1 = '0')
2. $(\text{acarreo1}) + '0' + '1' = '1'$ (acarreo2 = '0')
3. $(\text{acarreo2}) + '1' + '0' = '1'$ (acarreo3 = '0')
4. $(\text{acarreo3}) + '1' + '1' = '0'$ (acarreo4 = '1')
5. $(\text{acarreo4}) + '0' + '1' = '0'$ (acarreo5 = '1')
6. $(\text{acarreo5}) + '1' + '0' = '0'$ (acarreo6 = '1')
7. $(\text{acarreo6}) + '1' + '0' = '0'$ (acarreo7 = '1')
8. $(\text{acarreo7}) + '0' + '1' = 10_2$

Resultado = 100000110_2

Como se puede observar, el resultado posee un bit más (9 bits en total) que los dos operandos. Esto se debe al acarreo que se ha ido arrastrando desde la cuarta operación. En el ejemplo anterior, el primer operando 01101100_2 representa la cifra decimal 108, y el segundo operando 10011010_2 representa la cifra decimal 154. El resultado es 100000110_2 , cuyo valor decimal es 262, y corresponde a la suma de $108 + 154$.

Uno de los formatos utilizados por los sistemas embebidos para poder operar números positivos y negativos es el complemento a 2. Las cifras binarias pueden interpretarse de dos formas: positivas o negativas. Por ejemplo, la cifra 10010011_2 puede interpretarse de dos maneras: como 147 en base decimal o -109 (complemento a 2).

En el formato complemento a 2, una cifra se puede interpretar como positiva o negativa. En este formato, el bit MSB representa el signo. Si este bit es '1', entonces el número tendrá una representación negativa. Para obtener el valor negativo de una determinada magnitud en código binario, se debe obtener su complemento a 2. Esto se logra realizando los siguientes pasos:

1. Escribir la cifra binaria.
2. Realizar la operación NOT sobre toda la cifra.
3. Sumar el bit 1_2 a la cifra negada.
4. Si la cifra inicial representa una magnitud positiva, el resultado será la representación negativa en complemento a 2. Si la cifra inicial es una magnitud negativa, el resultado será la representación positiva en complemento a 2.

De esta manera, para determinar el valor representado por la cifra 10010011_2 en complemento a 2, el primer paso consiste en invertirla y transformarla en el siguiente valor negado: 01101100_2 . La siguiente fase consiste en sumar el bit '1', con lo cual cambia al siguiente estado: 01101101_2 . Finalmente, se interpreta esta cifra en magnitud positiva, cuyo valor es $64 + 32 + 8 + 4 + 1 = 109$. Entonces, el valor representado por 10010011_2 en complemento a 2 es -109 .

Las mismas reglas que se emplean para la adición de números sin signo aplican para los números complemento a 2. Por ejemplo, si se suma $-109 + 100$, el resultado es -9 . Esto, en binario, sería equivalente a obtener la suma de $10010011_2 + 01100100_2 = 11110111_2$. Dicho valor en complemento a 2 es 00001001_2 , que equivale a la cifra 9 decimal, lo cual implica que la suma debe interpretarse como -9 .

Las reglas de adición pueden combinar cifras sin signo y complemento a 2 sin ninguna distinción. Por ejemplo, en la siguiente sumatoria: $(-9) + 8 + 1 + (-4)$, el resultado es el complemento a 2 de 4. En cifras binarias sería lo siguiente:

$$11110111_2 (-9) + 00001000_2 (8) + 00000001_2 (1) + 11111100_2 (-4)$$

El procedimiento consiste en sumar cada bit empezando por los bits LSB de cada una de las cuatro cifras:

1. $1_2 + 0_2 + 1_2 + 0_2 = 0_2$ (acarreo1 = 1_2)
2. $(\text{acarreo1}) + 1_2 + 0_2 + 0_2 + 0_2 = 0_2$ (acarreo2 = 1_2)
3. $(\text{acarreo2}) + 1_2 + 0_2 + 0_2 + 1_2 = 1_2$ (acarreo3 = 1_2)
4. $(\text{acarreo3}) + 0_2 + 1_2 + 0_2 + 1_2 = 1_2$ (acarreo4 = 1_2)
5. $(\text{acarreo4}) + 1_2 + 0_2 + 0_2 + 1_2 = 1_2$ (acarreo5 = 1_2)
6. $(\text{acarreo5}) + 1_2 + 0_2 + 0_2 + 1_2 = 1_2$ (acarreo6 = 1_2)
7. $(\text{acarreo6}) + 1_2 + 0_2 + 0_2 + 1_2 = 1_2$ (acarreo7 = 1_2)
8. $(\text{acarreo7}) + 1_2 + 0_2 + 0_2 + 1_2 = 11_2$

$$\text{Resultado} = 111111100_2$$

Se puede observar que el resultado presenta 9 bits de longitud. Sin embargo, los bits más significativos son 1_2 , lo cual se puede interpretar como ceros a la izquierda o cifras no significantes para una magnitud complemento a 2. Si se niega Resultado y se suma 1, el resultado interpretado será 000000100_2 o -4 , lo cual es equivalente al resultado obtenido en base decimal.

Las técnicas utilizadas para restar números en base decimal también son muy similares a las que se aplican para la resta en base binaria. El proceso comienza restando los bits LSB hasta llegar a los bits MSB. En cualquier resta binaria en la que una cantidad mayor debe ser restada de una menor, se presentará un dígito de acarreo, que en este caso es prestado del bit de la posición izquierda. De esta manera, las reglas básicas de sustracción binaria son las siguientes:

$$\left. \begin{array}{l} '0' - '0' = '0' \\ '1' - '1' = '0' \\ '1' - '0' = '1' \\ '0' - '1' = '1' \end{array} \right\} \text{ Se presta un bit '1' del bit más significativo inmediato.}$$

Por ejemplo, si se desea restar $56 - 37$ en decimal, se debe realizar el siguiente procedimiento:

1. $6 - 7 = 9$ (se presta 1 del dígito más significativo, por lo cual se acarrea 1: acarreo1 = 1)
2. $5 - (\text{acarreo1}) - 3 = 1$

Resultado = 19

Si luego estas cifras se transforman a base binaria, en donde $56 = 00111000_2$ y $37 = 00100101_2$, la resta se realizará empezando desde los bits LSB hasta los bits MSB, de la siguiente manera:

1. $^10' - ^11' = ^11'$ (se presta '1' del siguiente bit más significativo, acarreo1 = '1')
2. $^10' - ^10' - \text{acarreo1} = ^11'$ (se presta '1' del siguiente bit más significativo, acarreo2 = '1')
3. $^10' - ^11' - \text{acarreo2} = ^10'$ (se presta '1' del siguiente bit más significativo, acarreo3 = '1')
4. $^11' - ^10' - \text{acarreo3} = ^10'$ (No se presta ya que el minuendo es mayor o igual al sustraendo, acarreo4 = '0')
5. $^11' - ^10' - \text{acarreo4} = ^11'$ (acarreo5 = '0')
6. $^11' - ^11' - \text{acarreo5} = ^10'$ (acarreo6 = '0')
7. $^10' - ^10' - \text{acarreo6} = ^10'$ (acarreo7 = '0')
8. $^10' - ^10' - \text{acarreo7} = ^10'$ (acarreo8 = '0')

Resultado = 00010011_2 o 19 en base decimal.

Uno de los detalles más interesantes se da cuando se restan dos números y el minuendo es menor que el sustraendo. Cuando esto ocurre, el número debe ser interpretado como negativo o complemento a 2. Por ejemplo, si se realiza la resta $72 - 94$, el resultado es -22 . Obviamente, para esto se han tenido que intercambiar las posiciones de los operandos, haciendo que 94 sea el minuendo y 72 el sustraendo.

En el caso de una resta binaria, el resultado de restar un minuendo menor que el sustraendo da el complemento a 2 del resultado de realizar la resta invirtiendo los roles del sustraendo y el minuendo. Por ejemplo, se sabe que 72 es 01001000_2 y 94 es 01011100_2 . Al realizar la resta, se obtiene el siguiente resultado: 11101010_2 , el cual es el complemento a 2 de 22.

Otra forma de realizar la misma operación resulta de sumar 72 con -94 . Para el caso del sistema decimal, la operación sigue siendo una resta. Sin embargo, para el sistema binario se pueden aplicar las mismas reglas de la adición, sumando 72 con el complemento a 2 de 94. En este caso, -94 es 10100010_2 . Entonces, sumando $01001000_2 + 10100010_2$, se obtiene el valor 11101010_2 , que resulta también ser -22 en complemento a 2.

La multiplicación binaria sigue un proceso muy similar al caso decimal. En esta última, los operandos son alineados uno encima del otro, y es el de menor longitud el de la posición inferior. Por ejemplo, si se desea multiplicar 123×14 , el proceso para el cálculo será el siguiente:

$$\begin{array}{r}
 123 \times \\
 14 \\
 \hline
 492 + \\
 123 \\
 \hline
 1722
 \end{array}$$

Como se puede observar, el operando con la cifra más extensa se coloca en la parte superior y el operando de menor longitud en la parte inferior. Se multiplica el dígito menos significativo del operando inferior por cada uno de los dígitos de la cifra superior. Luego, se multiplica el segundo operando más significativo por todos los dígitos de la cifra superior, pero el resultado se coloca en la parte inferior, dejando una columna derecha libre. Finalmente, se suman los dígitos de cada columna respectiva y se obtiene el resultado.

En el caso binario el procedimiento es muy similar, con la ventaja de que los dígitos por multiplicar son bits y los resultados de cada multiplicación solo pueden ser '0' o '1'. La multiplicación de cada dígito binario es en realidad una operación AND. El procedimiento consiste en ubicar el operando de mayor longitud sobre el de menor longitud, de la misma manera que en el procedimiento de multiplicación de cifras decimales. Luego, se debe realizar la operación AND entre cada dígito del operando corto y cada dígito de la cifra superior, empezando desde el bit LSB hasta culminar con el bit MSB del operador inferior.

Por ejemplo, si se realiza la misma operación anterior en formato binario, se tiene que $123 = 01111011_2$ y $14 = 00001110_2$. Entonces, el procedimiento de multiplicación sería como se muestra a continuación:

$$\begin{array}{cccccccccccc}
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1_2 \times \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0_2 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0_2 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1_2 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1_2 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1_2 \\
\hline
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0_2
\end{array}$$

Como resultado, se obtiene el valor binario 011010111010₂, que es equivalente a 1722. Cabe resaltar que este tipo de operación solo aplica para cifras positivas. Para multiplicar cifras en complemento a 2, se debe optar por otra metodología como sumas sucesivas.

1.1.5 Punto fijo y Coma flotante

En muchas ocasiones es necesario representar una cifra de forma fraccional; para esto, existen dos maneras de hacerlo en código binario. Estos formatos son el Punto fijo y la Coma flotante.

El formato de Punto fijo es el más simple y no requiere de mucha carga computacional por parte del sistema embebido para ser operado. Este formato fue introducido al inicio de la década de 1980 y se encuentra basado en un tamaño fijo de bits, tanto para la parte fraccional como para la parte entera.

En el formato de Punto fijo, se debe establecer un punto decimal imaginario dentro del código binario. Por ejemplo, para una cifra binaria con el siguiente formato: 1010.1011_2 , se muestra un byte que contiene un punto fijo entre ambos *nibbles*. Esta posición indica que los bits a la izquierda del punto decimal son considerados la parte entera de la cifra, mientras que los bits ubicados al lado derecho del punto son considerados la parte fraccionaria. En este caso, se han asignado 4 bits para la parte entera y 4 bits para la parte fraccionaria. Esto no necesariamente ha de ser así, ya que se pueden asignar diferentes tamaños para cada sección.

La interpretación de la cifra debe ser dada de dos maneras. Para la parte entera, se mantiene el formato entero. Para la parte fraccionaria, las potencias asignadas a la posición de cada bit presentan exponente negativo. En el gráfico 1.8 se muestra la forma como se obtiene el valor decimal de la cifra 1010.1011_2 .

Gráfico 1.8. Conversión del formato punto fijo a decimal fraccionario

$$\begin{array}{ccccccc}
 \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & . & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\
 \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow \\
 2^3 & 2^2 & 2^1 & 2^0 & & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4}
 \end{array}$$

$$8 + 2 = 10 + 0,5 + 0,125 + 0,0625 = 0,6875$$

$$10,6875$$

Elaboración propia.

Como se puede observar, la parte entera y fraccionaria se calcula por separado, para luego unirse en una sola cifra, incorporando el punto decimal. La mínima cantidad que puede ser expresada en formato Punto fijo es 0000.0000_2 , que representa 0.0, mientras que el valor máximo es 1111.1111_2 , que equivale a 15.9375. En muchos casos, el bit MSB de la parte entera es asumido como el signo de la cifra. En ese sentido, considerando el cuarto bit MSB de la parte entera el signo ('1' negativo y '0' positivo), el rango de valores máximo y mínimo que puede tener la cifra va desde -7.9375 hasta 7.9375 (tomar en cuenta que es diferente del formato complemento a 2, pues en este caso corresponde al formato complemento a 1^{11}).

La precisión del formato de Punto fijo aumenta conforme el número de bits asignados a la etapa fraccionaria es incrementado. En la práctica, este método es utilizado por muchos sistemas embebidos de baja capacidad, aunque en muchas ocasiones es dejado de lado, por su falta de precisión.

Para mejorar la precisión de las cifras fraccionarias existe el formato de Coma flotante. En este formato se divide un conjunto de bits (normalmente entre 24 y 32 bits) en tres partes: signo, exponente y mantisa. En el gráfico 1.9 se muestra una representación del formato de Coma flotante para un número de 32 bits.

11 En el formato complemento a 1 el bit MSB representa el signo ('1' negativo y '0' positivo) y los bits restantes corresponden a la magnitud de la cifra.

Gráfico 1.9. Representación en Coma flotante para un número de 32 bits

S	EXP	MANTISA
S: Bit de signo (0=>+, 1=>-)		
EXP: Exponente (8bits)		
MANTISA: Base decimal (23 bits)		
Número = (S)×2 ^(EXP-BASE) ×(1.0+fracción/(base fracción))		

Elaboración propia.

Por lo general, el bit de signo (S) toma la posición más significativa de todo el registro de 32 bits y ocupa un espacio de 1 bit. El valor ‘0’ indica que la cifra debe ser interpretada como positiva, mientras que el valor ‘1’ representa una magnitud negativa.

El exponente especifica directamente la magnitud entera de la cifra en Coma flotante. El sector del exponente está compuesto por un número de bits, por lo general, menor que el asignado a la mantisa. El código binario del exponente representa un factor, al cual se le restará una cifra base. El resultado de esta resta será el exponente de un factor 2 al cual se le multiplicará un número fraccionario. La base viene a ser el máximo valor que se puede formar con el número de bits menos uno asignado al exponente. Esto implica que, para 8 bits de exponente, la base será 127, ya que es el máximo valor posible formado por 7 bits. Evidentemente, el exponente puede ser negativo, con lo cual se tendría la representación de una cifra netamente fraccionaria y menor que 1.

La mantisa especifica el valor fraccionario de la cifra en Coma flotante. Mientras más bits posea la mantisa, mayor será la precisión decimal de la cifra. Mientras que la base de la fracción es 2ⁿ, donde n es el número de bits de la mantisa¹².

Como ejemplo, se cuenta con un número en Coma flotante compuesto por 13 bits. En este formato se utiliza un bit de signo, 4 bits para el exponente y 8 bits para la mantisa. El número presenta el siguiente código binario: 010001010000₂. Entonces, la cifra decimal se obtendrá de la siguiente manera:

Bit MSB = ‘1’ => Signo = -
Exponente = 1000₂ = 8
Base = 2⁴⁻¹ - 1 = 7
Fracción = 1010000₂ = 160
Base de la fracción = 2⁸ = 256

Por lo tanto, la cifra se calcula de la siguiente manera:

$$\text{Cifra} = (-1)^1 2^{8-7} \times \left(1 + \frac{160}{256} \right) = -3.25$$

En el mundo de los sistemas embebidos, el uso de números en formato Coma flotante es sumamente preferido por parte de los programadores para simplificar cálculos fraccionales y obtener una buena precisión. Sin embargo, la carga computacional que requiere este formato para realizar las ope-

12 Cfr: Pong Chu 2008.

raciones de suma, resta y multiplicación es bastante elevada para sistemas embebidos que no poseen una arquitectura optimizada para esto.

1.1.6 El sistema de numeración BCD

El formato BCD (*Binary Coded Decimal*) es bastante utilizado en los sistemas embebidos para el manejo de dispositivos periféricos, en los cuales se requiere presentar datos numéricos (por ejemplo, Displays 7 segmentos, módulos LCD alfanuméricos, entre otros). En este formato, cada dígito decimal es reemplazado por su equivalente en el sistema binario, pero solo utilizando cuatro bits. En el gráfico 1.10 se muestra el valor BCD para cada dígito decimal.

Gráfico 1.10. Códigos BCD para cada dígito decimal

Dígito decimal	Formato BCD
0	0000 ₂
1	0001 ₂
2	0010 ₂
3	0011 ₂
4	0100 ₂
5	0101 ₂
6	0110 ₂
7	0111 ₂
8	1000 ₂
9	1001 ₂

Elaboración propia.

Si la cifra que se representará en BCD es mayor que 9, se tendrán que agregar cuatro bits más al código para representar dos cifras decimales. En otras palabras, si la cifra decimal tiene *m* dígitos, esta misma cifra tendrá *4xm* bits en formato BCD.

Por ejemplo, la cifra 9512 se codifica en BCD de la siguiente manera: 1001010100010010₂. Esto es diferente del formato binario, ya que la cifra 9512 requiere 14 bits para su representación.

Aunque parezca sorprendente, la aritmética en el formato BCD no resulta muy sencilla para un sistema embebido, por lo cual, normalmente se deben operar las cifras en formato binario, y luego aplicar una técnica para realizar la conversión de base binaria a formato BCD.

1.1.7 El sistema de numeración ASCII

El formato ASCII (*American Standard Code for Information Interchange*) es un sistema de codificación binario empleado para representar caracteres en el alfabeto latino. Como se trata de un estándar estadounidense, este formato representa todos los caracteres usados en el idioma inglés. El formato ASCII actualmente se utiliza para que los sistemas embebidos puedan configurar diversos dispositivos, tales

como pantallas LCD alfanuméricas, módems, cámaras digitales, etc. Inclusive, los lenguajes de programación incluyen dentro de su sintaxis la opción de interpretación de caracteres ASCII.

El estándar ASCII utiliza 7 bits para representar los caracteres del alfabeto latino. Existen otros caracteres extendidos que se obtienen al generar códigos de 8 bits y que permiten representar caracteres de los idiomas español o japonés.

Los primeros 32 caracteres del código ASCII son conocidos como «comandos de control». Estos cumplen con la función específica de configurar algunas opciones del texto sin imprimir ningún carácter en pantalla. Originalmente, el código ASCII fue usado para enviar información a dispositivos de texto, como teleimpresoras, pero ellos, al estar actualmente en obsolescencia, han dejado muchos de estos códigos sin una función útil. En el gráfico 1.11 se muestran los códigos ASCII para los caracteres de control. Los comandos que aparecen con un * son los que actualmente son utilizados en las diversas aplicaciones de los sistemas embebidos.

Gráfico 1.11. Códigos ASCII para los caracteres de control

Comando ASCII	Código hexadecimal	Función
NUL	0x00	Carácter nulo
SOH	0x01	Inicio de encabezado
STX	0x02	Inicio de texto
ETX	0x03	Fin de texto
EOT	0x04	Fin de transmisión
ENQ	0x05	Consulta
ACK	0x06	Acuse de recibo
BEL	0x07	Timbre
BS	0x08	Retroceso *
HT	0x09	Tabulación horizontal
LF	0x0A	Salto de línea *
VT	0x0B	Tabulación vertical
FF	0x0C	De avance
CR	0x0D	Retorno de carro *
SO	0x0E	Mayúsculas fuera
SI	0x0F	En mayúsculas
DEL	0x10	Enlace de datos/Escape
DC1	0x11	Dispositivo de control 1
DC2	0x12	Dispositivo de control 2
DC3	0x13	Dispositivo de control 3
DC4	0x14	Dispositivo de control 4
NAK	0x15	Confirmación negativa
SYN	0x16	Síncrono en espera
ETB	0x17	Fin de transmisión de bloque
CAN	0x18	Cancelar

Comando ASCII	Código hexadecimal	Función
EN	0x19	Finalización del medio
SUB	0x1A	Substituto
ESC	0x1B	Escape *
FS	0x1C	Separador de fichero
GS	0x1D	Separador de grupo
RS	0x1E	Separador de registro
US	0x1F	Separador de unidad
DEL	0x7F	Eliminar *

Elaboración propia.

Posteriormente, quedan los caracteres ASCII imprimibles, los cuales son códigos que representan un símbolo alfanumérico. Estos caracteres están especificados entre los códigos del 33 al 126. En el gráfico 1.12 se muestra la lista de caracteres ASCII imprimibles.

Gráfico 1.12. Caracteres ASCII imprimibles

Código hexadecimal	Símbolo ASCII	Código hexadecimal	Símbolo ASCII	Código hexadecimal	Símbolo ASCII
0x20	Espacio ' '	0x40	@	0x60	`
0x21	!	0x41	A	0x61	a
0x22	"	0x42	B	0x62	b
0x23	#	0x43	C	0x63	c
0x24	\$	0x44	D	0x64	d
0x25	%	0x45	E	0x65	e
0x26	&	0x46	F	0x66	f
0x27	'	0x47	G	0x67	g
0x28	(	0x48	H	0x68	h
0x29	)	0x49	I	0x69	i
0x2A	*	0x4A	J	0x6A	j
0x2B	+	0x4B	K	0x6B	k
0x2C	,	0x4C	L	0x6C	l
0x2D	-	0x4D	M	0x6D	m
0x2E	.	0x4E	N	0x6E	n
0x2F	/	0x4F	O	0x6F	o
0x30	0	0x50	P	0x70	p
0x31	1	0x51	Q	0x71	q
0x32	2	0x52	R	0x72	r
0x33	3	0x53	S	0x73	s
0x34	4	0x54	T	0x74	t

Código hexadecimal	Símbolo ASCII	Código hexadecimal	Símbolo ASCII	Código hexadecimal	Símbolo ASCII
0x35	5	0x55	U	0x75	u
0x36	6	0x56	V	0x76	v
0x37	7	0x57	W	0x77	w
0x38	8	0x58	X	0x78	x
0x39	9	0x59	Y	0x79	y
0x3A	:	0x5A	Z	0x7A	z
0x3B	;	0x5B	[	0x7B	{
0x3C	<	0x5C	\	0x7C	
0x3D	=	0x5D	]	0x7D	}
0x3E	>	0x5E	^	0x7E	~
0x3F	?	0x5F	-		

Elaboración propia.

1.2 Sistemas embebidos

El término «sistema embebido» hace referencia a todo circuito electrónico digital capaz de realizar operaciones de computación, generalmente en tiempo real, que sirven para cumplir una tarea específica en un producto.

Los sistemas embebidos no son equivalentes a los sistemas de cómputo usados en las *laptops* o en computadoras de escritorio que se venden en las tiendas tecnológicas, ya que los sistemas embebidos suelen tener recursos limitados y aplicaciones específicas que los hacen sumamente útiles en múltiples ambientes, como en el campo automotriz (sistemas de inyección de gasolina, alarmas contra robos, control de climatización, sistema de frenado ABS) o en teléfonos móviles, ipad, reproductores Blu-ray, refrigeradoras, alarmas de casas, lavadoras, cámaras fotográficas, instrumentación industrial, equipos médicos, Set Top Boxes, entre otros.

La arquitectura de un sistema embebido contiene un microprocesador dedicado capaz de ejecutar instrucciones a una determinada velocidad, controlada por una señal de reloj. De acuerdo con la arquitectura del microprocesador del sistema embebido, los recursos internos (periféricos) y la máxima frecuencia de operación, se define la potencia de procesamiento. Normalmente, este parámetro se mide en unidades de MIPS (millones de instrucciones por segundo).

El programa que ejecuta un sistema embebido es por lo general elaborado en lenguajes como el ensamblador, ANSI C, C++ o Basic. Algunos sistemas embebidos tienen la capacidad de ejecutar sistemas operativos limitados, conocidos como RTOS (Real Time Operating Systems), que permiten que el procesador ejecute diversas tareas, asignándoles una prioridad y un orden de ejecución. Algunos de estos sistemas operativos son el MQX de Freescale, μ C/OS-II, FreeRTOS y μ Linux, entre los más conocidos.

Una de las características actuales de todo sistema embebido, y muy importante, es el bajo consumo de energía. Como muchas de sus aplicaciones involucran dispositivos que utilizan baterías, los modos de bajo consumo de corriente (modo *sleep*) son sumamente importantes para que puedan ser utilizados en este tipo de aplicaciones.

Un microcontrolador es un sistema embebido, y actualmente podemos encontrar microcontroladores en un solo chip, los cuales son sistemas computarizados completos, ya que contienen en su interior un microprocesador, unidades de memoria (de programa y datos), unidades de entrada-salida y periféricos. Hoy en día existen en el mercado microcontroladores de diversas arquitecturas (8 bits, 16 bits y 32 bits), diferentes marcas y modelos y dedicados para aplicaciones específicas.

1.3 Lenguajes de programación

El proceso de programación de un sistema embebido requiere inicialmente de un computador o *Host*, en el cual se ejecute una herramienta de *software* denominada IDE (*Integrated Development Environment*). Dentro de esta herramienta, se realiza la programación de instrucciones del sistema embebido, que, por lo general, puede incluir varios archivos fuente con partes del programa. La secuencia de instrucciones que especifica el programa en un archivo de texto que será grabado en el sistema embebido se conoce como *firmware*.

Cuando el programa es finalizado, se hace uso de un *software* llamado «compilador». Esta herramienta transforma el texto que contiene las instrucciones de forma secuencial y lo convierte en un código de máquina: un archivo compuesto de números binarios que va a ser enviado directamente a la memoria de programa del sistema embebido (denominado *Target*). En el gráfico 1.13 se muestra un esquema de la comunicación entre el *Host* y el *Target*.

Gráfico 1.13. Interfaz *Host-Target* para programación del sistema embebido

Elaboración propia.

Usualmente, se requiere una interfaz entre el *Host* y el *Target*, conocida como «el programador». Esta interfaz es un equipo que se puede conectar al *Host* a través del puerto RS232, USB, Ethernet o, antiguamente, en el puerto paralelo o LPT. Por otro lado, el programador se conecta al *Target* a través de un cable de conexión (cuyo estándar depende del fabricante del chip). El *software* IDE posee aplicativos que permiten enviar el archivo con el código máquina al programador. Este, a su vez, se comunica con una aplicación en el *Target* denominada «intérprete», que se encarga de grabar el código máquina en los registros de memoria de programa del sistema embebido.

Entre los lenguajes de programación que se pueden elegir para programar un sistema embebido, se tienen las siguientes opciones:

Para poder revisar todo el contenido de esta edición,
visite nuestra **tienda virtual**.