15

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ARAGÓN

PROGRAMACIÓN DE MICROPROCESADORES EN VHDL

T E S I S

QUE PARA OBTENER EL TITULO DE:
INGENIERIA EN COMPUTACIÓN
P R E S E N T A:
IVETTE CRUZ FELIPE

ASESOR:
M. EN C. DAVID J. GONZÁLEZ MAXINES

MÉXICO 2001

A MI MAMÁ

TERESA

Gracias por tu apoyo incondicional en todos los momentos de mi vida, porque has sido parte fundamental en cada uno de mis logros los cuales también son tuyos, eres mi fortaleza, la persona que me ayuda a seguir adelante a ser mejor dia con dia, me faltaria espacio para decirte lo mucho que significas para mi, gracias por ser parte de mi vida.

A MI PAPÁ

MIGUEL

Gracias por tu cariño incondicional por enseñarme a que debemos tener fortaleza y continuar hacia adelante sin importar las circunstancias que la vida nos presenta. Te quiero papá.

A MIS HERMANOS

MARY, MIGUEL, DANIEL, MARTHA, ELIZABETH Y ANABEL

Por su comprensión y apoyo incondicional en todo momento asi como su inmenso cariño que sirvió de aliciente para lograr mis objetivos.

INDICE

INTRODUCCIÓN

Los diseños digitales actualmente son cada vez mas complejos en los recursos físicos propios del diseño como en la cantidad de información que manejan; por tal razón es necesario estructurar los procesos de diseño, de una forma sistemática, flexible y confiable.

Una de las principales herramientas, que poseen los diseñadores, son los lenguajes de descripción de hardware, destacándose principalmente el VHDL. Este lenguaje nos permite el diseño de sistemas embebidos los cuales encontramos en el uso diario; algunos ejemplos son aparatos eléctricos, automóviles, usos industriales, computadoras, etcétera. Estas ultimas son el mas claro ejemplo de estos sistemas ya que son controladas por un microprocesador .

A lo largo de este trabajo abordaremos estos temas lo que nos permitirá al final ser capaces de diseñar dispositivos eléctricos cada vez mas eficientes, a menor costo y reducir el intervalo de tiempo de diseño.

En el *Capitulo I Programación en VHDL* se da una amplia descripción del lenguaje de descripción de hardware (VDHL), historia del lenguaje, arquitectura básica, sus ventajas y desventajas, herramientas de soporte, simuladores y aplicaciones en sistemas embebidos.

El *Capitulo II Estructura básica de lo sistemas embebidos en una sola pastilla* se da una explicación de los sistemas embebidos, fases de diseño, clasificación, factores a considerar para el diseño y aplicaciones.

En el *Capitulo III Programación e integración en VHDL de los módulos de un microprocesador* se muestra el diseño en VHDL de un microprocesador de cuatro bits, así como el funcionamiento de cada modulo y la lista de instrucciones para su funcionamiento.

En el *Capitulo IV Diseño de sistemas electrónicos digitales a través del microprocesador VHDL* nos muestra algunas aplicaciones que podemos realizar con el microprocesador así como las bases para nuevos diseños.

Al final del trabajo se comprenderá el uso del lenguaje VHDL, sus múltiples aplicaciones en la industria con el consecuente beneficio hacia la población en general. De ahí su importancia en la comprensión del buen uso y manejo de este tipo de lenguaje.

Capitulo I
Programación en VHDL

1.1 ANTECEDENTES.

El desarrollo electrónico de los últimos años se ha visto fuertemente dominado y conducido por la impresionante evolución de la microelectrónica desde su nacimiento en 1959-1960. Durante los años setenta, junto con la revolución de las memorias RAM y procesadores en forma de chip monolítico, se preparan las condiciones para el gran salto que el diseño microelectrónico daría en los años ochenta.

El desarrollo de nuevas tecnologías, alternativas de fabricación y diseño de circuitos integrados, junto con las metodologías y herramientas de diseño asistido por computadora han sido las innovaciones más importantes de la década de los años ochenta. Éstas se han reflejado en el continuo incremento de la complejidad de los chips, en los sistemas electrónicos y en la gran difusión de las técnicas, metodologías y herramientas de diseño de circuitos integrados que junto con las nuevas alternativas de fabricación, han ampliado el rango de posibilidades para los ingenieros permitiéndoles diseñar chips específicos llamados ASICs (Application Specific Integrated Circuits), para los productos que desarrollan.

Todos estos factores han contribuido directamente a la evolución de los recursos de Cálculo (procesadores, estaciones de trabajo, etc.), quienes a su vez tienen una incidencia decisiva en el desarrollo de nuevas herramientas y entornos integrados de diseño de sistemas electrónicos. Con el desarrollo y uso de tales herramientas para crear

nuevos componentes se cierra el ciclo del soporte mutuo entre ambas tecnologías: microelectrónica e informática.

En los años setenta hubo una fuerte evolución en los procesos de fabricación de los circuitos integrados y junto a las tecnologías bipolares surgieron los circuitos conocidos como MOS (Metal_Oxide Semiconductor).

No importado si eran circuitos digitales o analógicos, estos se desarrollaban con el objetivo de ser componentes estándar para su posterior uso en el diseño de sistemas electrónicos específicos; estos circuitos integrados se desarrollaban completamente en las fábricas.

El esfuerzo del diseño se concentraba en los niveles eléctrico (establecer características e interconexiones de los componentes básicos a nivel de transistor) y topográfico (dibujar las máscaras, layout, necesarias para la fabricación de los dispositivos y sus interconexiones). Prácticamente no había apoyo de herramientas de computación.

Pero a pesar de todas estas dificultades surgieron las primeras memorias DRAM de 1kbyte, el microprocesador 4004 de 4 bits de INTEL y a finales de la década ya habían surgido las primeras memorias de 16 kbytes y los microprocesadores de 16 bits.

En los años ochenta los procesos tecnológicos se volvieron más complejos para poder asumir mayores densidades de integración; se había identificado el conjunto básico de información que se necesitaba conocer a fin de poder realizar diseños para una determinada tecnología, es decir, parámetros eléctricos de los distintos componentes activos y pasivos para poder definir esquemas eléctricos, con ello el diseño de circuitos integrados comienza a hacerse accesible fuera de las fábricas de semiconductores. En ese momento se hace presente el desfase enorme entre tecnología y diseño.

La considerable complejidad de los chips que se pueden fabricar supone unos riesgos y costes de diseño desmesurado e imposibles de asumir, debido a una carencia de metodologías y herramientas de diseño; a raíz de esta situación se inicio una fuerte actividad sobre el desarrollo de metodologías y herramientas de diseño asistido por computadora(CAD), para el diseño de circuitos integrados.

En cuanto a tecnologías se mantienen las bipolares como las más utilizadas para desarrollos analógicos o mixtos, mientras que en el diseño digital los circuitos CMOS se imponen claramente a las antiguas NMOS; es a finales de esta década que surgen las tecnologías BICMOS, que permiten crear sobre un mismo sustrato dispositivos bipolares y CMOS facilitando el desarrollo de circuitos mixtos analógico-digital.

La estructura de herramientas CAD vario considerablemente durante esta década pasando de tener que enfrentarse a distintas herramientas con sus interfaces específicas para cubrir cada punto del diseño, a poder disponer de entornos integrados de CAD donde bajo una misma base de datos y un mismo entorno de usuario se agrupaban y encadenaban las distintas herramientas. Todos estos avances empujaron y se apoyaron en la propia evolución de las plataformas de hardware/software.

En los años noventa las aplicaciones sobre dispositivos programables (dispositivo lógico programable complejo (CPLDs) y Arreglos de compuertas programables en campo (FPGAs)) empiezan a tomar un mayor auge sobre los ASICs. Con el nacimiento del VHDL se empiezan a desarrollar métodos y herramientas para abordar el diseño a nivel funcional o de comportamiento.

Los diseños se basan en macrobloques funcionales (Unidad de Proceso Central (CPU), Procesador Digital de Señales (DSP), microcontroladores, etc.) desarrollados a nivel de software (código VHDL sintetizable) y optimizados a nivel hardware (layout).

En cuanto a metodologías de diseño los años noventa se han caracterizado por una implantación progresiva, en fase de consolidación, de los lenguajes de descripción de hardware (VHDL y Verilog), que junto con las herramientas de simulación y síntesis promueven el uso de las llamadas tecnologías de flujo descendente.

1.2 HISTORIA DE VHDL

Como vemos actualmente una de las principales herramientas que poseen los diseñadores, son los lenguajes de descripción de hardware, destacándose principalmente el VHDL, este lenguaje surgió debido a la necesidad de disminuir costos y de contar con herramientas más eficientes en el diseño ya que anteriormente los procesos tecnológicos se volvían más complejos, los problemas de integración aumentaban y la industria electrónica experimentaba una explosión en la demanda de computadoras, teléfonos celulares y diseños de comunicaciones de datos de alta velocidad, etc. además era cada vez más difícil depurar y dar mantenimiento a los diseños. En un principio los circuitos M.S.I.(Integración Media Escala) y L.S.I (integración a Larga Escala) se diseñaban por medio de prototipos basados en módulos sencillos, cada modulo estaba compuesto por puertas probadas pero este método se fue haciendo obsoleto a finales de los años setenta, es entonces cuando un grupo de investigadores desarrolla los "lenguajes de descripción de hardware" y es así como surgen IDL de IBM, TI-HDL de Texas Instruments, Zeus de General Electric y los prototipos desarrollados por las universidades pero ninguno de estos lenguajes tuvo relevancia.

Hacia 1981 el departamento de defensa de Estados Unidos desarrolla un proyecto cuyo objetivo era disminuir el costo de las inversiones en hardware haciendo más sencillo su mantenimiento con esto se pretendía resolver el problema de modificar el hardware diseñado para un proyecto y utilizarlo en otro así es como surge "VHSIC".

En 1983 Texas Instruments, IBM, Intermetrics empezaron a desarrollar un lenguaje de diseño estándar que permitiera el mantenimiento de diseños y la depuración de algoritmos por lo que en 1984 IEEE (Institute of Electrican and Electronics Engineers) propuso su estándar; a partir de entonces las universidades en conjunto con la industria desarrolló diferentes versiones hasta que en diciembre de 1987 el IEEE adoptó a VHDL como su lenguaje HDL (IEEE 1076-1987), ahora cada 5 años se revisa el proceso de estandarización, la última versión se realizó en 1992 (IEEE 1076- 1992).

Como podemos ver en la actualidad VHDL es un estándar de la industria para la descripción, modelado y síntesis de circuitos y sistemas digitales, el mercado entero de síntesis ha alcanzado aproximadamente los 100 millones de dólares y aumenta año con año es por esto que los ingenieros de todas las áreas deben aprender a programar en VHDL para incrementar su eficiencia.

1.3 ¿QUÉ ES VHDL?

Tal como lo indican sus siglas en ingles VHDL (Very High Speed Hardware Description Language) es un lenguaje de descripción de hardware cuyo objetivo es el desarrollo de sistemas electrónicos; este lenguaje cuenta con una amplia sintaxis que permite el diseño preciso de sistemas digitales, el desarrollo de modelos de simulación y la descripción de hardware, es decir, esto hace posible diseñar, modelar y comprobar desde una compuerta hasta un sistema completo, pero a pesar de esto hereda muchos de los conceptos de los lenguajes de programación de alto nivel como Pascal, C, ADA, etc.

Algunos puntos que VHDL hereda de los lenguajes de programación de alto nivel son el concepto de tipo de datos que a diferencia de otros lenguajes de descripción de hardware solo disponen de un reducido número de tipos de datos (la mayoría de ellos orientado al hardware) y con una escasa o nula capacidad para definir nuevos tipos de datos.

VHDL es un lenguaje que ofrece una enorme flexibilidad para definir nuevos tipos de datos, esta es una característica importante de VHDL, que nos permite describir sistemas electrónicos a distintos niveles de abstracción, y para cada nivel el tipo de datos adecuado a nuestras necesidades.

También hereda la potencia de control de flujo, es decir, control de condiciones (if, case) e iteraciones (for, while), esta capacidad de control y de datos permite al VHDL desarrollar algoritmos bastante complejos que de inicio no puedan tener una relación tan directa con la descripción de hardware.

Otra característica importante es que puede agrupar partes del código en subprogramas, ya sean funciones (function) o procedimientos (procedure), además heredada de los lenguajes de alto nivel la posibilidad de utilizar y desarrollar bibliotecas de diseño. Con todo esto al abordar el desarrollo de cualquier código en VHDL disponemos de un conjunto de datos, operadores, funciones y bibliotecas ya definidos para el área concreta donde vayamos a trabajar.

Para facilitar o permitir la descripción de hardware VHDL cuenta con tres puntos principales que son:

Un modelo de estructura; lo que permite a cualquier sistema electrónico poder dividirse en subsistemas más pequeños hasta llegar al nivel de compuertas que serían las primitivas del diseño digital. Con ello logramos referenciar a cualquier sistema a partir de las distintas partes que lo forman y especificando la conexión entre éstas.

Un modelo de concurrencia; ya que el hardware es por definición concurrente, cualquier sistema electrónico esta formado por un mar de compuertas lógicas todas ellas funcionando en paralelo. El elemento básico que ofrece VHDL para modelar

paralelismo es el proceso (process). De hecho cualquier descripción en VHDL es transformada en un conjunto de procesos concurrentes, estos procesos deben poder comunicarse entre ellos (sincronizarse); un elemento para poder comunicar dos o más procesos es la señal (signal), cualquier cambio en una señal se conoce como evento. Cada proceso es sensible a ciertas señales por lo que cada vez que ocurra un cambio en una señal se ejecutará un proceso.

Un modelo de tiempo; uno de los propósitos del modelado en VHDL es poder observar su comportamiento a lo largo del tiempo (simulación). Esto implica que las construcciones del lenguaje tendrán asociadas una semántica respecto a la simulación, e influirán en esta provocando distintos eventos que sucederán a lo largo del tiempo, y a su vez el modo en que se comportan las sentencias dependerá de los eventos que sucedan a lo largo de la simulación. El concepto de tiempo es fundamental para definir como se desarrolla la simulación de una descripción VHDL.

Como podemos ver debido a las características mencionadas anteriormente VHDL permite implementar un sin fin de circuitos lógicos secuenciales y combinacionales, también diseños más complejos, como CPU's, microprocesadores, manejar ficheros, retrasos en el tiempo, etc. ó diseños con una gran capacidad de CPLD´s (Dispositivo Lógico programable Complejo) y FPGA´s (Arreglos de Compuertas programables en Campo) que van desde 500 hasta mas de 100,000 compuertas, además de proporcionar una gran ventaja al poder crear diseños de gran capacidad y como consecuencia nos permite traer productos al mercado rápidamente.

También nos permite la sintesis automática de circuitos, este proceso consiste en partir de un determinado nivel de abstracción y llegar a una implementación mas detallada y menos abstracta, es decir, la síntesis es un proceso vertical entre niveles de abstracción, del nivel mas alto en la jerarquía de diseño se va hacia el mas bajo nivel de la jerarquía.

Por último, tenemos que el desarrollo en VHDL ésta compuesto de varios módulos en los cuales el diseño es determinado, compilado, simulado y particionado de acuerdo al dispositivo en el cual se va a implementar.

1.4 VENTAJAS Y DESVENTAJAS EN VHDL

Debido a la evolución de la microelectrónica desde su nacimiento, en poco más de 10 años, se pasó del primer dispositivo integrado (flip-flop con unos cuantos transistores) a las primeras memorias y microprocesadores. A los pocos años el diseño de circuitos integrados salió de las fábricas y se hizo accesible a los ingenieros de aplicación dando lugar a los ASICs. A mediados de los ochenta, se dispuso la necesidad de contar con un lenguaje estándar capaz de dar el soporte necesario al proceso completo de diseño de chips y sistemas electrónicos desde la idea hasta la implementación y explotación de un desarrollo en sus distintas etapas y niveles de abstracción y cuya complejidad de por si ya elevada se iba a incrementar en los años noventa. Todo esto dio origen a los lenguajes de descripción de hardware y en específico al VHDL; aunque sólo se debe considerar como una herramienta de base; la implantación de este lenguaje ha supuesto tal cantidad de cambios en el diseño electrónico como nuevas metodologías de diseño de alto nivel y flujos descendentes, herramientas CAD de simulación/síntesis de circuitos y sistemas, nuevos perfiles de los equipos de diseño, cambios en la organización y desarrollo de proyectos, etc; por lo que es necesario mencionar las ventajas que ha proporcionado este lenguaje, que, sin lugar a dudas, nos permite hablar de un antes y un después del VHDL.

Algunas ventajas que nos proporciona el lenguaje son:

- Consta de todas las características de los lenguajes de alto nivel.
- Un solo lenguaje para simulación y síntesis.

- Síntesis automática del diseño a nivel de puertas para una tecnología dada.
- Este lenguaje lo pueden interpretar tanto las personas como los ordenadores, lo cual nos permite proporcionar soporte a las tareas estrictas de diseño(modelado, síntesis, simulación, verificación), a las de comunicación e intercambio de modelos entre los distintos equipos de trabajo y a las de documentación y mantenimiento de los diseños.
- Permite diseñar, modelar, y comprobar un sistema desde el nivel mas alto de abstracción hasta el nivel de compuertas.
- Al estar basado en un estándar (IEEE 1076-1987) garantiza su estabilidad y soporte, y los ingenieros de toda la industria del diseño pueden utilizar dicho lenguaje.
- Simplifica la reducción de herramientas y formatos, así como la descripción/simulación multinivel que facilita el diseño independiente de los diferentes componentes o módulos y por lo tanto la distribución de tareas entre distintos equipos coordinados bajo un mismo entorno de simulación/verificación.
- VHDL utiliza el diseño Top-Down, esto quiere decir que permite describir (modelado) el comportamiento de los bloques de alto nivel, analizándolos (simulación), con lo cual podemos volver a definir la función de alto nivel requerida antes de llegar a niveles más bajos de abstracción en la implementación de este.
- Se puede amoldar a cualquier metodología donde el modelado de hardware tenga sentido.
- Por su definición y diseño como por los niveles de abstracción donde se usan.
- Pueden ser totalmente independientes de la tecnología final de implementación de los circuitos.
- La descripción VHDL de un circuito puede ser totalmente diferente a la tecnología de fabricación, pero se puede incluir información especifica de la implementación final como retrasos, consumo, etc.
- Permite dividir o descomponer un diseño y su descripción en bloques más pequeños.

- Los circuitos digitales capturados usando VHDL se pueden simular fácilmente, y pueden ser reducidos en tecnologías múltiples.
- Los diseños en VHDL se pueden archivar para su modificación y su reutilización posterior, esto facilita la evolución del producto mejorándolo o incrementando sus características funcionales.
- Un diseño en VHDL es fácil de volver a simplificar sobre distintas tecnologías.
- Es un lenguaje rico y de gran alcance.
- En general podemos decir que el VHDL puede proporcionar al diseñador cierta independencia frente a los fabricantes de chips.

Como podemos observar este lenguaje puede aportar ventajas importantes, pero no debemos ignorar que esta sujeto a limitaciones, a continuación veremos algunas desventajas del lenguaje:

- Al ser un lenguaje definido por consenso en una comisión, tiende a ser complejo.
- La revisión del lenguaje por la comisión es lenta ya que se realiza cada 5 ó 6 años como consecuencia siempre hay muchos cambios en cada versión.
- La falta de una sintaxis formal para síntesis dificulta la transparencia de los diseños entre distintos entornos de síntesis, ya que no todas las herramientas interpretan de la misma forma las construcciones del lenguaje e imposibilita abordar claramente la verificación.
- Debido a las características del lenguaje este funciona mejor en los altos niveles de diseño, sin embargo no es así en los niveles de compuertas.
- La enorme flexibilidad del VHDL dificulta la implantación de mecanismos estándar para facilitar el intercambio de módulos.

1.5 ¿POR QUÉ DISEÑAR CON VHDL?

La evolución de la microelectrónica ha sido tan rápida y profunda en los últimos tiempos, que ahora existe la posibilidad de que podamos fabricar circuitos integrados de alta complejidad. Por lo que el enfoque con el que se diseñaban los chips ha ido evolucionando a través de varios niveles de abstracción. Del diseño geométrico del circuito y resolución de las ecuaciones diferenciales, se pasó al diseño con transistores ayudado por simulación eléctrica, para evolucionar posteriormente al diseño lógico con ayuda de editores de esquemas y simulación lógica y eléctrica, hasta llegar a la actual metodología, basada en la descripción del comportamiento de los circuitos mediante lenguajes de descripción de hardware, simulación a nivel de comportamiento y utilización de herramientas de síntesis lógica. Esta metodología y herramientas asociadas son, hoy por hoy, instrumentos básicos y no podríamos pensar en el diseño de circuitos de la complejidad actual sin este tipo de ayudas. Es por ello que su conocimiento resulta obligado para todos aquellos profesionales en el diseño electrónico y de circuitos integrados.

Las metodologías de diseño electrónico denominadas "top down", basadas en el empleo de lenguajes de descripción de hardware, han transformado los procedimientos de diseño de sistemas electrónicos, muy especialmente de circuitos electrónicos.

El VHDL es su más claro exponente, abriendo enormes posibilidades al permitir la simulación con descripciones de partes del sistema con diferentes niveles de abstracción. Esto, unido a la posibilidad de realizar la síntesis automática, y a la concepción de bloques reutilizables y reconfigurables en función de las necesidades de la aplicación, ha permitido dotar al diseñador de enormes recursos que hacen posible abordar la creciente complejidad con mayores garantías de éxito.

1.6 HERRAMIENTAS DE SOPORTE.

Durante este tiempo han aparecido en el mercado herramientas comerciales que sustentan metodologías de diseño maduras, estas metodologías resultan imprescindibles en el diseño de la complejidad y prestaciones soportadas por la tecnología microelectrónica actual y se utilizan en un amplio rango de aplicaciones como sistemas de cómputo de propósito general, sistemas embebidos, sistemas de telecomunicaciones, sistemas de control, electrónica aerospacial, del automóvil y electrónica de consumo.

Las herramientas CAD (Diseño Asistido por Computadora) representan el medio para diseñar estos sistemas con la fiabilidad y la productividad requeridas. Estas herramientas CAD se pueden clasificar en cuatro grandes grupos dependiendo de su papel en el proceso de diseño completo: edición, análisis, síntesis y optimización.

- ***Herramientas de Edición.*** Los editores permiten la captura del comportamiento o la estructura del diseño, existen editores especializados en distintas representaciones del diseño como FSMDs, esquemáticos y layout.

- ***Herramientas de Análisis.*** . Las herramientas de análisis permiten extraer propiedades del diseño; una herramienta de análisis es el simulador que constituye la herramienta más importante en la actualidad ya que la simulación del circuito en las distintas etapas del proceso de diseño es el medio más frecuente para comprobar su corrección. Otras herramientas de análisis ampliamente utilizadas son el generador de patrones de test y el analizador temporal.

- ***Herramientas de Síntesis.*** Las herramientas de síntesis realizan automáticamente alguno de los pasos del proceso de diseño generando una implementación detallada de una descripción más abstracta. Esta herramienta mantiene una intima relación con la herramienta de optimización.

Dentro de la síntesis se lleva a cabo un refinamiento gradual del diseño que tiene por objeto la obtención de un esquema o lista de componentes y sus interconexiones basadas en bibliotecas de celdas y módulos. Estas herramientas de síntesis se implementan mediante distintos procesos de síntesis encadenados de forma transparente al usuario. Estos procesos son:

1.- Síntesis RT (Transferencia de registros), que determina los elementos de memoria y el conjunto de ecuaciones lógicas u operadores necesarios, en base a fases de partición, distribución (schedulling) y asignación (allocation) de recursos, bajo las restricciones impuestas por el diseñador.

2.- Síntesis Lógica, que se encarga de optimizar las ecuaciones lógicas para minimizar el hardware necesario a nivel de puertas y registros utilizando algoritmos de reducción y asignación de estados.

3.- Mapeo tecnológico que es muchas veces indistinguible de la síntesis lógica, en la que las puertas y registros utilizados se mapean de forma optimizada sobre los elementos disponibles en la biblioteca de celdas y módulos correspondientes a la tecnología utilizada para la implementación física del diseño.

- . ***Herramientas de Optimización.*** La herramienta de optimización permite mejorar la calidad del diseño a un determinado nivel de abstracción en función de los parámetros definidos por el usuario (coste, velocidad, consumo, etc.).

Como vemos las herramientas de *Diseño Asistido por Computadora* (CAD) hacen uso de notaciones de diseño, ya sean explícitas o transparentes para el usuario. Las notaciones explícitas son aquellas relevantes para el diseñador, entre ellas cabe señalar las tablas de verdad y de estado (para minimización lógica), la lista de puertas (para simulación lógica) o la lista de componentes (para simulación analógica).

Las notaciones transparentes para el usuario son formatos utilizados por la herramienta a los que el usuario normalmente no tiene acceso y en muchos casos ni siquiera tiene conocimiento de su uso. Cabe citar los formatos intermedios utilizados por una herramienta determinada o por un conjunto de ellas en un entorno CAD o las librerías de componentes en alguna tecnología.

Los lenguajes de descripción de hardware (HDLs), representan un medio de descripción explícito del circuito a diseñar. La novedad aportada por los HDLs reside en la utilización de conceptos de ingeniería de software a la descripción y modelado de hardware. La sintaxis de los lenguajes de descripción de hardware resulta en general muy similares a los lenguajes de programación. Esta similitud formal entre ambos lenguajes tiene consecuencias particularmente en su uso. El diseñador no debe olvidar que, aunque la sintaxis sea similar a la de un lenguaje de programación, el objetivo del código es la descripción del hardware que se quiere obtener.

1.7 ARQUITECTURA BASICA DE UN TRABAJO EN VHDL

Haciendo una comparación con cualquier otro lenguaje de programación podemos ver que el código es muy similar en cuanto a las instrucciones utilizadas, pero no es así ya que el orden de las instrucciones no importa como en el caso de los lenguajes de programación de software, además las instrucciones siempre se están ejecutando ya que VHDL asemeja el comportamiento real de un circuito.

La estructura básica de un diseño en VHDL está formado por módulos, llamados entidad, arquitectura, declaración y cuerpo del paquete; y configuración, en un programa VHDL la entidad y la arquitectura son unidades necesarias, mientras que la configuración, declaración y cuerpo del paquete son unidades opcionales. A continuación se explicara cada unidad:

Entidad (Entity). En la descripción de la entidad se definen las señales de los puertos de comunicación del exterior, es decir, se describen los pines que se van a utilizar en el circuito, cada pin se especifica con *in* para entrada, *out* para salida e *inout* si es de entrada/salida, cada señal lleva una especificación que indica su origen, algunas señales que podemos encontrar son bit, bit_vector, std_logic y std_logic_vector; una entidad puede, a su vez, estar constituida por otras entidades. Una entidad consta de: declaración de la Entidad y declaración de la Configuración. La sintaxis de la entidad es la siguiente:

entity identificador is
[genéricos]
[puertos]
end [entity] [identificador] ;

Donde :
El identificador es el nombre que va a recibir la entidad, los puertos son los que determinan la interfaz del dispositivo con el exterior, los genéricos son un conjunto de parámetros que permiten modificar la función de la entidad (opcional).

Arquitectura (Arquitecture). Indica los cambios que se van a realizar en los datos de entrada para transformarlos en salidas, es decir, nos describe el funcionamiento del módulo declarado como entidad . Existen diferentes tipos de arquitecturas como:

Estilo Algoritmico. Donde la función del dispositivo se define mediante un algoritmo secuencial como cualquier programa realizado en un lenguaje común como C.

Estilo Flujo de Datos. Donde la función del dispositivo se define por medio de un conjunto de ecuaciones que indican el flujo de los datos entre módulos.

Estilo Estructural. Esta arquitectura esta integrada por un conjunto de dispositivos interconectados y evaluados mediante señales.

Estilo Mixto. Es la arquitectura que nos permite mezclar uno o más de los estilos mencionados anteriormente e implementar una sola arquitectura.

Sintaxis de arquitectura:

architecture identificador of nombre is
[declaraciones]
begin
[sentencias];
end [identificador];

Donde :
El identificador es el nombre de la arquitectura, las declaraciones son las definiciones de las señales y procesos internos del diseño, las sentencias pueden ser: bloques, procesos, componentes, asignaciones y tablas, es decir la descripción del funcionamiento del circuito.

Paquetes (Packages). Los paquetes permiten agrupar un conjunto de declaraciones para que puedan ser usadas en el diseño de diferentes circuitos sin ser repetidas en la declaración de cada uno. La estructura básica en la declaración de un paquete está dividida en dos partes:

Declaración del Paquete(Package Declaration). Nos proporciona una visión externa y simplificada del componente ya que especifica el funcionamiento de los tipos, funciones, etc; debe llevar la palabra reservada **package**. Su sintaxis es la siguiente:

package identificador of package is
[declaración de funciones]
[declaración de tipos, etc]
end [identificado];

Donde:
El identificador es el nombre del paquete, las declaraciones es como la entidad donde se definen señales.

Cuerpo del Paquete(Package Body). Especifica el funcionamiento de funciones, tipos, etc. que nos permiten su implementación, de forma similar a la arquitectura, debe llevar las palabras reservadas package body. Su sintaxis es la siguiente:

package body identificador of package is
[definición de funciones]
[definición de tipos]
end [identificador];

Donde:
La estructura del cuerpo del paquete funciona igual que la arquitectura, es decir, describe el funcionamiento de los módulos en la declaración del paquete.

Todo esto hace nos permite que una vez declarados los subprogramas dentro de un paquete, podamos utilizarlos haciendo uso de una llamada al proceso, asignándole un nombre y la lista de parámetros necesarios.

Para poder usar o llamar a un paquete que ha sido creado anteriormente, debemos incluir la palabra reservada *use* antes del cuerpo de la arquitectura como se muestra a continuación:

use work. identificador del paquete. nombre del componente;

Además tenemos que especificar lo siguiente:

1. El nombre de la librería donde se encuentra el paquete.
2. El nombre del paquete.
3. El nombre del componente que se deseamos habilitar.

Cuando queremos habilitar todos los componentes declarados en un paquete, por comodidad o por no saber exactamente donde se encuentra el recurso que deseamos usar se utiliza la palabra *all* de la siguiente forma:

use work. identificador del paquete.all;

Configuración. Se utiliza cuando creamos varias arquitecturas en un solo diseño y seleccionamos cual es la más conveniente.

1.8 SIMULADORES

Dado que VHDL es un lenguaje estándar, empresas como Cypress, Xilinx, Altera, Actel, etc; que son fabricantes de PLD's (Dispositivo de Lógica Programable) y FPGA's (Arreglos de Compuertas Programables en Campo) han desarrollado su propio compilador con características especiales y diferentes funciones propias de cada compañía. Para la realizar este proyecto nos basaremos en el programa *WarpR4* de la compañía *Cypress Semiconductor*.

La herramienta de Cypress es tan potente como las demás, contando además con un gran soporte técnico vía e-mail. El nombre de la herramienta se llama WarpR4, que actualmente va por su versión 4.3. WarpR4 es un conjunto de programas que está orientado a la creación de un programa propio de VHDL (*.vhd), para compilarlo (Galaxy) y posteriormente simularlo (Nova). Para conseguir este software se puede solicitar por correo normal o electrónico en la página web de Cypress.

1.8.1 GALAXY DE CYPRESS

El programa Galaxy es el núcleo de WarpR4, ya que nos procesa los programas creados, nos permite editarlos y elegir alguna de las distintas opciones de compilación. Su pantalla principal es la siguiente (figura 1.1):

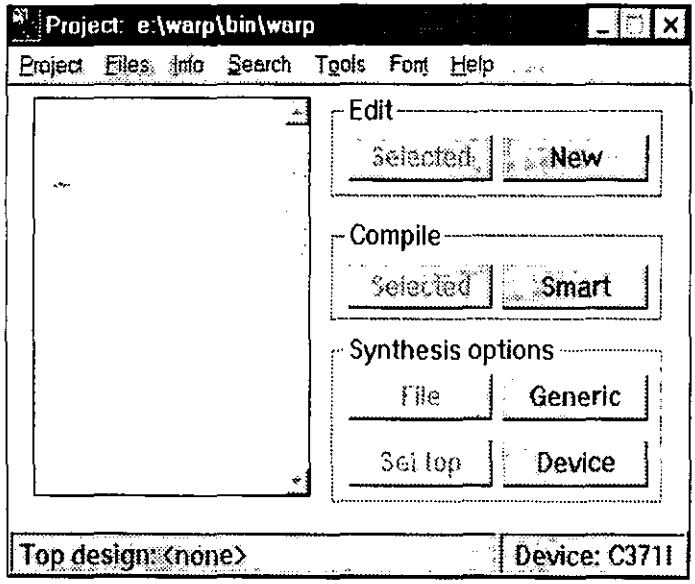


Figura 1.1

Al iniciar el programa por primera vez nos aparecerá una pantalla que nos permitirá crear un nuevo proyecto, para poder realizar esto debemos introducir en esta pantalla, la ruta y nombre del proyecto, según nuestras preferencias, a la cual se le añadirá automáticamente la extensión wpr:

Una vez hecho esto pasaremos a la ventana principal; si queremos crear un nuevo programa para compilar, debemos llamar al editor de texto, esto lo hacemos oprimiendo el botón *new* que se encuentra en el recuadro *edit*. Entonces se nos abrirá una ventana como la siguiente (figura 1.2):

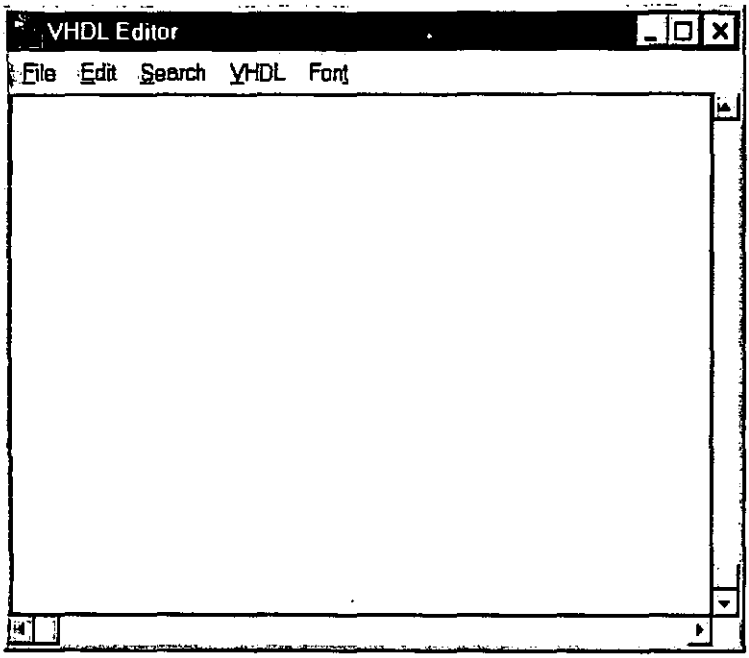


Figura 1.2

En esta ventana podemos crear o modificar el código fuente de VHDL, este editor tiene una serie de guías para ayudarnos a programar; una vez escrito todo el código del programa debemos guardarlo con extensión *vhd*, en el mismo subdirectorio en el cual creamos el proyecto extensión *wpr* para que pueda ser compilado sin ningún problema por el programa.

Antes de iniciar la compilación debemos seleccionar algunos parámetros como qué tipo de circuito vamos a usar, el tipo de optimización, etc; Esta tarea la realizamos con los botones de la parte inferior del programa: file, set top, generic y device.

El botón *file* nos servirá para decirle al programa que queremos usar alguna librería existente o si queremos usar un creada.

El botón *set top* sirve para indicar cual de los programas a compilar es el principal del cual dependen todos los demás.

El botón *generic* nos lleva al menú que nos permitirá elegir ciertas opciones como la optimización de la velocidad o el área usada del dispositivo.

El botón *device* nos permitirá elegir el tipo de dispositivo que utilizaremos para grabar nuestro diseño, que hacer con las salidas no usadas, que tipo de Flip-Flops queremos usar, etc.

Una vez seleccionadas las opciones anteriores podemos empezar a compilar oprimiendo los botones *selected* (permite compilar un diseño) o *smart* (para la programación de varios diseños a la vez, es decir, programación en alto nivel) dependiendo de los programas que se van a compilar.

Cuando se realiza la compilación nos aparece una pantalla con las librerías usadas, fecha, hora, errores, etc. Desde esta pantalla podemos acceder a las líneas de código erróneas directamente y modificarlas para solucionar el problema. En esta ventana nos damos cuenta de la forma interna de trabajar del programa, debido a que nos proporciona el nombre de los programas que van verificando el compilador en busca de errores, y como va transformando el código original en un programa que nos permitirá grabar posteriormente en un circuito.

La compilación es terminada cuando en la barra de estados de la pantalla aparece "*Compilation successful*" lo que nos indica que el proceso de compilación ha tenido éxito y que se genero el programa con extensión *jed*, necesario para la grabación de los dispositivos.

Una vez compilado el programa, llamaremos al simulador de VHDL llamado Nova, oprimiendo en el menú "Tools/Nova".

1.8.2 NOVA DE CYPRESS

Nova es el complemento de Galaxy, ya que va a simular los programas compilados previamente, para la simulación podemos introducir las entradas que deseamos para comprobar si las salidas que vamos a obtener son las correctas. La pantalla principal es la siguiente (figura 1.3):

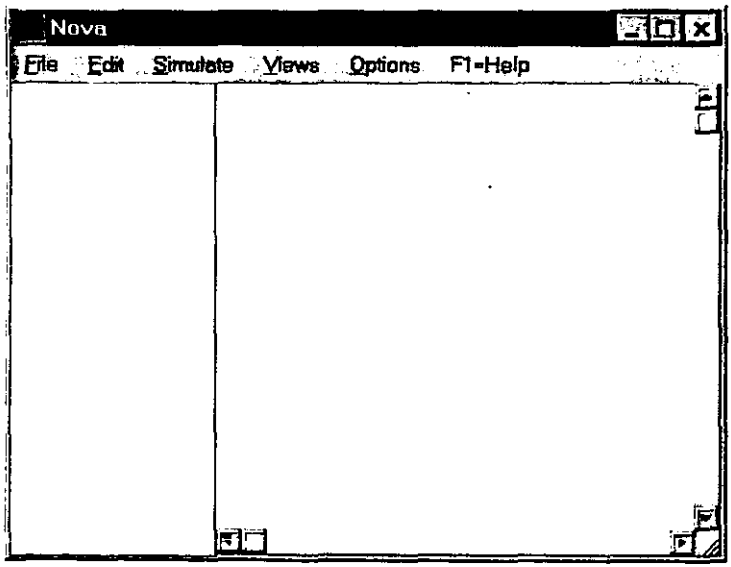


Figura 1.3

Para simular un programa *jed* debemos abrirlo con el comando "file/open". Lo siguiente que nos aparece es una pantalla con las entradas y las salidas que admitirá el dispositivo, nosotros podemos introducir las entradas y darle la forma de acuerdo a la opción que más nos convenga como nivel lógico alto o bajo, que sea un reloj, que tenga un pulso,.etc.

Una vez modificada la forma de las entradas, podemos acceder al comando *execute* del menú *simulate;* el programa nos pondrá en rojo las salidas que se obtienen con los datos de entrada.

Este simulador es muy útil ya que cuando tenemos los resultados de salida nos damos cuenta si funciona correctamente o si es incorrecto con lo cual podemos corregirlo para simularlo otra vez hasta que los resultados coincidan con lo que teníamos planeado.

Una vez que hayamos dado este paso, estaremos preparados para grabar nuestro dispositivo.

1.9 APLICACIONES EN SISTEMAS EMBEBIDOS Y MICROCONTROLADORES

Una definición de propósito general de los sistemas embebidos es que son microprocesadores (chips) o dispositivos electrónicos incorporados en componentes de aparatos o equipos. Todos los ordenadores contienen microprocesadores, pero además se encuentran en equipos utilizados en oficinas, factorías, plantas de proceso, maquinaria, automoción y en los hogares. Con frecuencia estos microprocesadores operan, controlan, protegen o monitorizan procesos vitales para la organización.

Todos los sistemas embebidos incluyen computadoras o microprocesadores. Algunas de estas computadoras sin embargo son sistemas muy simples comparados con una computadora personal. Los sistemas embebidos se pueden agrupar en tres grupos:

No programables. Son aquellos que en su fabricación han sido diseñados para realizar una función determinada y no se puede modificar. Esto implica que en la mayoría de los casos sea necesaria su sustitución para resolver otros problemas.

Programables. Son aquellos en los que su diseño permite variar la función o funciones iniciales para las que fue fabricado, permitiendo su adaptación a cualquier propósito.

Programables sólo en ROM (Read Only Memory). Son aquellos que se programan durante el proceso de fabricación mediante técnicas láser y posteriormente no pueden ser reprogramados, por lo que para realizar otra función con ellos es preciso sustituir la ROM.

Estos sistemas controlan gran cantidad de procesos que van desde sistemas biomédicos como equipos de diálisis, equipos de monitorización fetal, equipo hospitalario además de equipos de fabricación de productos médicos y equipos de laboratorio; sistemas de telecomunicación donde encontramos microprocesadores para controlar el tiempo y las fechas como en el GPS (Global Positioning System), centrales telefónicas, faxes, correo electrónico o videoconferencia, ó en sistemas de servicio como en energía, agua, transportes ya que también contienen microprocesadores.

En cuanto a su funcionamiento, los sistemas embebidos van desde los más simples que son capaces de realizar una sola función a los más complejos que requieren de programas de aplicación y cuyo objetivo final es el poder realizar funciones muy variadas.

Estos sistemas presentan una problemática diferente del resto de los sistemas, ya que son inseparables del software y del hardware en el que están desarrollando su función por lo que en algunos casos es difícil detectarlos además de que el usuario final desconoce en muchos casos de su existencia y por lo tanto es complicado determinar o solucionar sus fallas. A continuación presentamos algunos ejemplos donde encontramos sistemas embebidos:

- Aviones Comerciales y de Carga (motores, instrumentos, mantenimiento).
- Maquinas Contestadoras.
- Carros (manejo de los motores, sistemas de predicción para servicios).
- Sistemas de Control de Clima.
- Calculadoras.
- Sistemas Eléctricos.
- Elevadores
- Fax
- Sistemas de Protección Contra Fuego.
- Equipo Hospitalario.
- Swiches y Routers de Redes WAN y LAN.

- Hornos de Microondas.
- Teléfonos Celulares.
- Fotocopiadoras.
- Equipo de Hardware y Software en compañías de envío de paquetes.
- Sistemas de Seguridad de Control de Acceso, etc.

Bibliografía

1. Teres et al.1998.VHDL Lenguaje Estándar de Diseño Electronico.Ed.McGraw Hill/Interamericana.España.

2. Warp.1997.VHDL Development System User's Guide.Cypress Semiconmductor

3. Powel Douglass.1998.Real Time UML Developing Efficient Objects for Embedded Systems.Ed.Addison-Wesley.

4. Christian Tavernier.1994.Circuitos Lógicos Programables.Ed.Paraninfo.Madrid España

5. Skahill Kevin, Cypress Semiconductor.1996.VHDL For Programmable Logic. Ed.Addison-Wesley

6. Http://bugs.uv.es./dpt/atc/asignaturas/ti/curs/node21.html

7. Http://www.doulos.co.uk/hegv/index.htm

8. Http:// www.cseg.inaoep.mx/~ariasm/arqcomp/vhdlweb/inicio2.htm

Capitulo II

Estructura Básica de los Sistemas Embebidos en una Sola Pastilla

2.1 QUE ES UN SISTEMA EMBEBIDO

Los microprocesadores tienen un gran uso como componentes fundamentales en sistemas electrónicos modernos ya que incrementan su capacidad y proporcionan una mejor ejecución reduciendo su costo; cuando un sistema electrónico incorpora un microprocesador se cuentan con herramientas adicionales con tan solo un incremento en el costo de software de desarrollo, debido a esto cada semana millones de pequeños chips de computadora salen de fábricas como Motorola, Mitsubishi, etc. y encuentran su camino dentro de productos de uso común como sistema de transporte, producción de comida, defensa militar, sistemas de comunicación, etc.

Estos sistemas electrónicos basados en un microprocesador y diseñados para una aplicación especifica son llamados *sistemas embebidos*, es decir, en un sistema cuyo hardware y software es especialmente diseñado y optimizado para resolver un problema eficientemente. El termino *embebido* se refiere al hecho de que la microcomputadora es encapsulada en un solo circuito;. estos sistemas interactúan con todo lo que los rodea siendo el monitor o el control de algún proceso, este hardware es usualmente diseñado desde un nivel componente en un circuito integrado o a nivel tarjeta, y el software que controla el sistema es programado o arreglado en una ROM (Memoria de Solo Lectura) y no es accesible para el usuario del dispositivo.

Los sistemas embebidos siempre están compuestos por un microprocesador, memoria, entradas y salidas a periféricos y un programa de aplicación dedicado guardado permanentemente en la memoria.

Un programa de aplicación para un sistema embebido es una parte integral del sistema y usualmente debe ejecutarse sin el servicio de un sistema operativo, así la aplicación del programa debe incluir el software para controlar e interactuar con los dispositivos periféricos del sistema.

En la actualidad estos sistemas están presentes en la vida diaria formando parte de lavadoras, videos, televisores, calefacción, sistemas de alarma, procesadores de alimentos, computadoras, etc.; estos sistemas puede ser muy sencillos o muy complejos; ya que controlan desde las luces de los tenis hasta los sistemas de control de los aviones militares, cuentan con varias entradas y salidas a otros sistemas o a periféricos que son dispositivos conectados al sistema principal y controlados por el microprocesador. Los microprocesadores que controlan este tipo de sistemas se pueden dividir en:

Microprocesadores individuales. Que se encuentran en dispositivos como sensores de temperatura, detectores de humo y gas, interruptores de circuitos (www2.unam.edu.ar).

Conjuntos de microprocesadores sin funciones reloj. Que se encuentran en controladores de flujo, amplificadores de señal, sensores de posición, servomecanismos de válvula (www2.unam.edu.ar).

Conjuntos de microprocesadores con funciones reloj. Que se encuentran en equipos médicos de monitoreo, controladores, centrales telefónicas, sistemas de adquisición de datos (SCADA), sistemas de diagnostico y tipo real (www2.unam.edu.ar).

Sistemas Computarizados usados en control de procesos e industrias. Que son computadoras conectadas a equipos para controlarlos (www2.unam.edu.ar) .

A continuación tenemos una lista de aplicaciones con sistemas embebidos. Cada sistema acepta entradas, cálculos y salidas generales. Otro concepto general de estos sistemas es que los sistemas embebidos corren en tiempo real.

CAMPO	APARATOS	FUNCIÓN DE EJECUCION DEL SISTEMA
Hogar	Lavadora	Control del agua y ciclo de giros
	Control Remoto	Acepta la señal de las teclas y manda pulsos infrarrojos al sistema base
	Despertadores y Relojes	Mantiene el tiempo, alarma y el display
	Juegos y juguetes	Entrada del Jostick y salida de video
	Audio / video	Interactúa con el operador
Comunicación	Contestadoras Telefónicas	Reproductor de mensajes de salida , salva y organiza mensajes
	Teléfonos celulares	Teclado de entrada, sonido entrada y salida, comunicación con la estación central

(Tomado de Ganssle J.1999)

CAMPO	APARATOS	FUNCIÓN DE EJECUCION DEL SISTEMA
Automóviles	Freno Automático	Optimiza el frenado y la estación de parada
	Encendido eléctrico	Inyectores fuel y control de encendido
	Asientos y ventanas eléctricas	Recuerda las preferencias de cada conductor
Militar	Armas pequeñas	Reconoce el blanco
	Sistemas de Misiles	Orden directa la blanco deseado
	Sistema de posición global	Determina donde estas tu en el planeta
Industrial	Sistemas de control de trafico	Detecta la posición del carro y controla las luces del trafico
	Sistemas Robot	Control de motores y entradas de sensores
	Código de barras lectura y escritura	Entradas de lectura y salidas de escritura para inventario y control
Médico	Monitor cardiaco	Mide funciones del corazón
	Tratamiento de cáncer	Controla radiación, drogas o calor

(Tomado de Ganssle J.1999)

2.2 DISEÑO DE SISTEMAS EMBEBIDOS

Para tener una estructura fundamental común los sistemas embebidos tienen un ciclo de desarrollo representado por el proceso en el cual el sistema es creado. En este ciclo de desarrollo se nos muestran los avances en la ingeniería para maximizar la posibilidad de producir un sistema efectivo confiable y mantenible.

Cada fase de ciclo de desarrollo es identificado por un bloque con una actividad dominante, el resultado primario de cada fase es mostrado en la salida de bloque. Cada resultado esta representado por documentación que describe el sistema en esa etapa produciendo con esto una documentación completa y exacta de cada fase de desarrollo y del sistema final. El mismo ciclo es aplicable a un sistema diseñado para una sola persona o para un equipo.

A continuación se muestra el ciclo de desarrollo de un sistema embebido en la figura 2.1.

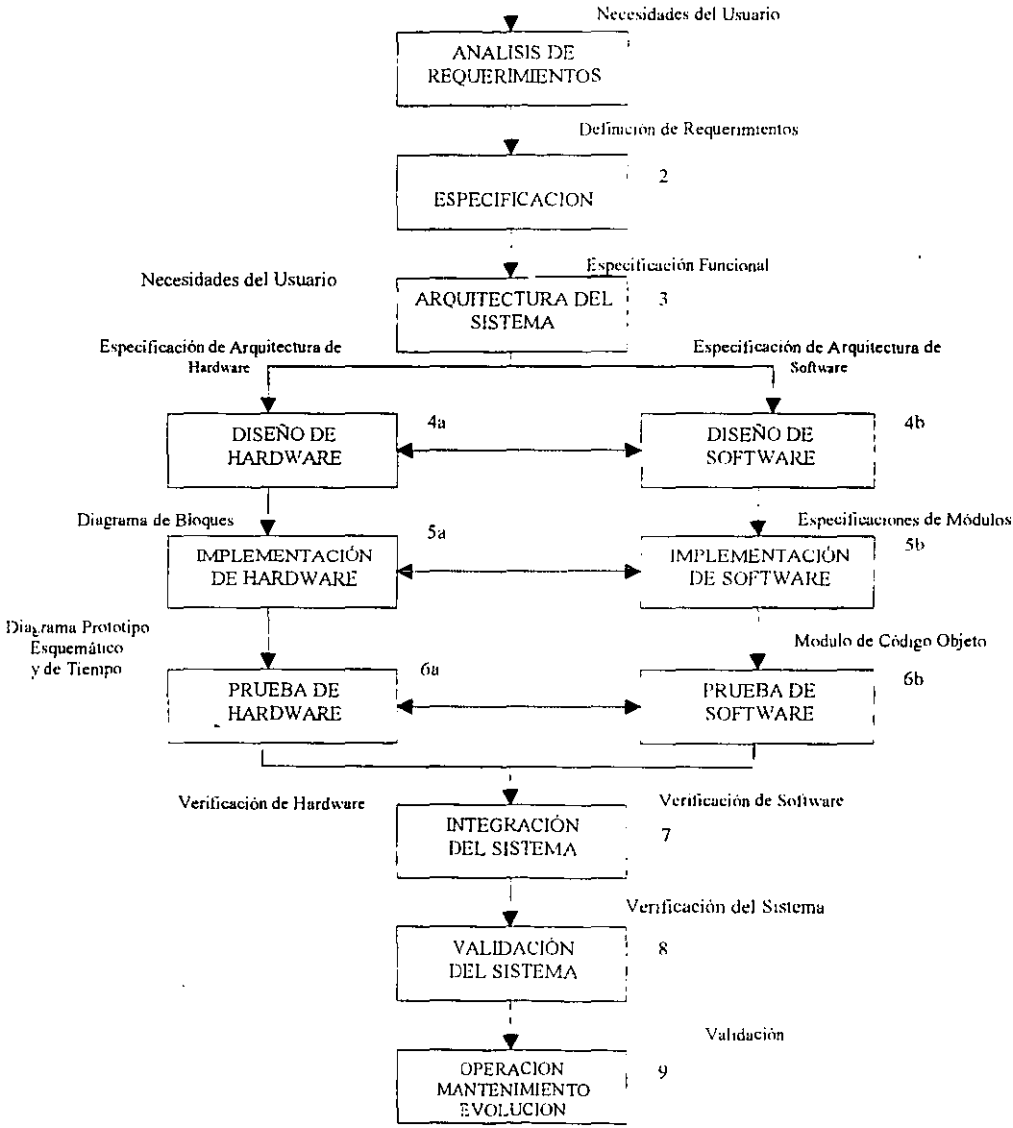


Figura 2.1.Ciclo de Desarrollo de un Sistema Embebido (Tomado de Short L.K. 1998)

El diagrama mostrado anteriormente nos muestra las fases que ocurren en forma separada o paralela pero solo una a la vez y en secuencia. En realidad el desarrollo de la información, los problemas encontrados y las decisiones hechas durante alguna fase pueden causar que la fase interactúe con fases anteriores y siguientes con lo que puede que las fases previas sean reactivadas. El resultado del proceso entero es iterativo y secuencial. Las fases del proceso las describimos a continuación.

Análisis de Requerimientos. El análisis de requerimientos involucra un examen a detalle de las necesidades del usuario final, es decir, el problema a ser resuelto. En esta etapa interviene el usuario final o cliente así como sus necesidades ya que estas son estudiadas para determinar que quiere el usuario o que necesita que haga el sistema. Estos requerimientos se definen del punto de vista del usuario y sin ninguna referencia para posibles soluciones. Los resultados del análisis se registran en la definición de los requerimientos.

Especificaciones. Las especificaciones se basan en los requerimientos, las funciones, operaciones y la interface con el usuario, estas especificaciones definen al sistema y lo asemejan para que sea directamente probado, es decir, determina que hará el sistema pero no como quedara.

Arquitectura del Sistema. La arquitectura del sistema son las especificaciones de software y hardware; en esta fase se define que tipo se va a utilizar, además de que cada elemento es descrito independientemente, todo esto nos permite hacer un balance entre el software y el hardware que es un punto critico para la ejecución y el costo del sistema. Esta etapa determina que parte del sistema es implementada primero en el software y que parte es implementada en el hardware.

Diseño de Hardware. Durante la fase de diseño de hardware la funciones y la interface de entrada y salida son definidas; la primera de forma general y la segunda de manera especifica, este diseño puede ser definido a gusto del usuario o siguiendo uno de los estándares aceptados por la industria. Esta fase incluye nombre de señales, funciones y características así como diagramas de bloques del sistema.

Implementación de Hardware. Es la etapa donde se seleccionan los dispositivos que se van a utilizar (microprocesador, memoria, periférico, etc.) para la integración del sistema; en esta etapa también se realiza un análisis de tiempo así como diagramas esquemáticos y diagramas de tiempo. Todo lo anterior permite la construcción de un prototipo.

Pruebas de Hardware. Esta etapa consisten en realizar pruebas individuales a los dispositivos para determinar sus especificaciones y el intervalo de tiempo para ejecutar un proceso.

Diseño de Software. Es la fase que permite el diseño de ejecución de flujo de datos así como la implementación de funciones y los procedimientos que contiene cada modulo para que interactuen entre si lo cual nos permite definir las jerarquías entre los módulos que componen el sistema.

Implementación de Software. Consiste en definir algoritmos a detalle y las estructuras de datos que van a ser desarrollados, además el software es diseñado para ser reusable en otras aplicaciones, sin embargo, siempre se diseña para una aplicación especifica. La programación del sistema se realiza por módulos independientes que posteriormente son ensamblados o compilados; todos los errores detectados son corregidos en esta etapa.

Pruebas de Software. Los módulos programados en la etapa de implementación de software son probados individualmente esto se realiza por medio de la simulación, después de realizar pruebas individuales a cada modulo estos son conectados para tener el programa completo y posteriormente se hacen pruebas de cada estado.

Sistema de Integración. Si las fases de software y hardware se ejecutaron individualmente y los resultados fueron exitosos es el momento de que el sistema de software y el prototipo de hardware sean integrados y probados. Los problemas en la interconexión o en las interfaces son detectados y corregidos.

Validación del Sistema. Esta fase nos permite conocer si el sistema funciona correctamente o si existen errores los cuales son corregidos, así como también se pueden realizar modificaciones solicitadas por el usuario.

Operación, Mantenimiento y Evolución. Esta es la fase final donde el sistema es implantado; pero cuando el sistema ya esta en servicio puede ser modificado para satisfacer nuevas necesidades e incrementar su calidad. En esta etapa se le puede dar soporte al usuario o corregir algún error posterior.

2.3 CLASIFICACIÓN DE LOS SISTEMAS EMBEBIDOS

La clasificación que veremos a continuación es utilizada por la IEEE.

Microprocesadores individuales. Estos pueden ser encontrados en dispositivos pequeños como sensores de temperatura, detectores de gas y humo, circuitos de interrupción de corriente, etc., sin embargo presentan dificultad para ser manejados.

Pequeñas líneas de producción. Estos pueden ser encontrados en controladores de flujo, amplificadores de señal, sensores de posición, actuadores de válvulas, etc. sin embargo su operación interna depende de un reloj.

Líneas de producción con funciones de tiempo. Son los dispositivos de distribución eléctrica como controles, centrales telefónicas, sistemas de monitoreo y adquisición de datos, diagnostico, sistemas de control en tiempo real, etc. Estos sistemas pueden ser elementos locales en sistemas más grandes ya que proporcionan información de sus sensores como en los sistemas de computación usados en la manufactura o en el control de procesos. Estos pueden formar parte de una PC y pueden involucrar algunas bases de datos (como eventos); aquí se relacionan casos donde la computadora se conecta a la planta o a la maquinaria para llevar el control de estas. Aquí los sistemas de computación se usan en conjunto para controlar y monitorear el proceso. Este tipo de sistemas frecuentemente involucra a otros sistemas embebidos que trabajan en conjunto.

2.4 FACTORES A CONSIDERAR PARA EL DISEÑO DE UN SISTEMA EMBEBIDO

Como podemos darnos cuenta en el diseño de sistemas embebidos tienen que considerarse diferentes factores, a continuación mencionaremos los mas importantes:

- Costos de sistema que incluyen entrenamiento, desarrollo y pruebas.
- Costos de material que incluyen partes y refacciones.
- Costos de manufactura que depende del número y complejidad de los componentes.
- Costos de mantenimiento que incluyen revisiones para arreglar errores y escalar el proyecto.

- El tamaño de la memoria de solo lectura (ROM) debe ser suficientemente grande para soportar instrucciones y arreglos de datos
- El tamaño de la memoria de acceso rápido (RAM) debe ser suficientemente grande para soportar variables globales y locales.
- La memoria EEPROM tiene que soportar arreglos de consultas que están en un arreglo configurable
- La velocidad debe ser suficientemente rápida para ejecutar el software en tiempo real.
- El ancho de banda de entrada/salida afecta la rapidez de datos de entrada/salida.
- El tamaño de los datos de 8,16,32 bits debe coincidir con los datos que van a ser procesados.
- Las operaciones numéricas que como multiplicaciones, divisiones, signo y punto flotante.
- Las funciones especiales como números complejos, lógica difusa, multiplicación / acumulador.
- Suficientes puertos paralelos para todas las señales digitales de entrada/salida.
- Suficientes puertos seriales para la interface con otra computadora o dispositivo de entrada/salida.
- Funciones de tiempo para generar señales, mediciones de frecuencia, periodo de mediciones.
- Tamaño del circuito.
- Fuentes disponible.
- Disposición de lenguajes de alto nivel, compiladores y simuladores
- Requerimientos de poder ya que mucho sistemas operan con pilas.

Además la mayoría de los sistemas embebidos incluyen componentes que miden y controlan parámetros del mundo real como posición, velocidad, temperatura y voltaje debido a esto el diseño siempre incluye dispositivos como amplificadores, filtros y convertidores digitales.

2.5 DISEÑO DE MICROPROCESADORES VHDL

Como vimos anteriormente los sistemas embebidos tienen un infinidad de aplicaciones, un ejemplo muy claro de esto son los microprocesadores los cuales están compuestos por varios dispositivos que interactúan entre si para el control de alguna aplicación ya que se encargan de realizar todas las operaciones y de controlar otros dispositivos recibiendo información y dando órdenes para que los demás elementos trabajen

El diseño de microprocesadores se facilita gracias a las múltiples herramientas que actualmente existen; como vimos en el capítulo uno el lenguaje VHDL es una herramienta muy poderosa para el diseño que nos permite la reducción de tiempo y costos además es muy flexible.

En el capitulo siguiente desarrollaremos un microprocesador de 4 bits al qué llamaremos UNAM4 utilizando el lenguaje VHDL.

2.6 Bibliografía

1. Ganssle Jack.1999.The Art of Designing Embedded Systems.Ed.Newnes.USA

2. Valvano W.Jonathan.2000.Embedded Microcomputer Systems Real Time Interfacing.Ed.Brooks/Cole.Ca.USA

3. Powel Douglass.1998.Real Time UML Developing Efficient Objects for Embedded Systems.Ed.Addison-Wesley.

4. Kenneth L. Short.1998.Embedded Microprocessor Systems Desing.Ed.PrenticeHall

5. www.imagecraft.com

6. www.cce.utexas.edu/~valvano/book.html

7. http://www2.unam.edu.ar/subprograma/metod_anex1.htm

Capitulo III

Programación e Integración en VHDL de los Módulos de un Microprocesador

3.1 GENERALIDADES.

Un microprocesador es un sistema digital complejo que realiza una gran variedad de operaciones lógicas, booleanas y aritméticas, además es capaz de generar direcciones para leer y escribir sobre un dispositivo de memoria o de algún otro tipo.

Básicamente un microprocesador se divide en tres secciones: Unidad Aritmético-Lógica (ALU), Unidad de Control y una cantidad de registros que varía dependiendo del microprocesador que se esta analizando, estos registros son necesarios para la operación del ALU y la Unidad de Control; algunos de estos registros son los acumuladores A y B, registro índice, contador de programa, registro de pila, etc.

El ALU como su nombre lo indica se encarga de realizar microoperaciones especificadas por la Unidad de Control con la información almacenada en los registros del procesador (acumulador A y B), además de almacenar el resultado en el registro destino. Estas microoperaciones pueden ser de suma, resta, incremento, decremento y lógicas como: AND, OR, INVERSOR, etc.

La Unidad de Control se encarga de generar las variables de control para los registros, memorias, ALU, etc. dentro de estas variables se encuentran. inicializar el microprocesador para que este ejecute las instrucciones, generar el pulso de lectura/escritura, además se encarga de atender las interrupciones e indica como y en que tiempo se debe realizar una microoperación, una transferencia de registros, etc.

Los registros A, B, pila, contador de programa, registro índice, registro de códigos de condición, etc. son auxiliares en la operación del ALU y de la Unidad de Control y es el contenido de estos registros los que determina el camino que seguirá el microprocesador para la ejecución de sus programas.

3.2 DISEÑO DEL MICROPROCESADOR UNAM-4.

En la figura 3.1 se muestra el diagrama a bloques del microprocesador que vamos a diseñar al cual llamaremos **UNAM-4**.

Dentro de sus principales características se encuentran: un bus de direcciones de 8 bits(A0-A7), un bus de datos de 4 bits (D0-D3), tres líneas de interrupción reset, IRQ1 e IRQ2, una línea de validación de direcciones (VMA), una línea para una señal de reloj, una línea de lectura/escritura y dos líneas para la alimentación.

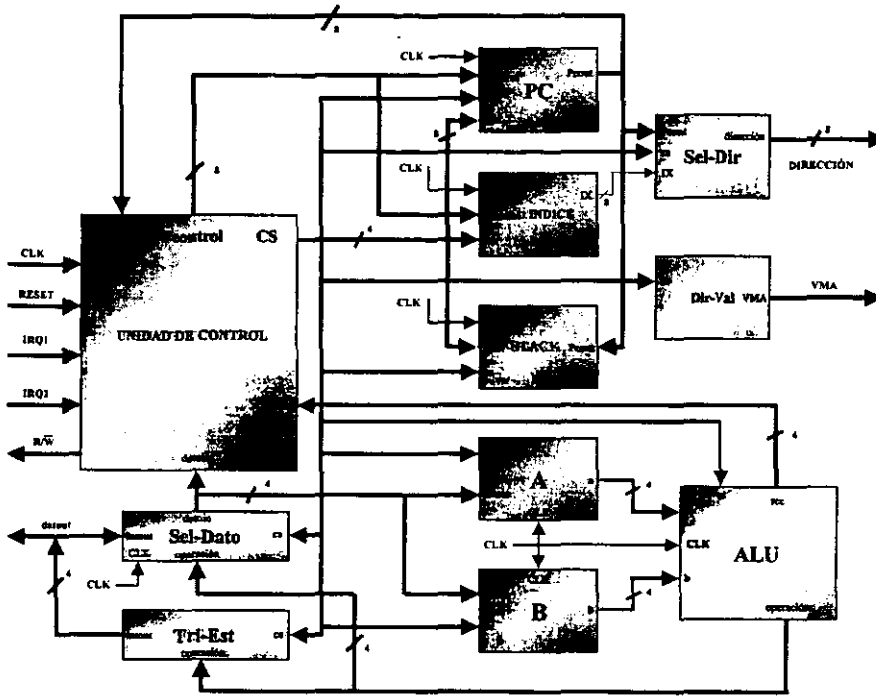


Figura 3.1 Diagrama a bloques del microprocesador UNAM-4

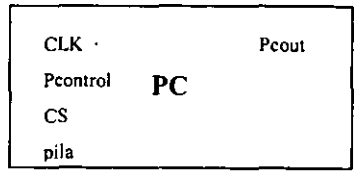
Como puede observarse en la figura 3.1 el microprocesador UNAM-4 se compone básicamente de 11 bloques cada uno con una función específica; en lo que resta del capítulo se analizará cada uno de estos bloques de manera detallada.

Este tipo de microprocesador es de clase Von Newman, es decir, que los datos e instrucciones se encuentran en una memoria externa; el microprocesador UNAM-4 por si solo no trabajaría ya que requiere de ciertos circuitos auxiliares para su operación, más adelante se muestra un diagrama esquemático que describe la forma en que estos circuitos auxiliares deben ser conectados.

3.2.1 CONTADOR DE PROGRAMA (PC).

El contador de programa es un registro de 8 bits que contiene la dirección del siguiente nibble (Conjunto de 4 bits) de la instrucción que será buscada en la memoria para su ejecución.

El hecho que el PC sea de 8 bits nos genera un total de 2^8 direcciones diferentes, es decir, 256 direcciones únicas; el microprocesador es el que debe habilitar cada una de estas direcciones. El bus de direcciones del microprocesador UNAM-4 es unidireccional y únicamente son salidas del microprocesador hacia los diferentes dispositivos. Cada vez que se ejecuta una instrucción el contador de programa se incrementa. Su diagrama es el siguiente :



Bloque del PC

El código en VHDL que describe la función del contador de programa es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity pc is port (
        clk: in std_logic;
```

```
    pcontrol: in std_logic_vector (7 downto 0);
          cs: in std_logic_vector(4 downto 0);
      reset : in std_logic;
        pila: in std_logic_vector (7 downto 0);
      pcout: inout std_logic_vector (7 downto 0));
end;
architecture arq_pc of pc is
begin

   process(clk, cs, reset)
   begin
     if reset = '1' then
      pc <= "00000000";
     else
if (clk'event and clk = '1') then
       case cs is
          when "11110" => pcout <= pcontrol;
          when "11111" => pcout <= pcout + 1;
          when "11101" => pcout <= pila;
          when others => null;
          end case;
       end if;
     end if;
   end process;
end;
```

Según la descripción anterior en VHDL el PC tiene tres variables que lo hacen trabajar el cs, clk y reset.

El reset sirve para inicializar al PC en la dirección "00000000", es en esta dirección donde el microprocesador ejecuta su primera instrucción.

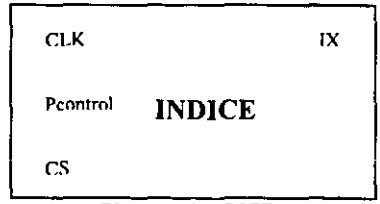
El cs le indica la función a realizar, cuando cs es igual a "11110" el pcout es igual al pcontrol, esta señal de pcontrol viene directamente de la Unidad de Control. Además puede ser igual a la dirección de inicio, a una nueva dirección cuando ha ocurrido una instrucción de salto anteriormente y finalmente a las direcciones de inicio de las interrupciones IRQ1 y 2.

Cuando el cs es igual a "11111" el valor de pcout se incrementa en una unidad, esto ocurre cuando se está ejecutando la instrucción presente en ese momento y requiere de un dato o bien de la siguiente instrucción. Cuando cs es igual a "11101" el pcout es igual al valor que esta almacenado en la pila, esto ocurre cuando el microprocesador atendió una petición de interrupción (IRQ1 y 2), y necesita regresar a la dirección donde originalmente estaba. Cualquier otra combinación en cs tiene un efecto nulo en el PC.

La señal de clk le indica en que momento debe ejecutar cada una de las acciones antes referidas. En todos los bloques del microprocesador la señal de clk tiene efecto cuando pasa de un bajo a un alto.

3.2.2 REGISTRO ÍNDICE (IX).

El registro de índice es un registro de 8 bits que se usa para almacenar una dirección, en ésta es posible escribir datos almacenados en el acumulador A. Este registro puede ser incrementado e inicializado a una dirección que el usuario especifique.



Bloque del INDICE

Su código en VHDL es el siguiente:

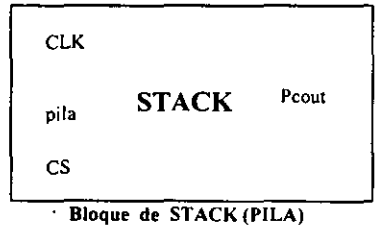
```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity indice is port (
          clk: in std_logic;
    pcontrol: in std_logic_vector (7 downto 0);
          cs: in std_logic_vector(4 downto 0);
       reset: in std_logic;
          ix: inout std_logic_vector (7 downto 0));
end;
architecture arq_indice of indice is
begin
   process(clk, cs, reset)
   begin
    if reset = '1' then
       ix <= "00000000";
   else
if (clk'event and clk = '1') then
          case cs is
          when "11010" => ix <= pcontrol;
          when "11011" => ix <= ix + 1;
          when others => null;
          end case;
      end if;
    end if;
   end process;
end;
```

El registro índice se ve afectado por las señales cs, clk y reset. La señal clk tiene el mismo efecto que en el PC.

La señal de reset cuando es igual a "1" coloca al ix con la dirección inicial "00000000". Cuando la señal de cs sea igual a "11010" el ix es igual a el pcontrol, el valor de pcontrol en este caso está determinado por el usuario dentro de su programa. Cuando cs es igual a "11011" el valor de ix se incrementa en una unidad, cualquier otra combinación de cs tiene un efecto nulo en el registro índice.

3.2.3 REGISTRO STACK (PILA).

La pila ó stack es un registro de 8 bits que se usa para almacenar la última dirección del PC cuando ocurre una solicitud de interrupción(IRQ1 y 2).



Bloque de STACK (PILA)

Su código en VHDL es el siguiente:

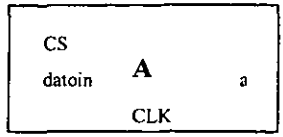
```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity stack is port (
        clk: in std_logic;
      pcout: in std_logic_vector (7 downto 0);
```

```
            pila: inout std_logic_vector (7 downto 0);
           reset: in std_logic;
              cs: in std_logic_vector (4 downto 0));
end;
architecture arq_stack of stack is
begin
   process (clk, cs, reset)
   variable q: std_logic_vector (7 downto 0);
   begin
     if reset = '1' then
       pila <="00000000";
    else
       if (clk'event and clk = '1') then
         case cs is
         when "11100" => pila <= pcout;
         when others => null;
         end case;
      end if;
    end if;
   end process;
end;
```

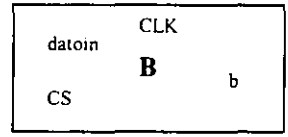
De acuerdo a la descripción VHDL del registro stack se ve afectado por las señales cs, clk y reset. La señal clk funciona como ya se había mencionado, la señal de reset cuando es igual a '1' inicializa a la pila con la dirección "00000000". Cuando el cs es igual a "11100" el valor de la pila es igual al pcout, esto ocurre cuando existe una interrupción, cualquier otro valor en cs tiene un efecto nulo sobre el registro de stack.

3.2.4 ACUMULADOR A Y B.

El microprocesador UNAM-4 contiene dos acumuladores (A y B) de 4 bits cada uno, que son utilizados como registros temporales los cuales contienen los operandos y el resultado de las operaciones efectuadas en la Unidad Aritmética-Lógica.



Bloque del Acumulador A



Bloque del Acumulador B

El código en VHDL del acumulador A es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity rega5 is port (
                cs:in std_logic_vector (4 downto 0);
               clk:in std_logic;
          datoin:in std_logic_vector (3 downto 0);
          reset: in std_logic;
              a:inout std_logic_vector (3 downto 0));
end;
architecture impedancia of rega5 is
begin
   process(cs, clk, reset)
   variable ares:std_logic_vector (3 downto 0);
     begin
      if reset = '1' then
          a <= "0000";
      else
```

```
        if (clk'event and clk = '1') then
          case cs is
            when "10001" =>
            a <= datoin;
            when "11100" =>
            ares := a;
            when "11101" =>
            a <= ares;
            when others =>
            null;
          end case;
        end if;
      end if;
    end process;
end;
```

El acumulador A se ve afectado por las señales clk, reset y cs. El contenido de A depende del usuario y del resultado de las operaciones efectuadas en la ALU. Cuando el reset tiene un valor de '1' el contenido del acumulador es "0000".
La variable interna ares juega un papel importante cuando existe una interrupción, ya que guarda el contenido del acumulador y posteriormente cuando el microprocesador regresa de dicha interrupción devuelve el valor original al acumulador con la finalidad de que la ejecución del programa principal no sufra alteraciones; en la descripción VHDL se observa que combinaciones de cs realizan esta función.

Además la combinación de cs igual a "10001" permite almacenar un dato en el acumulador para ser procesado.

El código en VHDL del acumulador B es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity regb5 is port (
                    cs:in std_logic_vector (4 downto 0);
            clk,reset:in std_logic;
              datoin:in std_logic_vector (3 downto 0);
                    b:inout std_logic_vector (3 downto 0));
end;
architecture impedancia of regb5 is
begin
     process(cs, clk, reset)
     variable bres:std_logic_vector (3 downto 0);
   begin
      if reset = '1' then
        b <= "0000";
       else
        if (clk'event and clk = '1') then
        case cs is
when "10010" =>
              b <= datoin;
              when "11100" =>
              bres := b;
              when "11101" =>
              b <= bres;
              when others =>
              null;
           end case;
```

```
        end if;
      end if;
          end process;
end;
```

El acumulador B realiza una función similar que el A. La única variación es el contenido que depende exclusivamente del usuario, para ello el cs debe ser igual a "10010. Al igual que en el acumulador A la variable bres tiene la misma función que ares.

3.2.5 REGISTRO SEL-DATO.

Este es un registro auxiliar que permite introducir datos al bus de datos interno del microprocesador, estos datos tienen como destino los acumuladores A o B y la Unidad de Control.

Bloque de Sel-Dato

El código en VHDL de este registro es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity sel_dato is port (
            clk:in std_logic;
        datout:in std_logic_vector (3 downto 0);
        datoin:inout std_logic_vector (3 downto 0);
```

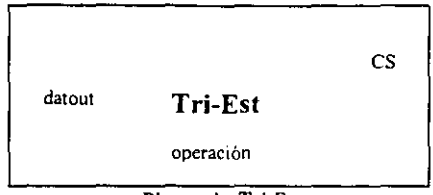
```
        operacion:in std_logic_vector (3 downto 0);
                cs:in std_logic_vector (4 downto 0));
end;
architecture arq_sel_dato of sel_dato is
begin
    process(cs,clk)
     begin
        if (clk'event and clk = '1') then
        case cs is
          when "10110" => datoin <= datout;
          when "10101" => datoin <= operacion;
          when others => null;
          end case;
        end if;
    end process;
end;
```

Como se observa en el código del Sel-Dato, el reset ya no tiene ningún efecto sobre éste y la señal clk tiene la misma función ya mencionada.

La señal de cs determina el tipo de dato a introducir, este dato puede ser externo ,de una memoria, y ocurre cuando cs es igual a "10110" o interno como resultado de una operación en la ALU y ocurre cuando cs es igual a "10101". En síntesis el registro Sel-Dato no es mas que un multiplexor de 2 entradas con una salida.

3.2.6 REGISTRO TRI-EST.

Este es un registro que permite sacar datos del microprocesador, que pueden ser el resultado de una operación en la ALU. Cuando no está en uso este registro sus salidas tienen un estado de alta impedancia.



Bloque de Tri-Est

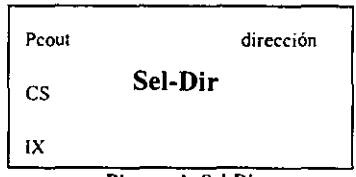
Su código en VHDL es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_esr is port (
                cs:in std_logic_vector (4 downto 0);
               clk: in std_logic;
      operacion:in std_logic_vector (3 downto 0);
          datout:inout std_logic_vector (3 downto 0));
end;
architecture arq_tri of tri_esr is
begin
   process(cs, clk)
   begin
      if (clk'event and clk = '1') then
      case cs is
        when "11000" => datout <= operacion;
        when others => datout <= "ZZZZ";
        end case;
     end if;
   end process;
end;
```

Como puede observarse tampoco es afectado por el reset y de acuerdo a su descripción se trata de un buffer unidireccional con estado de alta impedancia. En conjunto el registro Sel-Dato y el registro Tri-Est dan al bus de datos del microprocesador UNAM-4 la característica de ser bidireccional.

3.2.7 REGISTRO SEL-DIR.

El registro Sel-Dir permite que las direcciones generadas por el Contador de Programa o las del registro Índice salgan hacia el exterior del microprocesador, esto con la finalidad de leer o escribir un dato dependiendo del valor de la señal R/$\tilde{W}$.
La señal R/$\tilde{W}$ se genera a partir del momento en que el microprocesador necesita leer o escribir un dato, es decir, cuando va a leer un dato la señal R/$\tilde{W}$ se va a alto, esto le indica a los dispositivos que están conectados al microprocesador que se va a leer un dato de alguno de ellos, lo mismo ocurre cuando se va escribir un dato, solo que para este caso la señal R/$\tilde{W}$ se va a bajo; el dispositivo del cual se va a leer o escribir un dato es seleccionado por medio de la dirección generada en ese momento.



Bloque de Sel-Dir

El código que describe al registro Sel-Dir es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity sel_dir is port (
        pcout:in std_logic_vector (7 downto 0);
          clk:in std_logic;
```

```
            ix:in std_logic_vector (7 downto 0);
      direccion:out std_logic_vector (7 downto 0);
            cs:in std_logic_vector (4 downto 0));
end sel_dir;
architecture arq_sel_dir of sel_dir is
begin
    process(cs, clk)
        begin
        if (clk'event and clk = '1') then
        case cs is
        when "11000" => direccion <= ix;
        when others  => direccion <= pcout;
        end case;
        end if;
      end process;
end;
```

El registro Sel-Dir esta seleccionando la mayoria de las veces la dirección generada por el PC, debido a que el microprocesador en todo momento está leyendo datos o instrucciones y únicamente selecciona la dirección del registro índice cuando va a escribir un dato en la dirección señalada por el índice. Aunque en el código de VHDL de Sel-Dir no existe la señal de reset al momento de ser ésta igual a '1' el valor de dirección es igual a "00000000". Al igual que en el registro Sel-Dato, Sel-Dir no es más que un multiplexor de 2 entradas y una salida.

3.2.8 REGISTRO DIR-VAL (VMA).

La importancia de este registro reside en el sentido que su salida indica a los dispositivos periféricos que la dirección presente en el registro Sel-Dir es válida. En operación normal, esta señal debe ser utilizada para habilitar dispositivos periféricos como memorias, decodificadores o algún otro dispositivo de entrada/salida.

Esta señal no es de tres estados y cuando tiene un valor de '1' indica que la dirección es válida en caso contrario es una dirección no válida y esto es porque durante algunas operaciones internas del microprocesador, éste coloca una dirección en el bus que no esté usando en ese momento.

Bloque de Dir-Val

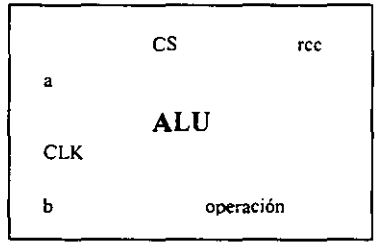
Su código en VHDL es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity dir_val is port (
        vma:out std_logic;
          cs:in std_logic_vector (4 downto 0));
end;
architecture arq_dir_val of dir_val is
begin
    process(cs)
        begin
        vma <= (not (cs(3)) and cs(2) and cs(1) and not (cs(0))) or (cs(3) and not (cs(2))
and not (cs(1)) and not (cs(0)));
    end process;
end;
```

Como puede observarse el registro Dir-Val no es más que un circuito combinatorio con una salida única (VMA) y su valor es igual a '1' cuando se efectúa una operación de lectura o escritura. El valor de VMA depende en todo momento de cs.

3.2.9 UNIDAD DE ARITMÉTICA-LÓGICA (ALU).

La Unidad de Aritmética-Lógica es una función multioperación digital de lógica combinacional. Esta puede realizar un conjunto de operaciones aritméticas básicas y otro de operaciones lógicas .



Bloque de Unidad Aritmética Lógica (ALU)

El código en VHDL de la ALU es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity alu4 is port (
        clk:in std_logic;
        a,b:in std_logic_vector (3 downto 0);
            cs:in std_logic_vector (4 downto 0);
    operacion:inout std_logic_vector (3 downto 0);
            rc:inout std_logic_vector (3 downto 0));
end alu4;
architecture sumar of alu4 is
signal c1:std_logic_vector (1 downto 0);
begin
    process(clk)
    variable Cout: std_logic;
```

```
   begin
   if (clk'event and clk = '1') then
--Selecciona una operación.
        case cs is
           when "00001" =>
           operacion <= (a + b);
           c1(0)<=(a(1) and b(1)) or ((a(0) and b(0)) and (a(1) xor b(1)));
           c1(1)<=(a(2) and b(2)) or (c1(0) and (a(2) xor b(2)));
           Cout:=(a(3) and b(3)) or (c1(1) and (a(3) xor b(3)));

           when "00010" =>
           operacion <= (a - b);
           if (a >= b) then
           Cout := '1';
           C1(0)<= '1';
           else
           Cout := '0';
           c1(0)<= '0';
           end if;

           when "00011" =>
           operacion <= (a and b);
           Cout := '0';
           c1(0)<= '0';

           when "00100" =>
           operacion <= (a or b);
           Cout := '0';
           c1(0)<= '0';
```

```
            when "00101" =>
            operacion <= (not a);
            Cout := '0';
            c1(0)<= '0';
            when "00110" =>
            operacion <= (a xor b);
            Cout := '0';
            c1(0)<= '0';

            when "00111" =>
            operacion <= (a and "1111");
            Cout := Cout;
            c1(0) <= c1(0);

            when others =>
            null;
        end case;
        end if;
            rc(3)<= Cout xor c1(1); -- Sobreflujo
            rc(2)<= not(operacion(3) or operacion(2) or operacion(1) or operacion(0));
            -- Cero
            rc(1)<= operacion(3); --Signo
            rc(0)<= Cout; --Acarreo
    end process;
end;
```

La ALU del microprocesador UNAM-4 se compone de 2 entradas de 4 bits cada una (contenido de los acumuladores A y B) y una salida de 4 bits (operación) con acarreo, puede realizar 7 operaciones, 2 aritméticas: suma y resta, y 5 lógicas: AND, OR, NOT, OR EXCLUSIVA y enmascarar (dejar pasar únicamente los bits que el usuario necesite) el contenido del acumulador A.

El diseño de esta ALU lleva consigo la generación del registro de códigos de condición (rcc), este registro es de 4 bits e indica como es el resultado de una operación en la Unidad de Aritmético-Lógica; estos 4 bits indican si el resultado es:

- ✓ Negativo(N)
- ✓ Igual a cero(Z)
- ✓ Si tiene sobreflujo(V)
- ✓ Si hay acarreo(C).

Acarreo (C).- Este bit (bit 0) del rcc se coloca a '1' si después de la ejecución de ciertas instrucciones, hay un acarreo del bit más significativo de la operación que se está ejecutando de otra manera se coloca a '0'.

Sobreflujo (V).- Este bit (bit 3) del rcc se coloca a '1' cuando un sobreflujo en complemento a 2 resulta de una operación aritmética y se coloca a '0' si el sobreflujo no ocurre en ese tiempo. Generalmente ocurre sobreflujo cuando la última operación resulta ser un número mayor que ± 7 de un registro de 4 bits.

Cero (Z).- Este bit (bit 2) del rcc se coloca a '1' , si el resultado de la operación lógica o aritmética es cero; de otra manera se coloca a '0'.

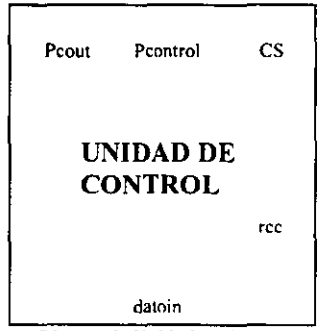
Negativo (N).- Este bit (bit 1) del rcc se coloca a '1', si el bit 4 del resultado de una operación lógica o aritmética es igual a '1' , de lo contrario se coloca '0'.

A pesar de ser pocas las operaciones que se pueden realizar con esta ALU, no hay que olvidar que al combinar estas operaciones es posible realizar decrementos, incrementos, corrimientos a la izquierda o derecha, corrimientos aritméticos, funciones NAND, NOR, etc.

En lo que se refiere a las señales que controlan el ALU, es el valor de cs el que determina que operación se va a llevar acabo en el ALU y la señal de clk en que tiempo se lleva a cabo.

3.2.10 UNIDAD DE CONTROL.

La Unidad de Control es la encargada de llevar la sincronía en cada una de las acciones realizadas por el microprocesador, determina en que tiempo, hacia donde o de donde viene los datos, se encarga de decodificar cada una de las instrucciones y que su ejecución se lleve a cabo, atiende cada una de las interrupciones y después de atenderlas se encarga de regresar al microprocesador a su operación normal.



Bloque de Unidad de Control

El código de la Unidad de Control es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity control is port(
        reset,clk: in std_logic;
              irq: in std_logic_vector (1 downto 0);
              rw: inout std_logic;
          datoin: in std_logic_vector (3 downto 0);
        pcontrol: inout std_logic_vector (7 downto 0);
               rc: in std_logic_vector (3 downto 0);
```

```
        pcout: in std_logic_vector (7 downto 0);
            cs: inout std_logic_vector (4 downto 0));
end control;
architecture arq_control of control is
type estados is (d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15, d16,
d17);
signal edo_presente, edo_futuro: estados;
signal f: std_logic_vector (7 downto 0);
begin
   proceso1: process (edo_presente, irq, reset)
        begin
          if reset = '1' then
            edo_futuro <= d0;
            pcontrol <= "11111111";
                rw <= '0';
                cs <= "11110";
            else
          case edo_presente is
              when d0 =>
              if irq = "10" then
                cs <= "11100";
                edo_futuro <= d1;
              elsif irq = "01" then
                cs <= "11100";
                edo_futuro <= d1;
              else
                edo_futuro <= d3;
                cs <= "11111";
              end if;
              when d1 =>
```

```
if irq = "10" then
  pcontrol <= "01000000";
  edo_futuro <= d1;
elsif irq = "01" then
  pcontrol <= "10000000";
  edo_futuro <= d1;
else
   cs <= "11110";
  edo_futuro <= d3;
end if;

when d2 =>
   cs <= cs;
  edo_futuro <= d5;

when d3 =>
      cs <= "10110";
      rw <= '1';
  edo_futuro <= d4;

when d4 =>
      rw <= '0';
if datoin = "0000" then
      cs <= "00001";
 edo_futuro <= d2;
elsif datoin = "0001" then
      cs <= "00010";
 edo_futuro <= d5;

elsif datoin = "0010" then
```

```
        cs <= "00011";
    edo_futuro <= d5;

elsif datoin = "0011" then
        cs <= "00100";
    edo_futuro <= d5;

elsif datoin = "0100" then
        cs <= "00101";
    edo_futuro <= d5;

elsif datoin = "0101" then
        cs <= "00110";
    edo_futuro <= d5;

elsif datoin = "0110" then
        cs <= "11111";
    edo_futuro <= d6;

elsif datoin = "0111" then
        cs <= "11111";
    edo_futuro <= d7;

elsif datoin = "1000" then
        cs <= "11111";
    edo_futuro <= d8;

elsif datoin = "1001" then
        cs <= "00111";
```

```
    edo_futuro <= d9;
elsif datoin = "1010" then
        cs <= "11011";
    edo_futuro <= d0;

elsif datoin = "1011" then
        cs <= "11111";
     if rc(0) = '1' then
    edo_futuro <= d10;
     else
    edo_futuro <= d0;
     end if;
elsif datoin = "1100" then

        cs <= "11111";
     if rc(2) = '1' then
    edo_futuro <= d10;
     else
    edo_futuro <= d0;
     end if;

elsif datoin = "1101" then
        cs <= "11111";
     if rc(1) = '1' then
    edo_futuro <= d10;
     else

    edo_futuro <= d0;
     end if;
elsif datoin = "1110" then
```

```
            cs <= "11111";
          if rc(3) = '1' then
        edo_futuro <= d10;

          else
        edo_futuro <= d0;
          end if;

      else
            cs <= "11101";
        edo_futuro <= d0;
      end if;


      when d5 =>
            cs <= "10101";
        edo_futuro <= d11;

      when d6 =>
            cs <= "10110";
            rw <= '1';
        edo_futuro <= d12;

      when d7 =>
            cs <= "10110";
            rw <= '1';
        edo_futuro <= d11;
```

```
when d8 =>
        cs <= "10110";

        rw <= '1';
  edo_futuro <= d15;


when d9 =>
        cs <= "11000";
  edo_futuro <= d0;

when d10 =>
        cs <= "10110";
        rw <= '1';
  edo_futuro <= d16;

when d11 =>
        cs <= "10001";
        rw <= '0';
  edo_futuro <= d0;

when d12 =>
        cs <= "11111";
        rw <= '0';
      pcontrol(0) <= datoin(0);
      pcontrol(1) <= datoin(1);
      pcontrol(2) <= datoin(2);
      pcontrol(3) <= datoin(3);
      pcontrol(4) <= pcontrol(4);
      pcontrol(5) <= pcontrol(5);
```

```
    pcontrol(6) <= pcontrol(6);
    pcontrol(7) <= pcontrol(7);

  edo_futuro <= d13;

when d13 =>
      cs <= "10110";
      rw <= '1';
  edo_futuro <= d14;

when d14 =>
      rw <= '0';
      cs <= "11010";
    pcontrol(0) <= pcontrol(0);
    pcontrol(1) <= pcontrol(1);
    pcontrol(2) <= pcontrol(2);

    pcontrol(3) <= pcontrol(3);
    pcontrol(4) <= datoin(0);
    pcontrol(5) <= datoin(1);
    pcontrol(6) <= datoin(2);
    pcontrol(7) <= datoin(3);
  edo_futuro <= d0;
when d15 =>
      cs <= "10010";
      rw <= '0';
  edo_futuro <= d0;
when d16 =>
      cs <= "10000";
     f(0)<= datoin(0);
```

```
                    f(1)<= datoin(1);
                    f(2)<= datoin(2);
                    f(3)<= '0';
                    f(4)<= '0';
                    f(5)<= '0';
                    f(6)<= '0';
                    f(7)<= '0';
                   if datoin(3) = '0' then
                  pcontrol <= pcout + f;
                  edo_futuro <= d17;
                  else
                  pcontrol <= pcout - f;
                  edo_futuro <= d17;
                  end if;

               when d17 =>
                     cs <= "11110";
                  edo_futuro <= d0;
            end case;
          end if;
end process proceso1;
proceso2: process(clk, reset)
begin
   if (clk'event and clk = '1') then
    if reset = '0' then
     edo_presente <= edo_futuro;
    else
    null;
   end if;
```

```
  end if;
  end process proceso2;
end;
```

De acuerdo al código en VHDL la Unidad de Control es una máquina de estados finita, en ella se esta utilizando un proceso combinacional para describir la función del próximo estado, las asignaciones de salida y un proceso secuencial para describir las asignaciones sobre el registro de estados en la transición activa de la señal de reloj.

La Unidad de Control se encarga de generar la señal cs para cada uno de los bloques, esta señal les indica la función a realizar y en que tiempo deben realizarla, de igual forma la Unidad de Control genera la señal de lectura/escritura, la señal de Pcontrol que va directamente al registro de PC encargado de generar las direcciones, la señal de clk, reset, IRQ1 y 2 y rcc son señales de entrada, la última resulta de las operaciones realizadas en la ALU indicándole a La Unidad de Control como es el resultado.

Al inicio del código en VHDL las interrupciones juegan un papel predominante porque le indican al microprocesador donde va a leer su primera instrucción.

El reset es la interrupción de mayor importancia, se encarga de inicializar al microprocesador empezando por la misma Unidad de Control, coloca al Pcontrol con valor de "11111111", y le dice al PC que se incremente. Después del reset pueden existir 2 interrupciones IRQ1 y 2. en caso de existir alguna de ellas el Pcontrol se modifica, pero antes el microprocesador guarda el resultado de los 2 acumuladores y de la última dirección(PC). Si IRQ1 existe el Pcontrol es igual a "10000000", en caso de existir IRQ2 el valor de Pcontrol es igual a "01000000", estas 2 direcciones indican donde se ejecutan las rutinas de interrupción, al terminar las rutinas los acumuladores vuelven a su valor original al igual que el valor del PC. En caso de no existir alguna interrupción el microprocesador genera la dirección "00000000" y la señal r/$\hat{w}$ se va a '1' con esto el microprocesador lee el contenido de esta dirección que es una instrucción e inicia la secuencia de ejecución.

En la figura 3.2 se puede observar un diagrama que indica el flujo de la operación que sigue el microprocesador al momento de empezar a trabajar y el camino que siguen las interrupciones.

La Unidad de Control se compone de 17 estados cada uno con una función específica, los estados '0' y '1' se encargan de las 3 interrupciones que existen. Se llega al estado '0' después de un reset o bien después de haberse ejecutado la última instrucción.

En éste se da la orden para incrementar el PC (cs ="11111"). Del estado '0' se va al estado '1', a menos que entre un reset, en éste la Unidad de Control verifica si ha entrado una interrupción, en caso positivo cambia el PC y pasa al estado '3', en caso contrario pasa de igual forma al estado '3'. Es en éste donde la Unidad de Control va a leer una instrucción para ello las señales r/ y VMA se van a '1', y la señal cs es igual a "10110".

Este valor de cs le indica al registro Sel-Dato que deje pasar al dato externo al bus de datos interno del microprocesador (datoin), para que sea analizado en la Unidad de Control.

De aquí se va directamente al estado '4' que es el más importante, porque cumple las funciones de decodificador de instrucciones y determina el camino que se debe seguir en la ejecución de las instrucciones.

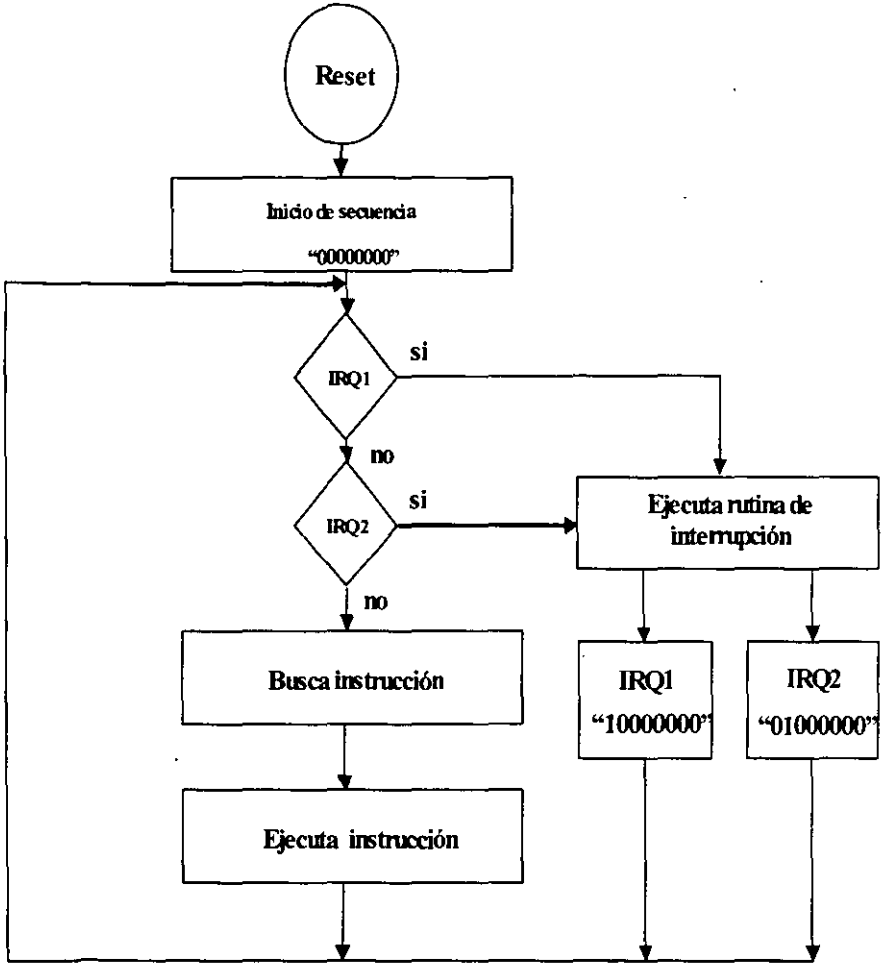


Figura 3.2 Diagrama de flujo de interrupciones.

En el estado '4' se tienen 16 combinaciones distintas de la señal datoin cada una de ellas representa una instrucción.

Las 16 instrucciones se dividen en las siguientes categorías:

Número de instrucciones	Tipo de Instrucciones
7	Aritmético-Lógicas.
4	Bifurcación.
4	Carga en acumulador, índice y dirección.
1	Retorno de interrupción.

Más adelante se da una lista detallada de las 16 instrucciones que puede ejecutar el microprocesador UNAM-4.

En el estado '4' únicamente las instrucciones aritmético-lógicas se ejecutan indicándole a la ALU el tipo de función a realizar. Para este tipo de instrucciones se pasa al estado '5' donde la Unidad de Control coloca el resultado de la operación en el bus de datos interno. Posteriormente se pasa al estado '11' donde el resultado se almacena en el acumulador A. Finalmente se pasa al estado '0' donde todo comienza de nuevo.

Para llegar al estado '6' es necesario que en el estado '4' el valor de datoin sea igual a "0110". Este dato corresponde a la instrucción de cargar una dirección en el registro índice. En el estado '6' la Unidad de Control da la orden para introducir un dato. De aquí se pasa al estado '12' donde el dato que se introdujo en el estado anterior corresponde a la parte baja de la dirección que se va a cargar al registro índice. En el estado '13' se da la orden de introducir un segundo dato y en el estado '14' este dato pasa a formar la parte alta de la dirección y desde este mismo estado se carga esta dirección ya formada en el registro índice. De aquí se pasa al estado '0' para ejecutar la siguiente instrucción.

Para llegar al estado '7' debe ocurrir que en el estado '4' el datoin sea igual a "0111" . este dato corresponde a la instrucción de cargar un dato en el acumulador A.

En el estado '7' se introduce el dato que se quiere cargar. Del estado '7' se pasa al estado '11' donde la unidad de control coloca el dato en el acumulador A. Finalmente se pasa al estado '0' para esperar la siguiente instrucción.

Al llegar al estado '8' debió haber ocurrido que en el estado '4' el valor de datoin fuera igual a "1000"; este valor corresponde a la instrucción de cargar un dato en el acumulador B. En el estado 8 se introduce el dato a cargar de aquí se pasa al estado 15 donde la Unidad de Control coloca dicho dato en el acumulador B y finalmente se pasa al estado '0'.

Cuando en el estado '4' el datoin es igual a "1001" significa que hay una instrucción de almacenar el dato del acumulador A en la dirección contenida en el registro índice, en este mismo estado la Unidad de control coloca el valor del acumulador A en el bus de datos interno. Posteriormente se pasa al estado '9' donde la dirección del índice se coloca en el registro Sel-Dir y el registro Tri-Est es habilitado para sacar el valor del acumulador fuera del microprocesador. De aquí se salta al estado '0'.

Las siguientes 4 instrucciones son de bifurcación y todas pasan por los mismos estados la única diferencia es la condición que tienen que verificar para hacer válida la condición de salto o no. Si en el estado '4' se dan alguno de los siguientes valores de datoin "1011", "1100", "1101" y "1110" son las instrucciones de salta si hay acarreo (C = 1), salta si es igual a cero (Z = 1), salta si es negativo (N = 1) y salta si hay sobreflujo (V = 0) respectivamente. En el mismo estado '4' se verifica si la condición de salto es verdadera, en caso de ser falsa se pasa al estado '0', pero en caso contrario se pasa al estado '10' donde se introduce el dato que determina el valor del salto; de aquí se pasa al estado '16', este salto puede ser hacia delante o hacia atrás, el valor máximo del salto es ± 7 direcciones a partir de la dirección que contiene el dato que determina la longitud del salto, el signo del salto se determina a partir del valor del tercer bit del dato si es igual a '0' el salto es positivo y si es igual a '1' el salto es negativo.

Posteriormente se pasa al estado '17' donde la nueva dirección es colocada en el PC. Finalmente se pasa al estado '0'.

Cuando en el estado '4' se tiene un valor de datoin igual a "1010", significa que hay una instrucción de incrementar el valor del registro índice, en el mismo estado '4' la Unidad de Control realiza esta operación. De aquí se regresa al estado '0'.

La última instrucción se utiliza cuando ha ocurrido previamente una interrupción (IRQ1 y IRQ2). Si en el estado '4' el valor de datoin es igual a "1111" significa que la rutina de interrupción ha terminado y que el microprocesador tiene que regresar a las condiciones previas a la interrupción para ello la Unidad de Control saca la dirección almacenada en la pila y la coloca en el PC y el valor original de los acumuladores es cargado nuevamente. Una vez realizado esto se salta al estado '0'.

3.3 DISEÑO JERÁRQUICO.

El diseño jerárquico es una herramienta de apoyo que permite la programación de extensos diseños mediante la integración de pequeños bloques los cuales fácilmente pueden ser detallados y simulados en forma individual.

Una vez ya estudiados cada uno de los bloques que conforman al microprocesador es necesario aclarar que de acuerdo a la programación en VHDL es necesario crear 2 programas, uno que define a cada uno de estos bloques como componentes y otro denominado de alto nivel que encadena cada uno de estos componentes y los hace trabajar en conjunto.

3.3.1 CREACIÓN DEL ARCHIVO DE COMPONENTES.

El archivo que define cada uno de estos bloques como componentes es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
package cmicro is
```

```
component alu4 port (
            clk:in std_logic;
            a,b:in std_logic_vector (3 downto 0);
             cs:in std_logic_vector (4 downto 0);
   operacion:inout std_logic_vector (3 downto 0);
             rc:inout std_logic_vector (3 downto 0));
end component;

component rega5 port (
            cs:in std_logic_vector (4 downto 0);
       reset:in std_logic;
         clk:in std_logic;
     datoin:in std_logic_vector (3 downto 0);
          a:inout std_logic_vector (3 downto 0));
end component;

component regb5 port (
       reset:in std_logic;
          cs:in std_logic_vector (4 downto 0);
         clk:in std_logic;
      datoin:in std_logic_vector (3 downto 0);
           b:inout std_logic_vector (3 downto 0));

end component;

component sel_dato port (
            clk:in std_logic;
       datout:in std_logic_vector (3 downto 0);
        datoin:inout std_logic_vector (3 downto 0);
```

```
    operacion:in std_logic_vector (3 downto 0);
            cs:in std_logic_vector (4 downto 0));
end component;

component tri_esr port (
            cs:in std_logic_vector (4 downto 0);
    operacion:in std_logic_vector (3 downto 0);
        datout:inout std_logic_vector (3 downto 0));
end component;

component pc port (
            clk: in std_logic;
          reset: in std_logic;
pcontrol, pila: in std_logic_vector (7 downto 0);
            cs: in std_logic_vector(4 downto 0);
         pcout: inout std_logic_vector (7 downto 0));
end component;

component stack port (
          clk: in std_logic;
       pcout: in std_logic_vector (7 downto 0);
        reset:in std_logic;
          pila: inout std_logic_vector (7 downto 0);
            cs: in std_logic_vector (4 downto 0));
end component;

component indice port (
            clk: in std_logic;
    pcontrol: in std_logic_vector (7 downto 0);
```

```
            reset:in std_logic;
               cs: in std_logic_vector(4 downto 0);
                ix: inout std_logic_vector (7 downto 0));
    end component;

    component sel_dir port (
           pcout:in std_logic_vector (7 downto 0);
              clk:in std_logic;

               ix:in std_logic_vector (7 downto 0);
       direccion:out std_logic_vector (7 downto 0);
               cs:in std_logic_vector (4 downto 0));
    end component;

    component dir_val port (
            vma:out std_logic;
             cs:in std_logic_vector (4 downto 0));
    end component;

    component control port (
       reset,clk: in std_logic;
              irq: in std_logic_vector (1 downto 0);
               rw: inout std_logic;
           datoin: in std_logic_vector (3 downto 0);

         pcontrol: inout std_logic_vector (7 downto 0);
               rc: in std_logic_vector (3 downto 0);
             pcout: in std_logic_vector (7 downto 0);
               cs: inout std_logic_vector (4 downto 0));
    end component;
end cmicro;
```

De acuerdo al código anterior cada componente es declarado de manera similar a su entidad de diseño y cada uno de estos componentes es almacenado en el paquete llamado c micro.

3.3.2 CREACIÓN DEL ARCHIVO DE ALTO NIVEL(TOP LEVEL).

Como se mencionó en el capítulo I en VHDL es posible diseñar en forma estructural, es decir, uniendo componente por componente. Esta metodología es la base del diseño jerárquico ya que cada bloque o componente del diseño es interconectado por medio de señales o buses internos, los cuales son declarados y asociados por medio de cláusulas propias del lenguaje.

La declaración de la entidad consiste en todas las terminales de entrada y salida del microprocesador, las cuales son nombradas de la misma forma en que se encuentran referenciadas en su módulo. Las señales encargadas de interconectar cada uno de los componentes se declara en la arquitectura, además se usara la librería de nombre diseno que contiene la función especifica de cada bloque.
El código en VHDL del archivo de alto nivel que permite trabajar en conjunto a todos los bloques del microprocesador UNAM-4 es el siguiente:

```
library ieee, diseno;
use ieee.std_logic_1164.all;
use work.std_arith.all;

use diseno.all;
entity anmicro is port (
        clk,reset:in std_logic;
        direccion:out std_logic_vector (7 downto 0);
```

```
            vma:out std_logic;
        datout:inout std_logic_vector (3 downto 0);
            rw:inout std_logic;
            irq:in std_logic_vector (1 downto 0));
end;
architecture arq_anmicro of anmicro is

signal a: std_logic_vector (3 downto 0);

signal b: std_logic_vector (3 downto 0);

signal datoin: std_logic_vector (3 downto 0);

signal operacion: std_logic_vector (3 downto 0);

signal rc: std_logic_vector (3 downto 0);

signal cs: std_logic_vector (4 downto 0);

signal pcontrol: std_logic_vector (7 downto 0);

signal pila: std_logic_vector (7 downto 0);

signal pcout: std_logic_vector (7 downto 0);
signal ix: std_logic_vector (7 downto 0);
begin

u1: alu4 port map (clk=>clk, cs=>cs, operacion=>operacion, rc=>rc, a=>a, b=>b);

u2: rega5 port map (cs=>cs, datoin=>datoin, a=>a, clk=>clk);

u3: regb5 port map (cs=>cs, datoin=>datoin, b=>b, clk=>clk);

u4: sel_dato port map (cs=>cs, datoin=>datoin, datout=>datout, operacion=>operacion,
clk=>clk);
```

u5: tri_esr port map (cs=>cs, operacion=>operacion, datout=>datout);

u6: pc port map (clk=>clk, pcontrol=>pcontrol, cs=>cs, pcout=>pcout, pila=>pila);

u7: indice port map (cs=>cs, clk=>clk, pcontrol=>pcontrol, ix=>ix);

u8: stack port map (cs=>cs, clk=>clk, pcout=>pcout, pila=>pila);

u9: sel_dir port map (clk=>clk, pcout=>pcout, ix=>ix, direccion=>direccion, cs=>cs);
u10: dir_val port map (vma=>vma, cs=>cs);

u11: control port map (clk=>clk, reset=>reset, irq=>irq, rw=>rw, datoin=>datoin, pcontrol=>pcontrol, rc=>rc, pcout=>pcout, cs=>cs);

end;

3.4 LISTA DE INSTRUCCIONES.

El microprocesador UNAM-4 contiene un conjunto de 16 instrucciones, dichas operaciones pueden ocupar 1,2 ó 3 nibbles según el tipo de operación a ejecutarse. En la tabla 3.2 se puede observar el conjunto de instrucciones. Al final de dicha tabla se describen cada uno de los símbolos utilizados en ésta.

Instrucción	**Mnemónico**	**~**	**#**	**RCC NZVC**	**Código De Operación**	**Comentario**
A + B A	ABA	6	1		0000(0)	Suma Los acumuladores y guarda el resultado en A.
A – B A	SBA	5	1		0001(1)	Resta los acumuladores y guarda el resultado en A
A and B A	ANAB	5	1		0010(2)	AND lógica entre acumuladores y guarda el resultado en A.

A or B A	ORAB	5	1		0011(3)	OR lógica entre acumuladores y guarda el resultado en A.
not A A	NEGA	5	1		0100(4)	Invertir A y guarda el resultado en A.
A xor B A	EOAB	5	1		0101(5)	XOR lógica entre acumuladores y guarda el resultado en A.
M X	LDX M	7	3		0110(6)	Carga el índice con la dirección de memoria inmediata.
M A	LDDA	5	2		0111(7)	Carga el acumulador A con el valor inmediato.
M B	LDDB	5	2		1000(8)	Carga el acumulador B con el valor inmediato.
A M	STTA	4	1		1001(9)	Almacena el contenido de A en la dirección del índice.
X + 1 X	INX	3	1		1010(A)	Incrementa en una unidad el valor del registro índice.
C = '1'	BCS	3, 6	2		1011(B)	Salta si C = '1'.
Z = '1'	BEQ	3, 6	2		1100(C)	Salta si Z = '1'.
N = '1'	BMI	3, 6	2		1101(D)	Salta si N = '1'.
V = '1'	BVS	3, 6	2		1110(E)	Salta si V = '1'.
PILA Pcout	RTI	3	1		1111(F)	Regresa de una interrupción.

Tabla 3.2 Lista de instrucciones del microprocesador UNAM-4

Descripción de los símbolos utilizados en la tabla 3.2:

- Transferencia a.
- Se pone a '1' si es cierto y en caso contrario a '0'
- A Acumulador A.
- B Acumulador B.
- X Registro de Índice.
- C Indica si hay acarreo.
- Z Indica si el dato es igual a cero.
- V indica si hay sobreflujo.
- N Indica si el dato es negativo.
- RCC Registro de códigos de condición.
- # Número de nibbles requeridos por la instrucción.
- ~ Número de ciclos de reloj empleados por la instrucción.
- M Dato de una dirección de memoria o una dirección de memoria.
- No afectado.

De acuerdo a la tabla 3.2 el usuario sólo puede manipular los acumuladores, el registro de índice y utilizar el registro de códigos de condición para saber como es el resultado. Todos los demás registros del microprocesador no pueden ser modificados por el usuario.

3.5 SISTEMA MINIMO.

Como ya se había mencionado en este capítulo el microprocesador UNAM-4 a pesar de su complejidad no puede trabajar solo, ya que requiere de ciertos circuitos auxiliares con los cuales forme un sistema que sea capaz de proporcionar las instrucciones y datos necesarios para que el microprocesador los procese. Así mismo el sistema debe estar diseñado de tal manera que permita una posible expansión del mismo haciendolo cada vez más poderoso, dando como resultado un sistema capaz de ejecutar diversas aplicaciones.

Finalmente el sistema debe ser una herramienta en la enseñanza de técnicas de programación del microprocesador UNAM-4.

En la figura 3.3 se muestra el diagrama esquemático de un sistema mínimo para el microprocesador UNAM-4; cabe aclarar que su funcionalidad no se ha comprobado de manera física, pero el diagrama servirá como una herramienta de apoyo para desarrollar el sistema en su totalidad.
En el siguiente párrafo se describe cada uno de los componentes que forman el sistema y la función que cumplen dentro de este.

3.5.1 MEMORIA EEPROM 2816.

La memoria 2816 se encuentra disponible en una amplia gama de fabricantes, contiene un bus de direcciones de 11 líneas (A0..A11), un bus de datos de 8 bits (D0..D7), tiene tres terminales de control CE que se utiliza para habilitar a la memoria, si esta línea esta en alto las líneas del bus de datos estarán en estado de alta impedancia. OE cumple las funciones de línea de lectura, cuando se encuentra en bajo coloca el contenido de una localidad de memoria específica en el bus de datos del sistema. Finalmente se tiene la línea WE cuya función es colocar un dato en una localidad de la memoria. La capacidad de almacenamiento de la memoria es de 2kbytes. Debido al tamaño de la memoria no es utilizada en toda su capacidad, de hecho solo se utilizan las primeras 160 localidades de memoria y los primeros 4 bits del bus de datos. Las líneas de dirección no utilizadas se colocan a tierra y las líneas de datos se colocan a Vcc por medio de una resistencia. La finalidad de esta memoria es la de proporcionar al microprocesador las instrucciones y datos que va a procesar, esta memoria puede ser grabada utilizando cualquier grabador comercial y tiene la ventaja de ser borrada sin necesidad de luz ultravioleta.

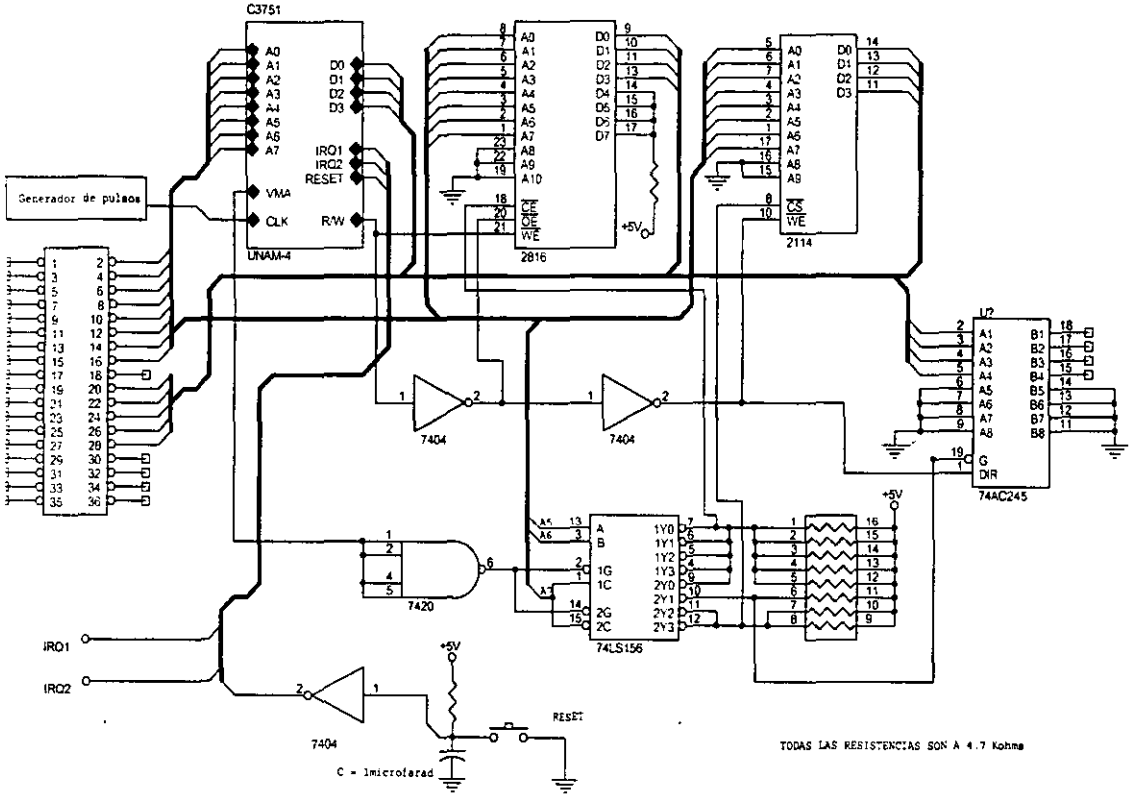


Figura 3.3 Sistema mínimo del microprocesador UNAM-4.

3.5.2 MEMORIA RAM 2114.

Es una memoria con una capacidad de almacenamiento de 1000 direcciones de 4 bits; tiene 10 líneas de dirección (A0..A9), 4 líneas de datos (D0..D3), una línea de selección CS, que habilita la memoria con un nivel bajo y una línea de lectura/escritura WE, cuando esta un nivel alto en esta línea permite la lectura de una localidad de la memoria y cuando el nivel es bajo escribe un dato en una localidad de memoria. Como se ve en el diagrama no todas las líneas del bus de direcciones se utilizan, sólo las primeras 7 y en cambio el bus de datos si se utiliza en su totalidad. La capacidad de la memoria utilizada es de sólo las primeras 64 localidades. Esta memoria es utilizada posteriormente para almacenar datos que son el resultado de un procesamiento interno de este y que se pueden volver a utilizar.

3.5.3 HEADER DE 18 PARES.

El header cumple sólo con la función de permitir monitorear por medio de un osciloscopio las líneas del bus de direcciones y las del bus de datos. Básicamente es para evitar algún posible daño a los pines de los circuitos.

3.5.4 BUFFER BIDIRECCIONAL 74LS245.

El circuito TTL 74LS245 es un buffer bidireccional de 16 líneas de entrada/salida (A0..A7 y B0..B7) donde la línea DIR indica el sentido de dirección habiendo 2 casos. si DIR es igual a '1' entonces Ax = Bx, es decir los valores de A son igual a los de B. Cuando Dir es igual a '0' entonces Bx = Ax. Existe además la línea G si su estado es igual a '1' entonces las líneas A y B serán de alta impedancia. En caso contrario la funcionalidad del bus estará determinado por la línea DIR. La función del buffer es la de

simular un puerto de entrada/salida de 4 bits, es decir, que me permita introducir y sacar datos del sistema, la función de entrada/salida está determinada por la línea del microprocesador R/□ y así mismo la línea G cumple la misma función que la línea CS o CE en las memorias del sistema.

La importancia del buffer es la de poder extender el sistema más allá del presentado en la fig. 3.3 como por ejemplo puede ser un puerto de comunicación con algún otro sistema o puede recibir información de un convertidor analógico/digital conectado este a un transductor que me permita saber el estado de un proceso, etcétera. Las 2 líneas de interrupción IRQ1 e IRQ2 pueden trabajar en conjunto con el buffer para hacer más eficiente la entrada y salida de datos.

3.5.5 DECODIFICADOR DE DIRECCIONES 74LS156.

El circuito integrado 74LS156 se encuentra configurado de tal manera que se convierte en un decodificador de 3 entradas (A, B y C) y 8 salidas (1Y1..1Y3 y 2Y1..2y3), las 8 salidas del decodificador dividen en 8 conjuntos las direcciones del sistema, cada conjunto es de 32 direcciones diferentes. La línea VMA del microprocesador está conectado a las entradas de una compuerta NAND que cumple las funciones de un inversor, la salida de esta compuerta va hacia las líneas 1G y 2G del decodificador; cuando la línea VMA se encuentra en estado alto indica que la dirección que se encuentra en el sistema es válida, al ser VMA igual a '1' la salida de la NAND es '0', con ello el decodificador coloca un estado bajo en alguna de sus líneas de salida, dejando las otras en estado alto. Al tener una salida en bajo permite seleccionar alguno de los dispositivos conectados al decodificador, en este caso pueden ser las memorias 2816, 2114 o el buffer 72LS245.

Al describir cada uno de estos dispositivos anteriormente se mencionó que son habilitados con un estado bajo (CS, CE y G). Como se observa las primeras 5 líneas de

salida del decodificador se encuentran unidas en un nodo común que conecta directamente la línea CE de la memoria 2816, esta configuración permite seleccionar 160 direcciones diferentes de la memoria. La 6 línea de salida del decodificador se conecta al buffer 74LS245, es por medio de esta que el buffer puede ser activado. Las 2 últimas líneas de salida del decodificador se unen formando una salida común hacia la línea CS de la memoria 2114, lo que permite seleccionar 64 direcciones diferentes.

Cuando VMA es igual a '0' la salida de la NAND es igual a '1', esta condición obliga al decodificador a enviar a todas sus salidas a un estado de alta impedancia, es decir el decodificador no selecciona ningún dispositivo. La tabla 3.3 describe el funcionamiento del decodificador 74LS156.

Entradas				**Salidas**								**Dispositivo**
Selección			**Ena**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**seleccionado**
C^{12}	B	A	G^{12}	1Y0	1Y1	1Y2	1Y3	2Y0	2Y1	2Y2	2Y3	
X	X	X	H	Z	Z	Z	Z	Z	Z	Z	Z	Ninguno
0	0	0	L	L	H	H	H	H	H	H	H	Memoria 2816
0	0	1	L	H	L	H	H	H	H	H	H	Memoria 2816
0	1	0	L	H	H	L	H	H	H	H	H	Memoria 2816
0	1	1	L	H	H	H	L	H	H	H	H	Memoria 2816
1	0	0	L	H	H	H	H	L	H	H	H	Memoria 2816
1	0	1	L	H	H	H	H	H	L	H	H	Buffer 74LS245
1	1	0	L	H	H	H	H	H	H	L	H	Memoria 2114
1	1	1	L	H	H	H	H	H	H	H	L	Memoria 2114

Tabla 3.3 Función del decodificador 74LS245.

C^{12} Conecta las líneas C1 y C2.

G^{12} Conecta las líneas G1 y G2.

Z = alta impedancia.

3.5.6 CIRCUITO DE RESET

El sistema requiere de un reset para inicializar al microprocesador a una estado que le permita leer y ejecutar su primera instrucción. Para ello el sistema cuenta con un arreglo RC en serie, en el nodo que une al resistor y al capacitor se conecta un interruptor conectado en el otro extremo a tierra y desde este mismo punto hacia un inversor y la salida del inversor hacia el reset del microprocesador. Al momento de encender el sistema el capacitor se empieza a cargar al valor de Vcc a través de la resistencia, es decir tendremos un estado de alto en la entrada del inversor y por supuesto un bajo a su salida, el microprocesador en este estado con reset bajo debe teóricamente de operar desde su condición de inicio, en muchas ocasiones no ocurre así, por lo que para inicializar al microprocesador es necesario presionar al interruptor de reset, esto ocasiona que el capacitor se descargue y se tenga un estado bajo a la entrada del inversor y un alto a su salida, este estado alto en el reset provoca que el microprocesador borre el contenido de todos sus registros internos y genere las señales de control necesarias para leer su primer instrucción. Al dejar de presionar el interruptor de reset el capacitor se vuelve a cargar a Vcc, teniendo un estado bajo en el reset que le permite al microprocesador empezar a trabajar.

3.5.7 MICROPROCESADOR UNAM-4.

El microprocesador UNAM-4 juega el papel central del sistema, se encarga de indicarle a cada uno de sus circuitos auxiliares el momento en que tienen que entrar en acción, además de estos depende la óptima operación del microprocesador, de ahí el nombre de sistema porque cada una de sus partes cumple una función específica para lograr un objetivo común. El microprocesador UNAM-4 tiene 8 líneas de dirección (A0..A7), este dato nos indica que el microprocesador puede generar 2^8 direcciones distintas, es decir, $2^8 = 256$ direcciones, éstas nos darán pie para formar el mapa de memoria del sistema.

Tiene un bus de datos bidireccional de 4 bits (D0..D3), además de 3 interrupciones que son el reset, IRQ1 e IRQ2; después de un reset la primera dirección es "00000000", después de una IRQ1 la dirección es "10000000" y finalmente después de una interrupción IRQ2 la dirección es "01000000". Tiene una línea de lectura/escritura (R/□) para leer y escribir datos de las memorias o de algún dispositivo externo.

Además de una línea de CLK donde se debe conectar el reloj del microprocesador. Finalmente la línea de VMA que le sirve al microprocesador para indicarle a los demás dispositivos conectados a él que la dirección es valida o no.

Internamente el microprocesador contiene 2 acumuladores de 4 bits cada uno, una ALU que puede realizar 7 operaciones distintas, un registro de códigos de condición que le señalan al usuario como es el resultado de una operación en la ALU, un registro de stack o pila que es utilizado por el microprocesador para almacenar el contenido de los acumuladores y la última dirección antes de que ocurra una interrupción IRQ 1 ó 2.

Un registro de índice que almacena una dirección hacia la cual se va enviar un dato, un registro de PC (contador de programa) que lleva el control sobre la siguiente dirección donde el microprocesador debe leer la siguiente instrucción o dato, una unidad de control que se encarga de manipular cada uno de estos registros, indicándoles en que tiempo entran en la ejecución de las instrucciones e indica en todo momento hacia donde y de donde viene la información que entra al microprocesador, ahí mismo en la unidad de control toda esta información es decodificada para su inmediato procesamiento.

Además de estos registros el microprocesador UNAM-4 tiene otros registros auxiliares para su buen funcionamiento.

3.6 MAPA DE MEMORIA.

El sistema mínimo del microprocesador UNAM-4 requiere de un mapa de memoria que es una herramienta de gran utilidad para todo aquel que desee manipular al sistema, este mapa cumple la función de indicarle al usuario las direcciones del sistema donde están localizados cada uno de los dispositivos interconectados al microprocesador, así por ejemplo si el usuario quiere guardar información en memoria RAM del sistema, sabrá que direcciones debe generar para elegir a la memoria que el quiere, de igual manera si lo que quiere es sacar o meter información del sistema, el mapa le indicará en que direcciones se encuentra ubicado el dispositivo que lo auxiliará en su propósito, en nuestro caso podría tratarse del buffer 74LS245.

El mapa del memoria de nuestro sistema es de 256 direcciones por lo que ya se había explicado anteriormente, si recordamos la memoria 2816 ocupa 160 direcciones, la memoria 2114 ocupa 64 direcciones y el buffer 74LS245 le corresponderían 32 direcciones, aunque en este caso sólo use una de las 32 direcciones asignadas, pero haciendo un pequeño arreglo al sistema puede utilizar las 32 direcciones en su totalidad. Como se vio en el decodificador 74LS156 este divide a todas la direcciones en 8 conjuntos de 32 direcciones distintas dándonos un total de 8 x 32 = 256 direcciones. Esto nos lleva al mapa de memoria de la tabla 3.4.

Dirección	Dispositivo seleccionado.
0000000 - 00011111	Memoria 2816
0010000 - 00111111	Memoria 2826
0100000 - 01011111	Memoria 2826
0110000 - 01111111	Memoria 2826
1000000 - 10011111	Memoria 2826
1010000 - 10111111	Buffer 74LS245
1100000 - 11011111	Memoria 2114
1110000 - 11111111	Memoria 2114

Tabla 3.4 Mapa de memoria del sistema mínimo UNAM-4.

De esta forma nos damos cuenta de donde se ubica cada uno de los dispositivos conectados al microprocesador, así sabremos donde guardar nuestros datos o de donde podemos leerlos o si los queremos sacar o meter al sistema.

Si queremos guardar o leer datos en memoria RAM debemos seleccionar las direcciones que van de la 11000000 a la 11111111 (C0 – FF). Si lo que deseamos es sacar o meter datos al sistema seleccionamos las que van de la 10100000 a la 10111111 (A0 – BF) y si lo que deseamos es ejecutar instrucciones debemos seleccionar las direcciones 00000000 a la 10011111 (00 – 9F), previamente se debieron haber grabado las instrucciones y datos requeridos en la memoria 2816.

3.7 PROGRAMACIÓN DEL MICROPROCESADOR UNAM-4.

Como vimos anteriormente el microprocesador UNAM-4 esta integrado por un set de 16 instrucciones, esto es producto del bus de datos de 4 bits el cual nos permite realizar 16 combinaciones diferentes de datos en binario; a esto se debe el reducido número de instrucciones. Al principio del capítulo se realizo una descripción de cómo funciona cada uno de los bloques que integran el microprocesador, el usuario final tendrá muy pocas posibilidades de poder modificar el funcionamiento de cada uno de estos bloques,

ya que el comportamiento de estos esta regido por las instrucciones e interrupciones del microprocesador a las cuales el usuario si tiene acceso; a continuación se describe cada una de las instrucciones del microprocesador UNAM-4.

En la tabla 3.2 se describen cada una de las instrucciones, la forma en que afectan al rcc, el número de ciclos de reloj que se requieren para cada instrucción, el numero de nibbles necesarios para que la instrucción se ejecute adecuadamente y el mnemónico que las identifica. Donde:

- La Unidad de control es la que determina el número de ciclos de reloj que se requieren para todas las instrucciones.
- El nibble es el que le indica al microprocesador la acción a ejecutar o en su defecto el dato que requiere la instrucción.
- La primera instrucción es sumar el contenido de los acumuladores A y B y guardar el resultado en el acumulador A, requiere de 6 ciclos de reloj y 1 nibble para ejecutarse;

El mnemónico que la identifica es ABA y su código de operación es 0000.

Acc A	Acc B	rcc	Dirección	Dato
0000	0001	0100	00000000	0000
0001	0001	0000	00000001	0000
0010	0001	0000	00000010	0000

Tabla 3.5 Ejemplo de programación con la instrucción ABA.

Tomando en cuenta la tabla 3.5 imaginemos que los datos se encuentran grabados en memoria (2816) y que en el sistema mínimo de la figura 3.3 el microprocesador genera la dirección "00000000" y el pulso r/w es igual a '1' con lo que el micro lee el dato almacenado en esa dirección y que es "0000", al momento de entrar esta información al micro la Unidad de Control la decodifica y determina que la operación a realizar es sumar los acumuladores y guardar el resultado en A.

Como se ve en la tabla en los acumuladores al principio tenemos en A "0000" y en B "0001" si sumamos estos números el resultado es "0001" y como indica la instrucción el resultado se debe guardar en A. El rcc indica como es el resultado, al principio tenemos "0100" (NZVC). Esta instrucción únicamente requiere de 1 nibble porque sólo le indica al micro que realice una operación de suma.

Posteriormente el PC se incrementa en '1' y la nueva dirección es "00000001" así mismo el valor de los acumuladores es A "0001" y b"0001" y el de rcc es "0000". Ahora bien A obtiene su actual valor producto de la instrucción anterior y rrc cambia porque el resultado de la operación de suma ejecutada en la ALU cambió, esto es el resultado de ejecutar la primera instrucción fue que no es negativo, no es cero, no hay sobreflujo y no hay acarreo.

Con la nueva dirección entra un nuevo dato que es igual a "0000" que la Unidad de Control decodifica como sumar nuevamente los acumuladores y guardar el resultado en A. Después de ejecutada la instrucción se incrementa el PC y obviamente la dirección. Como se ve en la tabla 3.5 el valor de A es "0010". El valor de A resulta de sumar el valor anterior de A "0001" y de B "0001" en general estamos realizando el incremento en una unidad de A y así seguiremos hasta que no aparezca una instrucción diferente.

Las siguientes 2 instrucciones son importantes porque nos permiten introducir datos de la memoria a los acumuladores y posteriormente procesar estos datos con operaciones aritméticas y booleanas como la instrucción ABA.

Mnemónico	Código	Nibble	rcc	Comentario
LDDA	0111	2	No afectado	Carga en A el valor inmediato.
LDDB	1000	2	No afectado	Carga en B el valor inmediato.

Tabla 3.6 instrucciones de carga en acumuladores A y B.

En el siguiente ejemplo se observa con más claridad el funcionamiento de estas 2 instrucciones.

Acc A	Acc B	rcc	Dirección	Dato
0000	0000	0000	00000000	0111
0000	0000	0000	00000001	0011
0011	0000	0000	00000010	1000
0011	0001	0000	00000011	0001
0011	0001	0000	00000100	xxxx

Tabla 3.7 Ejemplo de carga de acumuladores en el microprocesador UNAM-4.

Tomando como base la tabla 3.7 al momento de inicializar al sistema mínimo el microprocesador coloca en cero todos sus archivos sin excepción, generando su primera dirección donde busca su primera instrucción, en este caso la dirección "00000000" tiene el dato "0111" que es decodificado por la Unidad de Control como la instrucción de cargar el dato siguiente en el acumulador A, como siguiente paso se incrementa el PC para leer el contenido de la siguiente dirección que es el dato que se va a cargar en el acumulador A, ("00000001" →"0011") de ahí que esta instrucción requiera de 2 nibbles.

Después de ejecutar esta instrucción el resultado de los acumuladores es A→"0011"; B→"0000", el rcc no se afecta porque no se realiza ninguna operación en la ALU, (hay que notar que el dato de la dirección "00000001" es eso un dato y no una instrucción y cuya función es la de complementar a la instrucción que lo requiere). Siguiendo con el programa el PC se incrementa nuevamente apuntando la dirección "00000010" el dato "1000" que la Unidad de Control identifica como cargar en el acumulador B con el dato siguiente, el PC se incrementa para leer el dato siguiente que es "00000011"→"0001", este dato se carga en el acumulador B, finalmente se incrementa el PC teniendo como resultado en los acumuladores A→"0011", B→"0001", el rcc no se afecta nuevamente y el PC apunta la dirección "00000100" que tiene el contenido XXXX que significa que puede ser cualquier instrucción.

Las siguientes 5 instrucciones son similares a la primera instrucción de sumar los acumuladores y guardar el resultado en el acumulador A, la diferencia reside en la operación que se realiza con los acumuladores, pero el resultado de la operación se sigue guardando en A, finalmente el rcc se modifica de acuerdo al resultado

En la tabla 3.8 se muestran dichas instrucciones y en la tabla 3.9 se da un ejemplo con ellas.

Mnemónico	Código	Nibble	rcc	Comentario
SBA	0001	1	Según resultado	Resta A y B
ANAB	0010	1	Según resultado	AND lógica entre A y B
ORAB	0011	1	Según resultado	Or Lógica entre A y B
NEGA	0100	1	Según resultado	Invierte A
EOAB	0101	1	Según resultado	OR Exclusiva entre A y B

Tabla 3.8 Instrucciones aritmético-booleanas.

Para llevar acabo el ejemplo utilizaremos el de la tabla 3.7 y lo complementaremos con instrucciones de la tabla 3.8

Acc A	Acc B	rcc	Dirección	Dato
0000	0000	0000	00000000	0111
0000	0000	0000	00000001	0011
0011	0000	0000	00000010	1000
0011	0001	0000	00000011	0001
0011	0001	0000	00000100	0010
0001	0001	0000	00000101	xxxx

Tabla 3.9 Ejemplo de programación usando instrucciones de carga de acumulador y operación aritmético-booleana.

En la tabla 3.9 de la dirección "00000000" hasta la "00000011" se trata del mismo programa de la tabla 3.7 que carga los acumuladores A y B con los datos "0011" y "0001" respectivamente, posteriormente el PC se incrementa y marca la dirección "00000100", el dato apuntado por esta es "0010" que al momento de ser introducido al microprocesador la Unidad de Control lo decodifica como la instrucción que realiza la operación lógica AND entre los acumuladores A y B y guarda el resultado en A. Después de realizada dicha operación el PC se incrementa y los acumuladores quedan como sigue: A→"0001", B→"0001", el nuevo valor de A resultó de aplicar la AND bit a bit entre su valor anterior ("0011") y el valor anterior de B ("0001"); en el rcc se determinó que el resultado no es negativo, no es cero, no hay sobreflujo y no hay acarreo. En la dirección "00000101" el dato señalado es "xxxx" que indica que puede ser cualquier otra instrucción a ejecutarse. En la dirección "00000100" el dato "0010" puede ser sustituido por cualquier valor de la columna de código de la tabla 3.8 con lo cual la operación a ejecutarse se modifica.

Las siguientes 3 instrucciones se utilizan para que el usuario manipule el registro de índice, en conjunto éstas cumplen la función de colocar el resultado de una operación en la ALU en una dirección de memoria especificada por el registro de índice.

Mnemónico	Código	Nibble	Rcc	Comentario
LDXM	0110	3	No se afecta	Carga el índice con el dato inmediato
STAA	1001	1	No se afecta	Almacena el Acc A en la dirección señalada por el índice
INX	1010	1	No se afecta	Incrementa el índice en una unidad

Tabla 3.10 Instrucciones que manipulan el registro de índice.

El siguiente programa ejemplifica la manera de utilizar estas instrucciones, considerando que el acumulador A tiene un valor de "1000".

AccA	Indice	rcc	Dirección	Dato
1000	00000000	0000	00001000	0110
1000	00000000	0000	00001001	1100
1000	00000000	0000	00001010	1111
1000	11111100	0000	00001011	1001
1000	11111100	0000	00001100	1010
1000	11111101	0000	00001101	xxxx

Tabla 3.11 Ejemplo de programación utilizando el registro de índice.

En el ejemplo de programación de la tabla 3.11 el acumulador A contiene el dato "1000" y el PC ha generado la dirección "00001000" que contiene el dato "0110", al momento de entrar este dato al micro la Unidad de Control lo decodifica como la instrucción que carga al registro de índice con la dirección inmediata.

Esta instrucción consta de 3 nibbles, el primero de ellos le indica a la Unidad de Control el tipo de instrucción que es, el segundo y tercero le sirven a la Unidad de Control para formar la dirección que se va a almacenar en el registro de índice; debido a que el registro de índice es de 8 bits, utiliza el nibble 2 para formar la parte baja y el nibble 3 para formar la parte alta de la dirección, después de leer el primer nibble la Unidad de Control incrementa el PC señalando la dirección "00001001" que contiene el dato

"1100" que es la parte baja del registro de índice, nuevamente se incrementa el PC apuntando la dirección "00001010" con dato "1111", que será la parte alta del registro de índice, se incrementa el PC y el registro de índice tiene el valor de "11111100"; hay que recordar nuevamente que a los nibbles 2 y 3 la Unidad de Control no los reconoce como instrucciones sino como datos que se requieren para ejecutar la instrucción que esta en proceso.

La nueva dirección contiene el dato "1001" que la Unidad de Control lo decodifica como la instrucción que almacena el contenido del acumulador A en la dirección de memoria almacenada en el registro de indice, en este caso el valor del acumulador es "1000" y la dirección del índice es la recién formada "11111100". La Unidad de Control coloca la salida r/w en bajo indicando una operación de escritura en el dispositivo que tiene asignada esa dirección, que puede ser una memoria, un convertidor digital-analógico, un buffer de salida, etc. Nuevamente se incrementa el PC indicando la dirección "00001101" que contiene el dato "1010" que la Unidad de Control decodifica como la instrucción que incrementa en una unidad el valor del índice; el PC se incrementa nuevamente y el índice aparece ahora con el valor "11111101".

La siguiente instrucción se utiliza para regresar de una interrupción (IRQ1 y IRQ2); al momento de encontrar esta instrucción el microprocesador coloca los valores que tenían los acumuladores A y B, el rcc, y el PC antes de que ocurriera la interrupción y continua ejecutando el programa a partir de la dirección indicada en el PC.

Mnemónico	Código	Nibble	Rcc	Comentario
RTI	1111	1	No se afecta	Regresa de una interrupción.

Tabla 3.12 Instrucción de retorno de una interrupción.

Hasta el momento se han visto 12 instrucciones, faltan 4 instrucciones que tienen la característica de poder realizar saltos de acuerdo a sí se cumplen ciertas condiciones, en la tabla 3.13 se muestran estas 4 instrucciones.

Mnemónico	Código	Nibble	rcc	Comentario
BCS	1011	2	No se afecta	Salta si C = '1'
BEQ	1100	2	No se afecta	Salta si Z = '1'
BMI	1101	2	No se afecta	Salta si N = '1'
BVS	1110	2	No se afecta	Salta si V = '1'

Tabla 3.13 Instrucciones de bifurcación.

Como se ve en la tabla 3.13 ocurre un salto si uno de los bits del rcc es verdadero, es decir, es igual a 1. Todas utilizan 2 nibbles para ejecutar la instrucción, el primer nibble se utiliza para saber el bit del rcc que se va a evaluar es '1' o '0', el segundo se utiliza para determinar la longitud del salto; la longitud máxima del salto es +7 ó –7 direcciones. El bit cuatro de este segundo nibble nos indica si es hacia delante o hacia atrás el salto, si el bit es '1' el salto es hacia atrás y si es '0' el salto es hacia delante. En caso de que la condición del rcc evaluada no se cumpla el programa se salta 1 dirección(la dirección saltada es la que da el valor del salto) y a partir de esta continua ejecutándose el programa.

En la tabla 3.14 se muestra un ejemplo de programación utilizando una instrucción de salto.

Acc A	Acc B	rcc	Dirección	Dato
0000	0000	0000	0000000	0111
0000	0000	0000	00000001	1111
1111	0000	0000	00000010	1000
1111	0001	0000	00000011	0001
1111	0001	0000	00000100	0001
1110	0001	1000	00000101	1101
1110	0001	1000	00000110	1010
0000	0001	0100	00000111	xxxx

Tabla 3.14 ejemplo de programación utilizando instrucciones de bifurcación.

El ejemplo de la tabla 3.14 abarca desde cargar los acumuladores, realizar una operación aritmética con ellos y realizar un salto si la condición del rcc se cumple.

La primera dirección contiene el dato "0111" que la Unidad de Control interpreta como cargar el valor inmediato en el acumulador A, el PC se incrementa y la dirección "00000001" contiene el dato "1111" que va a ser cargado o almacenado en el acumulador A, se incrementa el PC y el acumulador A ya aparece con su nuevo valor, la siguiente dirección "00000010" contiene el dato "1000" que la Unidad de Control interpreta como cargar el valor inmediato en el acumulador B, el PC se incrementa y la dirección "00000011" contiene el dato "0001", se incrementa nuevamente el PC y el acumulador B ya aparece con su nuevo valor, el PC se incrementa y la dirección "00000100" contiene el dato "0001" que la Unidad de Control identifica como la operación de restar los acumuladores y guardar el resultado en A, se realiza esta primera operación y se incrementa nuevamente el PC, en donde el acumulador A ya aparece con el valor de "1110" resultado de haber restado "0001" de "1111", el rcc contiene el valor de "1000" que indica que el resultado es negativo (4to. Bit), no es cero (3er. bit), no hay sobreflujo (2do. Bit) y no hay acarreo (1er. Bit); la dirección apuntada por el PC es "00000101" con dato "1101" que la Unidad de Control decodifica como saltar si el resultado es negativo.

La Unidad de Control verifica el estado del rcc y encuentra que el según la última operación efectuada el resultado es negativo por lo que se cumple la condición de salto y este se efectuará; se incrementa el PC y la dirección ahora es "00000110" y con dato "1010" la Unidad de Control requiere de este dato para saber la longitud el salto y si es hacia delante o hacia atrás, para ello el dato "1010" tiene el siguiente significado: el 4to. Bit es igual a '1' con lo que el salto es hacia atrás, los siguientes bits determinan su longitud "010" en este caso se trata de 2 direcciones ("010" en decimal es igual a 2), por lo que el salto es hacia atrás y su longitud es de 2 direcciones, se empieza a contar a partir de la última dirección, es decir de la dirección "00000110", saltando 2 direcciones hacia atrás llegamos a la dirección "00000100", la Unidad de Control lee el dato de esta dirección que es "0001" y que decodifica como restar los acumuladores y guardar el resultado en A, posterior a esta operación llegamos nuevamente a la dirección "00000110" que contiene el dato "1101" que indica como ya vimos saltar si el resultado

es negativo, en caso de volverse a cumplir la condición de salto, el programa regresará a la dirección "00000100" y seguirá repitiéndose esta parte del programa hasta que la condición de salto ya no se cumpla, cuando esto ocurra estaremos en la dirección "00000101" y como la condición ya no se cumple el PC se incrementa en 2 ocasiones saltando hasta la dirección "00000111", saltándose el programa la dirección "00000110" que contiene el dato que indica la longitud del salto y si este es hacia atrás o adelante. De esta forma el programa se sale de este bucle y continua ejecutando las siguientes instrucciones a partir de esta última dirección.

Ya se han visto las 16 instrucciones del microprocesador UNAM-4, y podemos observar que son muy fáciles de utilizar y que depende del ingenio de cada persona el combinarlas adecuadamente para generar programas cada vez más complejos

Capítulo IV
Diseño de Sistemas Electrónicos A través del Microprocesador VDHL

4.1 INTRODUCCIÓN.

Como vimos en el capitulo anterior el uso que le podemos dar al microprocesador UNAM4 dependen de el ingenio de cada uno de nosotros, para realizar esto lo único que necesitamos es la lista de instrucciones del microprocesador , la cual recordaremos continuación:

Instrucción	**Mnemónico**	**~**	**#**	**RCC NZVC**	**Código De Operación**	**Comentario**
A + B A	ABA	6	1		0000(0)	Suma Los acumuladores y guarda el resultado en A.
A – B A	SBA	5	1		0001(1)	Resta los acumuladores y guarda el resultado en A
A and B A	ANAB	5	1		0010(2)	AND lógica entre acumuladores y guarda el resultado en A.

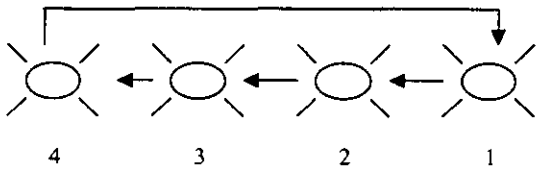
Instrucción	Mnemónico	~	#	RCC NZVC	Código De Operación	Comentario
A or B A	ORAB	5	1		0011(3)	OR lógica entre acumuladores y guarda el resultado en A.
not A A	NEGA	5	1		0100(4)	Invertir A y guarda el resultado en A.
A xor B A	EOAB	5	1		0101(5)	XOR lógica entre acumuladores y guarda el resultado en A.
M X	LDX M	7	3		0110(6)	Carga el índice con la dirección de memoria inmediata.
M A	LDDA	5	2		0111(7)	Carga el acumulador A con el valor inmediato.
M B	LDDB	5	2		1000(8)	Carga el acumulador B con el valor inmediato.
A M	STTA	4	1		1001(9)	Almacena el contenido de A en la dirección del índice.
X + 1 X	INX	3	1		1010(A)	Incrementa en una unidad el valor del registro índice.
C = '1'	BCS	3, 6	2		1011(B)	Salta si C = '1'.
Z = '1'	BEQ	3, 6	2		1100(C)	Salta si Z = '1'.
N = '1'	BMI	3, 6	2		1101(D)	Salta si N = '1'.
V = '1'	BVS	3, 6	2		1110(E)	Salta si V = '1'.
PILA Pcout	RTI	3	1		1111(F)	Regresa de una interrupción.

A continuación veremos algunas aplicaciones que podemos realizar con el microprocesador UNAM4 , utilizando el sistema mínimo que implementamos en el capitulo anterior y la lista de instrucciones.

4.2 APLICACIONES

4.2.1 LEDS EN SERIE

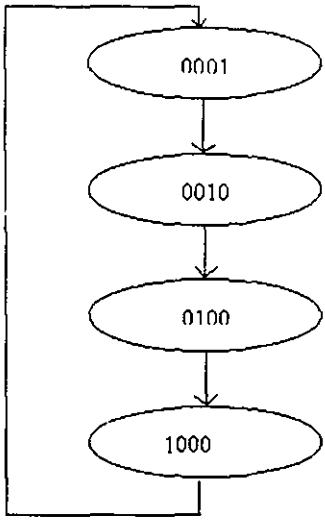
Se diseñará un programa que active 4 leds en serie de derecha a izquierda como se muestra a continuación.



LEDS EN SERIE

El orden en que se deben prender los leds es el siguiente: debe empezar por el led número uno , del uno debe pasar al número dos, del dos debe de pasar al número tres , del tres debe pasar al número cuatro y del cuatro debe regresar al número uno.

A continuación tenemos el diagrama de flujo que muestra el flujo que los datos deben seguir para prender en serie.



A continuación tenemos la tabla de programación del microprocesador, basados en la implementación en el sistema mínimo que se vio en el capítulo anterior.

NÚMERO DE LÍNEA	CÓDIGO EN BINARIO	DIRECCIÓN DE MEMORIA	CÓDIGO EN HEXADECIMAL	COMENTARIO
1	0110	00000000	6	Carga a el índice con la dirección de memoria inmediata.
2	0000	00000001	0	Parte baja del índice.
3	1010	00000010	A	Parte alta del índice.

4	0111	00000011	7	Carga el acumulador A con el valor inmediato.
5	0001	00000100	1	Valor inmediato que se carga en el acumulador A.
6	1001	00000101	9	Manda el contenido de A. a la dirección que tiene el índice.
7	1101	00000110	D	Salta si N es igual a 1
8	1101	00000111	D	Número de direcciones que salta
9	0111	00001000	7	Carga el acumulador A con el valor inmediato.
10	0010	00001001	2	Valor inmediato que se carga en le acumulador A.
11	1001	00001010	9	Manda el contenido de A. a la dirección que tiene el índice.
12	1101	00001011	D	Salta si N es igual a 1
13	1111	00001100	F	Número de direcciones que salta
14	0111	00001101	7	Carga el acumulador A con el valor inmediato.
15	0100	00001010	4	Valor inmediato que se carga en le acumulador A.
16	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.
17	1101	00001100	D	Salta si N es igual a 1
18	1111	00001101	F	Número de direcciones que salta
19	0111	00001110	7	Carga el acumulador A con el valor inmediato.
20	1000	00001111	8	Valor inmediato que se carga en le acumulador A.
21	1001	00010000	9	Manda el contenido de A. a la dirección que tiene el índice.
22	1101	00010001	D	Salta si N es igual a 1
23	1111	00010010	F	Número de direcciones que salta

El programa que se diseñó se explica a continuación.

Al iniciar el programa en la fila 1 la primera instrucción carga el índice con la dirección de memoria inmediata, la dirección del registro índice es de 8 bits, debido a esto se tiene que separar dos partes, los primeros 4 bits forman la parte baja (fila 2) y los otros 4 bits la parte alta (fila 3) , en este caso la dirección que se forma es la 1010000, podemos observar que las filas 2 y 3 esta el 0000 y 1010 respectivamente, que ayudan a formar la dirección. La dirección 10100000 permite el acceso al buffer de entrada / salida (74LS245) que se encuentra en el diagrama del sistema mínimo, recordamos que este buffer permite sacar e introducir datos, en este caso debemos sacar un dato que va a ir directamente a los leds.

En la fila 4 está la instrucción 0111 que indica la carga del acumulador A con el valor inmediato el cual está en la fila 5 (0001); posteriormente en la siguiente fila (6) encontramos la instrucción 1001 que le indica al micro que mande el contenido del acumulador A, a la dirección que está señalando el registro de índice, es decir manda el valor 0001 de A, a la dirección 10100000, pero si recordamos esta dirección es la del buffer, por lo que el valor 0001 va directamente a los leds.

En la fila 7 está la instrucción 1101 que indica saltar si el valor de A es negativo, es decir, el acumulador A es de 4 bits, pero el último bit (bit4) es de suma importancia, porque indica si el valor de A es negativo o positivo, en este caso si el bit4 es 0 por lo que el valor es positivo, pero si es 1 el valor será negativo, si vemos el valor de A hasta el momento es 0001 y el último bit es 0, por lo que el valor es positivo y no habrá salto ya que la condición para que ocurra un salto es que el valor sea negativo.

En la fila 8 esta el valor 1100 es el número que va a saltar si la condición de A fuera verdadera.

En la fila 9 está la instrucción 0111 que carga el valor inmediato en A, el valor inmediato es el 0010 que está en la fila 10, ahora el nuevo valor de A es 0010 y en la siguiente instrucción se manda este valor nuevamente a la dirección del índice y posteriormente viene otra instrucción de salto, pero como el último bit de A es 0, no habrá tal salto.

De la fila 11 a la 20 el procedimiento se repite el único cambio que ocurre es con el valor de A.

Ya en la fila 20 el último valor de A es 1000, el cual va a parar directamente al acumulador A, pero en la siguiente instrucción (fila 22) tenemos que ver si la condición de que A sea negativo se cumple y si analizamos el resultado el último bit de A es 1, por lo que la condición se cumple, como consecuencia habrá un salto.

La siguiente fila indica si el salto es hacia atrás o hacia delante y la distancia del salto. Si observamos en la fila 23 está el valor 1111, el último bit indica el sentido del salto y los otros tres la distancia del salto, como el último bit es 1 el salto será hacia atrás y la longitud de este será de 7 porque el 111 convertido a decimal es 7.

Con la ejecución del salto regresamos a la fila 17 donde está otra instrucción de salto y como el valor de A sigue siendo 1000 se cumple nuevamente la condición y salta hacia atrás 7 direcciones, para llegar a la fila 12 donde nuevamente tengo una instrucción de salto y como la condición de salto se cumple, vuelvo a saltar 7 direcciones para llegar a la fila 7, donde finalmente tengo otra instrucción de salto y como el valor de A no se ha modificado la condición de salto se sigue cumpliendo, ahora el valor del salto es de 5 por lo que llegamos hasta la fila 4 donde la instrucción a ejecutar es la de cargar el acumulador A con el valor 0001, por lo que el programa empieza de nuevo.

Si analizamos el programa, primero se cargó el acumulador A con el valor 0001, luego con el 0010, después con el valor 0100 y finalmente el 1000 y todos fueron enviados a la dirección indicada por el registro de índice, es decir fueron enviados a los leds, en general lo que se hizo fue recorrer un 1 hacia la izquierda y finalmente nos regresamos por medio de saltos a la instrucción donde el acumulador A se carga con el valor 0001, es decir el ciclo se repite otra vez.

A continuación se muestra le diagrama final.

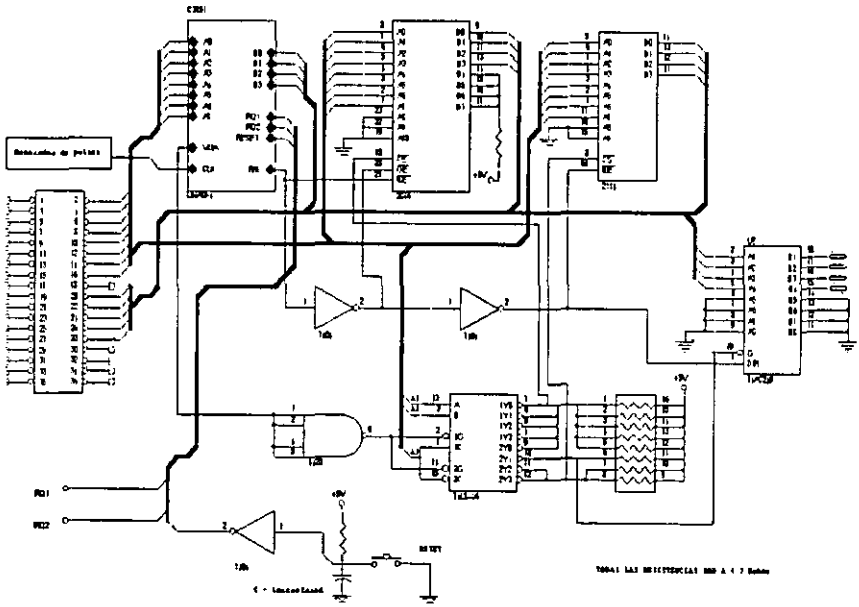
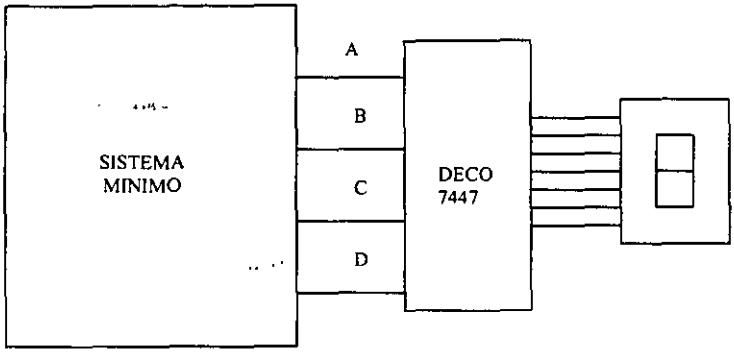


Figura 4.2 LEDS EN SERIE

4.2.2 CONTADOR DEL 0 AL 9 CON UN DISPLAY DE ANADO COMÚN.

A continuación se diseñará un contador del 0 al 9 basados en el sistema mínimo del capítulo anterior, esto se implementará colocando a la salida del sistema un decodificador BCD de 7 segmentos conectado a un display de ánodo común.



Las entradas y el diagrama de flujo se muestra a continuación:

Display	Decodificador (Entrada "D")	Decodificador (Entrada "C")	Decodificador (Entrada "B")	Decodificador (Entrada "A")
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

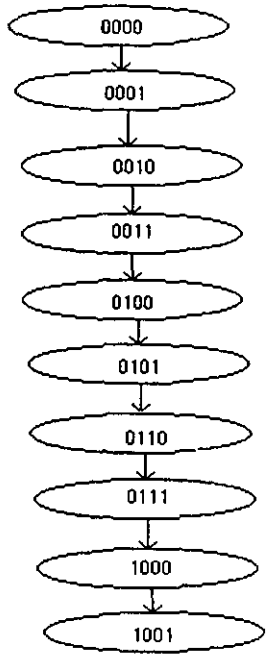


Diagrama de Flujo del Contador del 0 al 9

A continuación tenemos la tabla de programación del microprocesador, basados en la implementación en el sistema mínimo que se vio en el capitulo anterior.

NÚMERO DE LÍNEA	CÓDIGO EN BINARIO	DIRECCIÓN DE MEMORIA	CÓDIGO EN HEXADECIMAL	COMENTARIO
1	0110	00000000	6	Carga a el índice con la dirección de memoria inmediata.
2	0000	00000001	0	Parte baja del índice.
3	1010	00000010	A	Parte alta del índice.

4	1000	00000011	8	Carga el acumulador B con el valor inmediato.
5	0001	00000100	1	Valor inmediato que se carga en el acumulador A.
6	0111	00000101	7	Carga el acumulador A con el valor inmediato
7	0000	00000110	0	Valor inmediato que se carga en el acumulador A..
8	1001	00000111	9	Manda el contenido de A. a la dirección que tiene el índice.
9	0000	00001000	0	Suma A + B
10	1001	00001001	9	Manda el contenido de A. a la dirección que tiene el índice..
11	0000	00001010	0	Suma A + B
12	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.
13	0000	00001010	0	Suma A + B
14	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.
15	0000	00001010	0	Suma A + B
16	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.
17	0000	00001010	0	Suma A + B
18	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.
19	0000	00001010	0	Suma A + B
20	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.
21	0000	00001010	0	Suma A + B
22	1001	00001011	9	Manda el contenido de A. a la dirección que tiene el índice.

23	0000	00001010	0	**Suma A + B**
24	1001	00001011	9	**Manda el contenido de A. a la dirección que tiene el índice.**
25	0000	00001010	0	**Suma A + B**
26	1001	00001011	9	**Manda el contenido de A. a la dirección que tiene el índice.**

El programa que se diseñó se explica a continuación.

Al iniciar el programa en la fila 1 la primera instrucción carga el índice con la dirección de memoria inmediata, la dirección del registro índice es de 8 bits, debido a esto se tiene que separar dos partes, los primeros 4 bits forman la parte baja (fila 2) y los otros 4 bits la parte alta (fila 3) , en este caso la dirección que se forma es la 1010000, podemos observar que las filas 2 y 3 esta el 0000 y 1010 respectivamente, que ayudan a formar la dirección. La dirección 10100000 permite el acceso al buffer de entrada/salida (74LS245) que se encuentra en el diagrama del sistema mínimo, recordamos que este buffer permite sacar e introducir datos, en este caso debemos sacar un dato que va a ir directamente a los leds, esto funciona como en la primera aplicación que vimos en este capítulo.

En la fila 4 esta la instrucción 1000 que indica la carga del acumulador B con el valor inmediato el cual está en la fila 5 (0001); posteriormente en la siguiente fila (6) encontramos la instrucción 0111 que le indica a el micro que cargué el acumulador A con el valor inmediato (0000) posteriormente mande el contenido del acumulador A (fila 8) a la dirección que esta señalando el registro de índice, es decir manda el valor 0000 de A, a la dirección 10100000, pero si recordamos esta dirección es la del buffer, por lo que el valor 0000 va directamente al integrado, el cual es un decodificador BCD de 7 segmentos 74LS47 que está conectado al display con lo que tendremos a la salida el número 0.

En la fila 9 tenemos la instrucción de suma , es decir sumará los valores de A (0000) y B (0001) y el resultado (0001) lo colocará en A, posteriormente manda el resultado a la dirección que tiene el índice (instrucción 1001); como ahora tenemos el valor 0001 en A y el valor 0001 en B podemos hacer directamente la suma (instrucción 0000) como vemos en la fila 11 y mandar el resultado a la dirección del índice.

Este procedimiento se repite desde la fila 13 hasta la fila 26, incrementándose en uno el resultado el cual va a ir del 1 al 9 lo que nos permite que se habilite el decodificador y el conteo se muestre en el display.

A continuación se muestra el diagrama final:

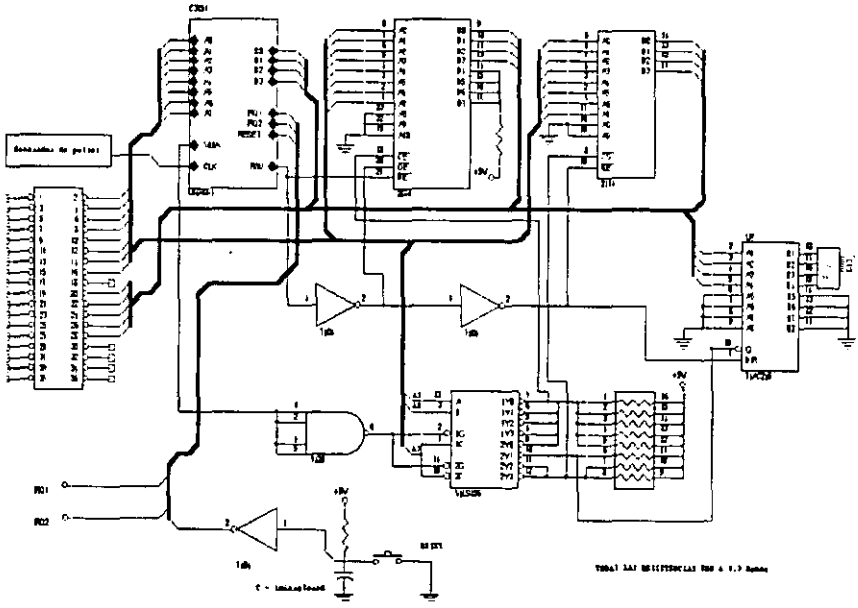
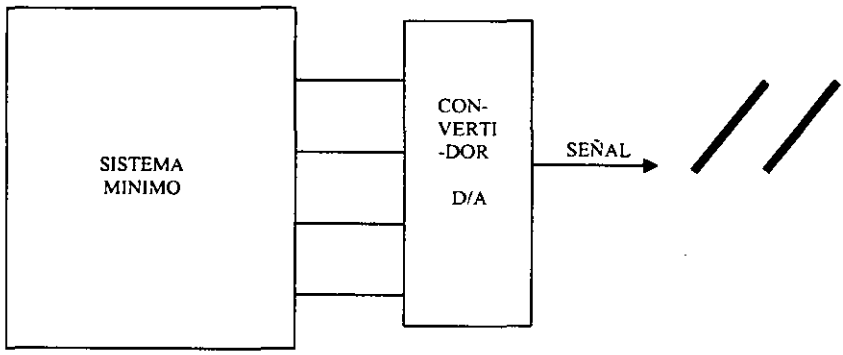


Figura 4.2.2 CONTADOR DEL 0 AL 9

4.2.3 GENERADOR DE SEÑAL DIENTE DE SIERRA

Se diseñará un programa que genere una señal diente de sierra, esto se va a implementar con el sistema mínimo que se diseñó en el capitulo anterior colocando a la salida un convertidor digital/analógico.



Esta señal se va generar cuando el sistema mínimo vaya incrementando en uno desde el dato 0001 hasta llegar al dato 1111 y estos valores pasen por el convertidor D/A dando como resultado la señal diente de sierra.

A continuación tenemos el diagrama de flujo que muestra el flujo que los datos deben seguir para generar esta señal.

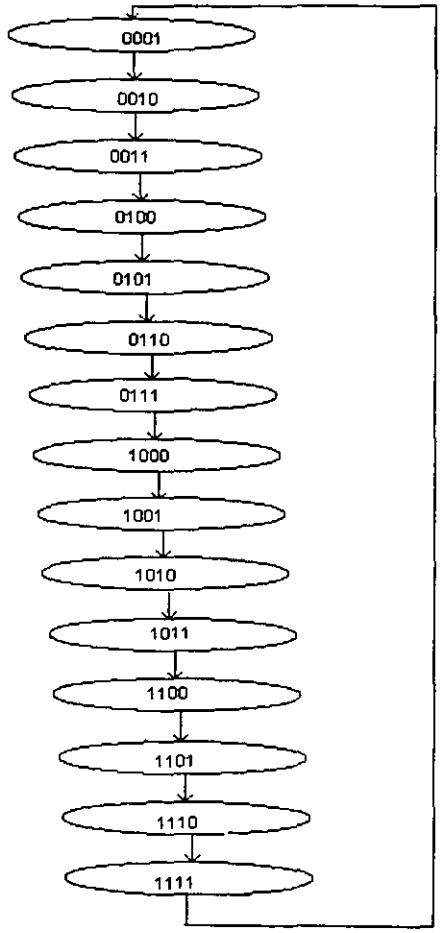


Diagrama de Flujo del Generador de Señal

A continuación tenemos la tabla de programación del microprocesador, basados en la implementación en el sistema mínimo que se vio en el capitulo anterior.

NÚMERO DE LÍNEA	CÓDIGO EN BINARIO	DIRECCIÓN DE MEMORIA	CÓDIGO EN HEXADECIMAL	COMENTARIO
1	0110	00000000	6	Carga a el índice con la dirección de memoria inmediata.
2	0000	00000001	0	Parte baja del índice.
3	1010	00000010	A	Parte alta del índice.
4	1000	00000011	8	Carga el acumulador B con el valor inmediato.
5	0001	00000100	1	Valor inmediato que se carga en el acumulador B.
6	0111	00000101	7	Carga el acumulador A con el valor inmediato.
7	0000	00000110	0	Valor inmediato que se carga en el acumulador A.
8	1001	00000111	9	Manda el contenido de A. a la dirección que tiene el índice.
9	0000	00001000	0	Suma A + B
10	1001	00001001	9	Manda el contenido de A. a la dirección que tiene el índice.
11	1110	00001010	E	Salta si V = 1
12	1111	00001011	F	Número de direcciones que salta
13	0000	00001100	0	Suma A + B
14	1001	00001101	9	Manda el contenido de A. a la dirección que tiene el índice.
15	1110	00001110	E	Salta si V = 1
16	1110	00001111	E	Número de direcciones que salta
17	0000	00010000	0	Suma A + B
18	1001	00010001	9	Manda el contenido de A. a la dirección que tiene el índice.
19	1110	00010010	E	Salta si V = 1
20	1110	00010011	E	Número de direcciones que salta

21	0000	00010100	0	Suma A + B
22	1001	00010101	9	Manda el contenido de A. a la dirección que tiene el índice.
23	1110	00010110	E	Salta si V = 1
24	1110	00010111	E	Número de direcciones que salta
25	0000	00011000	0	Suma A + B
26	1001	00011001	9	Manda el contenido de A. a la dirección que tiene el índice.
27	1110	00011010	E	Salta si V = 1
28	1110	00011011	E	Número de direcciones que salta
29	0000	00011100	0	Suma A + B
30	1001	00011101	9	Manda el contenido de A. a la dirección que tiene el índice.
31	1110	00011110	E	Salta si V = 1
32	1110	00011111	E	Número de direcciones que salta
33	0000	00100000	0	Suma A + B
34	1001	00100001	9	Manda el contenido de A. a la dirección que tiene el índice.
35	1110	00100010	E	Salta si V = 1
36	1110	00100011	E	Número de direcciones que salta
37	0000	00100100	0	Suma A + B
38	1001	00100101	9	Manda el contenido de A. a la dirección que tiene el índice.
39	1110	00100110	E	Salta si Z = 1
40	1110	00100111	E	Número de direcciones que salta
41	0000	00101000	0	Suma A + B
42	1001	00101001	9	Manda el contenido de A. a la dirección que tiene el índice.
43	1110	00101010	E	Salta si Z = 1
44	1110	00101011	E	Número de direcciones que salta
45	0000	00101100	0	Suma A + B

46	1001	00101101	9	Manda el contenido de A. a la dirección que tiene el índice.
47	1110	00101110	E	Salta si Z = 1
48	1110	00101111	E	Número de direcciones que salta
49	0000	00110000	0	Suma A + B
50	1001	00110001	9	Manda el contenido de A. a la dirección que tiene el índice.
51	1110	00110010	E	Salta si Z = 1
52	1110	00110011	E	Número de direcciones que salta
53	0000	00110100	0	Suma A + B
54	1001	00110101	9	Manda el contenido de A. a la dirección que tiene el índice.
55	1110	00110110	E	Salta si Z = 1
56	1110	00110011	E	Número de direcciones que salta
57	0000	00110100	0	Suma A + B
58	1001	00110101	9	Manda el contenido de A. a la dirección que tiene el índice.
59	1110	00110110	E	Salta si Z = 1
60	1110	00110111	E	Número de direcciones que salta
61	0000	00111000	0	Suma A + B
62	1001	00111001	9	Manda el contenido de A. a la dirección que tiene el índice.
63	1110	00111010	E	Salta si Z = 1
64	1110	00111011	E	Número de direcciones que salta
65	0000	00111100	0	Suma A + B
66	1001	00111101	9	Manda el contenido de A. a la dirección que tiene el índice.
67	1110	00111110	E	Salta si Z = 1
68	1110	00111111	E	Número de direcciones que salta

69	0000	01000000	0	Suma A + B
70	1001	01000001	9	Manda el contenido de A. a la dirección que tiene el índice.
71	1110	01000010	E	Salta si Z = 1
72	1110	01000011	E	Número de direcciones que salta

Ahora vamos a explicar la tabla de programación.

Como vimos en las aplicaciones anteriores lo primero que tenemos que hacer es direccionar al buffer de salida de igual manera que en las aplicaciones anteriores, es decir de la línea 1 a la 3 siempre va a ser el mismo procedimiento para el sistema mínimo.

A continuación explicaremos de la línea 4 en adelante .

En la línea número 4 tenemos la instrucción 1000 que carga el acumulador B, la linea 5 nos muestra cual es el valor que se va a cargar en el acumulador (0001), la línea 6 carga el acumulador A (instrucción 0111) con el valor 0000 (línea 7) a continuación va a sacar este valor enviándolo a la dirección que tiene el índice con la instrucción 1001; como necesitamos ir incrementando el valor en 1 vamos a utilizar la suma como ya tenemos los valores en A y B solo debemos utilizar la instrucción de suma (fila 9 instrucción 0000), después se manda el resultado a la dirección del índice con la instrucción 1001, a continuación tenemos una condición que nos ayuda a que el programa se repita cuando la condición se cumple, la condición está en la fila 11 y compara si en el resultado de la suma hay sobreflujo si es cierto regresa hacia arriba el número de direcciones que se indican en la fila 12 , si no continua el programa.

A continuación como ya tenemos los valores en los acumuladores vamos a sumarlos (fila 13), sacar el resultado (fila 14) y ver si la condición se cumple (fila15) si es cierta salta el numero de direcciones de la fila 16, si no sigue la ejecución del programa.

Esta secuencia se repite desde la fila 17 hasta la fila 72, así iremos incrementando hasta llegar al número 1111 donde la condición se cumple debido al sobreflujo (fila71), entonces va regresando hasta llegar nuevamente a la línea 4 y vuelve a empezar.

Estos valores van a ir generando la señal que al pasarlos por el convertidor digital / analógico no da la señal diente de sierra.

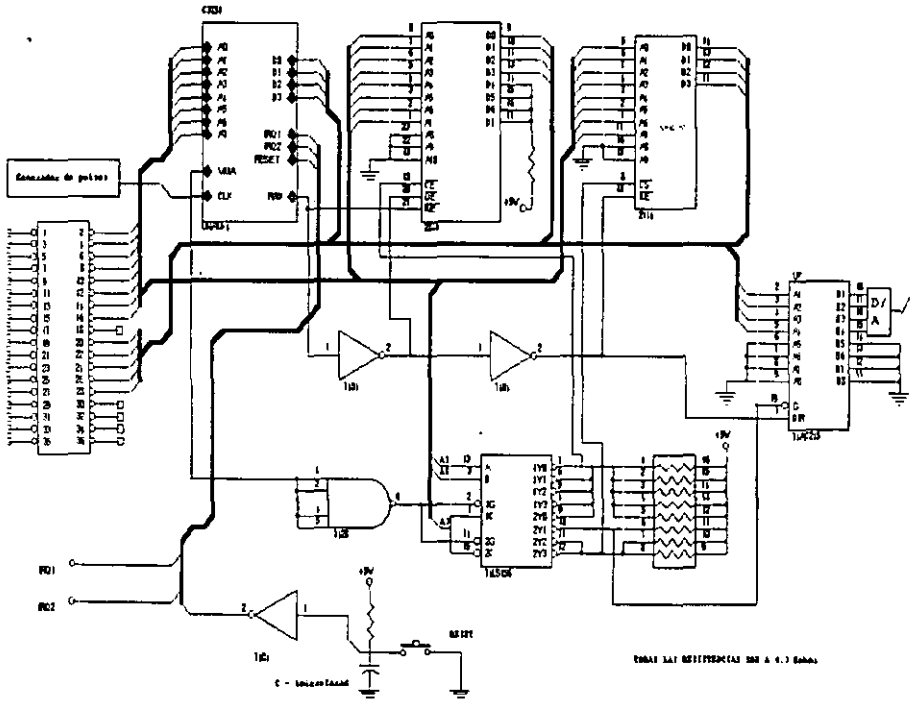
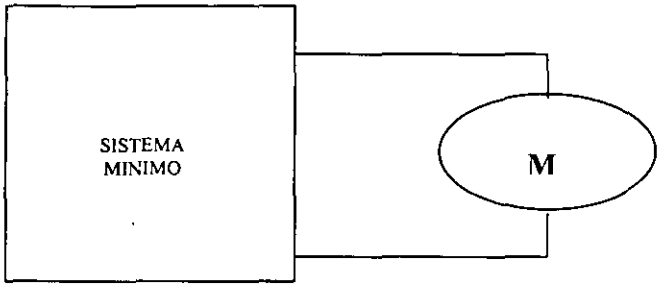


Figura 4.2.3 GENERADOR DE SEÑAL DIENTE DE SIERRA

4.2.4 ROTACION DE UN MOTOR A PASOS

Se diseñará un programa que mueva un motor a pasos en sentido contrario de las manecillas del reloj.



Para lograr la rotación del motor se necesitan

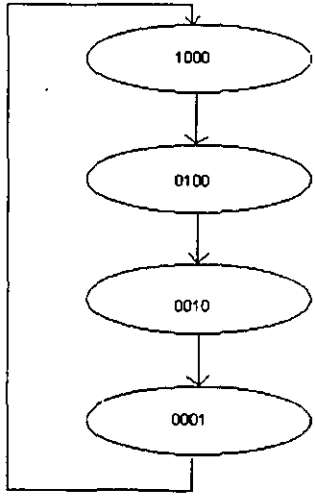


Diagrama De Flujo De Motor a Pasos

NÚMERO DE LÍNEA	CÓDIGO EN BINARIO	DIRECCIÓN DE MEMORIA	CÓDIGO EN HEXADECIMAL	COMENTARIO
1	0110	00000000	6	Carga a el índice con la dirección de memoria inmediata.
2	0000	00000001	0	Parte baja del índice.
3	1010	00000010	A	Parte alta del índice.
4	1000	00000011	8	Carga el acumulador B con el valor inmediato.
5	0001	00000100	1	Valor inmediato que se carga en el acumulador B.
6	0111	00000101	7	Carga el acumulador A con el valor inmediato
7	1000	00000110	8	Valor inmediato que se carga en el acumulador A
8	1001	00000111	9	Manda el contenido de A. a la dirección que tiene el índice.
9	1100	00001000	C	Salta si Z=1
10	1101	00001001	D	Número de direcciones que salta
11	0111	00001010	7	Carga el acumulador A con el valor inmediato.
12	0100	00001011	4	Valor inmediato que se carga en le acumulador A
13	1001	00001100	9	Manda el contenido de A. a la dirección que tiene el índice.
14	1101	00001101	D	Salta si Z=1.
15	1111	00001010	F	Número de direcciones que salta
16	0111	00001011	7	Carga el acumulador A con el valor inmediato.
17	0010	00001100	2	Valor inmediato que se carga en le acumulador A
18	1001	00001101	9	Manda el contenido de A. a la dirección que tiene el índice.

19	1101	00001110	D	Salta si Z=1.
20	1111	00001111	F	Número de direcciones que salta
21	0001	00010000	1	Resta A - B
22	1001	00010001	9	Manda el contenido de A. a la dirección que tiene el índice
23	0001	00010010	1	Resta A - B
24	1100	00010011	C	Salta si Z=0
25	1111	00010100	F	Número de direcciones que salta

La tabla anterior permite la rotación del motor de la siguiente manera.
Como vimos anteriormente todos las aplicaciones tienen en común las primeras tres líneas

La instrucción de la línea 4 instrucción 1000 carga el acumulador B con el valor inmediato (0001) este valor no va a servir posteriormente para hacer que el programa se repita n veces , a continuación se carga el acumulador A (fila 6) con el valor inmediato (1000) y manda este valor a la dirección del índice (instrucción 1001) en la fila 9 hay una condición la cual nos permite regresar si es verdadera para que el programa sea cíclico, esta condición es la instrucción 1100 la cual nos indica que si el valor es igual a cero debe regresara el número de direcciones que se indica en la fila 10.

En la fila 11 cargamos nuevamente el acumulador A (instrucción 0111) con el valor 0100 y después lo manda a la dirección del índice para que el valor salga (fila13), cuando ya tenemos el valor volvemos a ver si la condición se cumple si es verdadera regresa y si es falsa sigue con la ejecución del programa

Este ciclo se repite desde al fila 16 hasta la fila 20 , ya que en la fila 21 tenemos la instrucción 0001 la cual resta el acumulador A y el B en este caso restaría el valor de A 0010 y el valor de B 0001, dando como resultado 0001 y manda este resultado al acumulador A , debido a que queremos que el programa sea cíclico utilizamos a lo largo de este la condición de que el valor sea igual a cero para que regrese hasta este punto el

valor no es cero así que debemos forzarlo a que sea cero esto lo haremos restando nuevamente el valor de A 0001 y el valor de B 0001 fila 23 , el resultado de esta operación va a ser cero; si después se compara el resultado con la condición de la fila 24 se observa que es verdadera ya que el valor es igual a cero y va a saltar el número de direcciones que se indica en al fila 25, así va ir regresando hasta llegar a la fila 4 y se repite nuevamente .

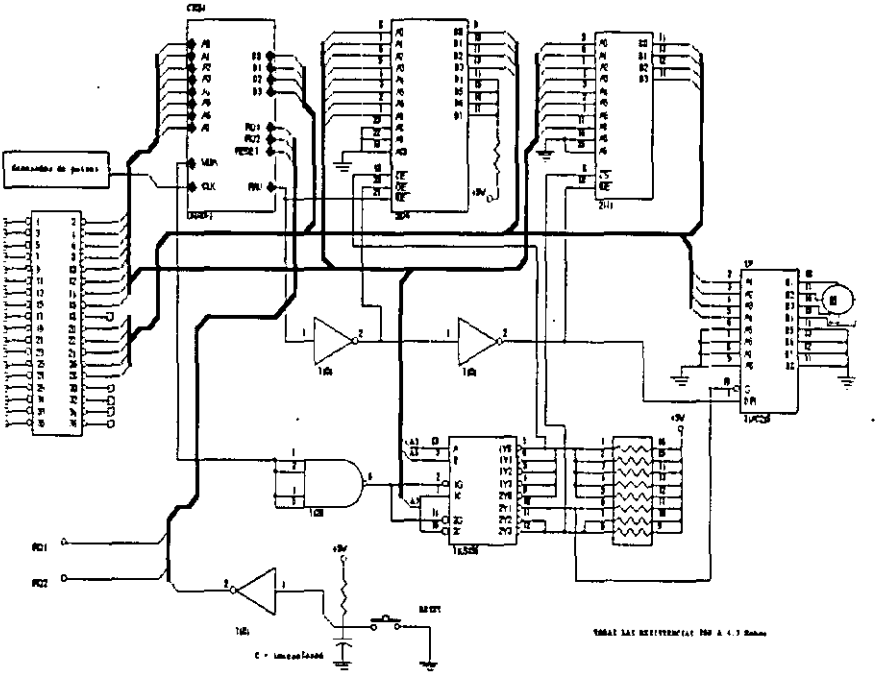


Figura 4.2.4 ROTACION DE UN MOTOR A PASOS

Como vemos podemos diseñar muchas aplicaciones todo depende de las necesidades y el ingenio de cada uno .

CONCLUSIONES

Como se observa la tecnología avanza rápidamente y todos los que contribuimos al desarrollo de esta debemos ir siempre a la par, es por eso que en este proyecto se muestra una herramienta de vanguardia como es el lenguaje VHDL.

Este lenguaje tiene múltiples beneficios en diversas áreas, como la automotriz, hogar, medicina, entre otras, hemos aprendido a lo largo de éste el uso del lenguaje, además de mostrar una perspectiva de uso para el diseño de nuevas aplicaciones de acuerdo a las necesidades de cada persona o compañía. Por lo que podemos concluir lo siguiente :

- ✓ El lenguaje VHDL es un lenguaje que tiene una gama muy amplia de desarrollo si se explota adecuadamente.
- ✓ Es fácil y flexible además de ser un estándar, por lo que garantiza su estabilidad y soporte.
- ✓ El lenguaje VHDL permite la creación de nuevos y mejores diseños de sistemas embebidos.
- ✓ Debido a las múltiples ventajas que ofrece como disminución de costo y tiempo, el campo de la industria puede hacer gran uso de él
- ✓ El uso a nivel internacional va aumentando día con día debido a las ventajas que ofrece para el diseño de dispositivos por lo que la mayoría de las grandes empresas del sector han definido VHDL como el lenguaje preferencial en todas las tareas de diseño, modelado, documentación y mantenimiento de sistemas electrónicos.
- ✓ Los diseños en VHDL pueden ser reutilizables en distintas tecnologías como en especificaciones.

- ✓ La implementación física se consigue rápidamente sin grandes costos.
- ✓ Permite el manejo de proyectos de grandes dimensiones.
- ✓ Proporciona la descripción del hardware en varios niveles de abstracción desde la especificación hasta la implementación final, lo que permite adaptarse a distintos propósitos .
- ✓ Diseño e implementación de circuitos muy complejos de manera sencilla.
- ✓ Debido a que VHDL es legible tanto por las personas como por las máquinas, soporta el desarrollo, verificación, síntesis y chequeo del diseño hardware; así como la comunicación de los datos del diseño hardware; y el mantenimiento, modificación y la obtención de hardware.
- ✓ Es un estándar no sometido a ninguna patente o marca registrada: así puede utilizarse sin restricciones por cualquier empresa o institución.
- ✓ Soporta diferentes metodologías de diseño.
- ✓ La utilización de un solo lenguaje en todo el proceso de diseño reduce el número de representaciones internas, simplificando el manejo de datos.
- ✓ Los componentes del sistema pueden diseñarse con mayor independencia, ya que el entorno permite la simulación multinivel, es decir, combina componentes descritos al nivel de especificación con componentes ya implementados al nivel de compuerta.
- ✓ Una vez obtenida la implementación final, la descripción VHDL puede usarse con objeto de especificar las características y prestaciones del producto.

En resumen, podemos decir que debido a las ventajas que ofrece el lenguaje VHDL para los diversos sectores productivos de un país, y que en México el uso de este lenguaje esta en su etapa inicial, resulta importante la capacitación de recursos humanos que implementen esta tecnología en el país para hacer más eficientes los procesos de diseño en todos los sectores.