Análisis estructural y funcional de un microcontrolador

Breve descripción:

El proceso de creación de un programa para microcontroladores incluye redactar instrucciones en lenguaje de programación, compilar el código y realizar el enlazado. Luego, el programa se simula para verificar su funcionamiento antes de quemar el firmware en la memoria del microcontrolador. Finalmente, se realizan pruebas en hardware y software para asegurar su correcto desempeño.

Diciembre 2024

Tabla de contenido

Introducción

El desarrollo de programas para microcontroladores es un proceso fundamental en la ingeniería electrónica y la programación de sistemas embebidos. Los microcontroladores son dispositivos que integran componentes de procesamiento, almacenamiento y comunicación en una sola unidad, lo que permite automatizar tareas en dispositivos y sistemas cotidianos. La creación de un programa funcional para estos sistemas requiere precisión y una comprensión detallada de su arquitectura.

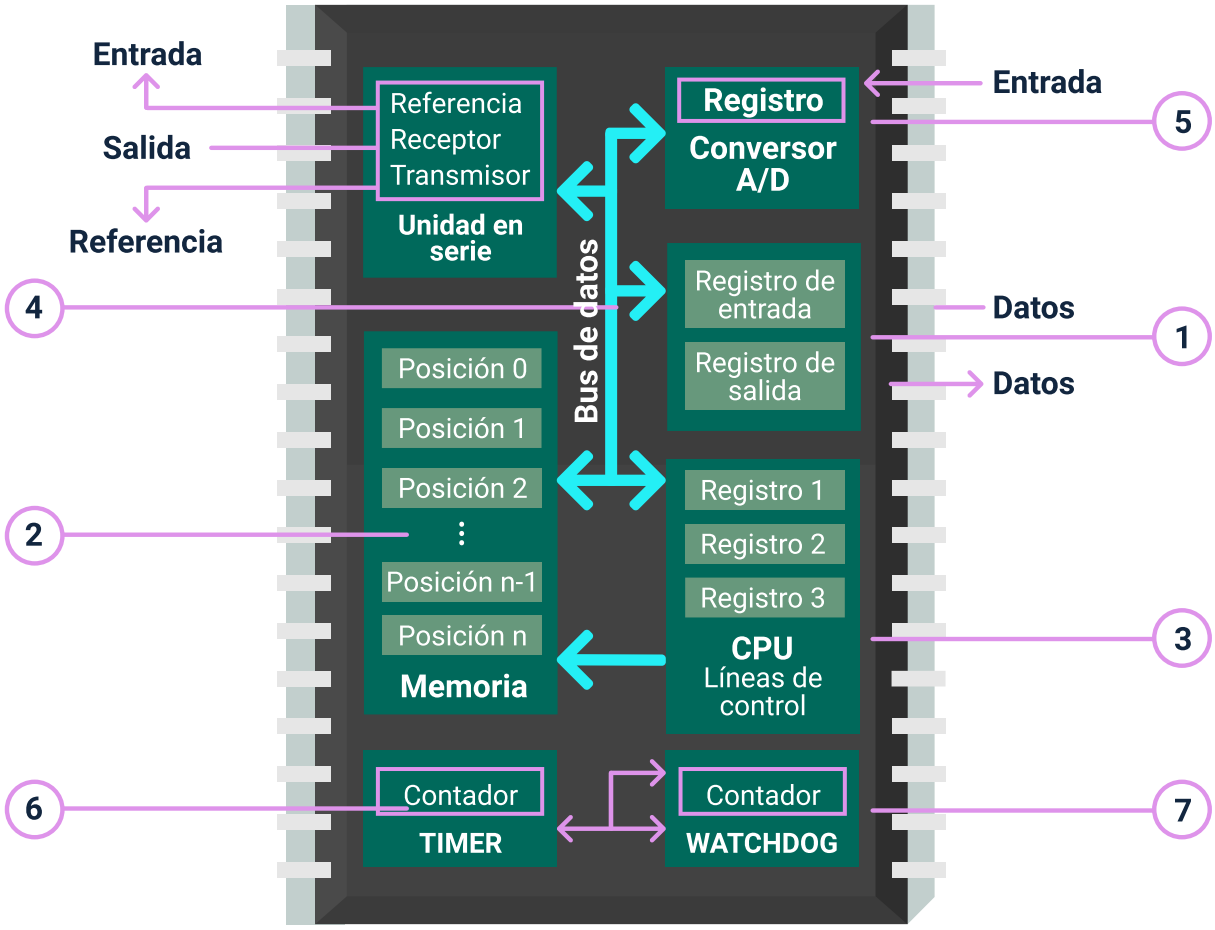
El proceso de programación de un microcontrolador implica varias etapas, comenzando con la redacción de instrucciones en un lenguaje de programación específico. Posteriormente, el código fuente se compila y enlaza para traducirlo a un formato comprensible por el hardware. Esta fase convierte las instrucciones en lenguaje máquina, permitiendo que el microcontrolador las interprete de manera adecuada y eficiente.

Una vez compilado, el programa se somete a simulación y pruebas antes de ser "quemado" en la memoria del microcontrolador. La simulación ayuda a predecir el comportamiento del programa sin el uso físico del dispositivo, mientras que el quemado transfiere el firmware directamente al hardware. Finalmente, se realizan pruebas tanto en hardware como en software para garantizar un funcionamiento óptimo y una integración precisa del microcontrolador en su entorno operativo.

1. Estructura básica de los microcontroladores

Los microcontroladores están constituidos, fundamentalmente, por los siguientes componentes:

Figura 1. Microcontroladores



a) Unidad de input – output (E/S)

Recibe y envía señales de información en las unidades funcionales de un sistema de procesamiento de datos, permitiendo la comunicación con el entorno externo.

b) **Unidad de memoria**

Almacena datos de manera organizada, similar a un armario con cajones, facilitando el acceso sin confusiones.

c) **Unidad de procesamiento central (CPU)**

Ejecuta operaciones como multiplicaciones, divisiones y restas, y transfiere datos entre registros de memoria.

d) **Bus de datos**

Transporta datos entre los bloques del microcontrolador; incluye el bus de dirección para la memoria y el bus de datos para la conexión entre bloques internos.

e) **Conversor A/D**

Convierte señales periféricas en un formato binario (ceros y unos) para que el microcontrolador pueda interpretarlas.

f) **Contador TIMER**

Genera intervalos de tiempo para que el microcontrolador ejecute tareas a intervalos específicos.

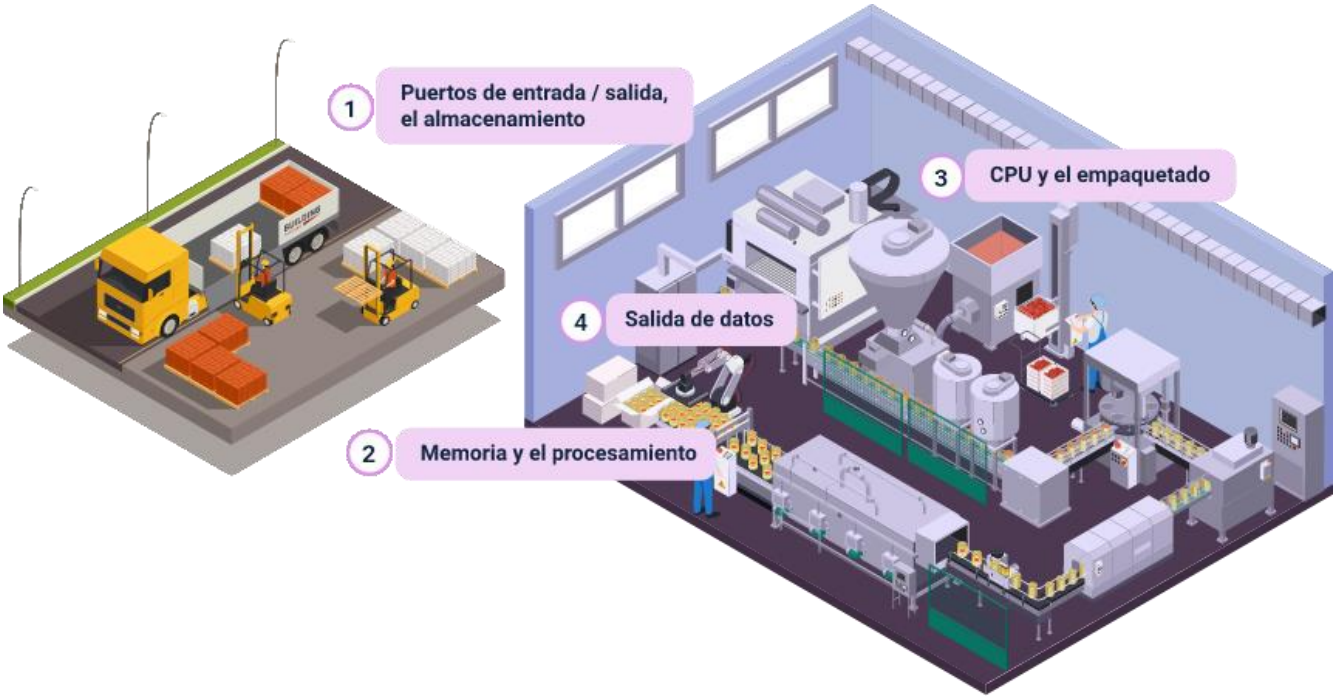
g) **Contador WATCHDOG**

Reinicia el sistema en caso de interrupciones, garantizando la continuidad del funcionamiento.

Analogía con una planta procesadora

La estructura de un microcontrolador puede asimilarse a una planta procesadora de papas fritas:

Figura 2. Empresa papas fritas



a) Unidad de input – output (E/S)

Similar a la zona de carga y descarga, donde se realiza el despacho y recepción de productos en la planta.

b) Unidad de memoria

Equivalente a la zona de almacenamiento o bodega, con áreas organizadas para el almacenamiento de productos e insumos.

c) Unidad de procesamiento central (CPU)

Corresponde a las zonas de proceso, donde se realizan las transformaciones del producto, desde el lavado hasta el empaque.

d) Bus de datos

Representa las zonas de transporte, los caminos internos por los que se movilizan insumos y productos dentro de la planta.

2. Lenguaje de programación

Cada lenguaje de programación posee características propias en cuanto a gramática y sintaxis. A diferencia del lenguaje humano, los lenguajes de programación transmiten cuatro ideas fundamentales y requieren precisión, claridad y ausencia de ambigüedad. Existen dos niveles principales de lenguajes de programación:

- **Lenguajes de bajo nivel**

  Son una representación simbólica del lenguaje máquina.

- **Lenguajes de alto nivel**

  Facilitan la programación mediante macroinstrucciones, que posteriormente se traducen al lenguaje máquina.

Lenguaje Assembly (ASM)

Cada lenguaje tiene características propias en cuanto a su gramática y sintaxis. A diferencia del lenguaje humano que permite transmitir múltiples ideas a la vez, los lenguajes de programación no requieren que comuniquen gran variedad de pensamientos, es suficiente que permitan la transmisión de solo cuatro pensamientos básicos. Además, la comunicación a través de estos lenguajes tiene que ser precisa, clara y sin ambigüedades.

Directivas en lenguaje Assembly

Las directivas en Assembly son similares a las instrucciones, pero no dependen del modelo específico del microcontrolador; representan una característica propia del lenguaje ASM y se utilizan para asignar significados o propósitos a través de variables o registros.

Directivas de control

Las directivas de control en Assembly son comandos que gestionan aspectos específicos del programa, como la inclusión de archivos, la definición de constantes y variables, y la configuración de memoria y bits.

- **#DEFINE**

  Permite intercambiar un fragmento de texto por otro.

  Sintaxis: #define <texto> [<otro texto>]

  Descripción: sustituye cada aparición de <texto> en el programa por <otro texto>.

  Ejemplo: #define encendido 1

  #define apagado 0

- #INCLUDE

  Incluye un archivo adicional en el programa.

  Sintaxis: #include <nombre_de_archivo> o #include "nombre_de_archivo"

  Descripción: Copia el contenido del archivo en el lugar donde se incluye. < > para archivos de sistema, comillas " para archivos de usuario.

  Ejemplo: #include <regs.h>

  #include "subprog.asm"

- **CONSTANT**

  Asigna un valor numérico constante.

  Sintaxis: constante <nombre> = <valor>

  Ejemplo: constant MAXIMO = 100

  constant Longitud = 30

- **VARIABLE**

  Define un valor numérico variable.

  Sintaxis: variable <nombre> = <valor>

  Ejemplo: variable nivel = 20

  variable tiempo = 13

- **SET**

  Define variables en ASM y permite redefinirlas.

  Sintaxis: <nombre_variable> set <valor>

  Ejemplo: nivel set 0

  longitud set 12

  nivel set 45

- **EQU**

  Define una constante.

  Sintaxis: <nombre_constante> equ <valor>

  Ejemplo: cinco equ 5

  seis equ 6

  siete equ 7

- **ORG**

  Establece la dirección de memoria donde se graba el programa.

  Sintaxis: <etiqueta> org <valor>

  Ejemplo: Inicio org 0x00

  movlw 0xFF

  movwf PORTB

- **END**

  Marca el final del programa.

  Sintaxis: end

  Ejemplo: movlw 0xFF

  movwf PORTB

  end

- **_CONFIG**

  Ajusta bits de configuración, como el oscilador.

  Sintaxis: _config <termino> o _config <direccion>, <termino>

  Ejemplo: _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

- **PROCESSOR**

  Define el modelo de microcontrolador.

  Sintaxis: Processor <tipo_microcontrolador>

  Ejemplo: processor 16F84

- **CBLOCK**

  Define un bloque de constantes nombradas.

  Sintaxis: Cblock [<termino>]

  <etiqueta> [:<incremento>], <etiqueta> [:<incremento>] endc

  Ejemplo: Cblock 0x02

  primero, segundo, tercero

  Endc

- **DE**

  Define un byte en la memoria EEPROM.

  Ejemplo: org H'2100'

de "Version 1.0", 0

- **DT**

  Define una tabla de datos.

  Ejemplo: dt "Mensaje", 0

Directivas de configuración

Las directivas de configuración en Assembly permiten ajustar parámetros esenciales del programa, como la configuración de bits, el modelo de microcontrolador, y la definición de bloques de datos o condiciones de ejecución, optimizando el control y funcionalidad del código.

a) **_CONFIG**

   Ajusta bits de configuración, como el oscilador.

   Sintaxis: _config <termino> o _config <direccion>, <termino>

   Ejemplo: _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

b) **PROCESSOR**

   Define el modelo de microcontrolador.

   Sintaxis: Processor <tipo_microcontrolador>

   Ejemplo: processor 16F84

c) **CBLOCK**

   Define un bloque de constantes nombradas.

   Sintaxis: Cblock [<termino>] <etiqueta> [: <incremento>], <etiqueta> [: <incremento>] endc

   Ejemplo: Cblock 0x02 primero, segundo, tercero Endc

d) **DB**

Define un byte de datos.

Ejemplo: db 't', 0x0f, 'e', 's', 0x12

e) **DE**

Define un byte en la memoria EEPROM.

Ejemplo: org H'2100' de "Version 1.0", 0

f) **DT**

Define una tabla de datos.

Ejemplo: dt "Mensaje", 0

g) **IF, ELSE, ENDIF**

Controlan la ejecución basada en condiciones.

Ejemplo: If nivel = 100 goto CARGA else goto DESCARGA endif

h) **WHILE, ENDW**

Ejecuta secciones de código mientras se cumpla una condición.

Ejemplo: While i < 10 i = i + 1 endw

i) **IFDEF, IFNDEF**

Ejecuta el código si una definición existe (IFDEF) o no (IFNDEF).

Ejemplo: #define prueba Ifdef prueba ... endif

j) **Operadores Matemáticos**

Incluyen +, -, *, /, % y operadores bit a bit como &, `

Instrucciones de condición

Las instrucciones de condición en Assembly permiten controlar el flujo de ejecución del programa según se cumplan o no ciertas condiciones, proporcionando flexibilidad en la toma de decisiones dentro del código.

- **IF, ELSE, ENDIF**

  Controlan la ejecución basada en condiciones.

  Ejemplo

  If nivel = 100

  goto CARGA

  else

  goto DESCARGA

  endif

- **WHILE, ENDW**

  Ejecuta secciones de código mientras se cumpla una condición.

  Ejemplo

  While i < 10

  i = i + 1

  endw

- **IFDEF, IFNDEF**

  Ejecuta el código si una definición existe (IFDEF) o no (IFNDEF).

  Ejemplo

  #define prueba

  Ifdef prueba

  ...

  endif

Operadores matemáticos

Los operadores matemáticos en Assembly permiten realizar cálculos aritméticos y lógicos en el programa, proporcionando una variedad de operaciones como suma, multiplicación, desplazamientos de bits y comparaciones, esenciales para manipular y evaluar datos de manera precisa en el código.

Tabla 1. Operadores matemáticos

Símbolo	Descripción	Ejemplo
$	Estado actual del contador del programa	goto $ + 3
(	Corchete izquierdo	1 + (d * 4)
)	Corchete derecho	(Length + 1) * 256
!	Complemento lógico	if ! (a = b)
~	Complemento	flags = ~flags
-	Negación (segundo complemento)	-1 * Length
high	Devuelve el byte más alto	movlw high CTR_Table
low	Devuelve el byte más inferior	movlw low CTR_Table
*	Multiplicador	a = b * c
/	Divisor	a = b / c
%	Divisor por módulo	entry_len = tot_len % 15
+	Sumando	tot_len =entry_len * 3 + 1
-	Restando	entry_len = (tot - 1) / 8

Símbolo	Descripción	Ejemplo
<<	Moviendo a la izquierda	val = flags << 1
>>	Moviendo a la derecha	val = flags >> 1
>=	Mayor que o igual	if entry_idx >= num_entries
>	Mayor que	if entry_idx > num_entries
<	Menor que	if entry_idx < num_entries
<=	Menor que, o igual	if entry_idx <= num_entries
==	igual	if entry_idx == num_entries
!=	No igual	if entry_idx != num_entries
&	Operación AND en los bits	flags = flags & ERROR_BIT
^	Exclusivo OR en los bits	flags = flags ^ ERROR_BIT
`	`	Lógica OR sobre bits
&&	Lógica OR sobre bits (if)	if (len ==512) && (b ==c)
=	Igual	entry_index = 0
+=	Añadir y asignar	entry_index +=1
-=	Restar y asignar	entry_index -=1
*=	Multiplicar y asignar	entry_index *= entry_length
/=	Dividir y asignar	entry_index /= entry_length
%=	Divide el módulo y asignar	entry_index %=1
<<=	Mueve a la izquierda y asigna	flags << = 3
>>=	Mueve a la derecha y asigna	flags >> = 3

Símbolo	Descripción	Ejemplo
&=	Lógica AND y asigna	flags &= ERROR_FLAG
`	=`	Lógica OR y asigna
^=	Exclusivo OR y asigna	flags ^= ERROR_FLAG
++	Incrementa por uno	1++

Ejemplo de lenguaje Assembly: "Hola Mundo" en x86 bajo DOS:

```
.model small

.stack

.data

Cadena1 DB 'Hola Mundo.$'

.code

programa:

  mov ax, @data

  mov ds, ax

  mov dx, offset Cadena1

  mov ah, 9

  int 21h

end programa
```

Este ejemplo imprime "Hola Mundo" en DOS usando lenguaje Assembly.

3. Estructura del proceso de creación de un programa para microcontroladores

En esta fase, se redactan las instrucciones del programa. El programa, escrito en un lenguaje de programación comprensible para el ser humano (frecuentemente utilizando lenguajes formales descritos por gramáticas independientes del contexto), no es ejecutado inmediatamente en una computadora.

La opción más común es compilar el programa, aunque también puede ejecutarse mediante un intérprete. El código fuente debe someterse a un proceso de transformación para convertirse en lenguaje máquina, interpretable por el procesador, conocido como compilación. La creación de un programa ejecutable (como un archivo .exe en Microsoft Windows) generalmente implica dos pasos:

- **Compilación**

  Traduce el código fuente a un código de bajo nivel (normalmente código objeto, sin llegar directamente al lenguaje máquina).

- **Enlazado**

  Combina el código de bajo nivel generado de todos los archivos compilados y agrega el código de funciones de las bibliotecas del compilador, permitiendo así la comunicación con el sistema operativo y la traducción del código objeto a código máquina.

Simulación del programa

Para probar el programa sin emplear físicamente un recurso electrónico, se pueden utilizar herramientas tecnológicas, ya sean de software o hardware, que

permitan verificar si la aplicación funciona correctamente; estas herramientas se denominan **simuladores**.

Un simulador es un **software** capaz de reproducir virtualmente el comportamiento real de un microcontrolador a partir del código .asm creado. Existen diferentes simuladores en el mercado que permiten desarrollar diseños complejos para programar microcontroladores, como los siguientes:

- **Proteus**

  Permite la simulación de circuitos electrónicos con microcontroladores y es el simulador más popular para microcontrola dores PIC.

- **GPSIM**

  Un simulador gratuito con un entorno gráfico sencillo.

Quemado del programa

Una vez simulado el programa y verificado su funcionamiento, se genera el archivo .asm con las instrucciones, conocido como **firmware** o **programación en firme**. Este firmware se descarga o **quema** en la memoria del microcontrolador en un proceso denominado quemado.

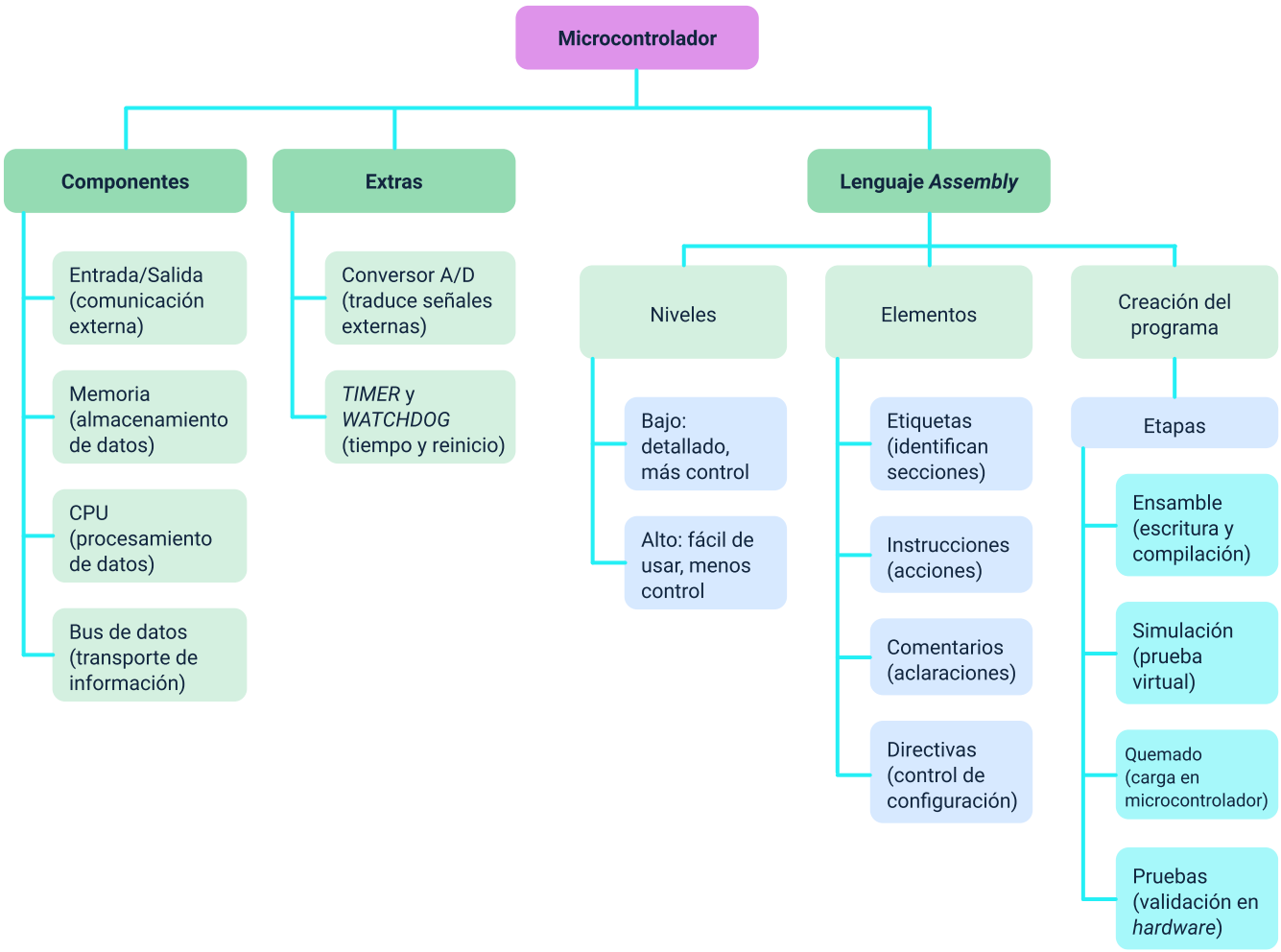
El **firmware** funciona como un bloque de instrucciones con las órdenes específicas de lo que se quiere ejecutar (por ejemplo, la activación de una alarma sonora). Este bloque establece la lógica de bajo nivel que controla los circuitos electrónicos del dispositivo. Funcionalmente, el **firmware** es el intermediario entre las órdenes externas que recibe el dispositivo y su electrónica, permitiendo la correcta ejecución de estas órdenes.

Pruebas del programa

Las pruebas permiten verificar si el código .asm funciona correctamente y si el microcontrolador está en buen estado. Para ello, es recomendable seguir un procedimiento básico que incluya tanto el hardware como el software del microcontrolador.

Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
Estructura básica de los microcontroladores	Delgado Miguel Angel. (2020). Estructura interna del microcontrolador. [Archivo de video] Youtube.	Video	https://www.youtube.com/watch?v=glLbD1aqJKo&ab_channel=DelgadoMiguelAngel
Lenguaje de programación	Coding Academy Perú. (2020). Assembler - Lo básico del lenguaje. [Archivo de video] Youtube.	Video	https://www.youtube.com/watch?v=eHA0WHFUinQ&ab_channel=CodingAcademyPer%C3%BA
Lenguaje de programación	Sergie Arizandieta (2022). Aprende Programación en Assembler. [Archivo de video] Youtube.	Video	https://www.youtube.com/watch?v=cFhUA7DnIVo&list=PLON3-BoIoWiV0Te8sxsvXw8u2k3DmBZun&ab_channel=SergieArizandieta
Estructura del proceso de creación de un programa para microcontroladores	Electrónica y Circuitos. (2020). BIENVENIDA AL CURSO DE PROGRAMACIÓN DE MICROCONTROLADORES PIC EN LENGUAJE C. [Archivo de video] Youtube.	Video	https://www.youtube.com/watch?v=SXl-VD5WU-s&list=PLONPO-iVba9nbY_KTCHt9GGj9dGSj1qYo&ab_channel=Electr%C3%B3nicayCircuitos
Estructura del proceso de creación de un programa para microcontroladores	Xataka (2020). ARDUINO: QUÉ ES, CÓMO FUNCIONA y PARA QUÉ se puede UTILIZAR. [Archivo de video] Youtube.	Video	https://www.youtube.com/watch?v=Zs9MZosVuqo&ab_channel=Xataka

Glosario

Código fuente: texto escrito en lenguaje de programación, que luego se compila para su ejecución en el hardware.

Compilación: proceso de traducción del código fuente en un lenguaje de bajo nivel para que el hardware pueda interpretarlo.

Enlazado: paso en el cual se une el código de bajo nivel con las bibliotecas necesarias para crear un programa ejecutable.

Firmware: conjunto de instrucciones específicas que controlan los circuitos electrónicos de un dispositivo.

Interprete: programa que ejecuta el código fuente línea por línea sin necesidad de compilarlo.

Lenguaje de programación: sistema formal de símbolos y reglas usado para escribir instrucciones entendibles por el microcontrolador.

Microcontrolador: dispositivo electrónico que integra procesamiento, almacenamiento y comunicación en una sola unidad.

Pruebas de hardware: evaluaciones que aseguran el funcionamiento correcto de los componentes físicos del microcontrolador.

Quemado: proceso de transferencia del firmware a la memoria del microcontrolador.

Simulador: software que permite verificar virtualmente el funcionamiento de un microcontrolador a partir del código programado.

Referencias bibliográficas

Barnett, R. H., Cox, S., & O'Cull, L. (2006). Embedded C programming and the Atmel AVR. Delmar Cengage Learning.

Hyde, R. (2010). The art of assembly language (2nd ed.). No Starch Press.

VanSickle, T. (2001). Programming microcontrollers in C. Academic Press.

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Castellanos	Responsable del ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de línea de producción	Centro de Servicios de Salud - Regional Antioquia
Paola Alexandra Moya Peralta	Evaluadora instruccional	Centro de Servicios de Salud - Regional Antioquia
Juan Daniel Polanco Muñoz	Diseñador de contenidos digitales	Centro de Servicios de Salud - Regional Antioquia
Luis Jesús Pérez Madariaga	Desarrollador full stack	Centro de Servicios de Salud - Regional Antioquia
Jaime Hernán Tejada Llano	Validador de recursos educativos digitales	Centro de Servicios de Salud - Regional Antioquia
Margarita Marcela Medrano Gómez	Evaluador para contenidos inclusivos y accesibles	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para contenidos inclusivos y accesibles	Centro de Servicios de Salud - Regional Antioquia