UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TEMA DE TESIS

"ROBOTCODE"

Para obtener el título de:
Ingeniero en Computación

P R E S E N T A N:

Hernández Lara Diana
Nava Hernández Dulce Tania
Savage Chávez Rodrigo

DIRECTOR DE TESIS:

M. I. NORMA ELVA CHÁVEZ RODRÍGUEZ

CIUDAD UNIVERSITARIA MÉXICO D. F. 2011

TEMARIO

INTRODUCCIÓN

Con la necesidad de crear nuevas herramientas que faciliten la programación de robots, este proyecto tiene como alcance principal crear una interfaz gráfica que ayude a los estudiantes de nivel medio y medio superior a programar sus propios robots ya sea de beneficio personal o social.

1. DEFINICIÓN DEL PROBLEMA

En la actualidad existe una creciente necesidad por parte de la industria, instituciones de investigación, instituciones militares de realizar proyectos de investigación relacionados con ingeniería y especialmente en la parte de robótica.
Hoy en México existen muy pocas personas capacitadas en el área de robótica y es necesario fomentar el estudio en esta área.

En Junio del 2012, se realizará por primera vez en México la RoboCup, proyecto internacional para promover, a través de competencias integradas por robots autónomos, la investigación y educación sobre inteligencia artificial.

En consecuencia habrá muchos estudiantes entusiasmados, dispuestos a aprender y comprar software el cual les ayudará y facilitará su aprendizaje hacia los lenguajes de programación, así como la programación del mismo robot y su comportamiento.

2. OBJETIVO

→ Programar fácil y dinámicamente un robot con la creación de RobotCode, una Herramienta de Software

→ Enseñar de una manera interactiva y divertida a estudiantes de secundaria, preparatoria o licenciatura que desean aprender a programar.

→ Mostrar resultados inmediatos del algoritmo que se acaba de realizar en un ente físico (robot).

→ Minimizar los errores comunes de programación facilitando la programación del robot.

3. ALCANCE

Este proyecto permitirá al usuario incrementar sus conocimientos de programación e implementación de algoritmos eficientes para el control y manejo de un robot. Esto gracias a que uno de los productos finales es una interfaz gráfica amigable y fácil de usar, pero también genera directamente el código en C y ensamblador; lo que facilita su implementación.

4. PRODUCTOS.

Robot Code: Interfaz gráfica, fácil de usar en la cual se arrastrarán con el mouse y se crearán estados para la creación de un algoritmo.

Robot Compiler: Usará los estados generados por Robot Code, y generará código en C, el cual podrá ser usado por el robot y la ejecución del algoritmo.

5. RESUMEN

La Tesis se encuentra dividida en cinco capítulos.

Los primeros dos capítulos, Marco teórico Hardware y Marco teórico Software, tratan sobre los fundamentos teóricos con los que se contaron para elaborar el software, RobotCode. Estos temas incluyen definiciones de algunos conceptos fundamentales como son la definición de robot, que es una carta ASM, que es un diagrama de flujo; también se tomaron en cuenta las características de los dos lenguajes que se utilizaron tanto para generar la interfaz, como el código generado por la misma: java y C, respectivamente. También se describen las características generales de los motores, sensores, protocolos utilizados en la parte de hardware, entre otros conceptos. Dentro del capítulo II, Marco teórico Software, se encuentra la esencia de la interfaz gráfica de Robot Code, debido a que en este tema se muestran los diagramas de flujo, y las cartas ASM; Robot Code toma la simbología de un diagrama de flujo, pero tiene el comportamiento de una carta ASM, esto es , para el fácil manejo del código generado.

El capítulo III, "Características del Robot", describe tanto las características del hardware como del software del robot utilizado para probar el código generado por RobotCode.

En el capítulo IV, "Plan de administración del proyecto", ya se habla propiamente del desarrollo de RobotCode, se describen los requerimientos , tanto funcionales como no funcionales, como son la portabilidad, usabilidad, las características con las que se debe contar para poder utilizar RobotCode, la memoria necesaria, el sistema operativo; también se da una pequeña introducción de las funciones con las que cuenta la interfaz gráfica; dichas funciones se describen a detalle en el siguiente y último capítulo , "Implementación". En este capítulo V, ya se muestra mediante imágenes tanto la vista final de la interfaz de usuario de RobotCode, como el diseño de las ventanas emergentes de cada uno de los bloques implementados, como son: declaración de variables, asignación de valores a las variables creadas, manejo de condiciones, y principalmente el Generador de Código C, el cual será implementado en el robot. Posteriormente profundiza en los requerimientos no funcionales implementados en Robot Code.

Después de los capítulos, se tiene un Glosario, dedicado a definir algunos conceptos manejados a lo largo de la tesis. Posteriormente se tienen los resultados de cada una de las pruebas realizadas a Robot Code. Finalmente se tienen las conclusiones y las referencias tanto bibliográficas como electrónicas.

CAPÍTULO I:

"MARCO TEÓRICO HARDWARE"

1. MARCO TEORICO HARDWARE

1.5 *ROBÓTICA*

La robótica es la ciencia encargada del estudio, diseño, fabricación y la implementación de máquinas programables capaces de emular el comportamiento de los seres humanos y resolver problemas de la mejor manera posible.
La robótica concentra seis áreas de estudio: física, matemáticas, mecánica, el control automático, la electrónica y la informática.

1.5.1 LOS ROBOTS

Un robot es un manipulador reprogramable y multifuncional concebido para transportar materiales, piezas, herramientas o sistemas especializados; con movimientos variados y programados, con la finalidad de ejecutar tareas diversas.
En 1921 Karel Capek en su obra R.U.R (Los Robots Universales de Rossum) introdujo el término "robot", que en checo es asociada con la palabra "robota" que quiere decir trabajo forzado.

1.5.2 CLASIFICACIÓN

La potencia del software en el controlador determina la utilidad y flexibilidad del robot dentro de las limitantes del diseño mecánico y la capacidad de los sensores. Los robots han sido clasificados de acuerdo a su generación, a su nivel de inteligencia, a su nivel de control, y a su nivel de lenguaje de programación. Estas clasificaciones reflejan la potencia del software en el controlador, en particular, la sofisticada interacción de los sensores. La generación de un robot se determina por el orden histórico de desarrollos en la robótica.

- **Robots Play-back:** los cuales regeneran una secuencia de instrucciones grabadas, como un robot utilizado en recubrimiento por spray o soldadura por arco. Estos robots comúnmente tienen un control de lazo abierto.
- **Robots controlados por sensores**: éstos tienen un control en lazo cerrado de movimientos manipulados, y hacen decisiones basados en datos obtenidos por sensores.
- **Robots controlados por visión:** éste tipo pueden manipular un objeto al utilizar información desde un sistema de visión.
- **Robots controlados adaptablemente:** los robots pueden automáticamente reprogramar sus acciones sobre la base de los datos obtenidos por los sensores.

- **Robots con inteligencia artificial:** los robots utilizan las técnicas de inteligencia artificial para hacer sus propias decisiones y resolver problemas.

- **Los robots médicos:** fundamentalmente se usan en prótesis para cubrir las necesidades físicas del cuerpo y se adaptan a él y están dotados de potentes sistemas de mando. Con ellos se logra igualar al cuerpo con precisión los movimientos y funciones de los órganos o extremidades que suplen.

- **Los androides:** son robots que se parecen y actúan como seres humanos.

- **Los robots móviles:** Están provistos de patas, ruedas u orugas que los capacitan para desplazarse de acuerdo su programación. Elaboran la información que reciben a través de sus propios sistemas de sensores y se emplean en determinado tipo de instalaciones industriales, sobre todo para el transporte de mercancías en cadenas de producción y almacenes. También se utilizan robots de este tipo para la investigaci*ón* en lugares de difícil acceso o muy distantes, como es el caso de la exploración espacial y las investigaciones de bajo del mar.

La Asociación de Robots Japonesa (JIRA) ha clasificado a los robots dentro de seis clases sobre la base de su nivel de inteligencia:

- Dispositivos de manejo manual, controlados por una persona.
- Robots de secuencia arreglada.
- Robots de secuencia variable, donde un operador puede modificar la secuencia fácilmente.
- Robots regeneradores, donde el operador humano conduce el robot a través de la tarea.
- Robots de control numérico, donde el operador alimenta la programación del movimiento, hasta que se enseñe manualmente la tarea.
- Robots inteligentes, los cuales pueden entender e interactuar con cambios en el medio ambiente.

Los programas en el controlador del robot pueden ser agrupados de acuerdo al nivel de control que realizan.

Nivel de inteligencia artificial: donde el programa aceptará un comando como "levantar el producto" y descomponerlo dentro de una secuencia de comandos de bajo nivel basados en un modelo estratégico de las tareas.

Nivel de modo de control: donde los movimientos del sistema son modelados, para lo que se incluye la interacción dinámica entre los diferentes mecanismos, trayectorias planeadas, y los puntos de asignación seleccionados.

Niveles de servosistemas: donde los actuadores controlan los parámetros de los mecanismos con el uso de una retroalimentación interna de los datos obtenidos por los sensores, y la ruta es modificada sobre la base de los datos que se obtienen de sensores externos. Todas las detecciones de fallas y mecanismos de corrección son implementadas en este nivel.
En la clasificación final se considerara el nivel del lenguaje de programación. La clave para una aplicación efectiva de los robots para una amplia variedad de tareas, es el desarrollo de lenguajes de alto nivel. Existen muchos sistemas de programación de robots, aunque la mayoría del software más avanzado se encuentra en los laboratorios de investigación. Los sistemas de programación de robots caen dentro de tres clases:

1. **Sistemas guiados:** en el cual el usuario conduce el robot a través de los movimientos a ser realizados

2. **Sistemas de programación de nivel-robot:** en los cuales el usuario escribe un programa de computadora al especificar el movimiento y las señales del sensor.
3. **Sistemas de programación de nivel-tarea:** en el cual el usuario especifica la operación por sus acciones sobre los objetos que el robot manipula.

1.6 *MICROPROCESADOR*

Es una unidad de proceso que consiste en el agrupamiento de un número relativamente pequeño de componentes integrados y realiza todas las funciones de la unidad central de procesa, ALU, unidad de control, registros, etc.

Realiza el procesamiento de datos, las operaciones que se llevan a cabo durante dicho procesamiento son controladas por un programa el cual le indica al microprocesador que es lo que tiene que hacer, por ejemplo, leer un dato de un teclado y mandarlo a pantalla.

Los microprocesadores han cambiado la forma en que realizaban las operaciones de tratamiento de información, desde que aparecieron en los años 70 no han dejado de evolucionar, el primer microprocesador fue el 4004 desarrollado por Intel el cual era muy limitado pues sólo sumaba y restaba datos de 4 bits.
La capacidad de proceso del microprocesador está definida en gran parte por su set de instrucciones. En la Figura 1 se observa la estructura general de un microprocesador.

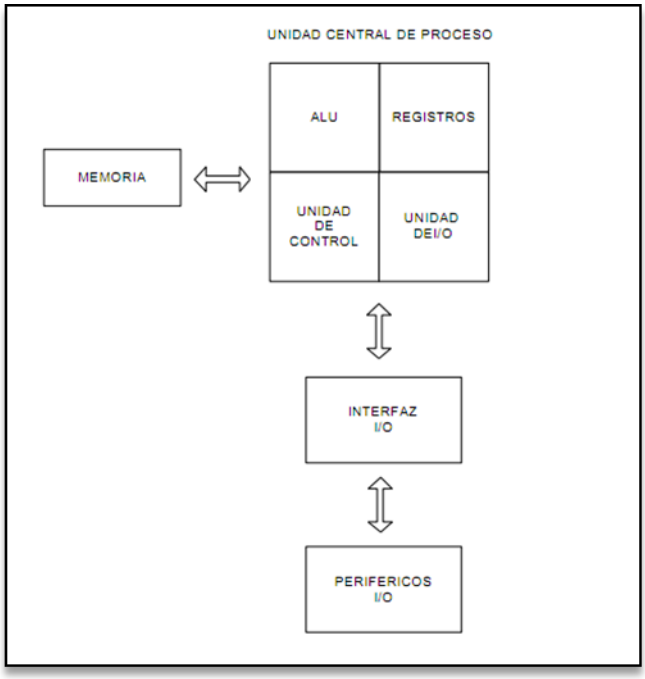


Figura 1. Estructura general de un microprocesador

1.7 *MICROCONTROLADOR*

Los microcontroladores son circuitos integrados en un chip que cuentan con un microprocesador o unidad de procesamiento central (CPU), una memoria RAM, memorias no volátiles (ROM, PROM, EPROM), puertos de entrada y salida. A diferencia de microprocesadores de propósito general, como los que se usan en las computadoras PC, los microcontroladores son unidades autosuficientes y más económicas.

El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria, éste puede escribirse en distintos lenguajes de programación. Además la mayoría de los microcontroladores actuales pueden reprogramarse repetidas veces.
Por las características mencionadas y su alta flexibilidad, los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan máquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, domótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

1.7.1 CARACTERÍSTICAS DE LOS MICROCONTROLADORES

- **Unidad de Procesamiento Central (CPU):** Típicamente de 8 bits, pero también los hay de 4, 32 y hasta 64 bits con arquitectura Harvard, con memoria/bus de datos separada de la memoria/bus de instrucciones de programa, o arquitectura de Von Neumann con memoria/bus de datos y memoria compartidas.
- **Memoria de programa:** Es una memoria ROM, EPROM, EEPROM o Flash que almacena el código del programa que típicamente puede ser de 1 kilobyte a varios megabytes.
- **Memoria de datos:** Es una memoria RAM que típicamente puede ser de 1, 2, 4, 8, 16, 32 kilobytes.
- **Generador de reloj:** Usualmente un cristal de cuarzo de frecuencias que genera una señal oscilatoria entre 1 a 40 MHz, o también resonadores o circuitos RC.
- **Interfaz de Entrada/Salida:** Puertos paralelos, seriales, interfaces de periféricos, red de área de controladores (CAN), USB.
- **Otras opciones:**
  - **Conversor Analógico-Digital:** convierte un nivel de voltaje en un cierto pin a valor digital manipulable por el programa del microcontrolador.
  - **Moduladores por Ancho de Pulso (PWM):** para generar ondas cuadradas de frecuencia fija pero con ancho de pulso modificable.

La alta integración de subsistemas que componen un microcontrolador reduce el número de chips, la cantidad de pistas y espacio que se requeriría en un circuito impreso si se implementará en un sistema equivalente usando chips separados.

1.7.2 ARQUITECTURA DE UN MICROCONTROLADOR

Según a arquitectura interna de la memoria de un microcontrolador se puede clasificar considerando como el CPU accede a los datos de instrucciones, en 2 tipos:

1.7.3 ARQUITECTURA VON NEUMANN

Fue desarrollada por Jon Von Neumann, se caracteriza por tener una sola memoria principal donde se almacenan datos e instrucciones de forma distinta. La CPU se conecta a través de un sistema de buses (direcciones, datos y control). Esta arquitectura es limitada cuando se demanda rapidez. En la Figura 2 se muestra la estructura general de la arquitectura Von Neumann.

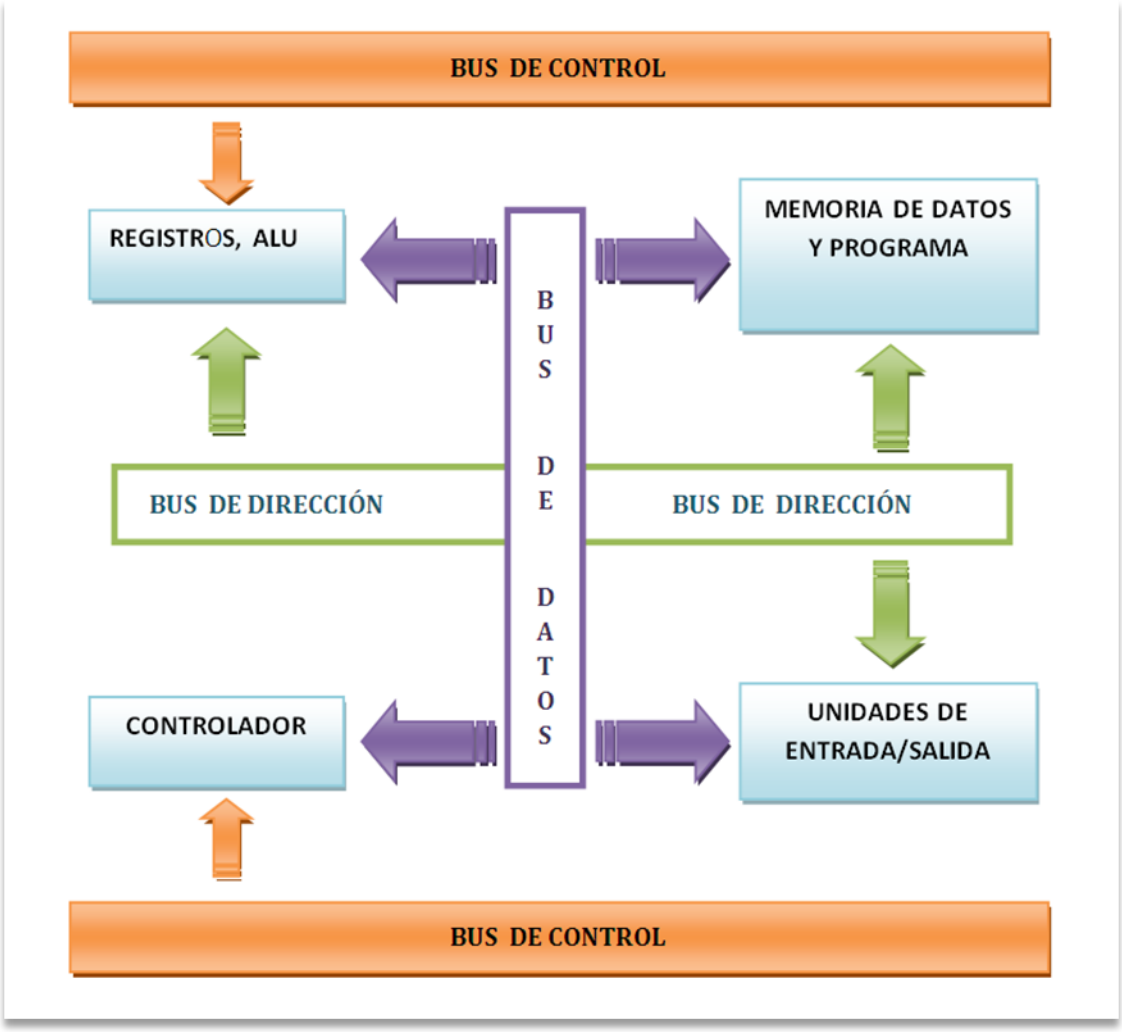


Figura 2. Arquitectura Von Neumann

1.7.4 ARQUITECTURA HARVARD

Fue desarrollado en Harvard, por Howard Aiken, esta arquitectura se caracteriza por tener dos memorias independientes una que contiene sólo instrucciones y otra que contiene sólo datos. Ambas, disponen de sus respectivos sistemas de buses para el acceso y es posible realizar operaciones de acceso simultáneamente en ambas memorias, como se muestra en la Figura 3.

Las principales ventajas de esta arquitectura son:

- El tamaño de las instrucciones no está relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.

- El tiempo de acceso a las instrucciones puede superponerse con el de los datos logrando una mayor velocidad de operación

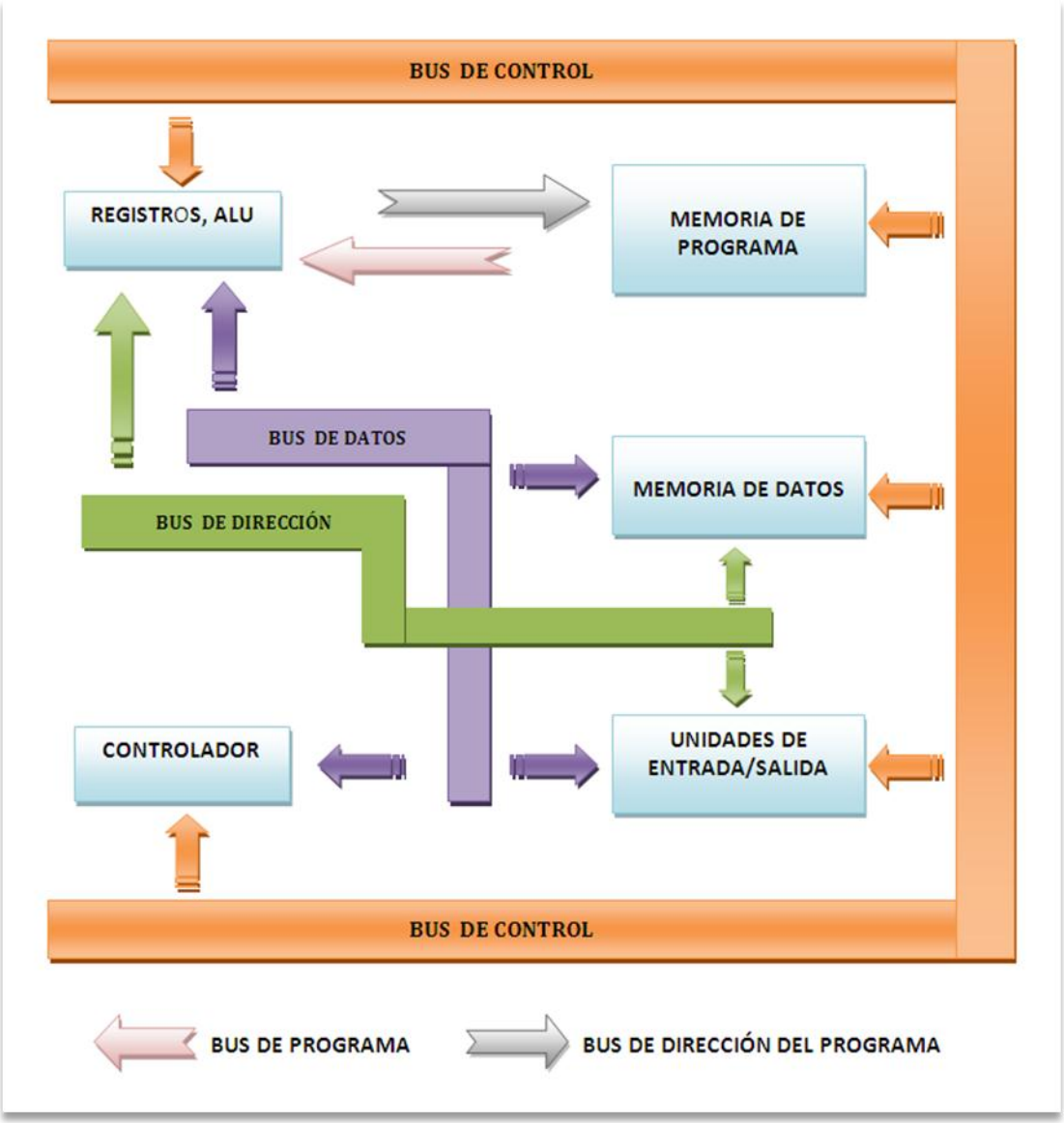


Figura 3. Arquitectura Harvard

1.8 *FPGA*

Es un Arreglo de Compuertas Reconfigurables (Fiel Programable Gate Array) es decir es un circuito diseñado para que sea configurado por el cliente o diseñador después de su manufactura. La configuración del FPGA es especificada usando un lenguaje de descripción de hardware (HDL) antes se utilizaban diagramas de circuitos para especificar la configuración. La habilidad de actualizar su funcionalidad, reconfigurarse y el bajo costo son grandes ventajas para muchas aplicaciones. Los componentes básicos de un FPGA se muestran en la Figura 5.

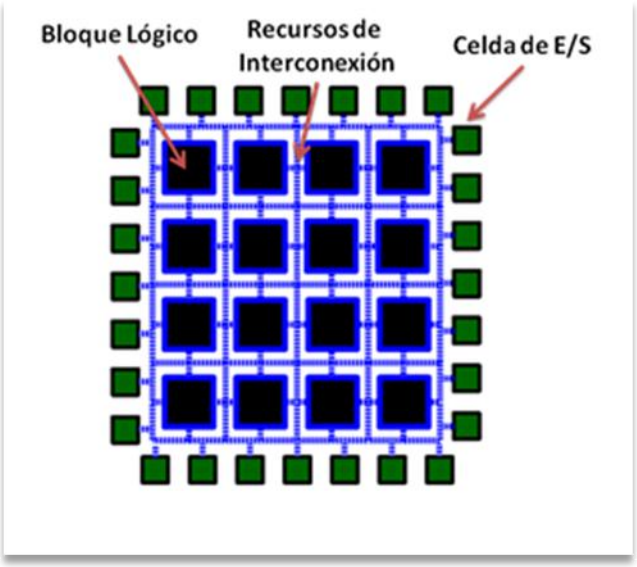


Figura 5. Componentes de un FPGA

1.9 *PUERTOS DE ENTRADA/SALIDA*

Los puertos de entrada/salida permiten realizar transferencias de información entre el exterior y el microprocesador. Existen dos modos de transferencia:

- **Paralelo:** el puerto utiliza un conjunto de líneas, tantas como bits a transmitir simultáneamente, por las que cada una pasa un bit en el intervalo de tiempo

- **Serie:** el puerto utiliza una única línea por la que, los intervalos de tiempos diferentes, se transmiten, uno a uno, todos los bits de dato.

1.10 *MICROCONTROLADOR PIC18F4550*

1.10.1 DESCRIPCIÓN

Familia PIC18XXXXX

- Arquitectura RISC avanzada Harvard: 16 bit con 8 bit de datos
- 77 instrucciones
- Desde 18 a 80 pines
- hasta 64 Kb de programa (2 Mbytes en ROMless)
- Multiplicador Hardware 8x8
- Hasta 3968 bytes de RAM y 1 Kb de EEPROM
- Frecuencia máxima de reloj 40 MHz Hasta 10 MIPS
- Pila de 32 niveles
- Múltiples fuentes de interrupción
- Periféricos de comunicación avanzados (CAN y USB)

CARACTERÍSTICAS	PIC18F4550
Frecuencia de Operación	Hasta 48 MHz
Memoria de Programa (bytes)	32
Memoria RAM de Datos (bytes)	2048
Memoria EEPROM Datos (bytes)	256
Interrupciones	20
Líneas de E/S	35
Temporizadores de 8 bits	1
Temporizadores de 16 bits	3
Módulos de Comparación/Captura/PWM (CCP)	1
Módulos de Comparación/Captura/PWM mejorado (ECCP)	1
Canales de Comunicación Serie	MSSP, EUSART
Canal USB	1
Puerto Paralelo de Transmisión de Datos (SPP)	1
Canales de Conversión A/D de 10 bits	13 Canales
Comparadores analógicos	2
Juego de instrucciones	75 (83 ext.)
Encapsulados	PDIP 40 pines QFN 40 pines TQFP 40 pines

Tabla 1. Características del Microcontrolador PIC18F4550

1.10.2 DIAGRAMA DE PINES

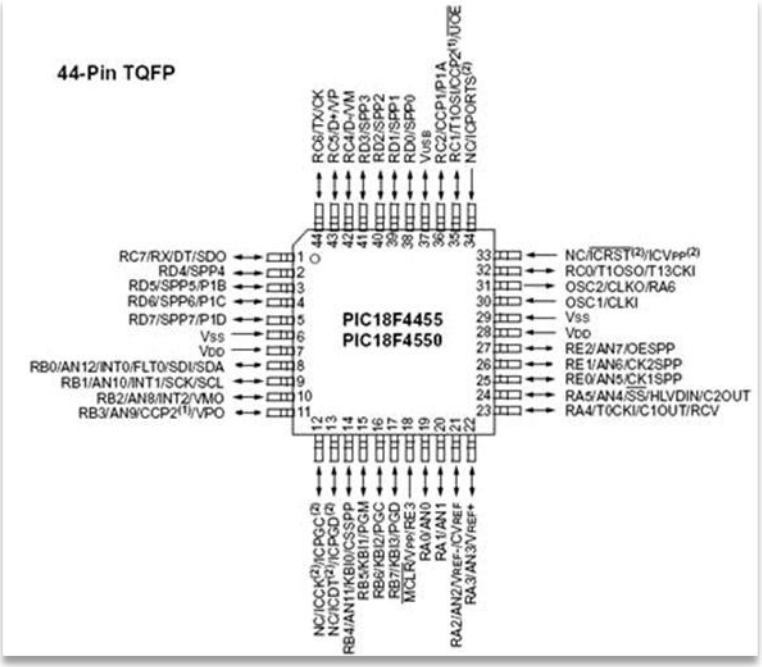


Figura 6. Diagrama de Pines del Microcontrolador PIC18F4550

1.10.3 DIAGRAMA DE BLOQUES

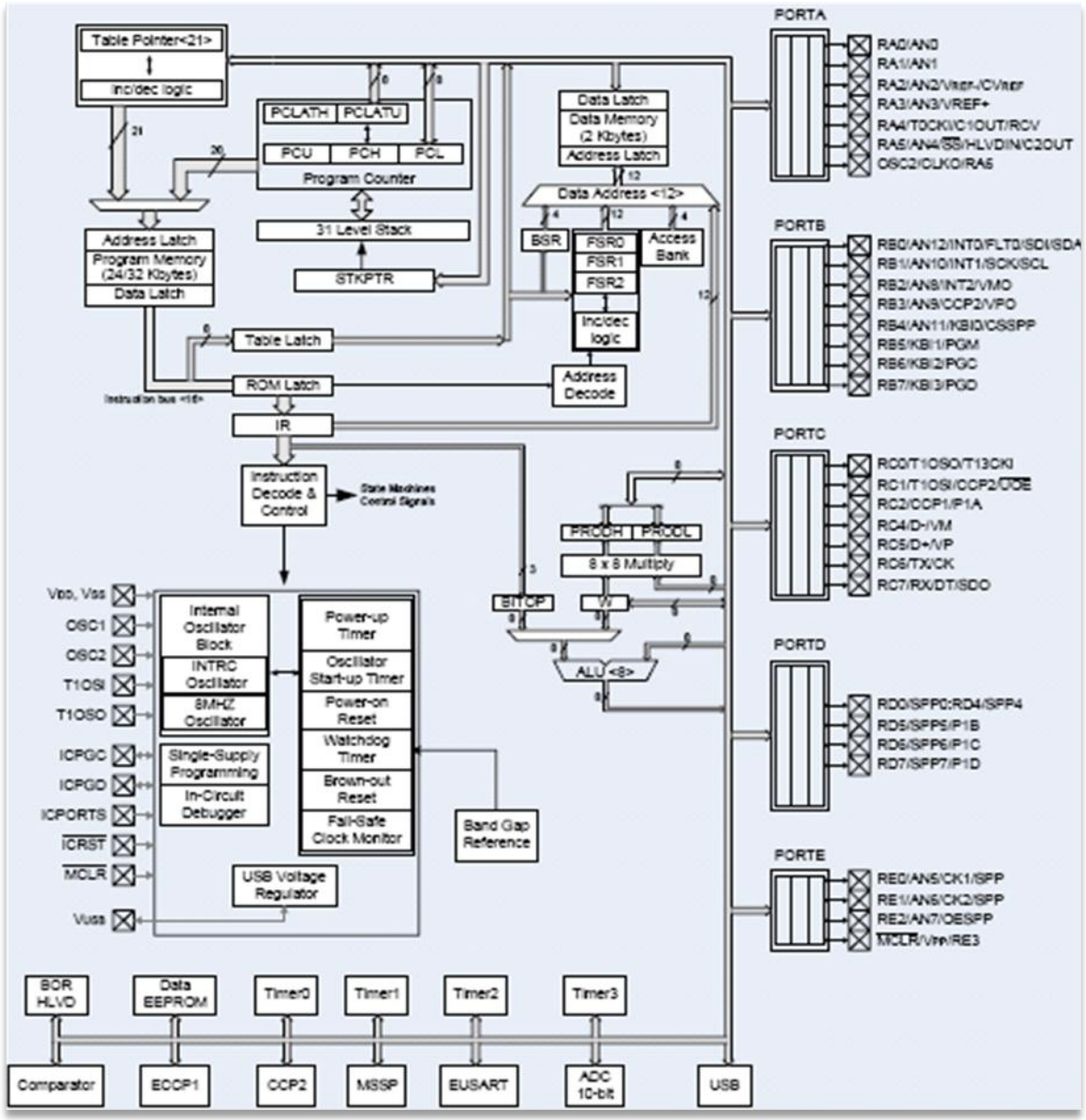


Figura 7. Diagrama de bloques del Microprocesador PIC18F4550

1.10.4 ORGANIZACIÓN DE LA MEMORIA

El microcontrolador PIC18F4550 dispone de las siguientes memorias:

- Memoria de programa: memoria flash interna de 32 bytes
  - Almacena instrucciones, constantes y datos.

  - Puede ser escrita o leída mediante un programador externo o durante la ejecución del programa mediante unos punteros.
- Memoria RAM de datos: memoria SRAM interna de 2048 bytes en la que están incluidos los registros de función especial.
  - Almacena datos de forma temporal durante la ejecución del programa
  - Puede ser escrita o leída en tiempo de ejecución mediante diversas instrucciones
- Memoria EEPROM de datos: memoria no volátil de 256 bytes.
  - Almacena datos que se deben conservar aun en ausencia de tensión de alimentación
  - Puede ser escrita o leída en tiempo de ejecución a través de registros
- Pila: bloque de 31 palabras de 21 bits
  - Almacena la dirección de la instrucción que debe ser ejecutada después de una interrupción o subrutina
- Memoria de configuración: memoria en la que se incluyen los bits de configuración (12 bytes de memoria flash) y los registros de identificación (2 bytes de memoria de sólo lectura)

1.10.5 MEMORIA DE CONFIGURACIÓN

Se trata de un bloque de la memoria situado a partir de la posición 30000H de memoria de programa. En esta memoria de configuración se incluyen:

- Bits de configuración: contenidos en 12 bytes de memoria flash, permiten la configuración de algunas opciones del microcontrolador como:

  - Opciones del oscilador
  - Opciones del reset
  - Opciones del watchdog
  - Opciones de la circuitería de depuración y programación
  - Opciones de protección contra escritura de memoria de programa y memoria EEPROM de datos

Estos bits se configuran generalmente durante la programación del microcontrolador, aunque también pueden ser leídos y modificados durante la ejecución del programa.

- Registros de identificación: se trata de dos registros situados en las direcciones 3FFFFEH y 3FFFFFH que contienen información del modelo y revisión del dispositivo. Son registros de sólo lectura y no pueden ser modificados por el usuario.

1.10.6 ARQUITECTURA HARVARD

El microcontrolador dispone buses diferentes para el acceso a memoria de programa y memoria de datos:

- Bus de la memoria de programa:
  - 21 líneas de dirección
  - 16/8 líneas de datos (16 líneas para instrucciones/8 líneas para datos)

- Bus de memoria de datos:
  - 12 líneas de dirección
  - 8 líneas de datos

1.10.7 INTERRUPCIONES

Las interrupciones en el microcontrolador pueden darse de varios tipos:

- Interrupciones externas
- Interrupciones por desbordamiento del contador
- Interrupciones de EUSART
- Interrupciones USB
- Interrupciones del CAD
- Interrupciones por periféricos externos

El microcontrolador puede tener varias interrupciones programadas a la vez, pero hay que tener en cuenta que una vez entra en una rutina de interrupción, el microcontrolador no puede acceder a otra interrupción hasta que la rutina de interrupción que se está ejecutando finalice.

1.10.8 TEMPORIZADOR

El PIC18F4550 tiene 4 temporizadores, de los cuales uno de ellos es de 8 bits y el resto de una precisión de 16 bits. En la Figura 8 se muestra el diagrama de bloques referente al Timer0.

Resolución de los temporizadores:

Timer0 -> Temporizador configurable de 8 ó 16 bits.
Timer1 -> Temporizador de 16 bits.
Timer2 -> Temporizador de 8 bits.
Timer3 -> Temporizador de 16 bits

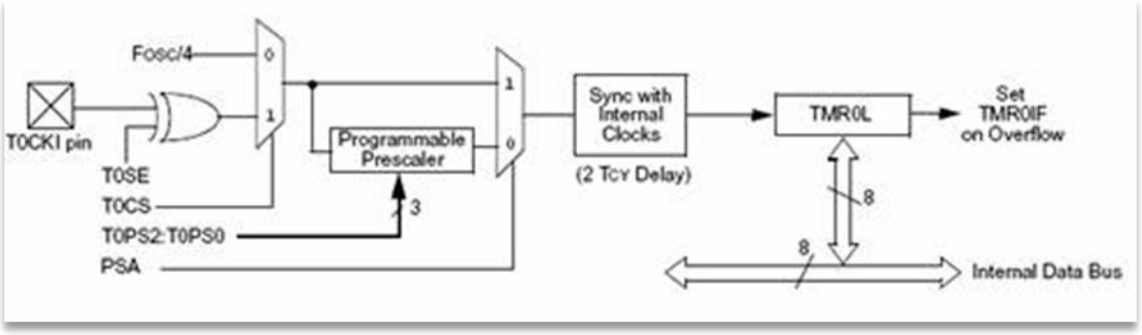


Figura 8. Diagrama de bloques Timer0 de 8 bits

Existe la posibilidad de activar un preescaler en los temporizadores de forma que se pueda alargar la duración del temporizador, dependiendo del temporizador puede ser de 2, 4, 8 e incluso 16, en la Figura 9 se muestra el diagrama de bloques de Timer1 de 16 bits.

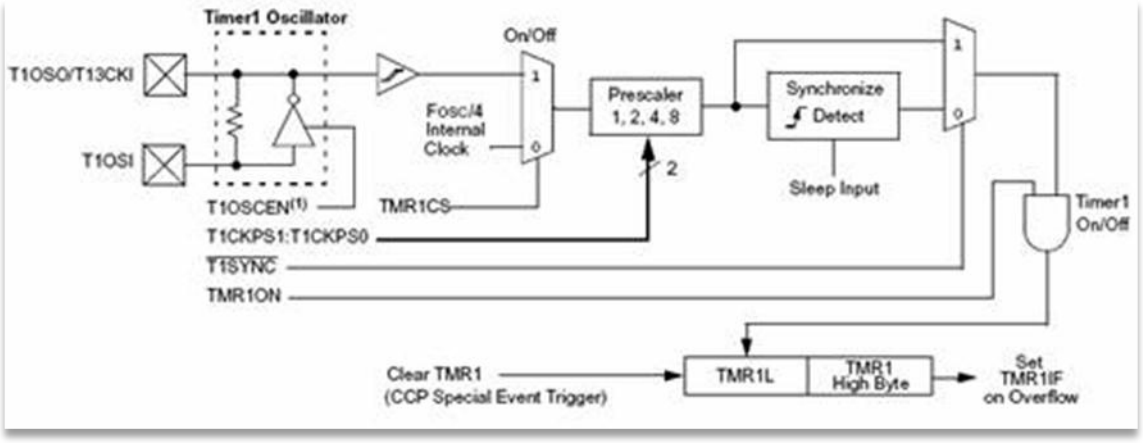


Figura 9. Diagrama de bloques de Timer1 de 16 bits

1.10.9 MÓDULO CPP

Son módulos (CCP1 y CCP2) con idéntico funcionamiento:

- Registro de captura de 16 bits
- Registro de comparación de 16 bits
- Registro de ciclo de trabajo PWM

Módulo CCP1

- Consta de dos registros de 8 bits: CCPR1H y CCPR1L
- Registro de control: CPP1CON
- Acción especial: Generada mediante una comparación, reinicia el Timer1

Módulo CCP2

- Consta de dos registros de 8 bits: CCPR2H y CCPR2L
- Registro de control: CPP2CON
- Acción especial: Generada mediante una comparación, reinicia el Timer1. Puede lanzar una conversión A/D.

1.10.10 TIMERS

Los temporizadores son contadores que al activarlos empiezan una cuenta y cuando esta cuenta se acaba se activa la bandera de interrupción por el temporizador, entrando el microcontrolador en la rutina de interrupción de temporizador Timer0 es un dispositivo que puede funcionar conceptualmente de dos formas: como contador de pulsos externos o como temporizador para calcular intervalos de tiempo, también es utilizado para generar interrupciones periódicas a través de una cuenta programada.

1.10.11 MÓDULO CONVERTIDOR ANALÓGICO/DIGITAL

El conversor ADC (Analog-to-Digital Converter - Conversor Analógico Digital) tiene que efectuar los siguientes procesos:

1. **Muestreo de la señal analógica.** : consiste en tomar diferentes muestras de tensiones o voltajes en diferentes puntos de la onda senoidal. La frecuencia a la que se realiza el muestreo se denomina razón, tasa o también frecuencia de muestreo y se mide en kilohertz (kHz)

2. **Cuantización de la propia señal**: representa el componente de muestreo de las variaciones de valores de tensiones o voltajes tomados en diferentes puntos de la onda sinusoidal, que permite medirlos y asignarles sus correspondientes valores en el sistema numérico decimal, antes de convertir esos valores en sistema numérico binario.

3. **Codificación del resultado de la cuantización, en código binario**: permite asignarle valores numéricos binarios equivalentes a los valores de tensiones o voltajes que conforman la señal eléctrica analógica original. [3]

El convertidor A/D (CAD) del PIC18F4550, posee las siguientes características:

- Microchip PIC18F4550 contiene 13 convertidores analógicos digitales, los cuales pueden ser seleccionados en modos de resolución de 8 ó 10 bits, para ello antes habrá que configurar las entradas en modo CAD, pues ya que estas están por defecto como I/O.
- Señal de reloj de conversión configurable.
- Tiempo de adquisición programable de 0-20 TDA.
- Posibilidad de establecer un rango de tensiones de conversión mediante tensiones de referencia externas. Se recomienda que la máxima resistencia de entrada (Rs) sea de 2.5K, pues sino la conversión no sería del todo fiable, por lo que habría que hacer una adaptación de impedancias entre las partes.

En la Figura 10 se muestra el diagrama de bloques del módulo A/D. [4]

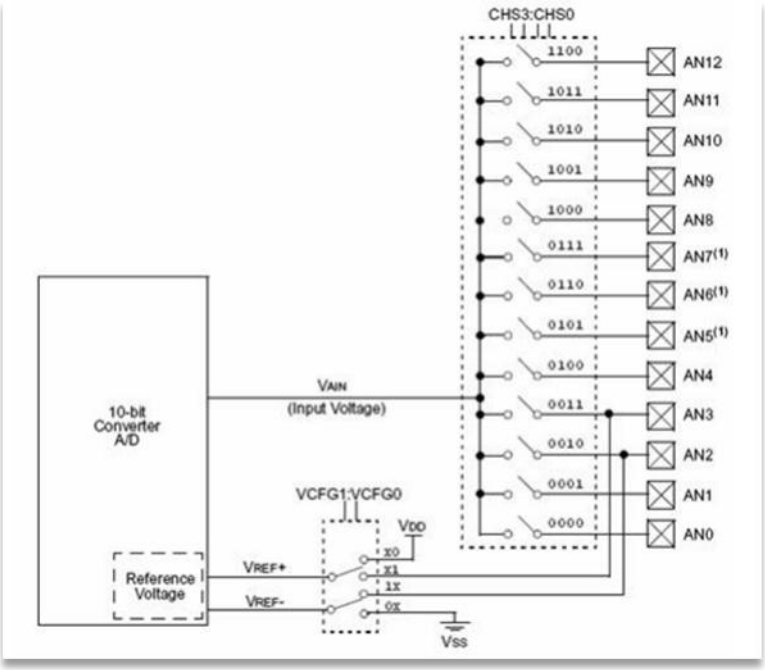


Figura 10. Diagrama de bloques del módulo A/D

1.11 *COMUNICACIÓN SERIAL*

La principal característica de la comunicación en serie es que los bits en la comunicación serie se reciben uno detrás de otro o "en serie".

1.11.1 RS-232:

RS-232 es un protocolo estándar de comunicación binaria serie entre un DTE (Equipo terminal de datos) y un DCE (Equipo de Comunicación de Datos).
La norma define las características eléctricas y sincronización de las señales, el significado de las señales, el tamaño físico y los pines de los conectores. La versión actual de la norma es TIA-232-F publicado en 1997. El puerto RS-232 fue un estándar para las computadoras personales para las conexiones a módems, impresoras, ratones, almacenamiento de datos, sin fuentes de alimentación externa, y otros dispositivos periféricos. Sin embargo, la velocidad de transmisión limitada, una oscilación de voltaje relativamente grande, y grandes conectores provocó que se creara el bus serie universal (USB) que ha desplazado a RS-232 usando lo mejor de ello.

1.11.2 PROTOCOLO TRANSMISIÓN RS232

El protocolo RS232 es una norma por medio del cual se estandarizan las velocidades de transferencia de datos. La norma RS232 fue definida para conectar una computadora a un módem. Además de transmitirse los datos de una forma serie asíncrona son necesarias una serie de señales adicionales que se definen en la norma. [1]

Consiste en un conector DB-25 de 25 pines, sin embargo es normal encontrar la versión de 9 pines DB-9 utilizado por cierto tipo de periféricos (como el ratón serie del PC). Los pines más importantes del conector DB9 y DB25 se observan en las Tablas 2 y 3 respectivamente.

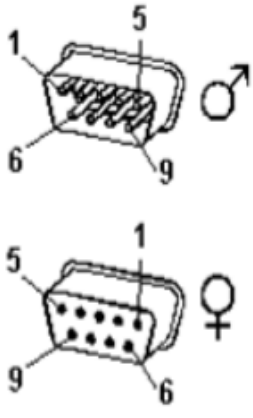
#	Pin	E/S	Función	Conector DB 9
1			Tierra de Chasis	
2	RXD	E	Recibir Datos	
3	TXD	S	Transmitir Datos	
4	DTR	S	Terminal de Datos Listo	
5	SG		Tierra de señal	
6	DSR	E	Equipo de Datos Listo	
7	RTS	S	Solicitud de Envío	
8	CTS	E	Libre para Envío	
9	RI	S	Timbre Telefónico	

Tabla 2. Pines más importantes del conector DB9

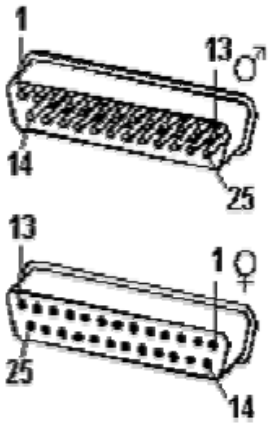
#	Pin	E/S	Función	Conector DB 25
1			Tierra de Chasis	
2	TXD	S	Transmitir Datos	
3	RXD	E	Recibir Datos	
4	RTS	S	Solicitud de Envío	
5	CTS	E	Libre para Envío	
6	DSR	E	Equipo de Datos Listo	
7	SG		Tierra de señal	
8	CD/DCD	E	Detector de Portadora	
15	TxC	S	Transmitir Reloj	
17	RxC	E	Recibir reloj	
20	DTR	S	Terminal de Datos Listo	
22	RI	S	Timbre Telefónico	
24	RTxC	S/E	Transmitir/Recibir Reloj	

Tabla 3. Pines más importantes del conector DB25

Las señales con las que trabaja este puerto serial son digitales, de +12V (0 lógico) y -12V (1 lógico), para la entrada y salida de datos. El estado de reposo en la entrada y salida de datos es de -12V. Cada pin puede ser de entrada o de salida, teniendo una función específica cada uno de ellos. Las señales TXD, DTR y RTS son de salida, mientras que RXD, DSR, CTS son de entrada. La Tierra de referencia para todas las señales es SG (Tierra de Señal). [2]

1.11.3 USB

USB (Universal Serial Bus) es un estándar desarrollado en la década de 1990 que definen los cables, conectores y protocolos usados para la conexión, comunicación y suministro de energía entre computadoras y dispositivos electrónicos.
USB fue diseñado para estandarizar la conexión de periféricos, como teclados, dispositivos de señalización, cámaras digitales, impresoras, reproductores multimedia portátiles, unidades de disco y adaptadores de red para computadoras. Se ha convertido en algo común en otros dispositivos, como teléfonos inteligentes, PDAs y consolas de videojuegos. USB ha sido sustituida por una variedad de interfaces anteriores, como puerto serie y paralelo, así como distintos cargadores de energía para dispositivos portátiles.

1.12 *CIRCUITO MAX232*

Este circuito integrado es usado para comunicar un microcontrolador o sistema digital con una PC o sistema basado en el bus serie RS232. [1]
El circuito integrado posee dos conversores de nivel TTL a RS232 y otros dos que a la inversa, convierten de RS232 a TTL. [5]
Estos conversores son suficientes para manejar las cuatro señales más utilizadas del puerto serie del PC, que son TX, RX, RTS y CTS; estas dos últimas son las usadas para el protocolo handshaking pero no es imprescindible su uso. Para que el MAX232 funcione correctamente deberemos de poner unos condensadores externos, todo esto lo podemos ver en la siguiente

Figura 11 en la que sólo se han cableado las líneas TX y RX que son las usualmente más usadas para casi cualquier aplicación. [1]

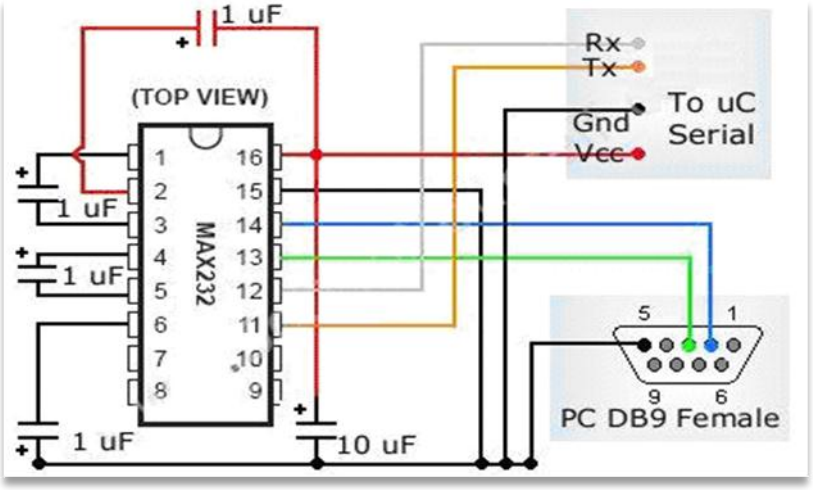


Figura 11.

Diagrama de conexión del circuito MAX*232*

1.13 *BOOTLOADER*

Es un programa guardado en la memoria flash del microcontrolador, el cual se ejecuta cuando el microcontrolador es encendido. Este programa es responsable de inicializar el microcontrolador y copiar el programa de memoria flash a la memoria RAM del microcontrolador e iniciar su ejecución.
En caso de ser activado un pin específico en el microcontrolador el bootloader cambiará su modo de operación haciendo que el programa se reciba directamente del puerto serie o USB cargándolo en la memoria flash del microcontrolador, con esto se facilita la programación del microcontrolador debido a que ya no es necesario grabar el programa con un programador diseñado especialmente para el microcontrolador.

1.14 *MOTOR*

Un motor es un dispositivo capaz de transformar energía como la eléctrica, combustibles de fósiles, etc., en energía mecánica para realizar un trabajo. Existen diversos tipos, siendo los más comunes los siguientes:

- **Motor térmico:** transforma energía térmica en trabajo mecánico por medio del aprovechamiento del gradiente de temperatura entre una fuente de calor (foco caliente) y un sumidero de calor (foco frío).
- **Motor de combustión interna:** es un tipo de máquina que obtiene energía mecánica directamente de la energía química producida por un combustible que arde dentro de una cámara de combustión.

- **Motor de combustión externa:** es una máquina que realiza una conversión de energía calorífica en energía mecánica mediante un proceso de combustión que se realiza fuera de la máquina, generalmente para calentar agua que, en forma de vapor, será la que realice el trabajo
- **Motor eléctrico:** Es una máquina eléctrica que transforma energía eléctrica en energía mecánica por medio de interacciones electromagnéticas. Algunos de los motores eléctricos son reversibles, pueden transformar energía mecánica en energía eléctrica funcionando como generadores.

1.15 *ELECTRÓNICA PARA CONTROL DE MOTORES*

1.15.1 PUENTE H

Un puente H es un dispositivo electrónico el cual permite a un motor eléctrico DC girar en ambos sentidos, *avance* y *retroceso.*

1.15.2 L298:

Controlador para 2 motores de corriente directa puede proveer hasta un ampere a cada motor; Controla la dirección, la habilitación y el freno de cada motor, se puede observar en la Figura 12.

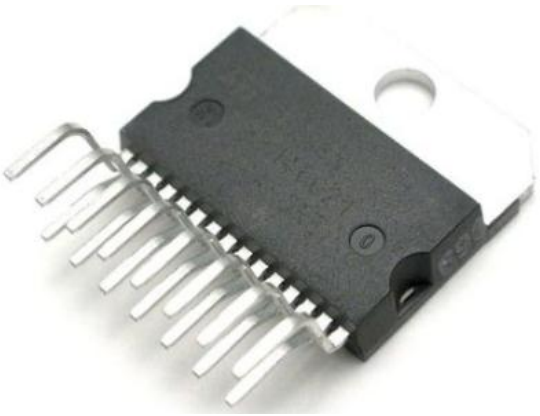


Figura 12. Controlador L298

1.15.3 TEORÍA PWM

La modulación por anchura de pulsos (PWM) es una técnica en la que se modifica el ciclo de trabajo de una señal periódica para, entre otras cosas, variar la velocidad de un motor. Implica obtener una serie de pulsos de igual anchura en cada medio ciclo; este tren de pulsos, en realidad hace que el motor marche alimentado por la tensión máxima de la señal durante el tiempo en que esta se encuentra en estado alto y que pare en los tiempos en que la señal está en estado bajo.{6}

1.16 *SENSORES*

Un sensor es un dispositivo que detecta variaciones en una magnitud física y las convierte en señal útil para un sistema de medida o control.

TIPOS DE SENSOR

Sensores de posición*:* Su función es medir o detectar la posición de un determinado objeto en el espacio, dentro de este grupo, podemos encontrar los siguientes tipos de captadores:

- **Los captadores fotoeléctricos***:* este tipo de sensores, se encuentra basado en la emisión de luz, y en la detección de esta emisión realizada por los fotodetectores. Según la forma en que se produzca esta emisión y detección de luz, podemos dividir este tipo de captadores en captadores por barrera o captadores por reflexión.

  - ✓ Captadores por barrera. Estos detectan la existencia de un objeto, porque interfiere la recepción de la señal luminosa.

  - ✓ Captadores por reflexión. La señal luminosa es reflejada por el objeto, y esta luz reflejada es captada por el captador fotoeléctrico, lo que indica al sistema la presencia de un objeto.

Sensores de contacto: Estos dispositivos, son los más simples, ya que son interruptores que se activan o desactivan si se encuentran en contacto con un objeto, por lo que de esta manera se reconoce la presencia de un objeto en un determinado lugar.

Captadores de circuitos oscilantes: se encuentran basados en la existencia de un circuito en el mismo que genera una determinada oscilación a una frecuencia prefijada, cuando en el campo de detección del sensor no existe ningún objeto, el circuito mantiene su oscilación de un manera fija, pero cuando un objeto se encuentra dentro de la zona de detección del mismo, la oscilación deja de producirse, por lo que el objeto es detectado. Estos tipos de sensores son muy utilizados como detectores de presencia, ya que al no tener partes mecánicas, su robustez al mismo tiempo que su vida útil es elevada.

Sensores por ultrasonidos: se basa en el mismo funcionamiento que los de tipo fotoeléctrico, ya que se emite una señal, esta vez de tipo ultrasónica, y esta señal es recibida por un receptor.

Captadores de esfuerzos: este tipo de captadores, se encuentran basados en su mayor parte en el empleo de galgas extenso métricas , que son unos dispositivos que cuando se les aplica una fuerza, ya puede ser una tracción o una compresión, varia su resistencia eléctrica, de esta forma podemos medir la fuerza que se está aplicando sobre un determinado objeto.

Sensores de movimientos: este tipo de sensores es uno de los más importantes en robótica, ya que da información sobre las evoluciones de las distintas partes que forman el robot, y de esta manera podemos controlar con un grado de precisión elevada la evolución del robot en su entorno de trabajo. Dentro de este tipo de sensores podemos encontrar los siguientes:

- ✓ Sensores de deslizamiento. Se utiliza para indicar al robot con que fuerza ha de coger un objeto para que este no se rompa al aplicarle una fuerza excesiva, o por el contrario que no se caiga de las pinzas del robot por no sujetarlo debidamente.
- ✓ Sensores de velocidad. Estos sensores pueden detectar la velocidad de un objeto tanto sea lineal como angular, pero la aplicación más conocida de este tipo de sensores es la medición de la velocidad angular de los motores que mueven las distintas partes del robot.

qrd1114: Es un dispositivo electrónico en el cual su encapsulado contiene un emisor infrarrojo y un receptor, con las siguientes características:

- ➢ Salida del fototransistor
- ➢ No necesita contacto con la superficie para sensar
- ➢ Desenfocado para sensar superficies difusas

- Encapsulado compacto
- Filtro de luz solar para minimizar el ruido [7]

El esquema del dispositivo se puede observar en la Figura 13, la distribución de los pines se muestra en la Tabla 4.

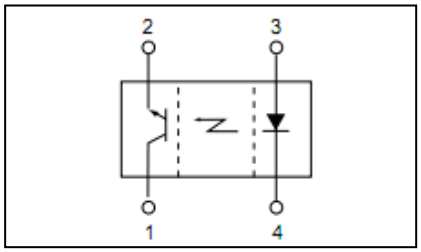


Figura 13. Esquema del dispositivo qdr1114

PIN 1: Colector	PIN 3: ÁNODO
PIN 2: Emisor	PIN 4: CÁTODO

Tabla 4. Distribución de los pines del dispositivo qdr1114

CAPÍTULO II:

“MARCO TEÓRICO SOFTWARE”

2. MARCO TEÓRICO SOFTWARE

2.1 *MÁQUINA DE ESTADOS*

Se denomina máquina de estados a un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas del sistema dependen de las entradas del estado actual y de las entradas anteriores, esto es logrado gracias a la abstracción de estados. El estado siguiente de un estado actual depende solamente de las entradas del estado actual y a su vez el estado actual es determinado por todos los estados anteriores y las entradas de todos los estados anteriores. Permitiendo así que el sistema tenga memoria.

2.1.1 CARTAS ASM

La Máquina del Estado Algorítmica (ASM) es un método para el diseño de máquinas de estados finitos. Se utiliza para representar diagramas de circuitos integrados digitales. La carta ASM es como un diagrama de estados, pero menos formal y por lo tanto más fácil de entender, su simbología se muestra en la Tabla 5. Una carta ASM es un método de descripción de las operaciones secuenciales de un sistema digital.

2.1.1.1 MÉTODO ASM

El método de ASM se compone de los siguientes pasos:

1. Crear un algoritmo, utilizando pseudocódigo, para describir la operación deseada del dispositivo.
2. Convertir el pseudocódigo en una carta ASM.
3. El diseño del flujo de datos basado en la carta ASM.
4. Crear un gráfico detallado ASM sobre la base de del flujo de datos.
5. Diseño de la lógica de control basada en la carta ASM.

NOMBRE DEL ESTADO	DESCRIPCIÓN

	Encerrado en un círculo, indica el nombre del estado actual.
CAJA DE ESTADOS	
	Adentro del rectángulo se indican las posibles salidas.
CAJA CONDICIONAL	
	Adentro del rombo existe una expresión condicional la cual se evaluara a verdadero o falso. Dependiendo del resultado el flujo de datos cambiará y se llegara a un nuevo estado.
CAJA DE SALIDAS CONDICIONALES	
	Van inmediatamente después de la condición. Indica la salida del estado condicional dependiendo de lo que valuó la expresión condicional.

Tabla 5. Simbología para la construcción de las cartas ASM

2.1.2 DIAGRAMAS DE FLUJO

Un diagrama de flujo es una representación gráfica de la secuencia de pasos que se realizan para obtener un cierto resultado. Este puede ser un producto, un servicio o bien una combinación de ambos.

2.1.2.1 CARACTERÍSTICAS

- **Capacidad de comunicación:** permite la puesta en común de conocimientos individuales sobre un proceso, y facilita la mejor comprensión global del mismo.

- **Claridad:** proporciona información sobre los procesos de forma clara, ordenada y concisa.

2.1.2.2 SIMBOLOGÍA

Los principales símbolos para la construcción de diagramas de flujo se muestran y se describen en la Tabla 6.

SÍMBOLO GRÁFICO	DESCRIPCIÓN
	Este se utiliza para representar el inicio o el fin de un algoritmo. También puede representar una parada o una interrupción programada.
	Se utiliza para un proceso determinado, se usa comúnmente para representar una instrucción, o cualquier tipo de operación que origine un cambio de valor.
	Se utiliza para representar una entrada o salida de información, que sea procesada o registrada por medio de un periférico.
	Usado para la toma de decisiones, ramificaciones, para indicación de operaciones lógicas o comparación entre datos.
	Enlaza dos partes cualesquiera de un diagrama a través de un conector de salida y un conector de entrada. Forma un enlace en la misma página del diagrama

Tabla 6. Simbología para la construcción de Diagramas de Flujo

2.2 *ELEMENTOS PARA EL ESTUDIO DE LOS COMPILADORES*

Un compilador es un programa que recibe como entrada un programa escrito en un lenguaje de nivel medio o superior (programa fuente) y lo transforma a su equivalente en lenguaje ensamblador (código objeto), e inclusive hasta lenguaje máquina (programa ejecutable) pero sin ejecutarlo. La forma de cómo llevará a cabo tal traducción es el objetivo central de un compilador, su estructura general se muestra en la Figura 14.

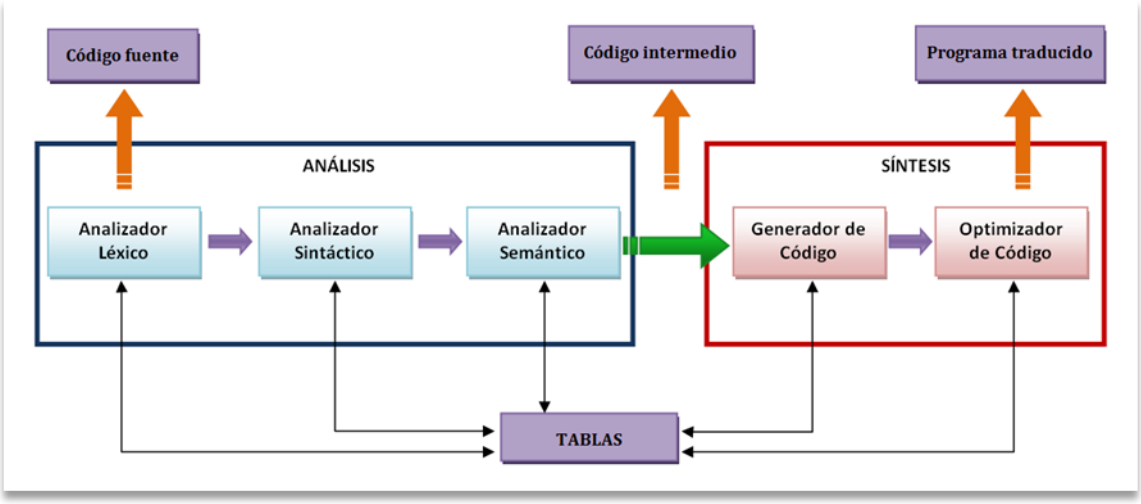


Figura 14. Estructura general de un compilador

2.2.1 ANALIZADOR LÉXICO

Etapa en donde se identifican los componentes que tienen un significado colectivo, su estructura general se muestra en la Figura 15.

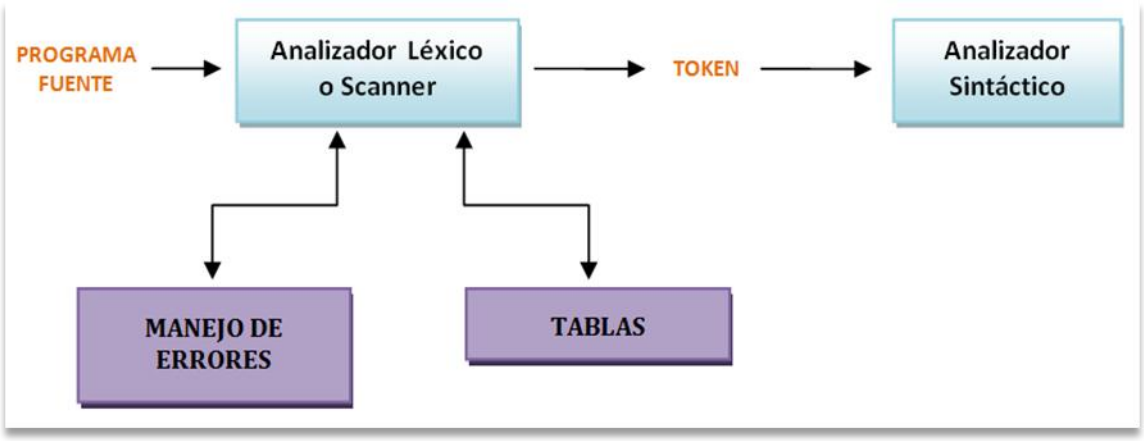


Figura 15. Estructura general de un Analizador Léxico

Un analizador léxico es un **AFD** que revisa gramáticas regulares.

Funciones de un analizador léxico

- Crear tokens, crear y actualizar la tabla de símbolos
- Descartar comentarios
- Identificar componentes léxicos
- Recuperación de errores y continuar con la revisión de errores
- Ser robusto: no debe fallar por errores en el programa fuente
- Debe leer carácter por carácter o por línea

- Debe identificar el **EOF**
- Expresión regular para operaciones aritméticas y operaciones

2.2.2 ANALIZADOR SINTÁCTICO

Es la fase del analizador que se encarga de checar el texto de entrada en base a una gramática dada, y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce, se observa su estructura general en la Figura 16.

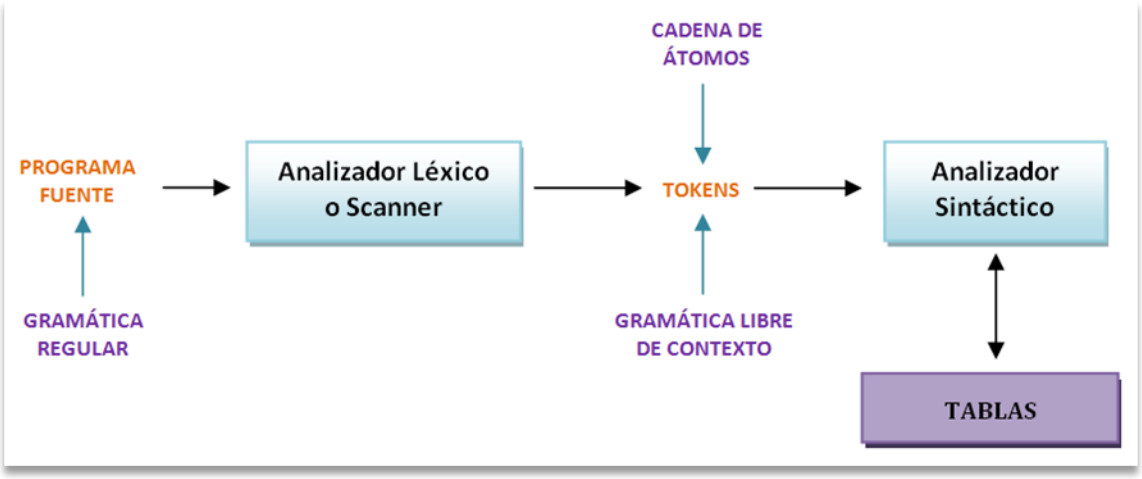


Figura 16. Estructura general de un Analizador Sintáctico

La salida del analizador sintáctico es alguna representación del árbol sintáctico que reconoce la secuencia de tokens suministrada por el analizador léxico.

Funciones de un analizador sintáctico

- Acceder a la tabla de símbolos (para hacer parte del trabajo del analizador semántico)
- Chequeo de tipos (del analizador semántico)
- Generar código intermedio
- Generar errores cuando se producen

Tipo de gramática que acepta un analizador sintáctico

La gramática que acepta el analizador sintáctico es una gramática de contexto libre.
Gramáticas Libres de Contexto **G = { N ∑ P S }**
N = no terminales (categorías gramaticales)
∑ = alfabeto de la gramática (terminales)
P = conjunto de reglas de producción
S = símbolo inicial de la gramática (s ϵ N)
Derivaciones: La idea central es que se considera una producción como una regla de reescritura, donde el no terminal de la izquierda es sustituido por la cadena del lado derecho de la producción.

- Derivación por la izquierda. Derivación donde sólo el no terminal de más a la izquierda de cualquier forma de frase, se sustituye en cada paso.
- Derivación por la derecha o Derivación canónica. Derivación donde el no terminal más a la derecha se sustituye en cada paso.

Árbol sintáctico de una sentencia de un lenguaje

Es una representación que se utiliza para describir el proceso de derivación de dicha sentencia. Los nodos internos del árbol, se sitúan como elementos no terminales de las reglas de producción que se vayan aplicando, y tanto hijos como símbolos existan en la en la parte derecha de dichas reglas.

Ambigüedad: una gramática es ambigua si derivando de forma diferente con el mismo tipo de derivación se llega al mismo resultado.

Forma Sentencial: es cualquier secuencia de terminales y no terminales obtenida mediante derivaciones a partir del axioma inicial.

Tipos de Análisis

De la forma de construir el árbol sintáctico se desprenden dos tipos o clase de analizadores sintácticos. Pueden ser descendentes o ascendentes.

- Descendentes: parten del axioma inicial, y van efectuando derivaciones a izquierda hasta obtener la secuencia de derivaciones que reconoce la sentencia.

  Pueden ser:

  - Con retroceso
  - Con recursión
  - LL(1)

- Ascendentes: parten de la sentencia de entrada, y van aplicando reglas de producción hacia atrás (desde el consecuente hasta el antecedente), hasta llegar al axioma inicial.

  Pueden ser:

  - Con retroceso
  - LR(1)

2.2.3 ANALIZADOR SEMÁNTICO

El analizador semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales.

Funciones principales:

- Identificar cada tipo de instrucción y sus componentes
- Completar la Tabla de Símbolos
- Realizar distintas comprobaciones y validaciones:

    - Comprobaciones de tipos
    - Comprobaciones del flujo de control
    - Comprobaciones de unicidad
    - Comprobaciones de emparejamiento

El análisis semántico finaliza la fase de análisis del compilador y comienza la fase de Síntesis, en la cual se comienza a generar el código objeto. Genera un árbol semántico o etiquetado.

Código intermedio

El código intermedio es un código abstracto independiente de la máquina para la que se generará el código objeto. El código intermedio ha de cumplir dos requisitos importantes: ser fácil de producir a partir del análisis sintáctico y ser fácil de traducir al lenguaje objeto.

Generación de código intermedio

El generador de código intermedio transforma un árbol semántico en una representación en un lenguaje intermedio cercano al código objeto.

2.2.4 OPTIMIZADOR DE CÓDIGO

El optimizador de código realiza modificaciones sobre el código intermedio para mejorar la eficiencia en la velocidad y tamaño.

Funcionamiento:

- Revisa el código generado a varios niveles de abstracción y realiza las optimizaciones aplicables al nivel de abstracción.
- Realiza representaciones de código para extraer información: grafos.

Condiciones que se tienen que cumplir:

- El código optimizado se ha de comportar igual que el código de partida excepto por ser más rápido o ocupar menos espacio.
- Buscar transformaciones que no modifiquen el comportamiento del código según el comportamiento definido para el lenguaje de programación.

Código objeto

Esta etapa constituye la fase final de un compilador, se genera el código objeto que por lo general consiste en código en lenguaje máquina (código relocalizable) o código en lenguaje ensamblador.

Tabla de símbolos

Almacena estructuras de datos como:

- Variables
- Constantes
- Etiquetas
- Tipos
- Valores
- Signatura de funciones

Se puede realizar sobre la tabla de símbolos las siguientes acciones:

- Insertar símbolo
- Consultar símbolo
- Borrar símbolo

2.2.5 MANEJO DE ERRORES

Considerar desde el principio el manejo de errores puede simplificar la estructura de un compilador y mejorar su respuesta a los errores.

Los errores en la programación pueden ser de los siguientes tipos:

- Léxicos, producidos al escribir mal un identificador, una palabra clave o un operador.
- Sintácticos, por una expresión aritmética o paréntesis no equilibrados.
- Semánticos, como un operador aplicado a un operando incompatible.
- Lógicos, puede ser una llamada infinitamente recursiva.

Un manejador de errores de un analizador sintáctico debe tener como objetivos:

- Indicar los errores de forma clara y precisa. Aclarar el tipo de error y su localización.
- Recuperarse del error, para poder seguir examinando la entrada.
- No ralentizar significativamente la compilación.

Estrategias para corregir errores, una vez detectados:

- Ignorar el problema (Panic mode). Consiste en ignorar el resto de la entrada hasta llegar a una condición de seguridad. Una condición tal se produce cuando nos encontramos un token especial (por ejemplo un ';' o un 'END'). A partir de este punto se sigue analizando normalmente.

- Recuperación a nivel de frase. Intenta recuperar el error una vez descubierto. En el caso anterior, por ejemplo, podría haber sido lo suficientemente inteligente como para insertar el token ';'. Hay que tener cuidado con este método, pues puede dar lugar a recuperaciones infinitas.

- Reglas de producción adicionales para el control de errores: La gramática se puede aumentar con las reglas que reconocen los errores más comunes.

- Corrección global: Dada una secuencia completa de tokens, si hay algún error por el que no se puede reconocer, consiste en encontrar la secuencia completa más parecida que sí se pueda reconocer. Es decir, el analizador sintáctico le pide toda la secuencia de tokens al léxico, y lo que hace devolver lo más parecido a la cadena de entrada pero sin errores, así como el árbol que lo reconoce.

2.3 *LENGUJAES DE PROGRAMACIÓN*

2.3.1 LENGUAJE C

Creado entre 1970 y 1972 por Brian Kernighan y Dennis Ritchie para escribir el código del sistema operativo UNIX. Es un lenguaje que conjuga la abstracción de los lenguajes de alto nivel con la eficiencia del lenguaje máquina. A mediados de los ochenta el lenguaje C se convierte en un estándar internacional ISO. Este estándar incluye tanto la definición del lenguaje como una enorme biblioteca de funciones para entrada/salida, tratamiento de textos, matemáticas, etc. [A]

Características:

- Es altamente transportable
- Es muy flexible
- Genera código muy eficiente
- Es muy expresivo (se pueden realizar muchas funciones escribiendo pocas líneas de código)
- Es muy poco modular
- Hace pocas comprobaciones
- Es difícil leer código escrito por otras personas

El lenguaje C utiliza 5 palabras reservadas para definir los tipos de datos fundamentales. Los tipos de datos fundamentales son: [B]

- char
- short int
- int
- long int
- unsigned char
- unsigned short int
- unsigned int
- unsigned long int
- double
- float
- long float
- void

Estructuras de control

El lenguaje C, esta basado en la programación estructurada, por lo que los programas son más fáciles de entender y pueden ser leídos de forma secuencial. El lenguaje C utiliza las estructuras de control que permiten modificar el flujo de ejecución de las instrucciones de un programa. Con las estructuras de control se puede:

- De acuerdo a una condición, ejecutar un grupo u otro de sentencias (If-Then-Else y Select-Case)
- Ejecutar un grupo de sentencias mientras exista una condición (Do-While)
- Ejecutar un grupo de sentencias hasta que exista una condición (Do-Until)
- Ejecutar un grupo de sentencias un número determinado de veces (For-Next)

Todas las estructuras de control tienen un único punto de entrada y un único punto de salida. Se puede clasificar en: secuenciales, iterativas y de control avanzadas. Esto es una de las cosas que permite que la programación se rija por los principios de la programación estructurada. [23]

2.3.2 LENGUAJE JAVA

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90

La tecnología Java está compuesta básicamente por dos elementos: el lenguaje Java y su plataforma. Con plataforma nos referimos a la máquina virtual de Java (Java Virtual Machine).
Es un lenguaje que es compilado, generando ficheros de clases compilados, pero estas clases compiladas, son en realidad interpretadas por la máquina virtual de java. Siendo la máquina virtual de java la que mantiene el control sobre las clases que se estén ejecutando.

Es un lenguaje multiplataforma: el mismo código java que funciona en un sistema operativo, funcionará en cualquier otro sistema operativo que tenga instalada la máquina virtual java.
Es un lenguaje seguro: La máquina virtual, al ejecutar el código java, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros. [24]

Características asociadas:

- Herencia
- Encapsulamiento
- Abstracción
- Polimorfismo.

Interfaces de Usuario en Java.

Java permite crear diferentes componentes para generar una interfaz gráfica simple, vistosa y usable. Incluye un paquete llamado Swing en el cual define múltiples controles los cuales se pueden agregar a la forma. A continuación se describen los componentes que se usaron en la realización de Robot Code.

- **Track Bar (Figura 17)**: Un componente que permite al usuario seleccionar gráficamente un valor deslizando un botón dentro de un intervalo acotado. El control deslizante puede mostrar tanto las marcas de graduación principales y las marcas de graduación secundarias entre ellos. El número de valores entre las marcas se controla con setMajorTickSpacing y setMinorTickSpacing.

Figura 17 . Track Bar

- **Button (Figura 18):** Esta clase crea un botón. La aplicación genera una acción al hacer click sobre el botón.

Figura 18. Button.

- **List Box (Figura 19)**: Un componente que combina un botón o un campo editable y una lista desplegable. El usuario puede seleccionar un valor de la lista desplegable; si se elije el cuadro combinado editable, entonces el List Box incluye un campo editable en el que el usuario puede escribir un valor.

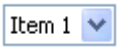


Figura 19. ListBox

2.4 *USABILIDAD*

Es la facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto.

En interacción humano-computadora, la usabilidad se refiere a la claridad y la elegancia con que se diseña la interacción con un programa de la computadora. El término también se usa a frecuentemente en el contexto de productos como la electrónica de consumo o en áreas de comunicación, y en objetos que transmiten conocimiento (por ejemplo, un libro de recetas o un documento de ayuda en línea).

También puede referirse al diseño eficiente de objetos mecánicos como, por ejemplo, un manubrio o un martillo.

El **grado de usabilidad** de un sistema es, por su parte, una medida empírica y relativa de la usabilidad del mismo.

- **Empírica** porque no se basa en opiniones o sensaciones, sino en pruebas de usabilidad realizadas en laboratorio u observadas mediante trabajo de campo.
- **Relativa** porque el resultado no es ni bueno ni malo, sino que depende de las metas planteadas (*por lo menos el 80% de los usuarios de un determinado grupo o tipo definido deben poder instalar con éxito el producto X en N minutos sin más ayuda que la guía rápida*) o de una comparación con otros sistemas similares.

El concepto de usabilidad se refiere a una aplicación de software o un aparato de hardware, aunque también puede aplicarse a cualquier sistema hecho con algún objetivo particular.

"Es decir la usabilidad se define como que tan fácil es usar una aplicación de software."

CAPÍTULO III:

"CARACTERÍSTICAS DEL ROBOT"

3. CARACTERÍSTICAS DEL ROBOT

3.1.1 LOCOMOCIÓN

El robot tiene locomoción de par diferencial. Trabaja con dos motores conectado cada uno directamente a la llanta. Tiene un punto de apoyo (rueda loca) debajo de las pilas para equilibrar el peso.
Cada uno de los motores tiene un codificador (encoder) que indica cuanto y en qué dirección va girado el motor.

3.1.2 ETAPA DE POTENCIA

El robot cuenta con 4 pilas AA con el cual alimenta los motores, sensores y el control. Para soportar la corriente que demanda los motores al microcontrolador PIC18F4550 se usa el puente H L298 con el cual permite una demanda de corriente hasta un ampere. Para alimentar el control y los sensores se utilizó un regulador de voltaje 7805 para regular a 5 volts adicionando un capacitor de 220µF para suavizar la señal de entrada y suprimir los picos generados por la corriente demandada de los motores. Se observa un esquema de la parte inferior del robot en la Figura 20.

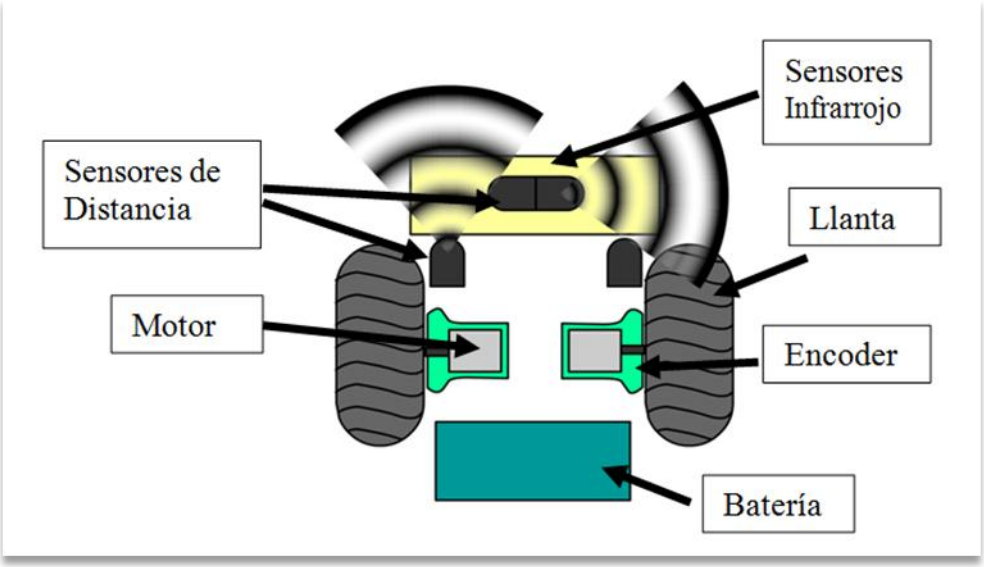


Figura 20. Sección inferior del robot vista desde arriba

3.1.3 SENSORES

El robot cuenta con dos tipos de sensores los cuales se conectan al convertidor analógico digital del microcontrolador haciendo uso de 9 ADC de los 12 disponibles en el PIC18F4550.

Sensores infrarrojos: cuenta con 5 sensores infrarrojos qrd1114 especialmente posicionados para seguir una línea de un cuarto de pulgada (6 mm), detectar cruces y giros con ángulos de 90° es decir fue diseñado para la competencia oficial de "line maze" aunque se puede utilizar para cualquier diseño.

Sensores infrarrojos de distancia: cuenta con cuatro sensores Sharp 2D120X posicionados especialmente para detectar el entorno en el que navega el robot. Los sensores pueden detectar objetos de 4 a 80 cm de distancia por lo que son ideales para la competencia de "micro mouse" aunque sus aplicaciones son ilimitadas.
Gracias a que los sensores están conectados a los convertidores analógicos digital del microcontrolador se puede saber con precisión la decisión que tomará, por ejemplo:

Sensores infrarrojos: cuando un sensor esta sobre la línea, está dejando la línea o está saliendo de la línea. Con esta información se puede censar más fácilmente el ambiente haciendo que la toma de decisiones del robot sea más correcta.
Sensores infrarrojos de distancia: se puede obtener una distancia aproximada de donde se encuentra el objeto a todo momento y con esto tomar decisiones más acertadas.

Cabe destacar que además de las aplicaciones específicas para las competencias de *micromouse y linemaze* se pueden usar los sensores de manera genérica y crear cualquier aplicación deseada, en la Figura 18 se observa la parte inferior del robot vista desde abajo.

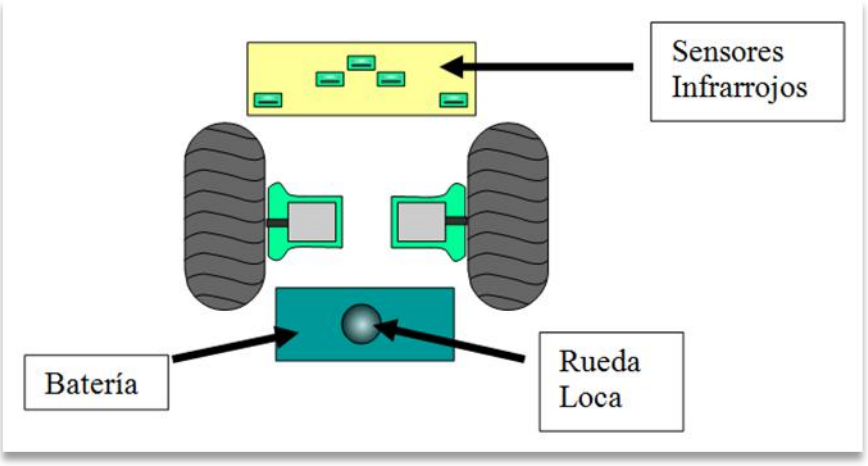


Figura 21. Sección inferior del robot vista desde abajo

3.1.4 CONTROL

Se utilizó el microcontrolador PIC18F4550 con el cual aprovechamos la versatilidad que posee. En la Figura 22 se observan los componentes electrónicos utilizados para el control.

Del microcontrolador PIC18F4550 utilizamos:

- Nueve convertidores analógicos digitales (sensores infrarrojos y a distancia).
- El puerto D para conectar la salida de 8 LED's.
- Dos push button's como entrada al microcontrolador.
- La comunicación USB para programar el microcontrolador a través del bootloader.
- La comunicación serie utilizando un MAX232 para regular voltaje entre la PC y el microcontrolador.
- Cuatro pines de salida para control la dirección de los dos motores.
- Dos módulos CCP conectados al habilitador (enable) de los dos motores para controlar la velocidad de ellos.
- Cuatro pines de entrada para leer la dirección y el numero de vueltas del codificador (encoder) posicionados en el rotor del motor.

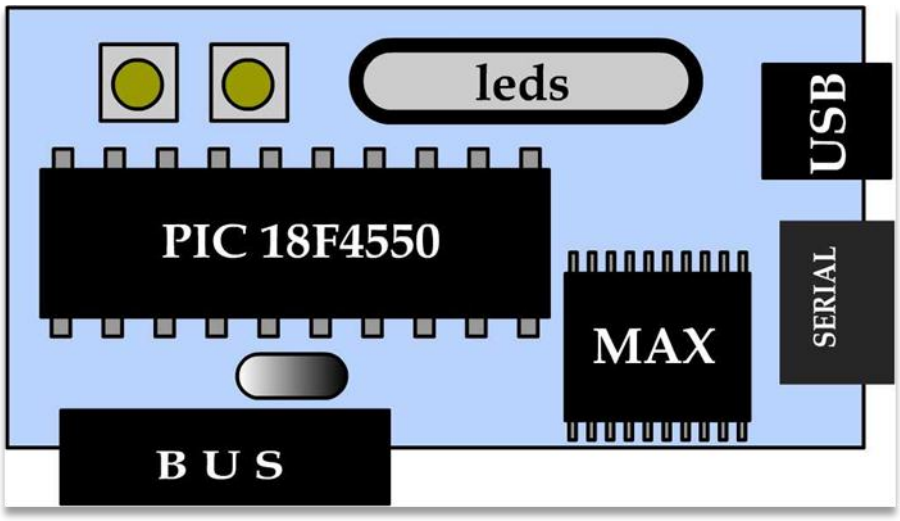


Figura 22. Componentes electrónicos utilizados

CAPÍTULO IV:

“PLAN DE ADMINISTRACIÓN DEL PROYECTO”

4. PLAN DE ADMINISTRACIÓN DEL PROYECTO

4.1 *ESPECIFICACIÓN DE REQUERIMIENTOS*

La Especificación de Requisitos incluye el propósito, alcance, definiciones, acrónimos, abreviaciones, referencias y generalidades de la especificación. Así como la descripción general del software y su uso en el ámbito de negocio del cliente. Algunos aspectos a mencionar en esta sección pueden ser, adicionalmente: uso del producto, principales características, naturaleza o características del usuario, restricciones generales.

4.2 *PROPÓSITO*

El propósito de este documento es dar a conocer de forma detallada de cada uno de los requisitos necesarios para desarrollar el proyecto, así como la aplicación, cada subsistema de ella y su estructura funcional (requisitos funcionales y no funcionales).

4.3 *DEFINICIONES, ACRÓNIMOS Y ABREVIACIONES*

- ***Bloque***: Estructura general que es la base de funciones, variables y controles de flujo.
- ***Función***: puede recibir una o más variables y además permiten modificar el valor de las mismas y ejecuta instrucciones atómicas de lenguaje C. Hay cuatro tipos de funciones básicas: Asignación, Lectura de sensor, Movimiento, Cambio de velocidad de motor, Cambio de sentido de motor.
- ***Variable***: Define un segmento donde se pueden almacenar valores: constantes enteras, flotantes, cadenas, y sensores.
- ***Tipo de Variable***: existirán dos tipos de variables:
- ***Memoria***: Variable que se encuentra en memoria, y pude ser leída y modificada al gusto.
- ***Sensor***: Variable la cual no se puede modificar, y está ligada con un puerto específico que recibe señal del sensor. Cada vez que se pregunta por ella, se actualizara el valor leído del sensor.
- ***Control de Flujo***: define la secuencia que llevara la carta ASM (Algoritmo).
- ***Área de Trabajo***: Lugar donde se puede arrastrar bloques.
- ***Sistema Embebido***: Sistema diseñado para realizar una o algunas pocas funciones dedicadas frecuente mente a un sistema de computación en tiempo real.

4.4 *GENERALIDADES*

A continuación se especificará a detalle como es la funcionalidad del sistema a ser desarrollado, como se ha presentado al cliente y como el usuario final podrá interactuar con él, y con ayuda de los requisitos funcionales, se tendrá un mejor entendimiento de

los mismos. También se señalan algunos de los requisitos con los que debe contar el sistema, sus características tales como portabilidad, eficiencia, confiabilidad, entre otras.

REQUISITOS	CARACTERISTICAS
Uso para estudiantes de Bachillerato	El producto está diseñado para ser empleado por alumnos de preparatoria que empiezan sus estudios en Robótica y Programación
Interfaz Gráfica	Distribuida en diferentes bloques que contendrán las funciones que permitirán al usuario crear el algoritmo.
Modo Aprendiz	Se trabajará con un robot estándar fabricado por SandroRobotics.
Arrastrar funciones	Selección de los botones disponibles para cada función
Asignación de Variables	Mediante el bloque de asignación se podrá cambiar el valor de las variables
Definir Variables	Se crearán nuevas variables , definiendo tipo, nombre y tamaño
Componentes de control de flujo	Permitirá unir las funciones previamente arrastradas
Definir funciones	A partir de funciones básicas , se podrán crear funciones más complejas
Guardar Algoritmo	Se designará una carpeta donde se almacenará el algoritmo dentro del sistema
Cargar Algoritmo al robot	Se verificará e implementará el algoritmo en el robot
Comunicación con MPASM y PM3Cmd	Para ensamblar y programar el Robot
Código en C y Ensamblador	Una vez compilado generará código en C , la sintaxis será similar a cartas ASM
Espacio ocupado	200 MB en memoria 500 MB después de instalación
Diagramas UML	El diseño de las clases se implementará en los diagramas
Lenguaje Java	Debido a la portabilidad del lenguaje, y se deberá usar el estándar CamelCAse para la compatibilidad

Tabla 7. Requerimientos Funcionales y no Funcionales

CAPÍTULO V:
"IMPLEMENTACIÓN"

5. IMPLEMENTACIÓN

DESCRIPCIÓN DE REQUISITOS

5.1 *REQUISITOS FUNCIONALES*

Arrastrar funciones:

- Se cuenta con diversos bloques funcionales los cuales interactúan con variables. Los bloques permiten leer y modificar el valor de las variables y ejecutar instrucciones atómicas de lenguaje C.
- Cada uno de los bloques funcionales están ubicadas en una barra de herramientas dentro de la interfaz gráfica localizada en el lado derecho.
- El proceso de arrastrar se lleva a cabo mediante la selección de los botones dispuestos para cada función, estos son colocados en el área de trabajo designada para el algoritmo.
- Funciones básicas:
  - Inicio
  - Declaración de Variables
  - Control de Motores
  - Asignación de una variable
  - Condicional

Definir variables:

- El usuario puede mediante el bloque de declaración de variables crear nuevas variables, con el cual puede definir su tipo, nombre, tamaño. Si la variable es de tipo sensor, se debe indica el puerto y el pin en el que se localiza el sensor.

Componentes de control de flujo:

- Permite al usuario unir cada uno de las funciones arrastradas en el área de trabajo mediante el uso de flechas e ir definiendo la secuencia de los bloques para estructurar el algoritmo.
- Además se cuenta con toma de decisiones representadas por diamantes para definir la secuencia de la carta ASM.

Definir funciones

- A partir de combinación de las funciones básicas de arrastre el usuario puede construir su algoritmo.
- Posibilidad de crear funciones más complejas con ayuda de las funciones básicas.

Guardar algoritmo

- Permite al usuario almacenar su algoritmo en una carpeta designada por el software.

Cargar algoritmo

- Una vez creado el algoritmo el sistema verifica el correcto funcionamiento del mismo.

Cargar algoritmo al robot

- Ya creado y verificado el algoritmo se procede a ser implementado en el robot especificado en el Capítulo III

5.2 *USABILIDAD*

Debido a que Robot Code fue diseñado para estudiantes de secundaria y preparatoria los cuales tienen limitado conocimiento de computación; fue muy importante pensar y tomar en cuenta los errores comunes que un estudiante puede generar cuando estaría creando el algoritmo. Por lo que se tomo la decisión de que el usuario deberá escribir la menor cantidad de texto posible. El único texto que escribirá será el nombre de las variables y constantes.

Gracias al uso de list box, track bars, botones y cuadros de textos fue posible reducir la posibilidad de errores, sin limitar la funcionalidad de Robot Code.

La barra indicadora tiene una gran utilidad para la funcionalidad del Robot Code

5.3 *INTERFAZ CON USUARIO*

La interfaz gráfica está compuesta por diferentes bloques distribuidos de una forma tal que facilita el manejo del sistema. Estos bloques contienen las diferentes funciones que permitirán al usuario construir los diversos algoritmos que desee, de una forma fácil y entendible.

En el lado izquierdo de la interfaz se muestran imágenes , las cuales representan a cada uno de los bloques (Bloque de Declaración de Variables, Asignación de variables, bloque de control de motores, cambio a modo flecha entre otros) además de un texto debajo del indicando su nombre, y un *tooltip* que de una breve descripción de lo que hace dicho bloque.

Estos bloques pueden ser conectados asignando así la secuencia (flujo) específica dependiendo del control de flujo que se desea.

Existirán dos modos principales: Modo Arrastre y Modo Flecha.

5.3.1 MODO ARRASTRE

Es el modo que esta seleccionado por default. El modo de arrastre permite la manipulación de los bloques que se encuentran en el área de trabajo, así como su modificación y su eliminación. Al colocarse encima de un bloque en el área de trabajo, el cursor cambiara al de una mano, como se muestra en la Figura 23. Indicando que ese bloque se puede mover.

Figura 23. Cursor manita

Para mover el bloque será necesario colocarse encima de él. Dar click izquierdo y sostener (arrastrar) hasta trasladar a la posición deseada. Al liberar el botón izquierdo del mouse el bloque permanecerá en su lugar.

Para editar el bloque, de click izquierdo una vez en el bloque y suelte en la misma posición. Se abrirá una ventana de diálogo específica del bloque en los cuales se colocarán los parámetros deseados.

5.3.2 MODO CONEXIÓN:

Para conectar cada bloque debe utilizar la herramienta de flecha contenida en el control de flujo. Se indicará en color rojo al momento que se active el modo conexión. El modo conexión permite definir la secuencia del diagrama de flujo además de dar la interconexión de dos bloques.

Cuando el modo conexión este activado y cuando el mouse este encima de un bloque el cursor se cambiará a crosshair indicando que se podrá interconectar ese bloque con otro para así poder indicar el flujo, como se muestra en la Figura 24.

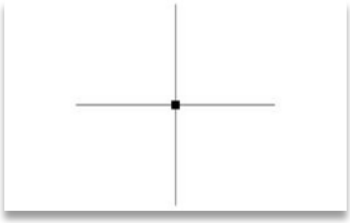


Figura 24. Cursor crosshair

Al momento de dar click izquierdo sostenido en un bloque, aparecerá una flecha en la parte inferior de éste y podrá ser arrastrada con el mouse hasta el siguiente bloque donde se desea continuar con el flujo, como se muestra en la Figura 25.

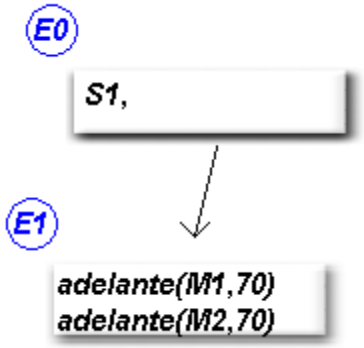


Figura 25. Ejemplo de conexión entre dos bloques

5.3.3 MODIFICACIÓN DE BLOQUES

Cada bloque tendrá distintas propiedades. Robot Code permite modificar dichas propiedades dando clic izquierdo sobre el bloque que se desea modificar. Se mostrará una ventana de dialogo específica de las propiedades del bloque que se desee modificar. Las propiedades almacenadas se mostrarán al momento de que se visualice la ventana de diálogo de propiedades del bloque. Las propiedades afectan el código final que generará.

5.3.4 CÓDIGO AUTOGENERADO

Por cada bloque en el área de trabajo Robot Code autogenera código. Existen dos tipos de códigos generado:

Código de Cabecera:

Principalmente usado en la declaración de variables, este código se colocará fuera del ***main*** de C.

Código de Cuerpo:

Usado principalmente por los bloques de: Asignación de variables, control de motores, condicionales. Esté código se coloca dentro de las llaves del main.

A continuación se describe cada bloque que existe en Robot Code, así como la ventana de dialogo de propiedades de dicho bloque y los cambios que realiza en el código final de C.

5.3.5 BLOQUE INICIO

Solo podrá existir un bloque de inicio. Este bloque indica el inicio del diagrama de flujo.

Figura 26: Bloque inicio

Si el usuario intenta crear dos bloques de inicio. El sistema indicará el error informando al usuario que solo puede existir un bloque de inicio. El bloque de inicio es el único bloque que no posee ventana de diálogo de propiedades. Debido a que solo indica el inicio del programa.

Código Generado:

```
int main()
{
E0: // Inicio
}
```

5.3.6 BLOQUE DECLARACIÓN DE VARIABLE

Permite crear y declarar nuevas variables. Las cuales serán visibles a todos los bloques. Los bloques de asignación de variable y bloque condicional podrán acceder y modificar tales variables. (Figura 27)

Figura 27. Bloque Declaración de Variable

Al nombrar una nueva variable se mostrara su nombre contenido dentro del bloque como se muestra en la figura. 28

Figura 28. Nombre de Bloque de Declaración de Variable

A cada una de estas variables se le puede asignar su propiedad específica. El cuadro de Diálogo del Bloque de Declaración de Variable permitirá especificar diferentes funcionalidades. A continuación se describe cada una.

5.3.7 CUADRO DE DIÁLOGO DE BLOQUE DECLARACIÓN DE VARIABLE

Sirve para indicar que una localidad de memoria será reservada y podrá ser identificada por el nombre de la variable. El cuadro de diálogo del bloque permite crear distintos tipos de variable, definir su nombre, tamaño y la declaración en cuanto a funcionalidad de la misma (véase Figura 29).

Figura 29. Ejemplo de declaración de una variable

Existen 3 diferentes tipos: Entero, Carácter y Decimal (Figura 30).

Figura 30. Tipos de variable

- **Entero:** almacena números enteros con signo, si es un byte el rango es de 0 a 255. Se manejan dos tipos de tamaño: Uno y dos bytes (Figura 31).

Figura 31. Tamaño de la variable

Cuatro diferentes tipos de declaración: Variable, Sensor, Entrada y Salida (Figura 32).

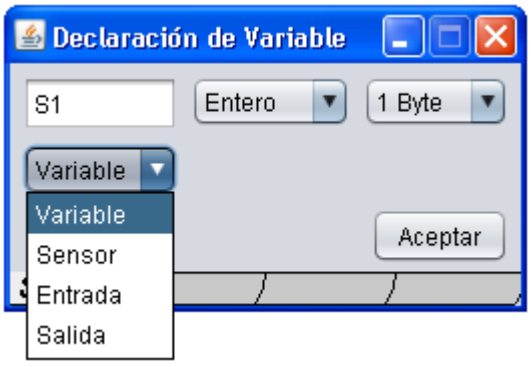


Figura 32. Tipos de declaración de variable

Variable: El valor seleccionado por default, indica que el valor se guardará en una localidad de memoria RAM.

Sensor: Variable que se almacena en una localidad de memoria, la cual sólo se puede leer. Cada vez que se accede a su valor, se actualiza automáticamente activando el convertidor analógico digital del puerto analógico seleccionado (véase Figura 33).

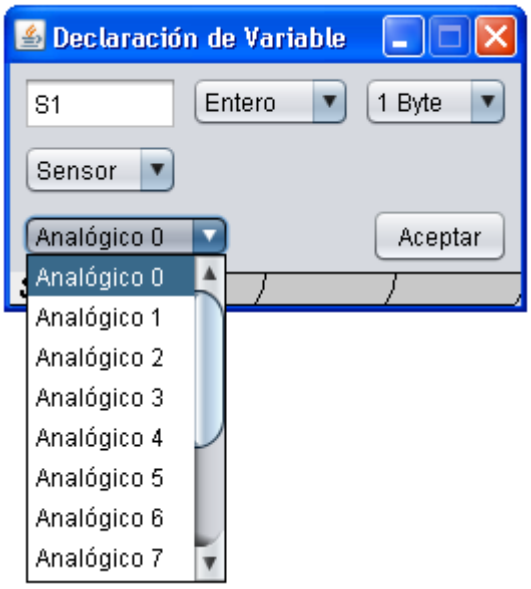


Figura 33. Tipo de variable Sensor

Entrada: Se almacena en una localidad de memoria, sólo es de tipo lectura, se puede indicar si se leerá todo el puerto o sólo un bit de él (véase Figura 34).

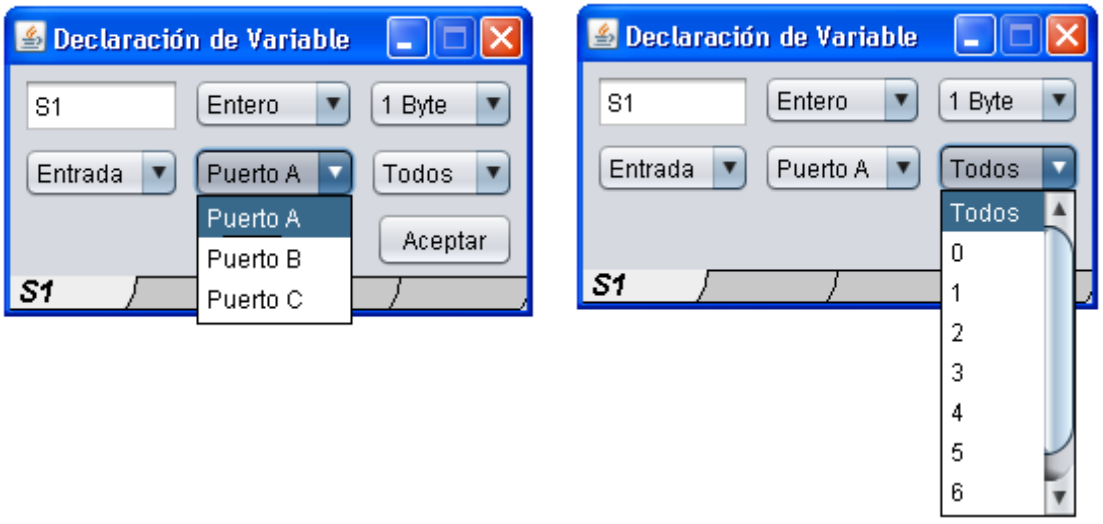


Figura 34. Tipo de variable Entrada

Salida: Se almacena en una localidad de memoria, sólo es de tipo escritura, permite la salida de datos de un puerto específico o un pin al asignar su valor.

Código Generado

El código generado depende de la propiedad elegida en la ventana de Diálogo de Declaración de Variable.

El código cabecera se muestra en la Tabla 8:

Propiedad Seleccionada	Código Generado
Tipo	
Entero	int
Carácter	char
Decimal	float
Tamaño	
1 byte	short
2 bytes	long
Tipo Variable	
Variable	-
Entrada	Crea una macro en C, la cual define la localidad física donde se encuentra el puerto, pin o entrada analógica elegida.
Salida	
Sensor	

Tabla 8. Codigo de cabecera

Ejemplo de Cabecera:

```
#define vS1 3 // donde 3 es la localidad física en memoria del sensor.
short int S1;
```

5.3.8 BLOQUE ASIGNACIÓN DE VARIABLE

Permite la asignación de variables que se encuentran definidas en el área de trabajo. (Figura 35)

Figura 35. Visualización del bloque asignación de variables

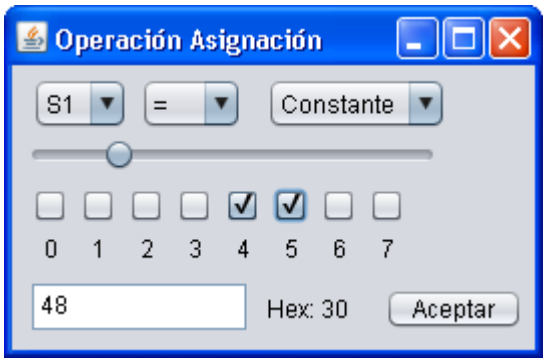


Figura 36. Bloque Asignación de Variable

Tiene tres listbox.

Selección de variables existentes: permite seleccionar la variable que se desea modificar (Figura 37).

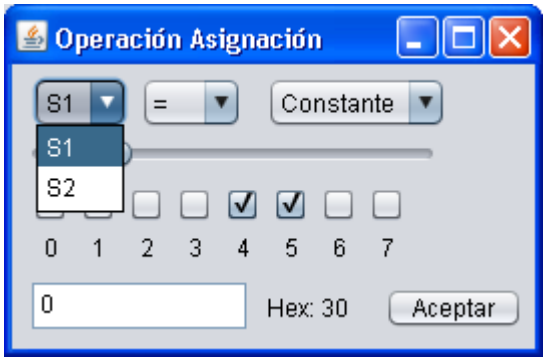


Figura 37. Selección de variables existentes

Operación de asignación: permite indicar la operación de asignación a realizar (Figura 38).

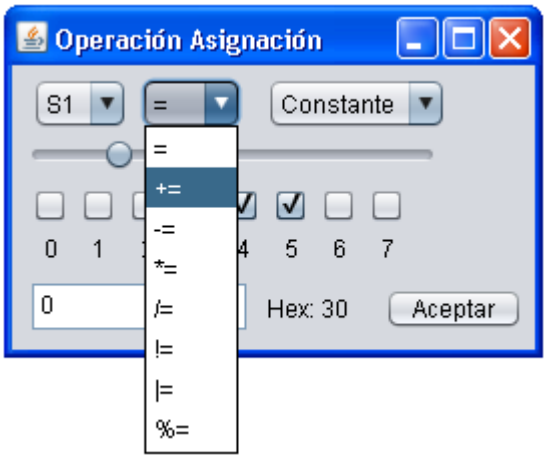


Figura 38. Selección del operador de asignación

Valor de asignación: indicará el nuevo valor que tomará la variable (Figura 39).

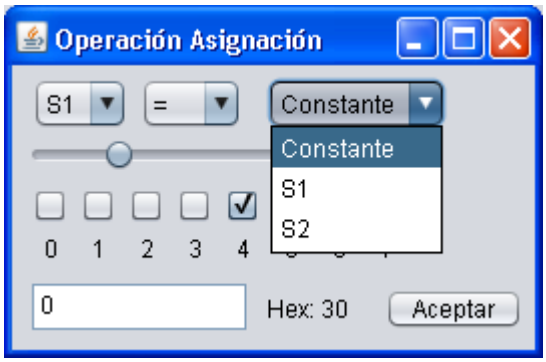


Figura 39. Valor de asignación de la variable

Si la opción constante es seleccionado, se podrá modificar el valor de la constante dando click a la barra o seleccionando los bits que se activarán (Figura 40).

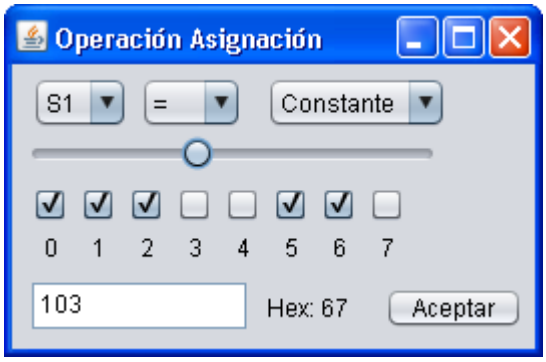


Figura 40 Modificación del valor de la constante

Si la variable es de dos bytes, se desactivarán los indicadores y sólo permitirá modificar su valor directamente en el cuadro de texto.

Código autogenerado:

No genera código de cabecera.
EL código de cuerpo que genera depende de las propiedades elegidas en el cuadro de diálogo de asignación de bloque variable.

Código generado por la figura 40:

```
a=103;
```

5.3.9 BLOQUE CONTROL DE MOTOR

Permitirá el control de uno o dos motores. Definir su dirección y su velocidad.

Figura 41. Visualización del Bloque Control de Motor.

Figura 42. Bloque Control de Motor

Tiene dos módulos en el cual cada uno contiene dos listbox y un track bar.

Listbox selección de motor: Permite seleccionar alguno de los dos motores (ver Figura 36), si se selecciona ninguno, se deshabilitará todo el módulo (ver Figura 43).

Figura 43. Selección de motor

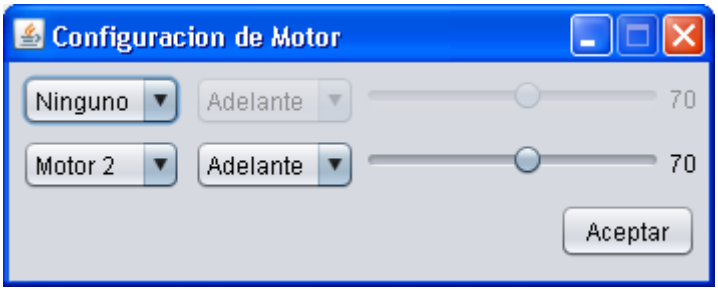


Figura 44. Deshabilitación del primer módulo

Listbox indicador de movimiento: Permite la selección de la dirección de giro del motor (ver Figura 45).

Figura 45. Listbox indicador de movimiento

Track bar de selección de velocidad: Permite modificar la velocidad de giro del motor (ver Figura 39).

Figura 46. Listbox selección de velocidad

Código autogenerado:

Código de Cabecera: Carga la biblioteca moteres.h, en la cual se definen constantes y funciones necesarias para el control de los motores.

Código de Cuerpo: por cada módulo habilitado en la ventana de diálogo de configuración de bloque de motor, se agrega el código necesario para realizar dicha función.

Ejemplo del código generado por la figura 46.

```
// adelante(M1,70)
   set_pwm1_duty(70);
   output_low(M1D);
   output_high(M1C);
// adelante(M2,40)
   set_pwm2_duty(40);
   output_low(M2D);
   output_high(M2C);
```

5.3.10 BLOQUE CONDICIONAL

El bloque condicional permitirá definir bifurcaciones en el diagrama de flujo (ver Figura 47).

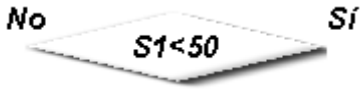


Figura 47 muestra la visualización del bloque condicional con la variable S1 y el operador elegido con la constante 50.

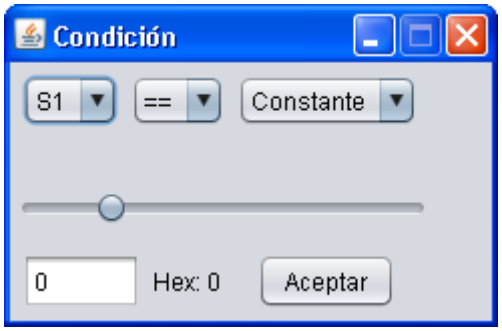


Figura 48. Bloque condicional

Tiene tres List boxes:

Selección de primera variable condicional:

Muestra todas las variables que han sido declaradas en el área de trabajo y permite seleccionar una específicamente (ver Figura 41).

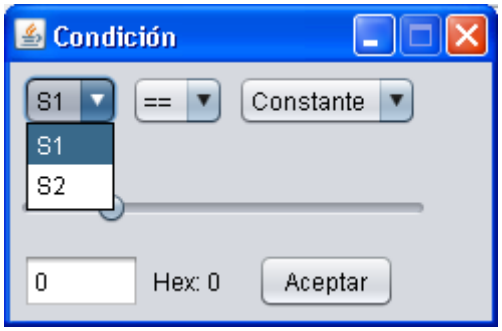


Figura 49. Selección de la primera variable condicional

Operador condicional: Permite seleccionar la operación condicional a realizar (ver Figura 50).

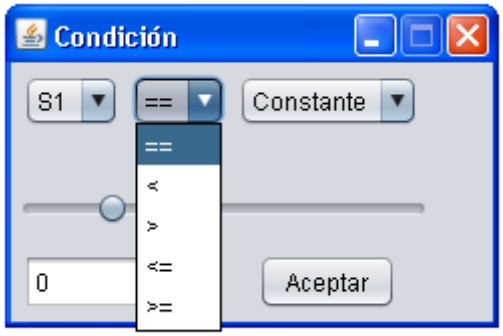


Figura 50. Selección de operador condicional

Constante o Selección de segunda Variable condicional:

Muestra todas las variables que han sido declaradas en el área de trabajo. Permite seleccionar una específicamente o el valor de una constante.

Si se selecciona constante, se habilita la barra indicadora, con la cual se puede seleccionar un valor pertinente para la comparación, si se selecciona una variable se deshabilita la barra indicadora por lo que no se puede indicar el número.

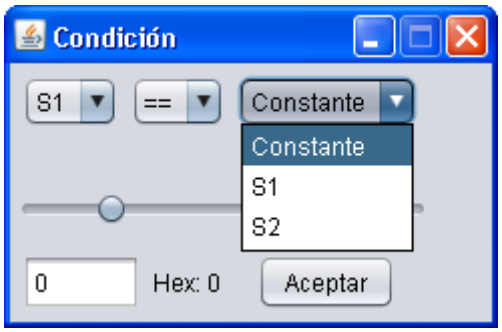


Figura 51. Selección de la segunda variable condicional

Código autogenerado:

Código de Cabecera: No genera código de cabecera.

Código de cuerpo: genera el código necesario para realizar la funciones indicadas en la ventana de diálogo del bloque de condición.

Ejemplo de código generado por la figura 51:

```
If(s1==0){
        // Liga verdadera
}else{
        // Liga falsa
}
```

Vista de la Interface gráfica completa de Robot Code

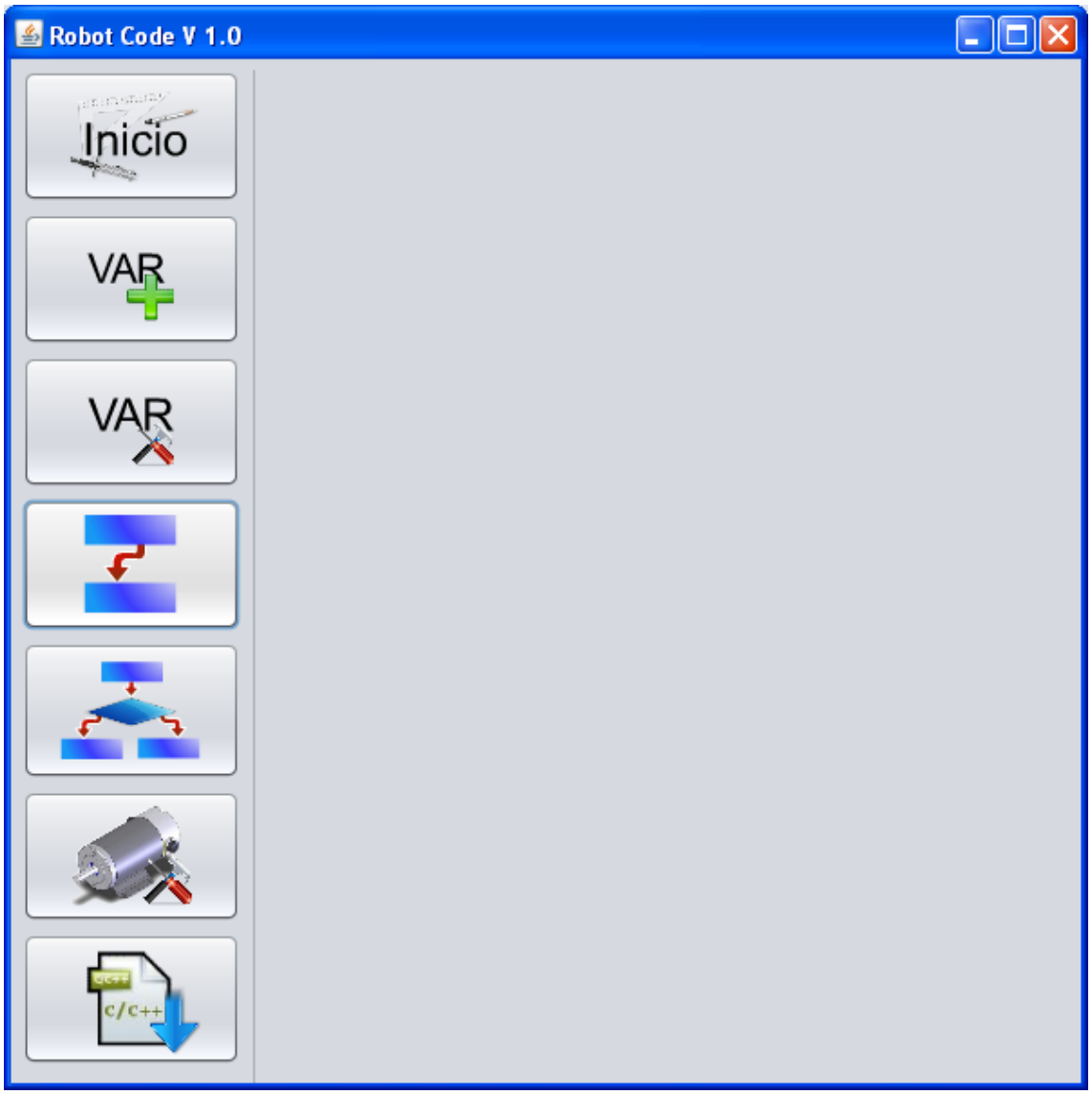


Figura 52. Interfaz Gráfica de Robot Code

El siguiente diagrama de flujo ejemplifica la funcionalidad de Robot Code, muestra el diseño de un algoritmo que controla el robot usando los dos sensores de distancia, el cual cambia de dirección y velocidad al detectar un obstáculo. El algoritmo se repite indefinidamente.

Diagrama de Flujo

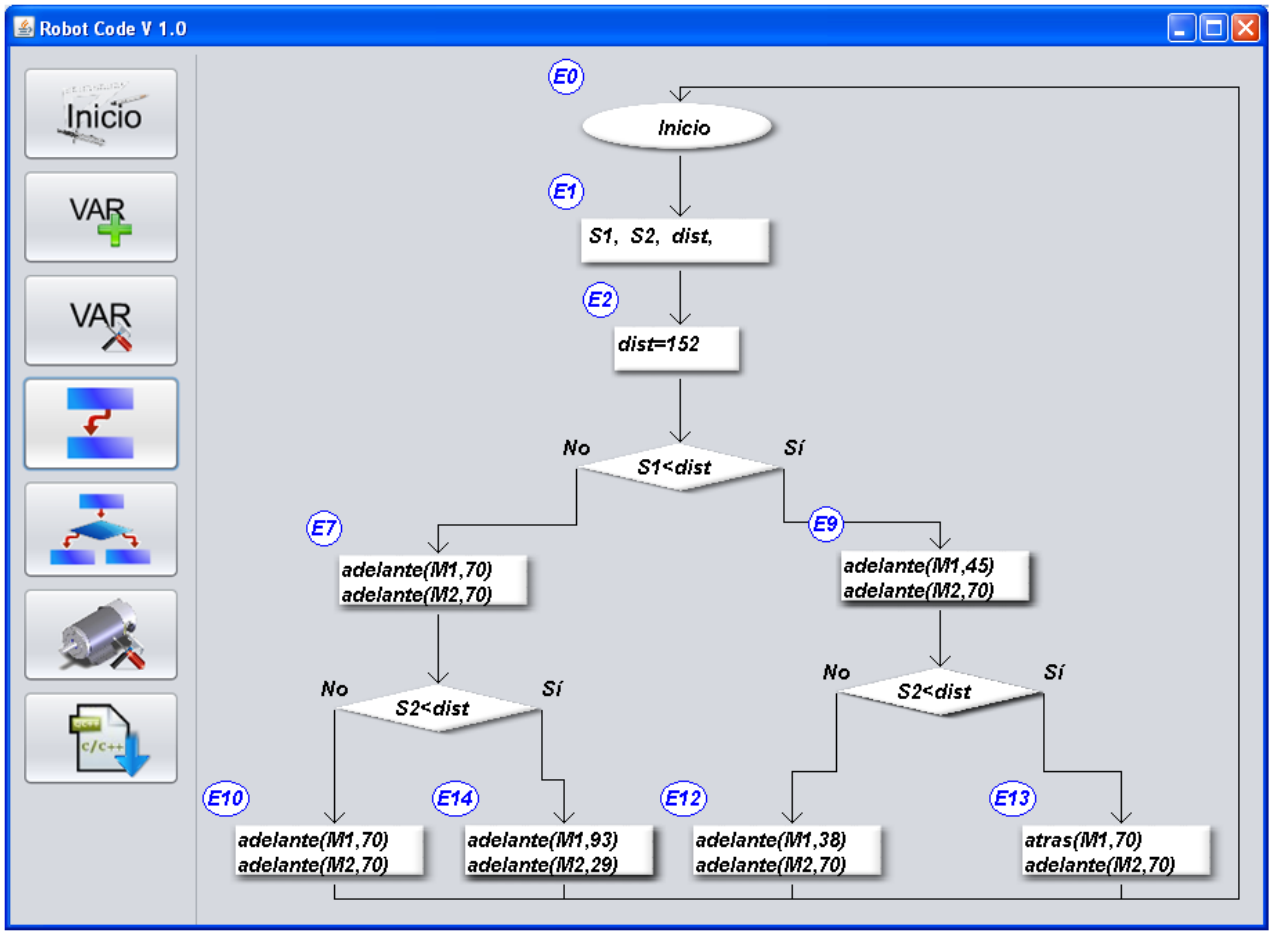


Figura 53. Diseño del diagrama de flujo para el control del robot

Código Autogenerado:

Compilador de Codigo

```
int main()
{
        E0: // Inicio
        E2:
        dist=152;
        if(S1<dist){
          E9:
            // adelante(M1,45)
            set_pwm1_duty(45);
            output_low(M1D);
            output_high(M1C);
            // adelante(M2,70)
            set_pwm2_duty(70);
            output_low(M2D);
            output_high(M2C);
          if(S2<dist){
            E13:
              // atras(M1,70)
              set_pwm1_duty(70);
              output_low(M1D);
              output_high(M1C);
              // adelante(M2,70)
              set_pwm2_duty(70);
              output_low(M2D);
              output_high(M2C);
            goto E0;
                        }
          else{
            E12:
              // adelante(M1,38)
              set_pwm1_duty(38);
              output_low(M1D);
              output_high(M1C);
              // adelante(M2,70)
              set_pwm2_duty(70);
              output_low(M2D);
              output_high(M2C);
            goto E0;
          }
                    }
        else{
          E7:
            // adelante(M1,70)
            set_pwm1_duty(70);
            output_low(M1D);
```

Figura 54. Código Generado por Robot Code a partir del diseño

Archivo generado:

```
#define vS1 4
#define vS2 5
short int S1;
short int S2;
int dist;

int main()
{
        E0: // Inicio
        E2:
        dist=152;
        S1=lee(vS1);
        if(S1<dist){
          E9:
            // adelante(M1,45)
            set_pwm1_duty(45);
            output_low(M1D);
            output_high(M1C);
            // adelante(M2,70)
            set_pwm2_duty(70);
            output_low(M2D);
            output_high(M2C);
             S2=lee(vS2);
              if(S2<dist){
            E13:
              // atras(M1,70)
              set_pwm1_duty(70);
              output_low(M1D);
              output_high(M1C);                    // adelante(M2,70)
              set_pwm2_duty(70);
              output_low(M2D);
              output_high(M2C);
            goto E0;
                        }
          else{
            E12:
              // adelante(M1,38)
              set_pwm1_duty(38);
              output_low(M1D);
              output_high(M1C);
              // adelante(M2,70)
              set_pwm2_duty(70);
              output_low(M2D);
              output_high(M2C);
```

```
            goto E0;
        }
                    }
    else{
        E7:
            // adelante(M1,70)
            set_pwm1_duty(70);
            output_low(M1D);
            output_high(M1C);
            // adelante(M2,70)
            set_pwm2_duty(70);
            output_low(M2D);
            output_high(M2C);

        S2=lee(vS2);
        if(S2<dist){
            E14:
                // adelante(M1,93)
                set_pwm1_duty(93);
                output_low(M1D);
                output_high(M1C);
                // adelante(M2,29)
                set_pwm2_duty(29);
                output_low(M2D);
                output_high(M2C);
            goto E0;
                        }
        else{
            E10:
                // adelante(M1,70)
                set_pwm1_duty(70);
                output_low(M1D);
                output_high(M1C);
                // adelante(M2,70)
                set_pwm2_duty(70);
                output_low(M2D);
                output_high(M2C);
            goto E0;
        }
    }

}
```

5.4 *INTERFACES CON OTRO SOFTWARE O HARDWARE*

El sistema a desarrollar deberá correr en las computadoras de los estudiantes, lo que significa que el sistema operativo en el que correrá al igual que el hardware será desconocido. Por esta razón se ha optado usar Java, debido a su independencia de hardware.

El sistema deberá correr a partir de una computadora con más de 512 MB de memoria RAM.

5.5 *CONFIABILIDAD*

El sistema será tolerante a fallas, si llegara a cerrarse inesperadamente el sistema el usuario podrá recuperar la información, ya que se creará un archivo temporal que se actualiza cada vez que el usuario agrega un bloque, o modifica el control de flujo del algoritmo.

5.6 *EFICIENCIA*

→ El sistema deberá usar pocos recursos de la computadora, deberá correr en computadoras de hasta 5 años de antigüedad.
→ El tiempo máximo que puede tardar el sistema en iniciarse es de un minuto.
→ El tiempo máximo que puede tardar el sistema en crear, arrastrar, y dibujar un bloque es de 100 milisegundos.
→ El tamaño máximo por bloque de una carta ASM, será de 1K incluyendo los bloques definidos por el usuario.
→ El sistema deberá ocupar un máximo de 200 MB de Memoria.
→ El sistema no deberá ocupar más de 500 MB después de la instalación en disco duro.

5.7 *MANTENIMIENTO*

Se usará diagramas UML en el diseño de todas las clases, además de que todo el software será codificado en Java.

Se codificará usando el estándar de CamelCase, para distinguir las variables, funciones y clases. Sólo se usarán las librerías de Java Standard Edition, por compatibilidad.

Todas las clases serán documentadas y deberán dar una breve descripción de lo que hacen, al igual que cada uno de sus métodos y atributos.

5.8 *PORTABILIDAD*

El software debido a que estará desarrollado en lenguaje Java, contará con las mismas características de portabilidad de dicho lenguaje, es decir, podrá ser utilizado en cualquier plataforma y sistema operativo.

5.9 *INTEROPERATIVIDAD*

El sistema, podrá interactuar con el compilador de C, para así poder generar el código máquina entendible por el robot utilizado.

5.10 *REUSABILIDAD.*

Con este sistema se podrán crear algoritmos los cuales pueden ser utilizados para crear comportamientos complejos de sistemas de cualquier dispositivo embebido.

5.11 *RESTRICCIONES DE DISEÑO Y CONSTRUCCIÓN*

→ No existen restricciones por parte del cliente, solo las que son impuestas por el equipo.
→ El sistema sólo deberá ser desarrollado lenguaje Java usando las bibliotecas estándares.

5.12 *LEGALES Y REGLAMENTARIOS.*

→ Al cliente no se le proveerá el código fuente, será parte del equipo.
→ Se venderán licencias individuales a las instituciones o usuarios externos.

GLOSARIO

GLOSARIO:

- **Turing completo**: En la práctica Turing completo, llamado así por Alan Turing, significa que las reglas seguidas por una secuencia de datos arbitrarios puede producir el resultado de cualquier cálculo. Esto requiere, como mínimo, la bifurcación condicional (un "if" y la declaración de "goto") y la capacidad de cambiar las ubicaciones de memoria arbitrarias. Para demostrar que es Turing completo, es suficiente demostrar que puede ser utilizado para simular la computadora más primitiva, ya que la computadora más simple puede ser utilizado para simular la más complicada.

- **ROBOCUP**: es un proyecto internacional para promover, a través de competencias integradas por robots autónomos, la investigación y educación sobre inteligencia artificial.

- **FMR**: Federación Mexicana de Robótica

- **Java:** Permite la ejecución de un mismo programa en múltiples sistemas operativos y hardware diferente, Usa la metodología de la programación orientada a objetos.

- **AWT:** Herramientas de Ventana Abstracta, es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.

- **Carta ASM**: Algorithmic State Machine, método para definir una maquina finita de estados. Es como un diagrama de estados.

- **Algoritmo**: es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien lo ejecute. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

- **MPASM**: Ensamblador gratuito de Microchip. Crea a partir de lenguaje ensamblador un archivo en lenguaje máquina (Hex) el cual puede ser entendido por el PIC.

- **PM3Cmd**: Interface de Microchip para programar el PIC por el puerto USB.

- **Programa fuente:** Código escrito en lenguaje de alto nivel.

- **RAM:** por sus siglas en inglés Random Access Memory.

- **ROM :** La memoria de solo lectura, conocida también como ROM (acrónimo en inglés de read-only memory), es un medio de almacenamiento utilizado en ordenadores y dispositivos electrónicos, que permite solo la lectura de la información y no su escritura

- **EPROM :** son las siglas de Erasable Programmable Read-Only Memory (ROM programable borrable). Una vez programada, una EPROM se puede borrar solamente mediante exposición a una fuerte luz ultravioleta.

- **EEPROM:** son las siglas de Electrically-Erasable Programmable Read-Only Memory (ROM programable y borrable eléctricamente). Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente, a diferencia de la EPROM que ha de borrarse mediante un aparato que emite rayos ultravioletas. Son memorias no volátiles.

- **HDL:** Un lenguaje de descripción de hardware (HDL, Hardware Description Language) permite documentar las interconexiones y el comportamiento de un circuito electrónico, sin utilizar diagramas esquemáticos.

- **SRAM:** Memoria Estática de Acceso Aleatorio es un tipo de memoria basada en semiconductores que a diferencia de la memoria DRAM, es capaz de mantener los datos, mientras esté alimentada, sin necesidad de circuito de refresco. Sin embargo, sí son memorias volátiles, es decir que pierden la información si se les interrumpe la alimentación eléctrica.

- **Tooltip :** Un tooltip (también llamada descripción emergente) es una herramienta de ayuda visual, que funciona al situar el cursor sobre algún elemento gráfico, mostrando una ayuda adicional para informar al usuario de la finalidad del elemento sobre el que se encuentra.

- **CamelCase:** es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello.

RESULTADOS

INSPECCIÓN DE USABILIDAD:

En esta sección inspeccionamos Robot Code al utilizar un encaminamiento de metodología cognitiva (cognitive walkthrough methodology). El encaminamiento cognitivo es una evaluación práctica, en el cual el usuario examina la interface y el sistema al intentar completar una serie de tareas. El encaminamiento cognitivo ayuda a identificar que tan fácil es de aprender usar nuestro sistema y que tan fácil es usarse.

5.13 *PERFIL DE USUARIOS*

La Inspección de usabilidad se realizó por 15 usuarios diferentes, 5 participantes de secundaria los cuales nunca habían usado un lenguaje de programación, 5 participantes de preparatoria y 5 estudiantes expertos de licenciatura en la Facultad de ingeniería, donde 2 de ellos había usado programas similares como Lego Mindstorms y tenían gran experiencia programando robots y 3 de ellos eran alumnos de quinto semestre de licenciatura.

5.14 *TAREAS*

A los usuarios se les asignó dos simples tareas, hacer un sencillo programa donde el robot tendría que avanzar indefinidamente hasta que el sensor de distancia midiera un valor menor a 50 en una escala de 0 a 255.
En la siguiente tarea, el robot tendrá que estar en alto. Al recibir la entrada 0F en el puerto A, deberá girar a la izquierda y al recibir la entrada F0, deberá girar a la derecha.
Con lo único que contaron para realizar la tarea fue con Robot Code y la guía rápida (solo dos páginas).

5.15 *RESULTADOS*

En la primera tarea los usuarios tenían problemas para familiarizarse con la interface. Con facilidad lograban arrastrar los bloques, pero algunos estudiantes de secundaria así como los de preparatoria tuvieron problemas al abrir el diálogo de propiedades de cada elemento, pero una vez entendido el procedimiento fácilmente podían repetirlo y configurar cada bloque. Algo desalentador fue el hecho de entrar al modo de conexión, la mayoría (12 usuarios) experimentó problemas en conectar sus bloques. Esto es debido a que los usuarios querían conectar los bloques como si estuvieran pintando líneas con Paint. La frustración de algunos usuarios era tan grande, especialmente en los usuarios expertos, que decidían fallar la prueba y pedir ayuda para terminarla.

Irónicamente los estudiantes de secundaria no se rendían y 3 de 5 lograron completar la prueba.

En la segunda prueba, debido a que los usuarios tenían la experiencia previa de la primera tarea, se tuvo un éxito de 13 de 15 estudiantes, los usuarios de secundaria comenzaron a adquirir el gusto de diseñar diagramas de flujos. Lo que más les gustaba es que podían ver los resultados en tiempo real del diagrama de flujo que habían diseñado siendo ejecutado en el robot.

La principal razón por lo que algunos estudiantes no lograron completar esta prueba fue debido a que el uso del bloque condicional se le complicaba así como el hecho de hacer que el diagrama de flujo regresara al bloque inicio. No tenían la intuición necesaria para descubrir que para hacer que se repitiera indefinidamente el código era necesario regresar al inicio.

Los resultados obtenidos por Robot Code fueron buenos, aunque hay muchas cosas que faltan mejorar. Un usuario experto nos mencionó que un bloque de espera (delay) y un bloque de distancia, mejoraría la capacidad de Robot Code. Ya que se le podría asignar un periodo de tiempo para que se ejecute el siguiente bloque.

A partir de los resultados nos percatamos que es necesario rediseñar los manuales para así minimizar el tiempo de aprendizaje de Robot Code.

CONCLUSIONES

CONCLUSIONES

Los objetivos planteados en la tesis, se lograron cumplir de forma exitosa, esto se puede observar en la parte de Resultados , en cada una de las pruebas realizadas a distintos usuarios . La facilidad y el gusto de programar un robot es mucho mayor al utilizar Robot Code en comparación con lenguajes convencionales, debido a que no hay necesidad de escribir el código, eliminando así los errores sintácticos, léxicos y semánticos.

El código autogenerado por Robot Code , es fácilmente potable , para ser compilado con éxito y posteriormente , puede ser almacenado en el robot que se este utilizando . Los alumnos dedican más atención al ver los resultados físicos funcionado en un robot por lo que se puede enseñar de forma interactiva y divertida.

RESULTADOS ESPERADOS:

La obtención de una Herramienta de Software que facilite tanto la programación como la comprensión del funcionamiento de Robots, con ayuda de una Interfaz Gráfica que permita la fácil implementación de procesos básicos realizados por un robot.

REFERENCIAS

REFERENCIAS:

Consultas Electrónicas:

[1]Tesis
[2]catarina.udlap.mx/u_dl_a/tales/documentos/lem/morales.../capitulo3.pdf
[3]http://www.asifunciona.com/electronica/af_conv_ad/conv_ad_5.htm
[4]http://www.electromicrodigital.com/micros/files/pic18f4550/MODULO_AD.pdf
[4]http://www.muchotrasto.com/SistemaElectronico.php
[5] http://robots-argentina.com.ar/Comunicacion_max232.htm
[6]
http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/salvatori_a_m/capitulo4.pdf
[7] http://www.profesormolina.com.ar/tecnologia/sens_transduct/que_es.htm
[8]Educatrónica: "Innovación en el aprendizaje de las ciencias y la tecnología", Ruiz-Velasco Sánchez, Enrique.
[9]http://informaticafrida.blogspot.com/2010/05/distintas-maneras-de-clasificar-los.html
[10]http://robotec11.tripod.com/id4.html
[11]http://www.muchotrasto.com/SistemaElectronico.php
[12]http://www.pinguino.org.ve/descargas/Manual%20PIC%2018F4550.pdf
[13]http://www.ate.uniovi.es/fernando/Doc2005/Sed_05/Ejemplos/CCPenPWM_ring.pdf
[14]www.robotstorehk.com/sensors/doc/QRD1113_1114.pdf
[15]http://www.datasheetarchive.com/
[16]http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf
[17]http://users.dsic.upv.es/~jsilva/uned/compiladores/Apuntes01.pdf
[18]http://compiladorsistemas.blogspot.com/2010/11/analisis-semantico.html
[20] http://sopa.dis.ulpgc.es/so/cpp/intro_c/
[21] http://webpages.ull.es/users/fsande/talf/cursoc/node3.htm
[22]
http://www2.ate.uniovi.es/fernando/Doc2007/Presentaciones/Programaci%F3n%20en%20
[23] http://laurel.datsi.fi.upm.es/~rpons/personal/trabajos/curso_c/node25.html
[24] http://tikal.cifn.unam.mx/~jsegura/LCGII/java3.htm
[25] Datasheet PIC18F4550

Bibliografía

1. AHO, A. V., SETHI, Ravi, ULLMAN, J.D. , Compiladores. Principios, técnicas y herramientas. México , Addison-Wesley Iberoamericana, 2000

2. LOUDEN, Kenneth C. , Compiler Construction. Principles and Practice

3. U.S.A. Thompson Learning, 1997

4. BROWN, Stephen; VRANESIC, Zvonko , Fundamentals of Digital Logic with VHDL Design , 2nd edition ,New York McGraw-Hill, 2005

5. MORRIS MANO, M. , Digital Design 3rd edition U.S.A. , Prentice Hall, 2002

6. ABEL, Peter , IBM pc assembly language and programming , 5a edición ,U.S.A. , Prentice Hall, 2001

7. BACA, Gabriel , Evaluación de Proyectos , 2a. edición , México ,McGraw-Hill, 2000

8. TOCCI, Ronald J. y Ambrosio, Frank J. , Microprocessors and Microcomputers Hardware and Software, 6a. edición , New Jersey Prentice Hall, 2002

Documentacion:

→ Manuales técnicos de microcomputadores comerciales, HC11, PIC´s, HC08, etc

hlpMPASMAsm.chm, ayuda de MPLAB.

→ *MoProSoft* Por Niveles de Capacidad de Procesos Versión 1.3, Agosto 2005

→ *Guía De los Fundamentos para la Dirección de proyectos* (guía del pmbok®) cuarta edición