UNIVERSIDAD RICARDO PALMA

Facultad de Ingeniería

Escuela de Ingeniería Electrónica

Sistema Autómata Secuencial por Tiempos Independientes (S.A.S.T.I)

Para optar por el título de Ingeniero Electrónico

Carlos Alberto Méndez Chang

Surco, Octubre 2010

A mis Padres Carlos Méndez Rodriguez y Blanca Chang Asmat y a mi asesor Ing. Gustavo Roselló Moreno por todo su apoyo

TABLA DE CONTENIDOS

INTRODUCCIÓN 6
CAPÍTULO I. ANTECEDENTES: SISTEMAS CONVENCIONALES 8
1.1 Breve marco histórico 8
1.2 La sincronización y algunos problemas 21
1.3 La velocidad como ideal 24
1.4 Algunas consideraciones sobre los Actuadores 26
1.5 Una solución en boga: el *Self-Timed* 29
1.6 Algunas conclusiones sobre lo expuesto 31
CAPÍTULO II EL SISTEMA AUTÓMATA SECUENCIAL POR TIEMPOS INDEPENDIENTES S.A.S.T.I. 33
2.1 Nacimiento de la idea 34
2.2 La Idea Primaria 35
2.3 Desarrollo del sistema prototipo 36
2.4 Tortuga Programable Amauta MCh-3A1 38
2.5 Proyecto Amauta-II 42
2.6 El concepto de Instrucción-Dato (I-D) 42
2.6.1 Datos e Instrucciones 43
2.6.2 Tamaño de Memoria 44
2.7 El primer SASTI real 45
2.8 El uso de FPGA y CPLD 46
2.8.1 Cajas Negras 47
2.8.2 VHDL y Modo Gráfico 48
2.9 El diseño de la *PseudoROM* 50
CAPÍTULO III EL S.A.S.T.I. TIPO I 53
3.1 SASTI – IA 55
3.1.1 Accionamiento y tiempos 55
3.1.2 Tiempos y posiciones de memoria necesarias 57
3.1.3 Instrucciones 58
3.2 SASTI – IB 61
3.2.1 Tiempos 61

3.2.2 Instrucciones y acciones ... 62
3.2.3 Tamaño de la memoria y Temporización ... 66
3.2.4 Decodificador de Instrucciones y Bus de Salida ... 66
3.3 El subsistema *MultiClock* ... 67
3.3.1 Diseño del MultiClock ... 68
3.3.2 Diseño del bloque *MultiClock* ... 70
3.4 S.A.ST.I. serie Mk-II ... 72
3.4.1 SASTI – IA Mk-II ... 73
3.4.2 SASTI – IB Mk-II ... 77
3.4.3 SASTI – IC y SASTI - ID ... 78
CAPÍTULO IV EL *S.A.S.T.I.* TIPO *II* ... 80
4.1 SleepingClock ... 80
4.1.1 Funcionamiento General ... 81
4.1.2 La Condición de *Igualdad Absoluta* ... 82
4.1.3 Temporización ... 83
4.1.5 Equivalente con sistemas SASTI de tipo I ... 85
4.1.6 El uso del concepto de Puerto ... 85
4.2 El SASTI-IIC y el SASTI-IC ... 87
4.2.1 Número de puertos ... 87
4.2.2 Decodificador de Instrucciones ... 88
4.2.3 Temporización ... 89
4.2.4 Memoria y Contador de Programa (C_P) ... 89
4.3 El SASTI –IID y SASTI – ID ... 90
4.3.1 Los Registros Especializados (R_E) ... 91
4.3.2 La Unidad de Comparación (U_{Comp}) ... 91
4.3.3 Unidad de Saltos (U_S) ... 92
4.3.4 Contador de Programa (C_P) ... 93
CAPÍTILO V PRUEBAS PRÁCTICAS ... 94
5.1 Metodología general ... 94
5.2 Prototipo Experimental de *SASTI – IA* ... 96
5.2.1 Palabra de Memoria ... 96
5.2.2 Tamaño de Memoria ... 96
5.2.3 Instrucciones ... 97
5.2.4 Contador de Programa (C_P) ... 98

5.2.5 Sistema de Reset ... 98
5.2.6 Conclusiones ... 98
5.3 Prototipo Experimental SASTI – IB ... 99
5.3.1 Determinación de la *Palabra de Memoria* e Instrucciones ... 99
5.3.2 Tamaño de la Memoria ... 99
5.3.3 Tamaño del Bus de Salida ... 100
5.3.4 Contador de Programa (C_P) ... 100
5.3.5 Sistema de Reset ... 100
5.3.6 Conclusiones ... 101
5.4 Prototipos Experimentales de clase *Mk-II* ... 101
5.4.1 Caso del tipo IA Mk-II ... 101
5.4.2. Caso del tipo *IB Mk-II* ... 102
5.5 Prototipos Experimentales del tipo *IIC y IC* ... 104
5.5.1 Elementos estándares ... 104
5.5.2 Diseño de la memoria ... 104
5.5.3 Sistema de Puertos ... 105
5.6 Prototipo Experimental *SASTI – IID* ... 106
5.6.1 Elementos estándares ... 106
5.6.2 Unidad de Comparación (U_{Comp}) ... 106
5.6.3 La Unidad de Saltos ... 106
5.6.4 La Instrucción-Dato ... 107
CAPÍTULO VI OBSERVACIONES Y CONCLUSIONES ... 108
6.1 Observaciones y Recomendaciones ... 108
6.2 Conclusiones del Proyecto ... 111

INTRODUCCIÓN

En la actualidad, la miniaturización ha llegado a extremos tales que su propia capacidad está al límite. La idea inicial de una miniaturización radical desde la *LSI* (*Large Scale Integration*, gran escala de integración) hasta la *ULSI* (*Ultra Large Scale Integration*, Ultra gran escala de integración), que se basaba en realizar sistemas más complejos y rápidos en escalas cada vez más pequeñas, se ha visto en la práctica frenada con la llegada al límite físico de miniaturización; y sin mencionar que a mayor frecuencia, mayor consumo de los sistemas. Esto es fácilmente observable en la nueva tendencia de los procesadores: en vez de pretender hacerlos más rápidos o más pequeños, se está apostando por la multiplicidad de núcleos; es decir, varios procesadores realizando tareas en paralelo.

Lo anteriormente expuesto es una generalidad de lo que es observable en el mundo de las computadoras, pero es suficiente para darnos cuenta cómo se está tratando de lidiar con un problema límite en la arquitectura de sistema procesador existente. Una arquitectura nacida de una necesidad: el cálculo.

Por ello, es lícita la pregunta *¿no existe otra alternativa?* En la actualidad se han llevado a cabo experimentos con otras arquitecturas de sistemas (los llamados *Self-Timed* o Autotemporizados) para tratar de reducir la velocidad de procesamiento (uno de los problemas de los sistemas actuales); sin embargo, en líneas generales sigue estando basado en solucionar problemas de cálculo. Hasta ahora, no se ha podido encontrar ninguna referencia hacia una solución orientada en otro sentido.

Estando el panorama de la manera antes mencionada, se plantea una alternativa que no nace de necesidades de cálculo. Esta alternativa nace más de la observación y la experiencia, con miras a solucionar una actividad específica: el control de actuadores. Inicialmente no se le plantea como una unidad de cálculo que pueda realizar un control, sino como una unidad que controle actuadores.

Inclusive, el sistema abandonará la estructura convencional basada en la sincronización, para dar énfasis a una estructura basada en la temporización; siendo quizás lo más

cercano a la idea primaria no un sistema electrónico sino un sistema mecánico: un sistema de relojería.

Aunque se pudiera considerar que el hecho de ser una arquitectura con miras a desarrollar un sistema autómata (con todo lo que esto implica), el sistema podría ser bastante limitado, los hechos experimentales demuestran que es posible inculcarle más elementos que podrían alejarlo del simple autómata a un sistema más complejo.

CAPÍTULO I. ANTECEDENTES: SISTEMAS CONVENCIONALES

Los sistemas procesadores digitales convencionales en la actualidad están basados en los microprocesadores y sus variedades integradas (llámese microcontroladores); inclusive, con una casi tácita tendencia a llevarlo prácticamente a cualquier campo y lugar: desde un equipo de telefonía móvil hasta dentro de nuestros propios organismos. Esta tendencia a su uso extendido está basado en sus años de éxitos; éxitos originalmente orientados a sólo una actividad: el cálculo.

1.1 Breve marco histórico.

Cuando se habla de del *Microprocesador*, éste se ve inmediatamente asociado al *Computador.* Este hecho que podría ser considerado casi anecdótico, tiene en realidad mucho que ver con la razón por la cual se realizaron los esfuerzos para dar nacimiento a la idea del microprocesador.

En líneas generales se puede decir que la palabra ***"computador"*** puede rastrearse hasta 1613[1]; en aquellos tiempos, el término se refería a un ser humano (en muchos casos, mujeres). El trabajo de un computador era realizar cálculos repetitivos con la finalidad de computar las tablas de navegación, tablas de mareas y las posiciones planetarias para los almanaques astronómicos[2]. Es obvio que un trabajo repetitivo y lento con alta exigencia mental, puede llevar al cansancio mental y éste a su vez ser la causa de errores. A pesar de la existencia de ayudas tales como el ***ábaco*** (ejemplo más antiguo: Babilonia, 300 AC; erróneamente se consideraba al ábaco de origen chino[3]), éste no dejaba de ser sólo una ayuda que intentaba suplantar a las manos.

---

[1] **http://en.wikipedia.org/wiki/History_of_computing_hardware, Before computer hardware**

[2] **http://www.computersciencelab.com/ComputerHistory/History.htm, Part 1**

[3] **Ídem a referencia 2**

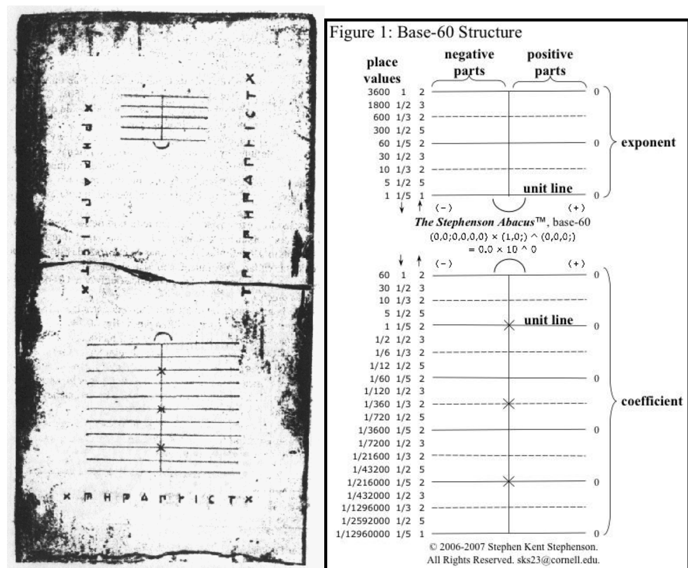
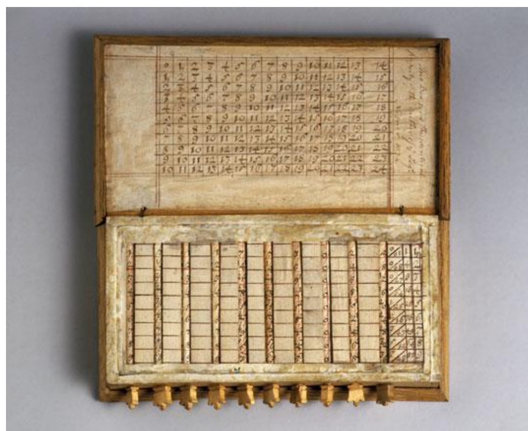


Imagen de un ábaco Babilonio y su interpretación moderna.

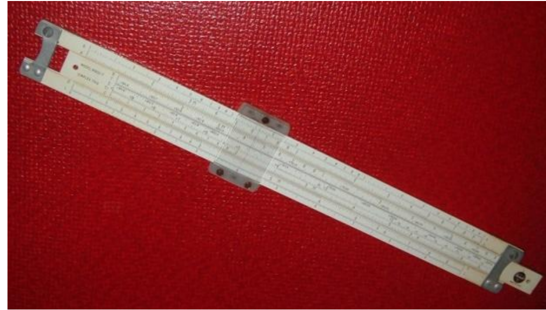
Es por ello que se trató de buscar soluciones más efectivas tales como la invención de los *logaritmos* por John Napier (1617), los cuales permitía mediante sumas realizar multiplicaciones; para realizar los cálculos logarítmicos se recurrían a las tablas de logaritmos, inicialmente en placas de marfil denominadas *Huesos de Napier*.



Conjunto de *Huesos de Napier*, precursor de las tablas de Logaritmos escritas

Pero, a pesar de todo no dejaban de ser simples ayudas pasivas. Inclusive, la llamada *Regla de Cálculo* (Inglaterra, 1632) no fue más que una ayuda elaborada; aunque fue usada aún en la década de los 60's por Ingenieros de la NASA en sus programas

Mercury, Gemini y Apollo[4]. La necesidad de calcular más eficientemente y más rápido llevó a varios desarrollos que fueron realmente los precursores de los sistemas microprocesadores actuales.

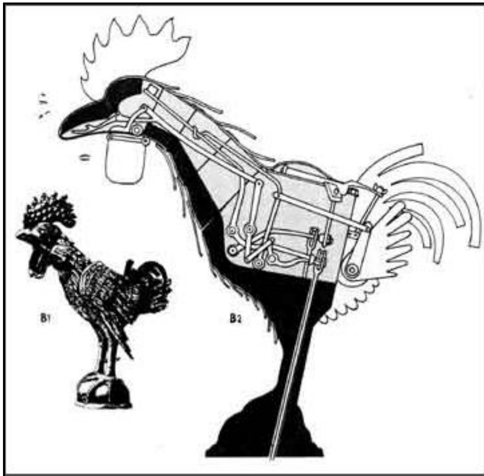


Regla de Cálculo, muy usada en ingeniería. Hoy prácticamente en desuso.

La idea de una máquina que pudiera realizar cálculos en vez de un ser humano no era visionaria, teniendo en cuenta que existieron antecedentes de mecanismos autómatas elaborados; aunque sus aplicaciones no eran precisamente relacionadas al cálculo. El reloj de agua de Al-Jazari (1136 - 1206)[5] o el gallo mecánico de la Catedral de Estrasburgo (1354[6] y operativo hasta 1789[7]) son ejemplos de dispositivos que demostraban la existencia de tecnología y conocimientos capaces de desarrollar una máquina autómata elaborada. La necesidad del cálculo hizo que los conocimientos volcados en autómatas y relojes fueran usados para dar solución al problema de calcular a la mayor velocidad posible.

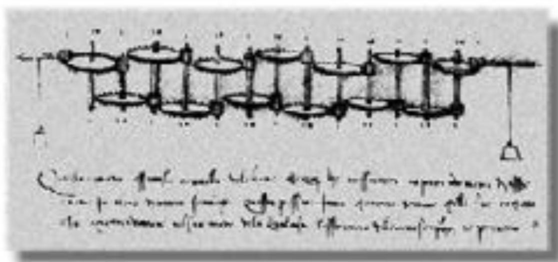
---

[4] **Ídem a referencia 2**
[5] **http://www.muslimheritage.com/uploads/Automation_Robotics_in_Muslim%20Heritage.pdf**
[6] **Robots: una revolución, pág. 42**
[7] **http://novella.mhhe.com/sites/dl/free/8448156366/507530/Cap_Muest_Barrientos_8448156366.pdf**

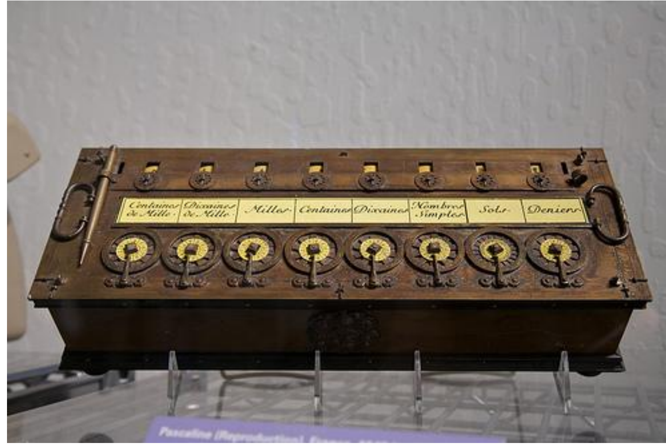


El Gallo de Estrasburgo y el Reloj de Agua de Al-Jazari: ejemplos de destreza técnica.

Evidentemente los primeros intentos fueron netamente de tipo mecánico. Leonardo da Vinci (1452 - 1519) dejó dibujos que aparentan ser juegos de engranajes para máquinas calculadoras[8]. Wilhelm Schickard desarrolló el *Reloj Calculador* (1623). Blaise Pascal inventó la *Pascaline* (1642), aunque con problemas debido a la falta de precisión en los engranajes (algo que persiguió hasta al mismo Charles Babbage posteriormente).



[8] **Ídem a referencia 2**

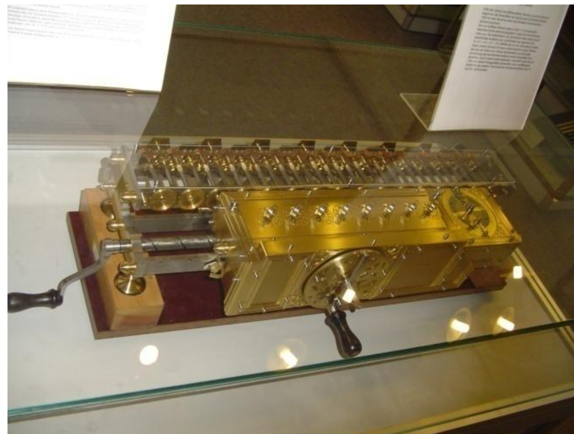


Dibujos enigmáticos de Leonardo da Vinci, la máquina de Schickard y la Pascaline: primeros ejemplos de la aplicación mecánica al cálculo

Muchas máquinas que aparecieron tenían nombres que ahora son poco comunes, tales como la *Máquina de Leibniz* (*Stepped Reckoner* o *Staffelwalze*) , el *Aritmómetro* (Charles Xavier Thomas, 1820), el *Comptómetro* (Dorr E. Felt, 1887), el *Sumador* (*Addiator,* por Troncet, 1889) o la *Curta* (1948). Máquinas como las antes descritas, sin ser en sí antepasados directos, pavimentaron el camino para el nacimiento de la idea del computador moderno y del microprocesador.

El Aritmómetro, el Comptómetro, el Addiator (sumador y sustractor) y la Curta.

Muchos consideran como el más arcaico antecesor de un sistema procesador al llamado *Motor Analítico* (*Analytical Engine*) de Charles Babbage, descrito por primera vez en 1837 y cuyo desarrollo se llevó a cabo hasta 1871 cuando fallece el inventor[9]. Sin embargo, no se pueden dejar de mencionar antes a dos contribuciones muy interesantes. Primeramente, la contribución de Gottfried Wilhelm Leibniz (co-inventor del cálculo junto a Newton)[10].

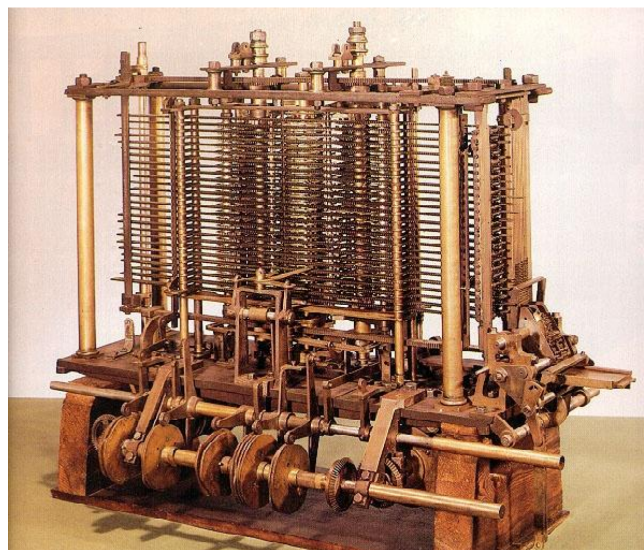


Su ***Staffelwalze*** presentaba una innovación: en vez de usar los clásicos engranajes, él usó unos tambores con canales de escala variable (lo que demuestra que siempre ***es posible encontrar otra forma de realizar las cosas***); pero, quizás su mayor aporte sea ser el primero en abogar por el uso del sistema binario (fundamental en los microprocesadores actuales), a pesar que las máquinas calculadoras de su época usaban el sistema decimal e inclusive su propia máquina era así. La segunda contribución viene

9 **http://en.wikipedia.org/wiki/Analytical_engine**
10 **http://www.computersciencelab.com/ComputerHistory/History.htm, Part 2**

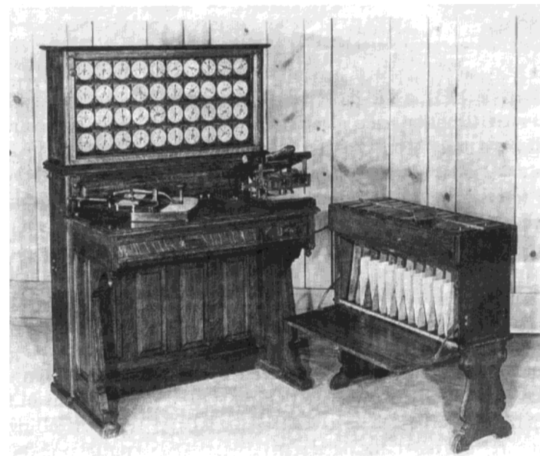
de Joseph Marie Jacquard, quien en 1801 inventó un telar autómata que realizaba sus tejidos mediante el uso de tarjetas perforadas.

Esta innovación ***no relacionada al cálculo*** representó una revolución aplaudida por muchos y atacada por otros. Charles Babbage usó la idea de las tarjetas perforadas (aunque no las ideas de Leibniz, quien murió pobre y olvidado) y la llevó a un nuevo nivel: el de medio de almacenamiento[11]. Inclusive, definió dos elementos importantes: el ***Almacén*** (***Store***, lo que sería en la actualidad la ***Unidad de Memoria***) y el ***Molino*** (***Mill***, lo que sería en la actualidad la ***Unidad Central*** de ***Procesamiento*** o ***CPU***) como las partes principales de su futura máquina. Inclusive, tenía algo que diferenciaba a su máquina de las máquinas calculadoras de su época: instrucciones condicionales, con las cuales los resultados pueden variar de acuerdo a las condiciones existentes.



[11] **Ídem a referencia 11**

Sin embargo, su máquina nunca se completó. Primero, por problemas técnicos: la habilidad para realizar los muy precisos engranajes requeridos no existía. Segundo: un constante cambio de diseño. Por último, falta de fondos: el propio gobierno británico no quiso inmiscuirse en este proyecto debido al antecedente del ***Motor Diferencial*** del mismo inventor. Por su parte, las tarjetas perforadas siguieron su propia evolución, siendo Herman Hollerith quien a finales de 1880 se definió como el inventor de la grabación de datos en un medio que puede ser luego leído por una máquina[12]. El llamado ***Escritorio de Hollerith*** (***Hollerith Desk***) fue la máquina con la cual se pudo computar el censo norteamericano de 1890.



El Hollerith Desk de 1890

Los antecesores directos de los sistemas procesadores actuales aparecen recién casi a mediados del siglo XX. La Segunda Guerra Mundial fue uno de los alicientes más grandes para el desarrollo de complejas máquinas que pudieran computar a altas velocidades. Los años previos a ésta, durante su duración y los años posteriores por las mismas consecuencias del conflicto, permitieron desarrollos que a la larga devendría en la velocidad de cambio tecnológico que alcanzamos en la actualidad.

El primer avance reconocido es el de Konrad Zuse (1910 - 1995) y sus series de máquinas computadoras (tomando el concepto original) de serie Z[13]. **Z1,** su primera máquina (1936), era mecánica (aunque manejada por electricidad), era programable en

---

[12] **Ídem a referencia 1**

[13] **http://en.wikipedia.org/wiki/Konrad_Zuse**

forma limitada y leía sus instrucciones desde tarjetas perforadas. **Z3** (1941) es considerada la primera computadora programable automática y leía sus programas desde una cinta perforada; aunque no tenía saltos condicionales. Estaba basada en relays (era electromecánica). Usaba un sistema binario sencillo, tenía palabras de memoria de 22 bits y una memoria de 32 palabras. **Z22** (1955) fue la primera computadora con memoria basada en almacenamiento magnético. Adicionalmente, cabe recalcar que fue Zuse quien diseñó entre 1943 y 1945 el primer lenguaje de programación de alto nivel: el ***Plankalkül***[14].

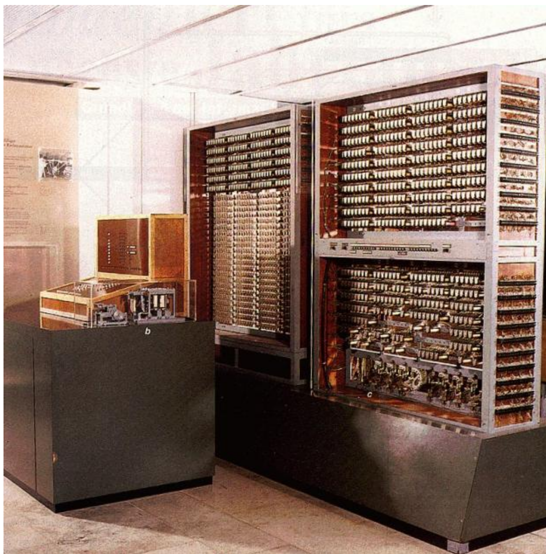


Fig. 1.

Ein Beispiel aus der Schachtheorie

Als Beispiel sei kurz auf die Schachtheorie eingegangen. Zunächst ist der Aufbau der auftretenden Angabenarten interessant.

$S0$	Ja-Nein-Wert
$S1 \cdot n$	n-stellige Folge von Ja-Nein-Werten

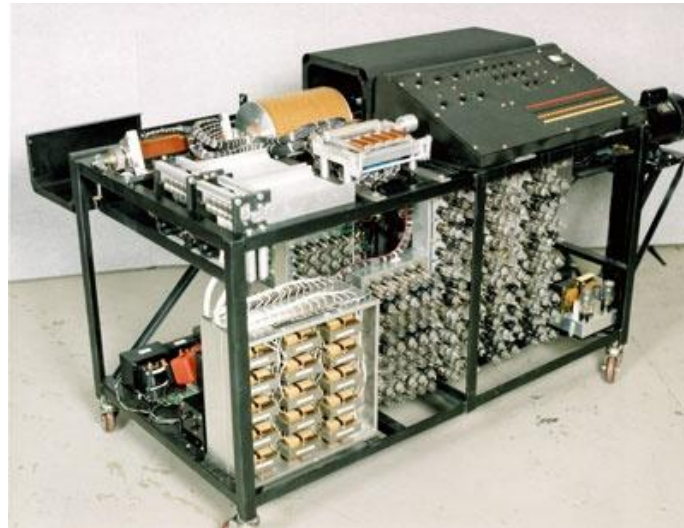
$A1$	$S1 \cdot 3$	= Koordinate
$A2$	$2 \times A1$	= Punkt (z. B.: L00, 00L entspricht Punkt e2 in üblicher Darstellung)
$A3$	$\begin{pmatrix} S1 \cdot 4 \\ B3 \end{pmatrix}$	= Besetzt-Angabe (z. B.: 00L0, Weißer König)
$A4$	$(A2, A3)$	= Punkt-besetzt-Angabe (z. B.: L00, 00L; 00L0 „Punkt e2 mit weißem König besetzt")
$A5$	$64 \times A3$	= Feldbesetzung: $C5$ Anfangslage (Aufzählung der Besetzung der 64 Punkte in fester Reihenfolge)
$A6$	$64 \times A4$	= Feldbesetzung mit Punktangabe, C6 Anfangslage
$A7$	$12 \times S1 \cdot 4$	= Anzahlliste der Steine; $C7$ Anfangslage (Gibt an, wieviel Steine von jeder Sorte auf dem Feld sind, z. B. für Bewertungsrechnungen wichtig).
$A9$	$(A5, S0, S1 \cdot 4, A2)$	= Spielsituation; $C9$ Anfangssituation (Feldbesetzung [$A5$]; Angabe, ob Weiß oder Schwarz am Zuge [$S0$]; Angaben über Rochade-Möglichkeiten [4 Ja-Nein-Werte] Angabe der Punkte mit den Möglichkeiten, „en passant" zu schlagen).
$A10$	$(A6, S0, S1 \cdot 4, A2)$	= Spielsituation mit Punktangabe; $C10$ Anfangslage
$A11$	$(A2, A2, S0)$	= Zugangabe (zwei Punktangaben, gesetzt von ... nach ... Ein Ja-Nein-Wert „Es wird geschlagen").

Una Zuse Z3 y un ejemplo del lenguaje *Plankalkül*

Se ha tratado de poner en forma independiente los esfuerzos de Konrad Zuse ya que mucho de su trabajo fue desconocido hasta tiempos relativamente recientes por las circunstancias de la Guerra: fue alemán.

[14] **http://en.wikipedia.org/wiki/Plankalkül**

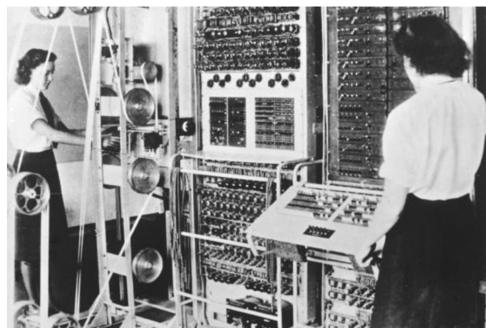
Adicionalmente a Zuse, hay varios hitos en el desarrollo de los sistemas procesadores, cada uno con contribuciones que al final fueron usadas en los diseños posteriores. La ***ABC*** (***Atanasoff - Berry Computer***) es considerada el primer dispositivo digital de cómputo, concebida en 1937 y probada en 1942. No era programable y servía para resolver ecuaciones lineales. Como innovaciones están el uso de procesamiento en paralelo y el uso de memoria capacitiva regenerativa.

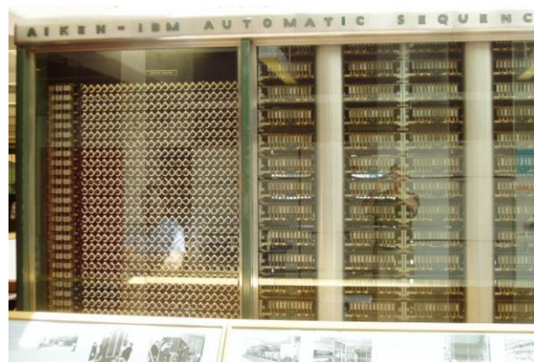


Reconstrucción de la ABC.

La serie de computadores *Colossus* (1944) eran máquinas usadas por los descifradores de códigos ingleses en la Segunda Guerra Mundial. *Colossus* fue el primer dispositivo computador programable que usó dispositivos electrónicos (tubos al vacío). El control se realizaba mediante cables de conexión e interruptores. También fue el primer dispositivo en usar registros de desplazamiento y arreglos sistólicos (*Systolic Arrays*)[15]. La IBM *ASCC* (*Automatic Sequence Controlled Calculator*), también conocida como la *Harvard Mark I* (1944), fue un sistema de tipo electromecánico que no necesitaba de operación humana una vez activada. Se le considera el primer sistema computador totalmente automático completado. Era controlada por una cinta perforada de 24 canales, aunque sin saltos condicionales; sin embargo, podía realizar lazos de programación uniendo los extremos de su cinta de programa (realizado lazos físicos). La llamada *Arquitectura Harvard* nació de éste desarrollo.

[15] **http://en.wikipedia.org/wiki/Systolic_array**





Una de las pocas fotografías de *Colossus* y una sección de la ASCC

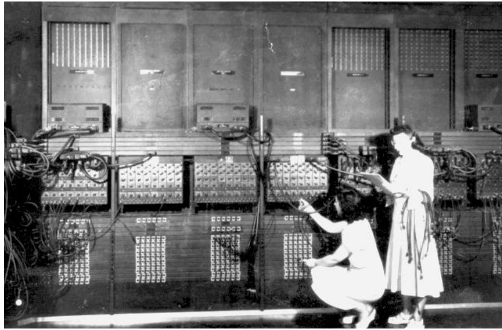
ENIAC (*Electronic Numerical Integrator And Computer,* 1946) es considerado el primer sistema computador de propósito general. Una característica a resaltar es que era modular: tenía diversos paneles que realizaban funciones distintas. Su programa era controlado mediante cables de conexión e interruptores. Tenía como entrada una lectora de tarjetas perforadas y como salida una perforadora de tarjetas (ambas propietarias de IBM). Un modelo mejorado (la *Modified ENIAC*, 1948) usaba una *ROM* primitiva adicional, desarrollada a través de Tablas de Función, donde se almacenaban su programa. Dato interesante: sus registros realizaban operaciones aritméticas decimales, en vez de binarias. La *SSEM* (*Small-Scale Experimental Machine,* 1948), llamada *Manchester SSEM* o *Baby*, no fue pensada como una computadora práctica (sólo restaba y negaba): fue un banco de pruebas para los llamados *Tubos de Williams*[16], una forma electrónica de almacenar datos binarios en un tubo de rayos catódicos. A pesar de ser sólo un banco de pruebas, fue el primer dispositivo operativo en tener todos los elementos esenciales de un computador moderno[17]. Usaba palabras de 32 bits. Su diseño fue tan exitoso que en sirvió de base para la *Manchester Mark I* en 1949, la cual fue la primera en usar registros indexados; además de usar una memoria de tambor magnético. Este sistema se convirtió en el prototipo de la *Ferranti Mark I*[18] (1951), el primer computador multipropósito comercial.

---

[16] http://en.wikipedia.org/wiki/Williams_tubes

[17] http://en.wikipedia.org/wiki/Manchester_Small-Scale_Experimental_Machine

[18] http://en.wikipedia.org/wiki/Ferranti_Mark_I



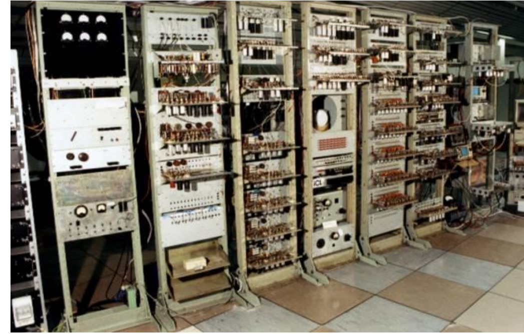
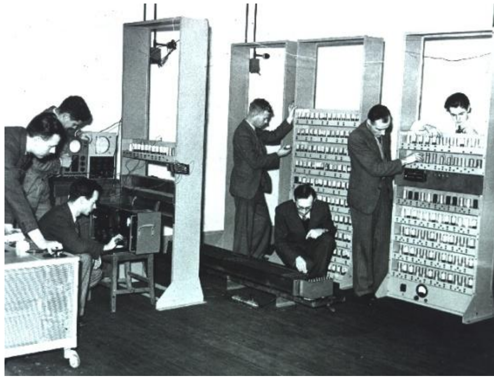


Foto de *ENIAC*, sección de la *SSEM* y consola de la *Ferranti Mark I*

La *ESDAC* (*Electronic Delay Storage Automatic Calculator,* 1949) es considerada el primer computador electrónico práctico. Usaba una memoria de *Línea de Retardo de Mercurio*[19] (un tipo primitivo de memoria serial) para almacenar su programación. Como se puede observar, la idea del cálculo siguió presente como la función fundamental para la cual estos sistemas fueron concebidos. A pesar de ello, algunos vieron otras aplicaciones no relacionadas al cálculo. La *CSIRAC* (*Council for Scientific and Industrial Research Automatic Computer,* 1949) fue la primera en tocar música digital; uno de los primeros juegos de computadora, un juego de ajedrez, fue implementado para la *Ferranti Mark I* (1951). En el transcurso del tiempo y del desarrollo de éstos sistemas, nuevas tecnologías aparecieron, tales como el transistor (1954)[20] y luego el advenimiento del circuito integrado (1958)[21].

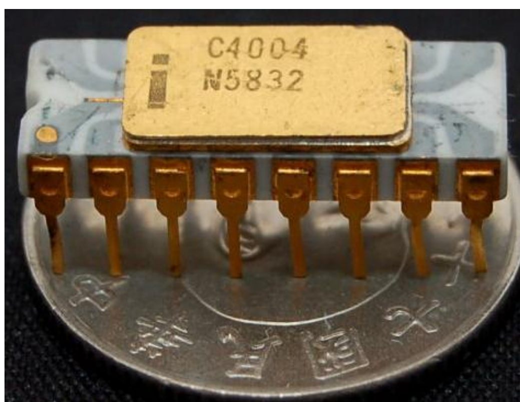
---

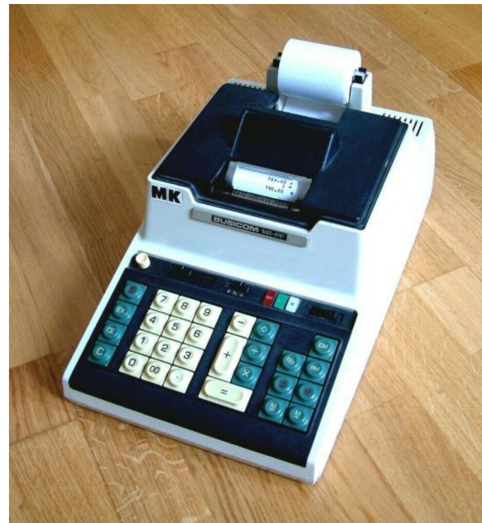
19 **http://en.wikipedia.org/wiki/Delay_line_memory**
20 **http://en.wikipedia.org/wiki/Transistor**
21 **http://en.wikipedia.org/wiki/Integrated_circuit**



La *ESDAC* y la *CSIRAC*: ejemplos de los primeros “cerebros artificiales”

El primer sistema microprocesador fue el *Intel 4004* (1971). El proyecto fue iniciado ante un requerimiento de la compañía japonesa *Business Computer Corporation* (*Busicom*)[22] para un chipset de 12 circuitos integrados especializados para su nueva calculadora electrónica *Busicom 141 PF*[23]. En vez de realizar el chipset, al considerarlo muy complejo de realizar con la tecnología que disponía Intel, se acogió la propuesta de un grupo de ingenieros: Marcian Edward Hoff, Federico Faggin, Stanley Mazor y Masatoshi Shima, la cual daría como resultado el nacimiento del chipset *MCS-4*, es decir, la familia de circuitos del microprocesador *Intel 4004*.





Un ejemplo del microprocesador *4004* y el equipo que lo usó: la *Busicom 141 PF*

22 http://en.wikipedia.org/wiki/Intel_4004
23 http://www.intel.com/museum/archives/4004.htm

Este se convirtió en la práctica en el primer microprocesador de uso general disponible comercialmente disponible y empezó la carrera que posteriormente llevó al desarrollo de otros microprocesadores y de otras compañías tales como *ZiLOG, AMD*, *Fairchild Semiconductors*, *Hewlett-Packard Company*, entre otros. Igualmente, se acuñaron términos como *VLSI* (*Very Large Scale Integration*), *RISC* (*Reduced Instruction Set Computer*), *VLIW* (*Very Long Instruction Word*), entre otros términos. Últimamente, nuevos términos tales como *Multi-core* han aparecido; y en el futuro, se espera que muchos términos, tecnologías y conceptos nuevos aparezcan.

Esta reseña pone en evidencia cómo ha sido el desarrollo de los sistemas procesadores y cuál fue su verdadero origen: resolver problemas de cálculo, problemas originalmente realizados por seres humanos y que la misma necesidad determinó que se busque una solución más rápida y fiable. En el ínterin, se fue nutriendo de muchos adelantos (no necesariamente relacionados al cálculo); y cuando los sistemas empezaron a mostrar capacidades más poderosas, nuevas aplicaciones fueron proyectadas y desarrolladas. Desde el entretenimiento hasta la industria, la medicina, y prácticamente en todos los campos del desarrollo humano, los sistemas microprocesadores se han diversificado siendo uno de los pilares del desarrollo que en la actualidad la sociedad goza.

Esto no quiere decir que debemos olvidar las lecciones del pasado. Siempre se puede ver una forma distinta de hacer las cosas. Siempre se puede buscar una solución sin seguir necesariamente los lineamientos existentes.

1.2 La sincronización y algunos problemas

En la actualidad, la gran mayoría de los sistemas procesadores son de carácter síncrono; es decir, sus operaciones internas son controladas mediante un tren de pulsos externo (la denominada *Señal de Reloj*). La elección de un sistema síncrono se debe principalmente por las ventajas que el propio diseño permite. Una de las razones para optar por el desarrollo de un sistema síncrono radica en su relativa comodidad al momento del diseño: tiende a eliminar factores desconocidos (o de incertidumbre) durante el proceso de diseño y permite concentrarse en el correcto funcionamiento lógico del mismo. Inclusive, entradas de naturaleza asíncrona como las interrupciones tienden a ser tratadas de sincronizar lo más rápido posible para que los problemas de tiempo estén

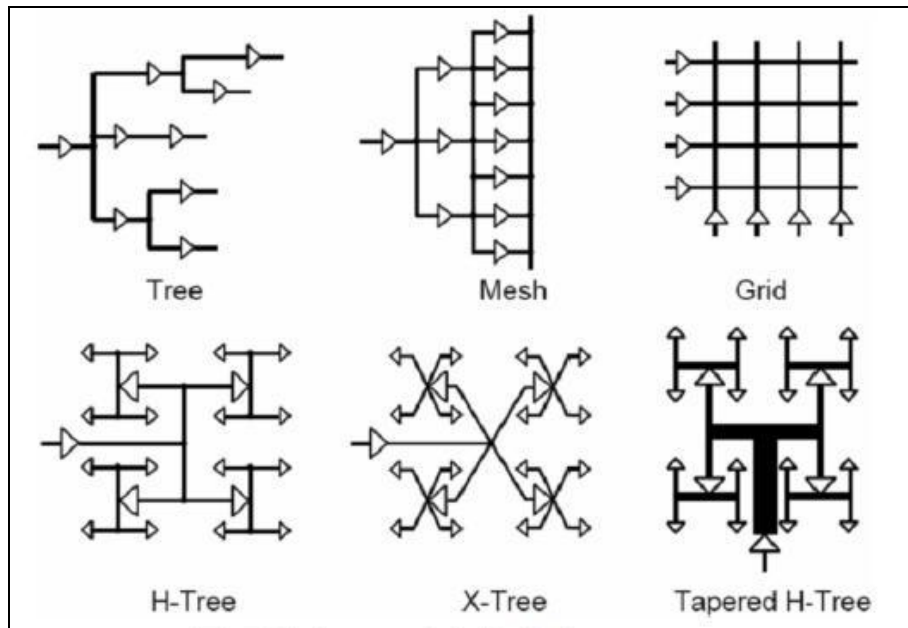
focalizados en pequeñas zonas del dispositivo[24]. Sin embargo, los sistemas síncronos presentan algunos inconvenientes.

Un inconveniente de tener una mayor velocidad es el hecho de realizar muchos cambios de estados, lo cual se traduce en una mayor potencia a disipar. Hay que considerar que la corriente consumida aumenta con respecto a la velocidad, ya que en los sistemas *CMOS* la mayoría de potencia es disipada durante la conmutación (sin referirnos a las disipaciones con la tecnología *TTL*). Y esto no sólo se circunscribe al proceso de conmutación, ya que el sólo hecho de tratar de distribuir una única señal de reloj de alta frecuencia a través de todo el sistema puede acarrear que se tenga un 40% de la disipación total sólo en éste menester[25]. En aplicaciones que son sensibles en cuanto al consumo de potencia, existen algunas técnicas para disminuir el consumo usando el manejo del *Clock* mediante compuertas (*Clock Gating*) en secciones que no lo usen. Sin embargo, en el esquema general síncrono esto crea problemas pues adiciona complicaciones como desfases si es que no se maneja correctamente.

Otro problema que se ha tenido es el propio hecho de la complejidad del sistema, la cual exige una correcta distribución del *Clock*. Una de las ventajas de la miniaturización ha sido la posibilidad de realizar una integración de diferentes subsistemas propietarios (bloques *IP*, *Intellectual Property*, para formar un *SoC*, *System-on-Chip*), lo cual ha generado más de un problema práctico. Uno de ellos es justamente la correcta distribución de la señal del reloj, para evitar que existan desfases que puedan acarrear problemas. Esto sin considerar que la frecuencia de reloj tiende a aumentar cuando el número de dispositivos integrados aumenta. El sincronizar correctamente muchos bloques que pueden realizar incluso tareas distintas (por ejemplo, sistemas multinúcleo o *Multicore*) a muy altas velocidades resulta ser una tarea compleja[26].

---

[24] **J.Nurmi (ed), Processor Design: System-on-Chip Computing for ASIC's and FPGA, p. 367**
[25] **J.Nurmi(ed), Processor Design: System-on-Chip Computing for ASIC's and FPGA, p. 368**
[26] **J.Nurmi(ed), Processor Design: System-on-Chip Computing for ASIC's and FPGA, p. 368-369**



Ejemplo de distribuciones de *Clock* en los sistemas actuales.

También hay que considerar que existen ciertas aplicaciones en los campos de las comunicaciones en las cuales la *Compatibilidad Electromagnética* (*Electromagnetic Compatibility*) es crucial. En un sistema síncrono, los picos de corriente suelen ser correlativas al reloj. Esto genera picos de frecuencia constante que producen emisiones de radiofrecuencia concentrada en determinadas armónicas del reloj, las cuales generan interferencia electromagnética; este hecho debe ser considerado al momento del diseño para evitar justamente fallas de funcionamiento[27].

Uno de los hechos más resaltables con respecto a la industria electrónica sea su avance tecnológico avasallador. Cada cierto tiempo el mundo se ve sorprendido por la aparición de nuevos y más rápidos sistemas, en tamaños cada vez más pequeños. Esto está de acuerdo con la predicción de 1965 de G.E Moore (llamada *Ley de Moore*), la cual (revisada en 1975) plantea que cada 2 años el número de transistores en un circuito integrado se duplicaría[28], lo cual se puede corroborar; a pesar que la mayoría de referencias indiquen tradicionalmente que Moore planteó 18 meses para tal fin[29]. Esta

---

[27] **J.Nurmi(ed), Processor Design: System-on-Chip Computing for ASIC's and FPGA, p. 369**
[28] **Excerpts from A Conversation with Gordon Moore: Moore's Law, ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf**
[29] **historia.pdf, p. 56-57, profesora Silvia Mª. del Pino Gordo, Universidad Complutense Madrid, Facultad de Informática: http://www.fdi.ucm.es/profesor/sdelpino/ETC/historia.pdf**

integración a gran escala (conocida como *VLSI*), o Ultra Gran Escala (*ULSI*), es conocida comúnmente como miniaturización. El estado actual de la misma ha llegado al nivel de poder fabricarse componentes cuyo tamaño está alrededor de los 45 nm; inclusive, ya se habla de 32 nm y de 1900 millones de transistores en un solo circuito integrado[30]. Sin embargo, también se habla de un problema: se está llegando al límite de la miniaturización, ya que al continuar con la reducción del tamaño de los elementos, se alcanzará el tamaño de moléculas y átomos. Cuando los tamaños de los elementos se acercan al nivel de los átomos, las leyes físicas cambian de seguir la Física Clásica a depender de los enunciados de las leyes mecánicas cuánticas de la *Nanofísica*[31].

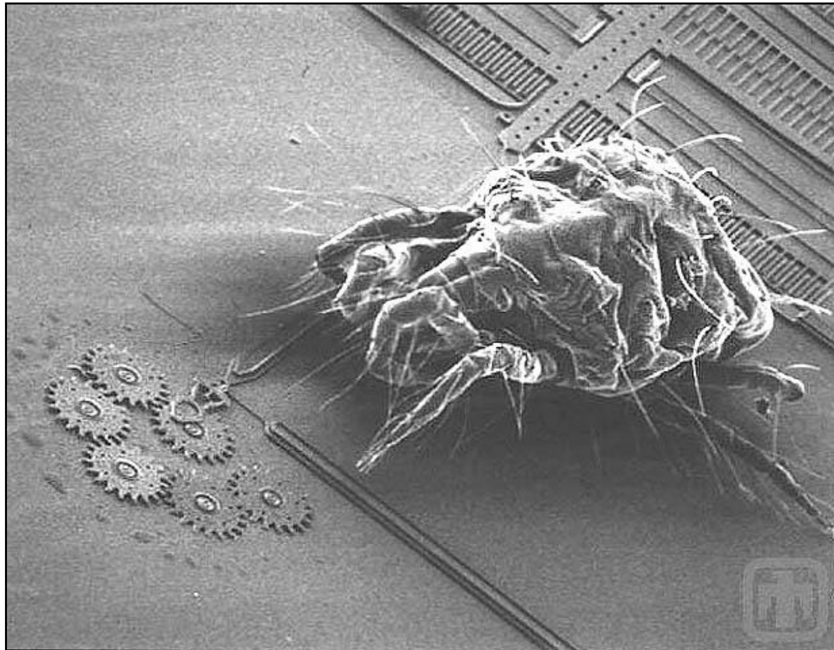
1.3 La velocidad como ideal.

Hay que resaltar un detalle ya observado al exponer brevemente el desarrollo de los sistemas procesadores, el cual es sin lugar a dudas su idea original: sistemas concebidos para ayudar a los usuarios en realizar operaciones matemáticas referidas al cálculo. Si bien es cierto, suelen enfocarse a resolver problemas relacionados a elementos físicos (como las áreas de terrenos o la velocidad de desplazamiento), los cálculos matemáticos son fundamentalmente abstractos[32]. Por ejemplo: $I = \frac{V}{R}$ es una representación abstracta de un hecho real. Esto quiere decir que si sustituimos sus valores, es posible encontrar una correspondencia con los fenómenos que se dan en el mundo real. Definitivamente, existen muchos fenómenos reales cuyas ecuaciones son en definitiva más elaboradas; para ellas, el simple cálculo mental promedio deja de ser una alternativa y ya son necesarios elementos adicionales. Desde simplemente lápiz y papel, pasando por calculadoras y computadores, al realizar éstos cálculos se puede por propia experiencia deducir qué sería lo más ideal: resolverlos sin demora.

Lo anteriormente expuesto permite aseverar que el cálculo matemático ideal es instantáneo. Aunque en el mundo real, el cálculo de cualquier enunciado matemático toma su tiempo, mientras más rápido se realicen los cálculos más rápidos se obtendrán los resultados deseados (considerando el ideal de cálculo sin errores o con desviaciones dentro de parámetros aceptables). Sin embargo, ésta valiosa capacidad choca con el

[30] http://www.tauzero.org/2009/06/computacion-cuantica-un-nuevo-paradigma/
[31] Edward L. Wolf, Nanophysics and Nanotechnology: An Introduction to Modern concepts in Nanoscience
[32] http://es.wikipedia.org/wiki/Matem%C3%A1ticas

mundo real. A diferencia de las casi instantáneas necesidades del cálculo abstracto, el mundo real está sujeto a magnitudes de todo tipo. Éstas pueden cambiar a velocidades distintas, inclusive a velocidades altas; sin embargo, tienen un límite físico en el cual al existir cambios tan pequeños en una determinada magnitud, éstos se vuelven en la práctica, despreciables. Cabe anotar que en ésta idea expuesta, no se tomaron en cuenta aplicaciones en las cuales, al ser de nivel microscópico, pequeños cambios son apreciables. Un ejemplo de ello es en el caso de los *micromotores* que usan la tecnología *MEMS* (*Micro Electro Mechanical Systems*)[33], los cuales inclusive a esos niveles tienen que lidiar con magnitudes físicas reales. Por más pequeños que estos sistemas sean, cambios mucho más pequeños de los que ellos manejan se vuelven también despreciables.



Mecanismo de tecnología MEMS en comparación a un ácaro. Incluso a éste nivel, existe una magnitud mínima de respuesta al movimiento del sistema.

---

[33] **http://eelinux.ee.usm.maine.edu/courses//ele498/Lecture%20Material/MEMS-Overview.PDF**

1.4 Algunas consideraciones sobre los Actuadores

En forma general los actuadores son dispositivos mecánicos que toman energía originada ya sea por un gas, la electricidad o un líquido, y la convierte en movimiento; conformando una subdivisión de los *Transductores*[34]. De hecho, muchos actuadores, como el caso de las electroválvulas, controlan magnitudes tales como flujo en forma indirecta. Un hecho a resaltar es que en la actualidad, en forma electrónica es posible controlar un gran abanico de actuadores para prácticamente cualquier actividad; y si no existiera uno adecuado, es indudable que se trabaja en diseñar uno. Una cosa hay que aclarar: en la presente tesis no se desea hacer un análisis sobre los actuadores, tan sólo se dejarán presentes algunas consideraciones que se tomaron en cuenta para el tema.

Ejemplos de actuadores. Los mostrados son usados con fines educativos, de prototipos y hobby.

En líneas generales, todo actuador es posible de ser controlado a partir de un sistema electrónico digital. Aunque los sistemas digitales suelen manipular bajos voltajes y

[34] http://en.wikipedia.org/wiki/Actuator

bajas corrientes, no es de extrañar que pueden manejar elementos de alta potencia (con voltajes o corrientes muchas veces superiores a los que pudiera soportar); esto debido a su uso de elementos de interface que sirven como sistemas de manejo de los actuadores, los cuales suelen recibir en forma general el apelativo de *Driver*. Haciendo un análisis generalizado, se puede distinguir 3 características propias comunes entre todos:

1. Activación / Desactivación.
   Independientemente de sus elementos constituyentes o las magnitudes que se controlen con un actuador, éste sólo puede permanecer activado o desactivado durante un determinado lapso de tiempo; siendo este tiempo variable dependiendo de la aplicación (inclusive, algunos están casi permanentemente activados). Que esté activado no implica necesariamente que un actuador esté actuando. Un motor DC pequeño, cuando se le alimenta (su activación típica), empieza a rotar a una determinada velocidad. Un servomotor de hobby puede ser alimentado (activado); y sin embargo, sin una señal de control, su actuación puede ser errática o nula.

2. Sentido de Actuación.
   Baje este enunciado se ha tratado de generalizar la gama de acciones que pueden realizar los actuadores. Pueden manipular una válvula, rotar, realizar movimientos lineales, entre otros. Toda esta gama de acciones (que en la presente tesis serán denominadas *Actuaciones*), suelen realizarse en los casos más comunes en un solo sentido a la vez: rotación a derecha o izquierda, apertura o cierre de una válvula, elongación o encogimiento, entre otros. En algunos casos, la activación o desactivación no implica que el elemento podrá o no accionarse en un sentido determinado dependiendo de la influencia de fuerzas externas. Por ejemplo, un motor DC rotará en un sentido dependiendo de la polarización; cuando no está polarizado, es posible moverlo aplicando una fuerza al eje.

3. Magnitud de Actuación
   Aquí se hace referencia a cuánta acción desarrollará el actuador. Por ejemplo, cuántas vueltas girará, cuánto abrirá o cerrará una válvula, cuántos grados se moverá, cuánto se extenderá o contraerá, entre otras acciones posibles de realizar. No siempre es posible considerar absolutamente proporcional la acción de un actuador. Variaciones en su fabricación o elementos externos a él pueden influenciar para que su accionar no sea necesariamente proporcional. Es por ello que siempre

que se desee un control adecuado será necesario un lazo de realimentación con información sobre la magnitud que se desee controlar y así hacer las correcciones necesarias.

Un elemento actuador que interactúa con el mundo real debe operarse considerando la acción que va a realizar y el tipo al que pertenece. Dependiendo si son neumáticos, hidráulicos, eléctricos, etc., cada tipo de actuador tiene características conocidas y condiciones de funcionamiento[35]. E invariablemente, los cambios que puedan efectuar en su accionar se llevarán a cabo en un determinado tiempo. Si bien es cierto que una rapidez en el actuar es siempre deseable, existe un límite temporal para actuar. Si se es más rápido que éste límite es probable que no se puedan tener los resultados deseados. Por ejemplo, si a un sistema de posición se le indica que se coloque en un determinado Por ejemplo, si a un sistema de posición se le indica que se coloque en un ángulo (una magnitud con respecto a un punto de origen) y posteriormente se le indican más posiciones a velocidades típicas de los sistemas microprocesadores, es muy probable que el sistema sólo pueda reaccionar a la última posición; físicamente tiene que tener un tiempo para que los sistemas realmente puedan variar una posición por otra. Para lidiar con este inconveniente, se han tenido que implementar técnicas que permitan al sistema poder interactuar con estos dispositivos lentos. Entre otros métodos, los más conocidos son aquellos que usan bucles de retardo(los más comunes en las áreas educativas) y aquellos que utilizan las interrupciones para poder interactuar con ellos.

Para la presente tesis, no se ha planteado de antemano interactuar con elementos muy complicados. Es por ello que se ha considerado un acercamiento desde el punto de vista de la industria del entretenimiento. En ella, no necesariamente se usan elementos de mucha potencia o que requieran manejos muy complicados (dependiendo de la aplicación). Teniendo en cuenta que se plantea un sistema de carácter experimental al ser una arquitectura particular, se usarán actuadores sencillos tales como motores *DC,* motores de paso, entre otros elementos. Sin embargo, se tendrá en cuenta a futuro su posibilidad de uso para actuadores de prestaciones más elaboradas.

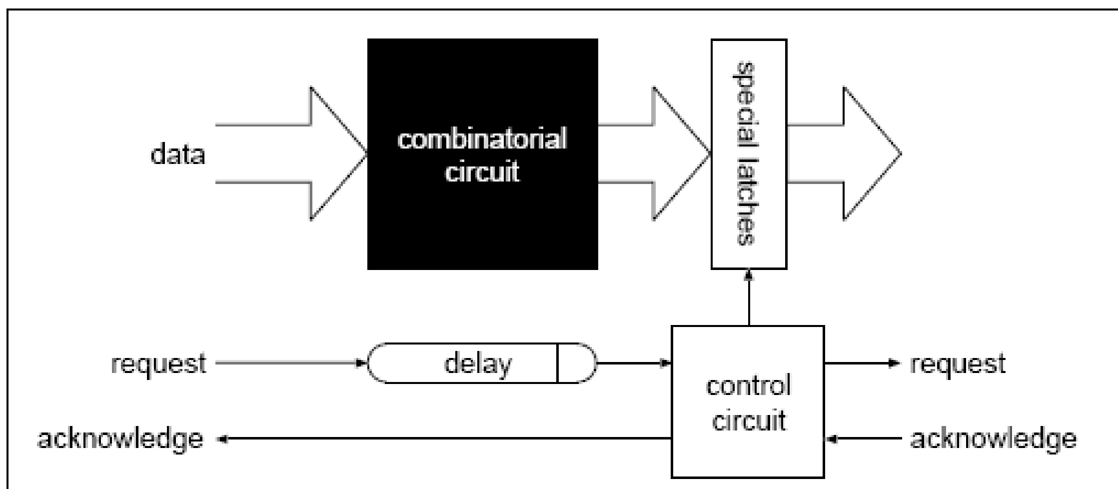
---

[35] **Referencia: http://www.zerobots.net/manuales/Teoria_Actuadores.pdf**

1.5 Una solución en boga: el *Self-Timed*

Lo opuesto a síncrono es asíncrono; sin embargo, esto no basta para considerarlo una posibilidad. ¿Por qué se debe escoger un esquema asíncrono? Las mismas consideraciones antes mencionadas en los sistemas síncronos pueden usarse como base para defender un esquema asíncrono.

Un sistema asíncrono funciona sólo cuando se le necesite, lo cual quiere decir que solo las secciones que necesiten funcionar consumirán energía. En un sistema asíncrono, los subsistemas funcionarán a las velocidades que cada uno necesite; aunque un módulo lento pueda hacer más lento al sistema, no hará que el sistema falle. Un correcto diseño de sistemas más rápidos con sistemas que sean lentos pueden hacer que globalmente el tiempo de retardo esté dentro de los parámetros establecidos[36]. También hay que considerar que no es necesario diseñar una forma de alinear el funcionamiento de interfaces de distinta velocidad con la frecuencia principal; y que inclusive se puede obviar el costo y el diseño de distribución del reloj evitando retardos y asimetrías en la señal (*Clock Skew*).

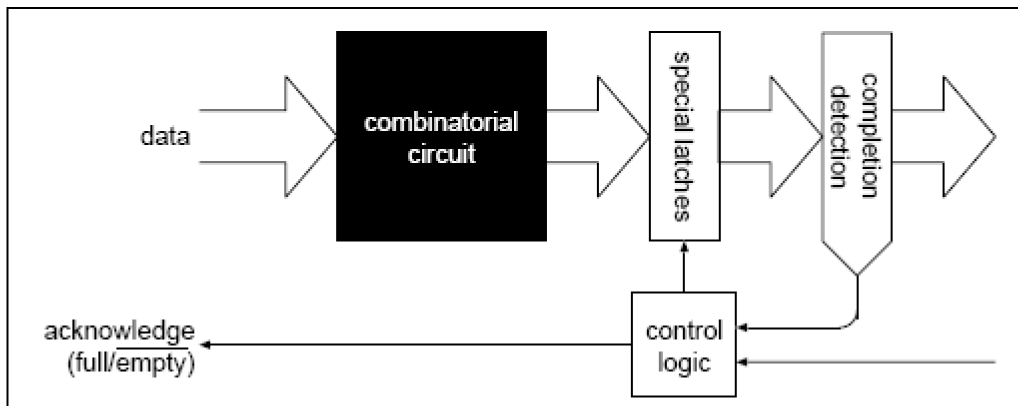


Esquema de un proceso *Self-Timed* de tipo *Delay Based*

La solución asimétrica planteada fue la de los sistemas denominados ***Self-Timed*** o ***Autotemporizados***. A diferencia de los sistemas convencionales síncronos, donde es necesaria la presencia de un reloj global, estos sistemas presentan funcionamientos

[36] **Ídem a referencia 26**

dependientes sólo de la sección y de la acción que se lleve a cabo. Las demás secciones se encuentran a la expectativa, funcionando sólo si las condiciones que esperan para ello se cumplen[37]. En forma general éstos sistemas presentan los procesos de transmisión de datos y procesamiento de los mismos de una manera particular: estos procesos son negociados localmente mediante dos líneas de protocolo: ***Request*** (***Requerimiento***) y ***Acknowledge*** (***Reconocimiento***)[38]. Se suele usar las estructuras de tipo ***Delay Based*** (***Basadas en Retardo***) y las de tipo ***Completion Based*** (***Basadas en Finalización***) para realizar los procesos de éstos sistemas[39]



Esquema de un proceso *Self-Timed* de tipo *Completion Based*

Sin embargo, se ha descubierto que el sistema propuesto en esta tesis no es compatible con un sistema *Self-Timed*. Su propia arquitectura es particular y no es compatible con las ideas de un sistema asíncrono como el presentado. La presente información general mostrada en esta sección sirve sólo para demostrar primeramente que ya existen investigaciones referidas a sistemas no sincronizados (con sistemas ya disponibles), y en segundo lugar, para poner énfasis que el sistema propuesto en la presente tesis no es del tipo *Self-Timed.* No se ha podido encontrar alguna referencia a ningún sistema parecido al propuesto, por lo que se puede inferir que éste es un sistema particular.

[37] **http://www.cs.virginia.edu/~robins/Computing_Without_Clocks.pdf, p. 65 -66**
[38] **http://arantxa.ii.uam.es/~mcts/papers/6)%2059_ortega.pdf, p. 1**
[39] **http://www.cl.cam.ac.uk/teaching/Lectures/compwoclocks/selftimed.4up.pdf, p. 6-7**

El sistema propuesto de hecho será un sistema de tipo asíncrono, aunque en realidad usa una sección síncrona como parte de uno de los módulos; esto en sí ha puesto en duda si es que realmente se le debe considerar asíncrono o un híbrido entre un sistema síncrono y asíncrono. Sin embargo, la denominación de *Computador Híbrido* (*Hybrid Computer*)[40] y la de *Núcleo Híbrido* (*Hibrid-Core*)[41] ya existen y ninguna de ellas hace referencia a lo que se ha planteado: la primera habla de sistemas que muestran características de computadores digitales y computadores análogos; y la segunda hace referencia a procesadores con instrucciones de uso específico para acelerar su desempeño (parte de la llamada *Computación Heterogénea* o *Heterogeneus Computing*).

Bajo un análisis de su funcionamiento, la parte síncrona es sólo un subsistema con una tarea específica, mientras que el verdadero funcionamiento tendrá tiempos propios e independientes de tipo configurable por instrucción. Es por ello que ha prevalecido la noción de estar desarrollando una propuesta asíncrona; y así se consignará.

1.6 Algunas conclusiones sobre lo expuesto

- Primero:

  Los sistemas procesadores de la actualidad deben su razón de ser al cálculo. La pugna por la alta velocidad fue inicialmente alimentada por la necesidad de calcular más rápido y el hecho de que sea una máquina fue justamente encontrar la forma de lidiar con la fatiga del cálculo manual.

- Segundo:

  Los sistemas precursores usaron adelantos particulares (no necesariamente relacionados al cálculo) para su construcción y para mejorar su funcionamiento.

- Tercero:

  Los sistemas síncronos, base de la mayoría de sistemas actuales, presentan algunos inconvenientes que han sido tratados de solucionar mediante el desarrollo de alternativas asíncronas.

[40] http://en.wikipedia.org/wiki/Hybrid_computer
[41] http://en.wikipedia.org/wiki/Hybrid-core_computing

- Cuarto:

  El sistema propuesto en la presente tesis no es de tipo *Self-Timed* a pesar que se trata de un sistema definitivamente asíncrono. Se trata de una arquitectura particular no vista en ninguna referencia previa. Es por ello que se espera que resulte novedosa.

- Quinto:

  Se orientará los modelos prácticos al uso de elementos y actuadores de tipo sencillo como una forma de probar y depurar la arquitectura y su desempeño.

CAPÍTULO II
EL SISTEMA AUTÓMATA SECUENCIAL POR TIEMPOS INDEPENDIENTES
S.A.S.T.I.

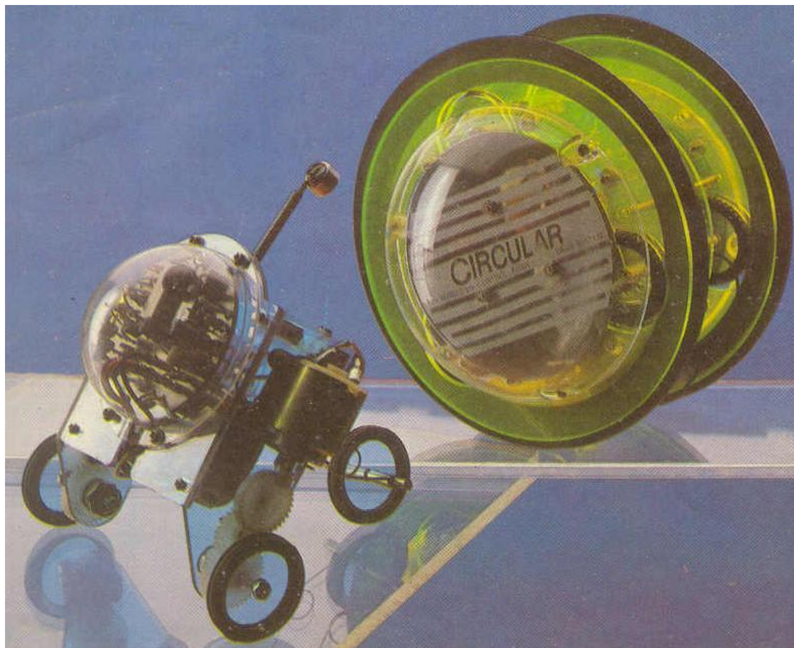
El nombre que se le ha dado a la presente tesis, *Sistema Autómata Secuencial por Tiempos Independientes* (*S.A.S.T.I*) es en realidad la última denominación por la cual se puede identificar a éste dispositivo. Inicialmente, no se consideró un nombre específico para ésta arquitectura. Su origen se basa en la *Observación* y *Experimentación*; ambas motivadas por la *Necesidad*. El plantear este tipo de arquitectura se originó en un hecho acaecido hace ya bastante tiempo, antes de inclusive haber ingresado a la Universidad.

Se le denomina *Autómata Secuencial* por el hecho de realizar sus acciones basadas en los datos que existen en su memoria de sistema. Esto quiere decir que sólo ejecutará un programa que encuentre en su memoria en forma repetitiva, a menos que se le indique lo contrario. En forma inicial, su capacidad de decisión será mínima; o simplemente, no existirá, dependiendo del modelo. El sistema puede ser reprogramado si así se considera. De hecho, originalmente todos los sistemas deberían tener un subsistema programador que pudiera ejecutar la alteración de los datos de memoria; pero, por consideraciones operativas, se usarán bloques de *PseudoROM* como memoria de sistema. Estos bloques son ni más ni menos que el equivalente a la ROM primitiva del computador *Modified ENIAC*.

La razón de incluir la idea de *Tiempos Independientes* está relacionada con el hecho que la temporización de las instrucciones es arbitraria por programa. Dependiendo de qué subsistema de temporización se use y cuál es la frecuencia de referencia del sistema (*Reloj de Temporización,* R_T), el valor de temporización puede variar. Así, la temporización resultante tiene en realidad tres variables que deberán ser consideradas para obtener una temporización adecuada. Este punto será abordado en la sección correspondiente para cada sistema que se expondrá.

2.1 Nacimiento de la idea

La llamada *Idea Primaria* se esboza a partir de un experimento acontecido en la década de los 90's. Dicho experimento se basó en la necesidad de emular un modelo electrónico aparecido en la revista mexicana *Mi Computer*. El artículo detallaba una serie de juguetes electrónicos, los cuales eran presentados como *"Los Movits son "kits" de robot, económicos y pre programados, de gran demanda en el floreciente mercado de este tipo de productos…"*[42].



Dos de los MOVITS de Kaho Musen: el Piper Mouse y el Circular.

El dispositivo objetivo del experimento fue el Memocon Crawler, descrito como "… una máquina que se acerca mucho a nuestra definición de lo que es un robot…" y que "… está alimentado por dos motores eléctricos CD. El sistema de guía se compone de un teclado conectado al Crawler mediante un cable plano… Estas instrucciones se almacenan en un chip de RAM estática con una memoria que contiene 256 bytes,

---

[42] **mi computer. CURSO PRACTICO del computador personal el micro y el minicomputador., Fascículo 79, pág. 1564**

constando cada byte de cuatro bits."[43]. Las razones fundamentales para empezar con el experimento fueron principalmente:

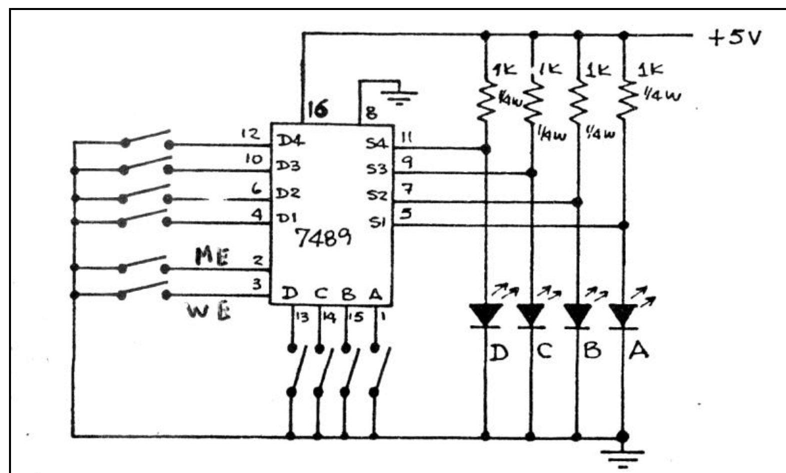
Dificultad para la obtención de un sistema microprocesador por factor de costos.

Dificultad para la adquisición de un sistema programador para memorias adecuado.

Cabe indicar que al momento de la realización del experimento, sólo se disponía como experiencia previa un libro y circuitos de experimentación de la editorial Cekit[44]. Una revisión de las características generales a obtener y la experiencia previa en el manejo de circuitos conllevaron a la hipótesis por la cual era posible buscar otra forma de realizar un sistema que pudiera hacer lo mismo que se indicaba en el artículo sobre el Memocon Crawler; pero, usando componentes conocidos.

2.2 La Idea Primaria

Este concepto se desarrolla a partir de un análisis de posibilidades operativas realizado sobre un circuito demostrativo de la RAM 7489; una memoria de acceso aleatorio ya descontinuada, pero de la cual aún se tenían algunos chips. El circuito experimental está presente en un folleto sobre circuitos digitales[45] y cuyo diagrama es el que se muestra a continuación:



43 mi computer. CURSO PRACTICO del computador personal el micro y el minicomputador., Fascículo 79, pág. 1565

44 Curso básico de Microprocesadores Nivel 1, Editorial Cekit

45 CIRCUITOS DIGITALES, García Villareal, Jorge R. y García Villareal, Juan R., Consorcio Integrado de Electrónica e Informática (CIEI), págs. 84 a 87.

Diagrama original del Ejercicio para el manejo de la RAM 7489.

El análisis del ejercicio permitió sacar algunas conclusiones que se tomaron en cuenta para la arquitectura propuesta. Primeramente, se requiere un *Dato* estable durante un *lapso de tiempo*, el cual debe coincidir con una *Dirección* estable. Mediante la manipulación de los bits de *Control*, es posible guardar los datos en la posición de memoria que se desee. Si una *Dirección* estable es implementada y se manipulan los bits de *Control* adecuados, la salida de la memoria será el dato almacenado en la posición apuntada por los bits de *Dirección.* Si los bits de *Dirección* varían, la salida variará. Bajo éstos lineamientos y considerando la relativa facilidad para la grabación manual de datos, se pudo empezar el análisis general. La idea del *Contador de Programa*, llevado a un extremo literal, condujo al uso de un circuito contador para poder mostrar en forma secuencial los datos almacenados en las posiciones de memoria de una forma autónoma. Considerando que el reloj del contador es estable en frecuencia y ciclo de trabajo, se dedujo que obtener una temporización efectiva de cualquier dato mostrado era posible mediante la propia temporización del reloj. Así, durante el *Ciclo de Ejecución* (estando el contador en una posición estable), el sistema mostraría un dato específico; y cuando entrara al *Ciclo de Traspase,* el pulso de reloj realizaría el cambio de posición de memoria para entrar a un *Ciclo de Ejecución* nuevo. En forma real, el cambio de dato se ejecutaría casi instantáneamente (segundos de ejecución versus nanosegundos de retardo digital promedio), lo cual garantizaría un desempeño aceptable. Para lidiar con temporizaciones menores, se usaría un reloj más rápido; mientras que para temporizaciones mayores se consideró suficiente la duplicidad de instrucción para tener una sumatoria de temporizaciones.

2.3 Desarrollo del sistema prototipo

El plantear la posibilidad de realizar casi las mismas funciones del *Memocon Crawler* con pocos elementos y sin la necesidad de un microprocesador generó una serie de interrogantes: ¿qué se sabe de lo que se quiere armar? ¿qué elementos se mencionan que tiene? ¿Qué hay disponible para realizarlo? ¿Es posible de hacerlo más barato?

En primer lugar, se definió lo siguiente: se requiere un sistema que sea *programado*; es decir, que pueda *almacenar una secuencia de instrucciones* y que luego sea capaz de

reproducirlas e interpretarlas de una manera específica para obtener como resultado las acciones deseadas. Esto permitía inferir que se necesitaba un elemento para programar el sistema (*Programador*), un elemento para guardar los datos (*Memoria*), un elemento para controlar la memoria y realizar las acciones (*Procesador*) y un elemento que permita el manejo de los motores o luces que se desean (*Driver de Potencia*).

El sistema del *Memocon Crawler* no era completamente detallado en la publicación[46]; siendo lo único claro que contaba con una RAM estática de 256 posiciones de memoria de 4 bits pos posición. Dato interesante para notar es que sí mencionaba su accionar: movimiento de 2 motores, encendido de LEDs y encendido de un zumbador (*Buzzer*). Como dato adicional: usaba 2 motores. Así, estos datos permitían deducir que los 4 bits podían corresponder a un control *On/Off* de cada uno de los 4 elementos. El hecho de la necesidad de 2 bits para realizar el control de giro fue algo que desconcertó al principio (inicialmente no se consideró usar un *Decodificador de Instrucciones*); sin embargo, al ser simplemente un experimento, se optó por obviar este inconveniente y se construyó un prototipo considerando un control *On/Off* sencillo.

Memocon Crawler de Kaho Musen

Lo primero a considerar en forma práctica fue un modo de enviarle información de una manera sencilla y entendible. Al menos, la representación general debería ser lo más reconocible posible y la forma de ingresar los datos debía de ser de alguna manera bastante accesible. Es por ello que se planteó el uso de un circuito decodificador de teclado, un grupo de registros y decodificadores de display de ánodo común para

[46] **Ídem a referencia 43**

obtener una visualización de datos más sencilla. Se tomó como modelo la estructura de la tarjeta programadora del módulo de Cekit[47].

Al no requerir de un Decodificador de Instrucciones, el único inconveniente en el sistema Procesador fue el desarrollar un aislamiento de la memoria con el sistema a controlar. Eso principalmente debido a una precaución de proteger de alguna manera a la memoria y adicionalmente obtener un aislamiento del driver de potencia con miras de obtener una forma de depuración de programa (ACP, Aislamiento Circuital por Programación) sin tener que gastar energía en los sistemas motrices y luminosos. Es por ello que se planteó una salida con Buffers Triestados (Tristate) y adicionalmente una etapa de opto acopladores. En ese momento, esta solución fue planteada como suficiente para poder desarrollar el prototipo. Para que los datos sean accesibles de forma secuencial, se optó por usar un contador binario de 4 bits. Se trató de elegir uno que sea relativamente sencillo de manejar y que pudiera tener algunas características que pudieran ser usadas en un futuro. El Driver planteado fue del tipo más sencillo. Un par de luces podían ser reemplazadas por dos LEDs, lo cual permitía simplemente realizar una red resistiva sencilla para controlar. Para los motores, se planteó el uso de dos transistores de mediana potencia trabajando en corte y saturación para obtener el control On/Off requerido. Si bien es cierto, no se podía obtener una marcha para atrás, aún podía avanzar y dar giros; aunque no sobre su propio eje.

Todas estas consideraciones fueron analizadas, tratando de obtener un sistema que pudiera ser armado con relativa facilidad y con un costo que fuera lo suficientemente económico como para realizar experimentos con ellos y que pudieran ser fácilmente adquirible y fácilmente reemplazables. Todo este proceso como resultado el primer sistema funcional del concepto: el móvil tipo tortuga Amauta MCh-3A1.

2.4 Tortuga Programable Amauta MCh-3A1

La Tortuga Amauta MCh-3A1 es el primer prototipo de sistema programable que usa la arquitectura de la Idea Primaria. Obvia el uso de microprocesadores o microcontroladores y se concentra únicamente en realizar un manejo directo de los actuadores que tiene; en se caso, un par de motores DC y un par de LED.

[47] **Ídem a referencia 44**

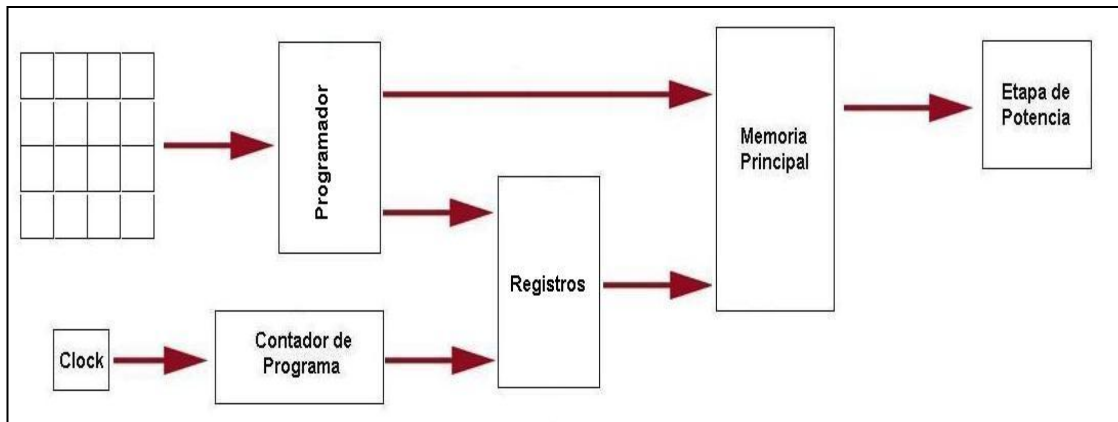
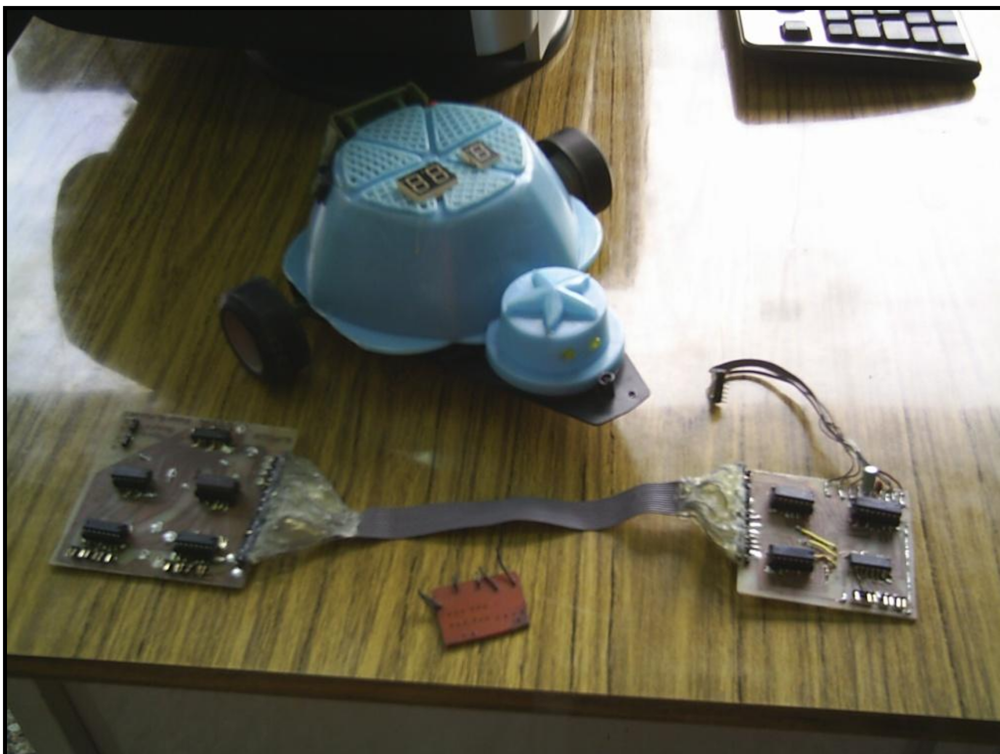


Diagrama de Bloques general del Amauta MCh-3A1

El ***Programador*** está formado por un decodificador de teclado 74C922, registros 74LS173, decodificadores de display 74LS47, Displays de ánodo común, interruptores y elementos adicionales. Esta etapa se encarga de detectar la tecla activada y enviar dicho dato en forma de un valor equivalente en formato binario. El valor es traspasado en forma secuencial a los registros y a su vez éstos envían los datos a decodificadores de display para su observación. Debido a que no se pudo encontrar decodificadores de binario a hexadecimal, se ha usado un decodificador genérico que muestra los caracteres ***A, B, C, D, E, F*** como símbolos especiales. El sistema permite separar los datos de ***Instrucción*** y ***Datos*** para que éstos sean grabados en la ***RAM 7489***.



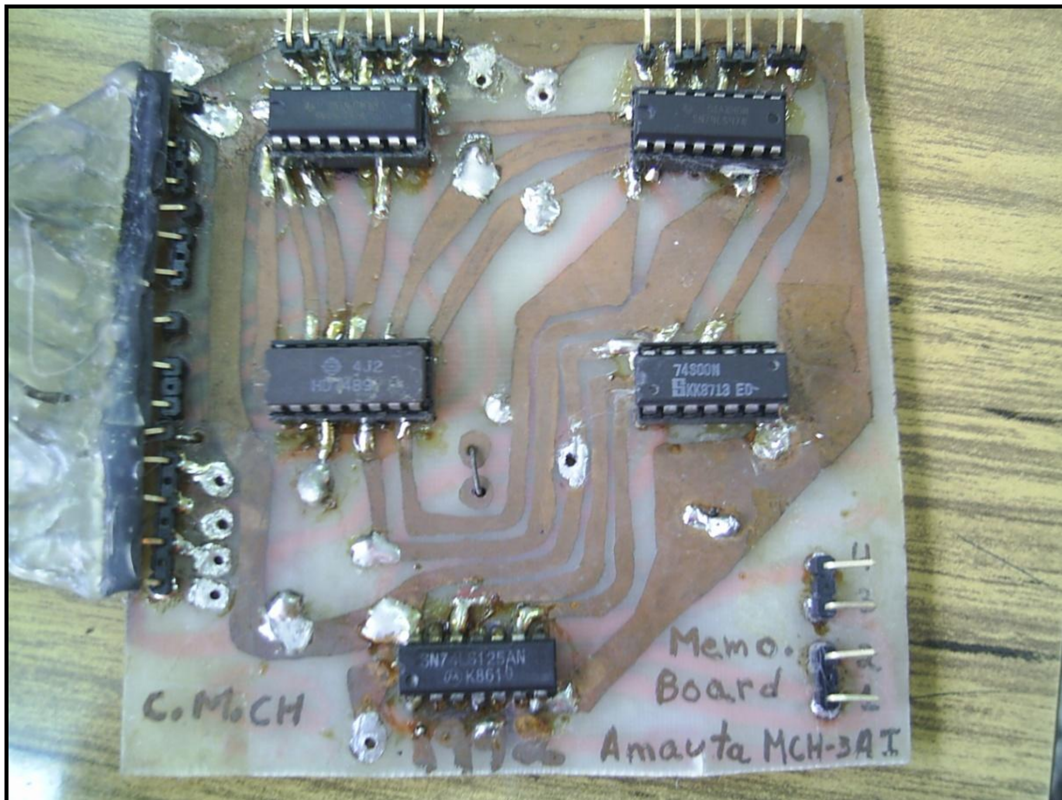
Vista general de las partes recuperadas del *Amauta MCh-3A1*. Aún no se ha podido recuperar la tarjeta de reloj. Se aprecia la falta de algunos componentes.

El *Procesador* está constituido en base a una memoria 7489, un contador 74193, decodificadores de display, buffers triestados, un oscilador LM555 y elementos adicionales. Durante el estado de *Programación*, el contador y oscilador permanecen inactivos, siendo los datos de posición y los datos del sistema manejados íntegramente por el usuario. La salida triestado permite aislar la etapa de Potencia de la etapa de Procesador y evitar activar los actuadores durante la fase de programación. La etapa de *Potencia* está conformada por salidas opto acopladas que manejan 2 *LED* y 2 motores *DC*. En realidad, no había una necesidad de opto acopladores para los LED; sin embargo, éste fue el diseño original. Los motores recibían los datos directamente a transistores de mediana potencia que activaban los motores.

Originalmente, era posible programar 16 eventos de 1 segundo. Las instrucciones eran las siguientes:

Instrucción (Hexadecimal - Binario)	**Acción**
00 – 0000	Parada, Luces apagadas
01 – 0001	Sólo Motor 1 encendido.
02 – 0010	Sólo Motor 2 encendido
03 – 0011	Ambos motores encendidos, sin luces
04 – 0100	Sólo LED 1 encendido
05 – 0101	LED 1 encendido, Motor 1 encendido
06 – 0110	LED 1 encendido, Motor 2 encendido
07 – 0111	LED 1 encendido, ambos motores encendidos
08 – 1000	Sólo LED 2 encendido
09 – 1001	LED 2 encendido, Motor 1 encendido
10 – 1010	LED 2 encendido, Motor 2 encendido
11 – 1011	LED 2 encendido, ambos motores encendidos
12 – 1100	Ambos LED encendidos
13 – 1101	Ambos LED encendidos, Motor 1 encendido
14 – 1110	Ambos LED encendido, Motor 2 encendido
15 – 1111	Ambos LED encendido, ambos motores encendidos

El circuito fue implementado en placa impresa de fibra de vidrio de doble cara. Todo el conjunto fue montado sobre una base con motores reciclado de un viejo juguete y como caparazón se usó una antigua carcasa de un juguete rascaplaya (una tortuga). Su alimentación inicialmente fue mediante pilas y una batería pequeña; posteriormente, se cambió a una fuente externa mediante un conector.



Tarjeta Original del Amauta MCh-3A1. Se observa el acabado *amateur* y la fecha: 1992.

En la parte central izquierda: se aprecia la memoria TTL 7489 de 16 posiciones de 4 bits.

Se realizaron varias pruebas y se confirmó que el sistema funcionaba de manera satisfactoria. Por desgracia, la necesidad de reciclar los proyectos para realizar nuevas cosas conllevó al desmantelamiento total de éste sistema; siendo necesario una labor de búsqueda y reconstrucción. La experimentación con éste sistema influyó en el desarrollo de un segundo modelo de mayor capacidad.

2.5 Proyecto Amauta-II

El Proyecto *Amauta – II* fue pensado bajo la presunción de mejorar la capacidad del sistema añadiendo una memoria *RAM 2114* para obtener una mayor capacidad de memoria. Originalmente se pensó colocarlo en una estructura tipo tanque, la cual poseía una tapa que permitía descubrir un pequeño teclado, LEDs indicadores y Displays.

El sistema estaba planeado para tener las mismas características de instrucciones; y en general, la misma arquitectura. Durante el proceso de implementación, ante problemas para obtener los elementos mecánicos adecuados para el sistema de tracción y dirección, se realizó un alto en el ensamblado. Durante el tiempo en el cual se realizó la búsqueda de partes adecuadas, se aprovechó para realizar una evaluación adicional del sistema diseñado con la finalidad de depurarlo. Fue en éstas circunstancias que las desventajas del sistema tal y como hasta ese momento se había planteado, salieron a relucir. Este factor fue decisivo para dejar detenido cualquier tipo de desarrollo para reevaluar la situación y determinar las necesidades para poder construir un sistema eficiente; a la larga, esto conllevó a que el Amauta – II nunca fuera terminado.

2.6 El concepto de Instrucción-Dato (I-D)

La idea de realizar una estructura de tipo *Instrucción-Dato* (*I-D*) es principalmente para tratar de ahorrar espacio de memoria realizando algunas acciones al mismo tiempo. Los primeros sistemas microprocesadores (como el *Intel 8085*) solían leer de la memoria una instrucción y su dato correspondiente en forma secuencial. Poder realizar las dos cosas al mismo tiempo, se veía prometedor. Esto, sin embargo, no aparenta ser algo nuevo. Los microprocesadores modernos han evolucionado y pueden realizar un procesamiento en paralelo de varias instrucciones; así, es posible hablar de la *Very Long Instruction Word* (*VLIW*)[48]. Sin tratar de ahondar en los procesos que éstos realizan para lograr este procesamiento (distinto en cuanto a funcionamiento y fines al sistema propuesto), se pretender describir las razones para usar éste tipo de Instrucción.

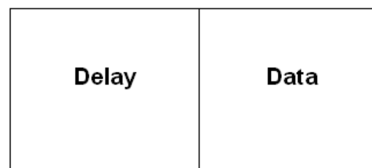
[48] **http://en.wikipedia.org/wiki/VLIW**

2.6.1 Datos e Instrucciones.

En primer lugar, se debe considerar de qué tipo de Instrucción y de qué tipo de dato se está hablando. La *Instrucción* a la cual se hace referencia inicialmente era sólo para realizar un control básico. Principalmente era para *encender o apagar* un actuador; y en forma extendida, variar su *sentido de actuación*. Inicialmente no se había considerado un control usando un valor proporcional a la magnitud de la actuación. Bajo ésta concepción, es fácil suponer que sólo enviando algunos bits que conformen la acción deseada éste nivel sería suficiente.

El *Dato* inicialmente no había sido considerado. Como originalmente la instrucción era de carácter predominante y cualquier temporización era lograda mediante mayor frecuencia y sumatoria de retardos por duplicidad de instrucción, simplemente no había una necesidad para la existencia de un dato. La primera idea real de *Dato* para esta arquitectura fue, como tenía que ser, la relacionada a la temporización. Este dato tomaba la forma de un *Delay* (retardo) que indicaba cuánto tiempo debía permanecer la señal de control (la *Instrucción*) para realizar una determinada acción. Indudablemente, cuando fuera requerido un valor numérico adicional, éste tendría que ocupar un campo particular.

Así, el formato original de la *Palabra de Memoria* está formado por dos campos específicos:



El ***Delay*** estaba relacionado al subsistema de temporización, el cual posteriormente tomó la forma del ***MultiClock*** y del ***Sleeping Clock.*** Aunque cronológicamente ***Sleeping Clock*** antecede en concepto e implementación a ***MultiClock***, la razón por la cual ***MultiClock*** fue el primer tipo de sistema de retardo considerado (***SASTI - I***) se debe a que era el más cercano a la ***Idea Primaria***. Inclusive, originalmente el subsistema de temporización primitivo que se implementó no tenía un nombre específico. El número de bits necesarios para la Palabra de Memoria dependerá del diseño del propio sistema. Estos detalles serán ahondados en las secciones correspondientes.

El desarrollo posterior de toda la idea de ***SASTI*** dio como resultado que el formato final de la Palabra de Memoria estaría formado por tres campos específicos: Datos de Temporización, Instrucción y Datos numéricos. Así, la ***Palabra de Memoria*** definitiva está formada por los campos:

Delay	Instrucción	Dato

2.6.2 Tamaño de Memoria

El tamaño de la memoria necesaria para realizar las acciones deseadas debe ser estudiado con detenimiento. Ya que el Sistema por sí está construido alrededor de una memoria, su tamaño (relacionado a las necesidades de funcionamiento) influirá en todos los subsistemas. Un acercamiento bastante sencillo es pensar en ella como un elemento finito, por lo cual un contador diseñado para sólo recorrer le extensión del programa sería suficiente. Si se desea un sistema con posibilidad de modificaciones en extensión de programa, se requerirá considerar un máximo de extensión con relación a los tiempos máximos que se puedan manejar con el sistema.

Debido a que el sistema va a manejar varios tipos de datos, estos campos son enormes; y dependiendo del tipo de sistema de temporización, el Dato de Temporización podría mayor que los otros campos. Esto implica que la extensión de la *Palabra de Memoria* sería muy grande; aunque no es una *VLIW* en el sentido operativo. Es por ello que una característica importante de todo sistema que utilice esta idea de *Instrucción-Dato* es que la memoria usada tendería a aumentar su número de bits de Palabra y a disminuir las posiciones de memoria usadas. Es decir, la memoria aumenta en *Magnitud de Palabra* y disminuye en *Extensión de Capacidad*. Estos hechos deben de ser considerados en el diseño, ya que una menor extensión de la memoria implica un contador más pequeño; aunque podría implicar una magnitud de palabra de memoria mayor. La mayoría de memorias comerciales que se disponen en la actualidad (de tipo estático para este trabajo) tienden a dar mayor énfasis en la capacidad de la memoria (su extensión) que en la magnitud de la palabra. Típico es un estándar de 8 bits; inclusive, es posible encontrar bloques de 4 bits. En contraposición, es posible encontrar

memorias de varios *kilobytes*, mientras memorias de menores capacidades tienden a desaparecer; esto con referencia a la RAM 7489, actualmente descontinuada.

2.7 El primer SASTI real

El primer sistema *SASTI* implementado se llevó a cabo en el marco del *Concurso de Proyectos 2004 - I* de la *Facultad de Ingeniería Electrónica*. De hecho, el mismo nombre no había sido acuñado, y al momento de implementar el sistema (con el concepto de *I-D)* había dudas de cómo catalogarlo. Debido al hecho de procesar sus instrucciones en forma asíncrona a pesar de usar un R_T síncrono, se optó presentarlo como *Secuenciador Asíncrono Programable*.

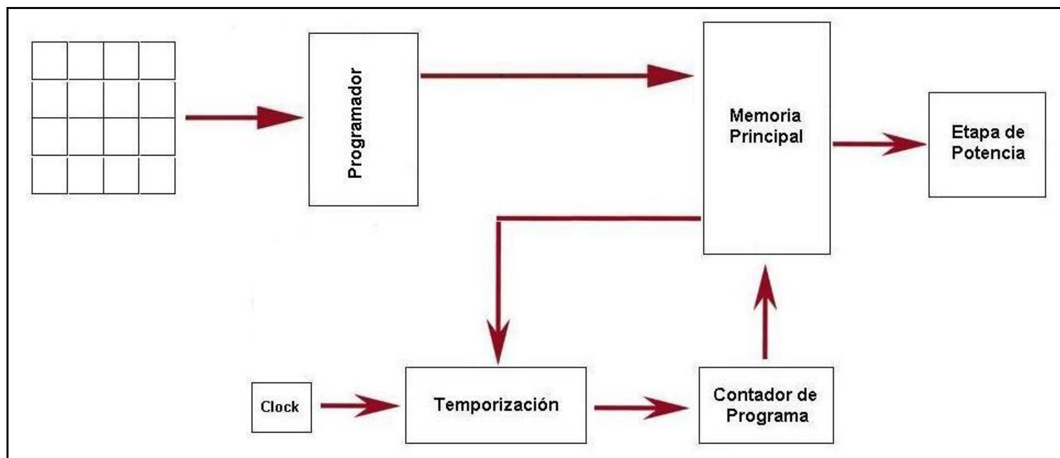
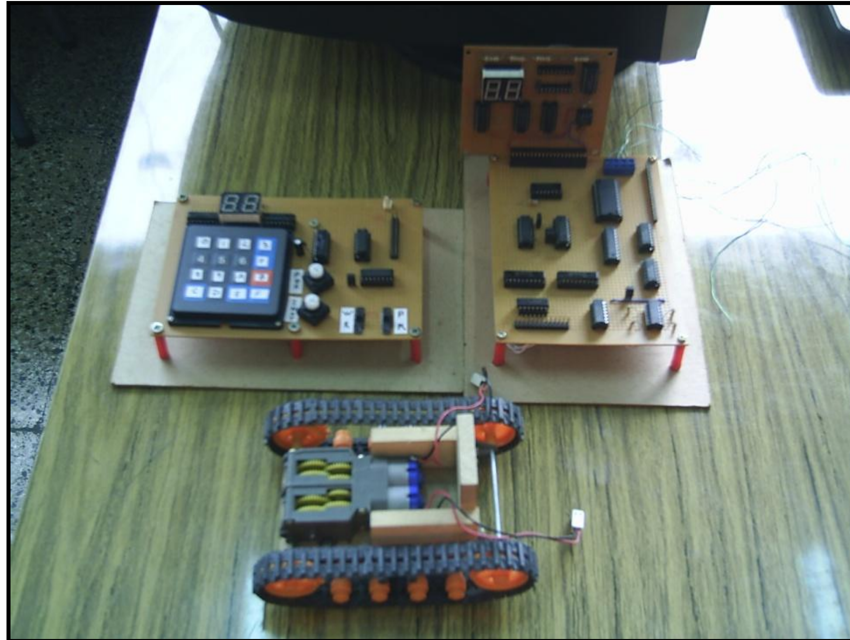


Diagrama de Bloques general del Secuenciador Asíncrono Programable

El sistema presentado estaba basado en un programador parecido al del **MCh-3A1,** un par de ***RAM 2114***, dos contadores 74LS193 (con lo que obtenían 256 posiciones efectivas y cuatro bloques de programa, aunque inicialmente estaba permanentemente seleccionado el bloque 0), un comparador 74LS85 y algunos circuitos adicionales con los cuales se tenía un $\boldsymbol{R_T}$ primitivo de 1 Hz. Como elemento actuador usó un módulo de motores y orugas Tamiya y transistores como driver de potencia. El sistema resultó absolutamente operativo a tal punto que inclusive se pensó que se estaba usando un microcontrolador PIC; cuando se notaba la ausencia de cualquier tipo de procesador convencional, las miradas de incredulidad eran elocuentes. De hecho, en esa ocasión se recibió un diploma de primer puesto: el primero que se recibía por un sistema ***SASTI***.



Vista general de las partes recuperadas del proyecto *Secuenciador Asíncrono Programable*.

Falta el cable principal, un display doble y la tarjeta de potencia para unir al móvil.

Al igual que en el caso de las *Tortugas,* la necesidad de reciclar componentes hizo que dicho sistema ganador fuera canibalizado para otros proyectos y experimentos (personales y de carrera). Una búsqueda en el *Scrapyard* de todos los elementos que originalmente conformaban éste sistema ha permitido ubicar buena parte del conjunto; sin embargo, faltan aún algunos sistemas principales.

2.8 El uso de FPGA y CPLD

La implementación del sistema *SASTI* originalmente se planteó mediante circuitos integrados convencionales: contadores, registros de desplazamiento, comparadores, entre otros (sin mencionar otros elementos adicionales). Sin embargo, la misma construcción planteaba un problema: sería voluminoso. Implicaría muchas líneas de interconexión, una gran área de circuito impreso, muchos elementos y subsistemas a conectar. Esto hace que las probabilidades de fallas debidas a falsos contactos, malas soldaduras, pistas defectuosas, entre otros, sean muy altas. Ya que se esperaba hacer algo nuevo, no era deseable problemas adicionales a simplemente depurar la

arquitectura. Es por ello que se buscó una solución que permitiera tener un sistema confiable en el menor espacio posible y con capacidad de experimentar. Es por ello que se optó por el uso de las *FPGA* y *CPLD*.

2.8.1 Cajas Negras

Una *FPGA* (*Field Programmable Gate Array, Matriz de Compuertas Programables en Campo*) introducidas por Xilinx en 1985[49], es en líneas generales un dispositivo programable por el usuario o diseñador en el cual se puede configurar un determinado diseño digital. Denominados también *LCA* (*Logic Cell Array*), están formados por una matriz bidimensional de bloques configurables que pueden ser interconectados para así configurar un diseño digital; esta posibilidad de ser usada en el campo de aplicación directamente por el usuario es lo que en parte le da su nombre[50]. Por lo general, su configuración suele hacerse mediante un lenguaje descriptor de hardware (tales como *HDL* o *VHDL*); aunque en realidad, en una FPGA se programa los conmutadores para las conexiones entre bloques y adicionando la configuración de los mismos. Suelen ser de carácter volátil: una vez sin alimentación, pierden su configuración. Por ello, requieren de una memoria de configuración externa para almacenar la configuración y poder cargarla al ser energizada.

Un *CPLD* (*Complex Programmable Logic Device*) es un dispositivo lógico programable con una complejidad entre una *PAL* (*Programmable Array Logic, Matriz Lógica Programable*) y una *FPGA*[51]. Popularmente se considera que tiene internamente varios dispositivos lógicos programables. En forma general está conformado por varias unidades programables denominadas *Macro Celdas* (*Macro Cell*), las cuales contienen bloques lógicos especializados y bloques lógicos de tipo *DNF* (*Disjuntive Normal Form*, Forma Normal Disyuntiva)[52]. Esto último implica que sólo se tienen operadores lógicos de tipo AND, OR y NOT, bajo ciertas condiciones; por lo que requiere usar equivalencias lógicas (como las *Leyes de Morgan*) para poder implementar una fórmula lógica. Son de tipo no volátil, aunque el número de regrabaciones depende del modelo.

Lo anteriormente expuesto son sólo lineamientos generales de ambos elementos pensados para implementar la tesis expuesta. Lo más importante que resaltar de ambos

---

[49] **http://www.miky.com.ar/fpga_2004.pdf, p. 2**
[50] **http://en.wikipedia.org/wiki/Fpga**
[51] **http://en.wikipedia.org/wiki/Complex_programmable_logic_device**
[52] **http://en.wikipedia.org/wiki/Disjunctive_normal_form**

es su capacidad de poder ser configuradas como un diseño completo. Se vuelven una especie de *Caja Negra* con las cuales se puede implementar un diseño y depurar su funcionamiento hasta obtener el sistema deseado. Dos hechos a tener en cuenta son los relacionados a su capacidad y su rango de programación. Una *FPGA* suele tener muchos más elementos lógicos y biestables que una FPGA, aunque es volátil; mientras que un *CPLD*, a pesar de tener menos elementos, es no volátil y regrabable un número finito de veces. Ambos presentan una ventaja interesante: sus pines de salida son configurables por programa, lo que permite tener las salidas necesarias en el lado correcto del dispositivo y en el orden que uno elija, lo cual facilita las labores de diseño. Esta flexibilidad y las características permiten sacar dos conclusiones importantes:

Para la experimentación propiamente dicha es necesario el uso de una *FPGA*. Al ser reprogramable un número elevado (por no decir casi infinito) de veces, es ideal para probar arquitecturas, interconexiones y realizar modificaciones a los diseños previos hasta depurar un sistema completo.

Para la implementación de prototipos estables, es necesario el uso de un *CPLD*, el cual deberá ser elegido de acuerdo a la capacidad que se requiera para cada diseño. Con éste elemento, se puede tener en un espacio menor todo un circuito de muchos elementos.

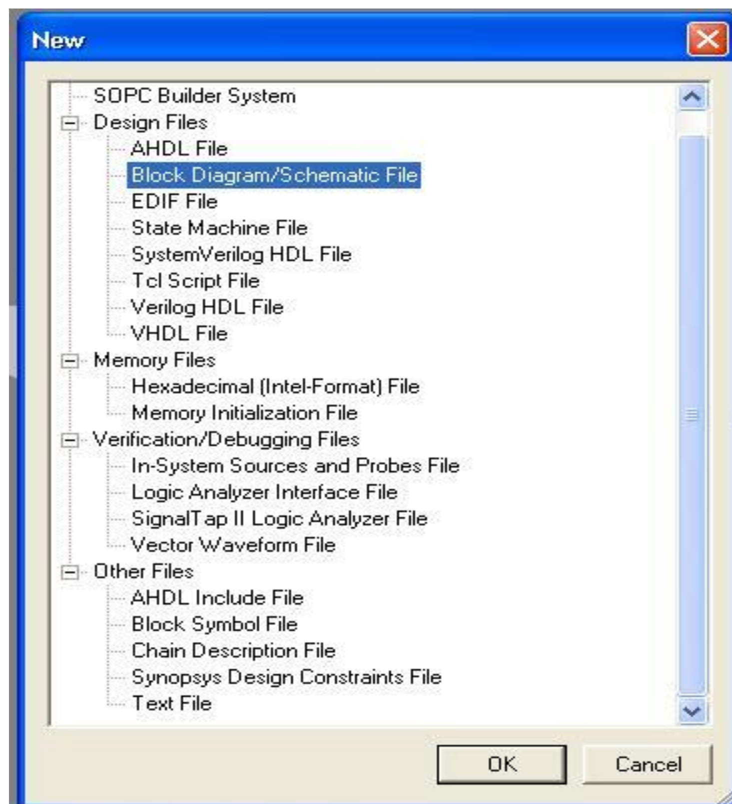
Para la etapa experimental se ha usado el Módulo Educativo *UP2* de la compañía *ALTERA*, junto al software de desarrollo *Max+Plus II V.10.2*© y *Quartus II V.9.0*© de la misma compañía; siendo los programas mencionados específicos para diseño con *FPGA, CPLD* y *ASIC* (*Application-Specific integrated Circuit*), mas no son simuladores de circuitos digitales (tienen herramientas de simulación de diseños pero sólo orientados a su aplicación en sus dispositivos programables). Adicionalmente, se ha contado con el Módulo de Desarrollo *SY-03091* de la compañía *Siphre*. La *FPGA* usada es la *EPF10K70RC240-4*, que tiene unas 70,000 compuertas lógicas. Se usaran como *CPLD* tres tipos: la *EPM7032SLC44, EPM7064SLC84* y *EPM7128SLC84*; las cuales tienen 32, 64 y 128 macroceldas respectivamente.

2.8.2 VHDL y Modo Gráfico

VHDL (*Very High Speed Integrated Circuit Hardware Description Software,* o simplemente *VHSIC Hardware Description Software*) es un lenguaje descriptor de Hardware para circuitos integrados de muy alta velocidad. En líneas generales, implica

que mediante un programa textual es posible configurar un bloque lógico cualquiera (una *Caja Negra* de propósito específico); es bastante flexible y permite realizar desde lógica sencilla hasta bloques tan elaborados como filtros digitales.

En el caso de la presente tesis se optó por seguir un método más tradicional: el uso del modo gráfico de los programas mencionados en el apartado anterior para realizar los módulos necesarios. Una ventaja de un entorno de bloques en vez de un texto complejo es que es posible *visualizar* las conexiones de los mismos y definir variantes que, en un texto plano, es más difícil de dilucidar. Se dice que el ser humano es un ser visual; esto se puede demostrar con facilidad al verificar cómo un entorno gráfico (como el del Sistema Operativo *Windows©)* desplazó al entorno textual (como el *DOS*) en los computadores. No es de extrañar que inclusive el programa *Quartus©* incluya un entorno gráfico y que convierta sus programas de *VHDL* en bloques operativos que pueden ser usados como tales en cualquier diseño. Esto no quiere decir que un lenguaje descriptor sea malo por su naturaleza; sólo se desea enfatizar que la razón de usar el entorno gráfico radica en la flexibilidad para realizar los experimentos.



Menú de Quartus© 9.0 con sus opciones de diseño.

En forma práctica, el uso del ***VHDL*** será para los casos que se necesiten bloques de características especiales, prefiriendo los bloques predeterminados para realizar el diseño general. Específicamente se ha usado para generar bloques contadores especializados y para generar las ***PseudoROM*** necesarias para los experimentos. ¿Por qué usar una ***PseudoROM*** en vez de usar bloques de alto nivel? Se sabe que los programas de diseño antes mencionados tienen como parte de sus librerías unos bloques especializados configurables que permiten tener tanto bloques de RAM como bloques de ROM; siendo así, en apariencia la opción de generar una ROM al estilo de la ***Modified ENIAC*** sería un arcaísmo. Sin embargo, la razón fundamental para ésta opción es muy práctica: los bloques de alto nivel mencionados no pueden ser implementados en las ***CPLD***; por lo tanto, para evitar problemas de diseño, se optó por esta opción. Adicionalmente, es posible configurar un listado visual de las instrucciones completas y adicionarles comentarios, nemotécnicos y valores que permiten entender mejor el trabajo realizado; inclusive, configurando completamente las posiciones de memoria no usadas. Todo esto inclinó la balanza al uso de la ***PseudoROM*** como memoria.

2.9 El diseño de la *PseudoROM*

Debido a que la ***PseudoROM*** es un bloque general y común para todos los tipos de ***SASTI***, se ha preferido detallarlo en esta sección y no como parte de las consideraciones de diseño de los capítulos siguientes.

Lo primero a considerar es el bloque básico a definir. Por comodidad, dicho bloque no contará con pines de control tales como ***CE*** (***Chip Enable***, Habilitador del ***IC***) al considerársele absolutamente dedicada para fines experimentales. Esto permite definir que el bloque tendrá ***A*** pines de ***Dirección*** y ***D*** pines de ***Datos***. Lo segundo a considerar es qué tipo de instrucciones de programación serán útiles. Se sabe que en VHDL es posible implementar bloques de memoria tanto ***RAM*** como ***ROM***[53] ; sin embargo su uso será dejado de lado ya que se usará la idea de una tabla lógica. Esto principalmente por la facilidad de elaboración.

[53] **http://esd.cs.ucr.edu/labs/tutorial/**

Para realizar el diseño primeramente se deben escoger las librerías. Por comodidad, se escoge la ***Librería ieee***, la cual contiene la ***ieee.std_logic_1164.all***, la cual permite tener un "… ***sistema de nueve niveles lógicos de señal, definiciones de tipos junto con sus operaciones aritméticas y relacionales, y funciones de resolución…***"[54] . Así, se declara como:

library ieee;

use ieee.std_logic_1164.all;

El segundo paso es definir la estructura general del sistema, denominado ***Entidad*** (***Entity***). Ésta debe tener un nombre igual al nombre del archivo donde se grabará (procedimiento clásico en programas de ***Altera***©). Como sólo se necesitan entradas y salidas simples, se definen los elementos como ***Port***s (Puertos). Como los puertos son multielementos, deben ser definidos como un vector en lógica estándar (se denomina ***std_logic_vector***). Si suponemos que la ***SeudoROM*** tendrá ***n*** bits para ***Direcciones*** y ***m*** bits pata ***Datos***, la declaración de la entidad queda definida como:

entity [Nombre del Archivo] ***is***

port(I: in std_logic_vector (n ***downto 0);***

O: out std_logic_vector (m ***downto 0)***

);

end [Nombre del Archivo]***;***

Luego de haber declarado los elementos principales, se procede a declarar el funcionamiento del propio dispositivo; es decir, su ***Arquitectura*** (***Architecture***). Esta arquitectura en particular es única y con un proceso establecido; el cual es dependiente de la variable de entrada ***I***. Para la ***SeudoROM*** se ha visto conveniente el uso del comando ***Case***. Éste permite colocar las opciones (en éste caso ***Posiciones de Memoria***) y su correspondiente salida (***Datos*** grabados). Así, es posible tener como comentarios los equivalentes en hexadecimal y cualquier indicación nemotécnica que se desee. Por ende, se puede definir la arquitectura general como:

---

[54] Exposición: ***VHDL. Laboratorio de Arquitectura de Computadoras***

architecture [Nombre de la arquitectura] ***of*** [Nombre del Archivo] ***is***

begin

process (I)

begin

case I is

when [Posición] => ***O*** <= [Dato]***;***

< Se repite tantas veces como posiciones se requieran >

when others => ***O*** <= [Dato por defecto: vacío]***;***

end case;

end process;

end [Nombre de la arquitectura]***;***

La razón para el uso de éste formato para generar la ***PseudoROM*** es arbitrario: se ha tomado en consideración la comodidad del manejo (consideración personal) y le emulación del proceso históricamente usado en uno de los precursores de los computadores (la ***Modified ENIAC***).

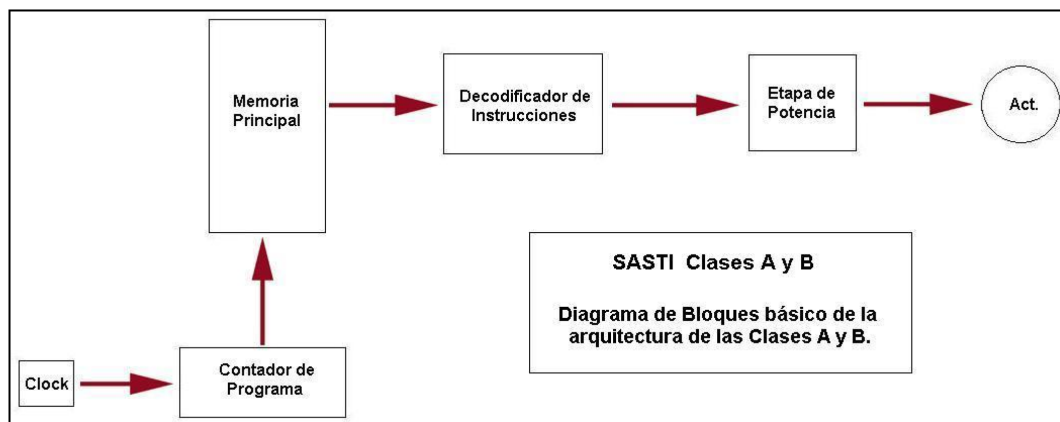
El bloque que éste programa en ***VHDL*** permite generar se ha trabajado para así poder definir un bloque de aspecto más "circuital". Así, se han separado adecuadamente sus entradas y salidas para poder conectarlas directamente si se necesitan.

CAPÍTULO III

EL S.A.S.T.I. TIPO I

El sistema ***S.A.S.T.I.*** de tipo ***I*** es la primera configuración sobre la cual se trabajó para obtener un sistema que sea operativo, modular y escalable. Paradójicamente el sistema tiene como características el ser ***Sencillo*** y ***Complejo***. Es sencillo porque la ***Idea Primaria*** es sencilla y también la idea inicial es de hacer un sistema de tipo sencillo con circuitos comunes: una reminiscencia de la idea original de implementarlo físicamente con circuitos integrados comerciales de tipo ***TTL***. Se vuelve ***Complejo*** al momento mismo de definir los sistemas de temporización y también los esquemas relacionados a las variantes. Al ser el primer tipo sobre el cual se trabaja, todas las dificultades y contratiempos que puedan encontrarse (desde simplemente ajuste lógico o hasta fallas de bloques complejos) recaerán en éste tipo. El ***S.A.S.T.I.*** tipo ***II*** usa bloques ya probados y definidos en el tipo ***I***; por lo que su experimentación es más sencilla. Aun así, en una cierta etapa de la experimentación, recaerá en el tipo ***II*** la mayor parte del trabajo, dejando al tipo ***I*** disfrutar de los bloques alcanzados.

El **SASTI - I** está basado íntegramente en el primer concepto de temporización usado para el funcionamiento de los Sistemas SASTI: el uso del ***Reloj de Sistema*** como medio de temporización general; el reloj es denominado ***Reloj de Temporización*** ó R_T.



Como es un sistema experimental no se ha considerado el efecto de un sistema programable por usuario en forma directa, tal como fue el prototipo original. Ya que la idea es probar en forma práctica el concepto general, se ha considerado el uso de un

bloque que haga la labor de una memoria ROM (llamada en el diseño ***PseudoROM***), la cual se encargará de proporcionar los datos necesarios para realizar las acciones que se deseen.

El uso de una capacidad limitada en este tipo inicial ha sido arbitraria: el sistema está capacitado para tener tanta capacidad de memoria como se desee según la aplicación a realizarse. Simplemente se tiene que modificar los bloques necesarios (***PseudoROM***, Contador y Latch de salida). El uso de un ***Latch de Salida*** tiene como finalidad aislar la memoria de cualquier elemento externo, basado en la idea de la ***IDCIC*** (***Implementación Directa con Circuitos Integrados Conocidos***); adicionalmente, ésta idea se junta a la idea inicial de ***ACP*** (***Aislamiento Circuital por Programación***). Si bien es cierto no se ha implementado totalmente esta idea final al mantener activado el Latch de Salida y utilizar un bloque de Latch sin salidas en ***Tri-State***, ésta concepción está latente para cualquier aplicación que se considere implementar. Una muy sencilla variación permitirá implementar ésta opción y otras.

La idea fundamental consiste en el manejo de bits de control más que en el manejo de valores numéricos relacionados a posiciones, proporcionalidades, constantes, etc. Es por ello que para el sistema presente no se tomará en cuenta estas posibilidades.

En general, SASTI – I fue desarrollado siguiendo los lineamientos básicos de la ***Idea Primaria***, la cual manejaba transiciones de memoria comandadas por el reloj. Es por ello que SASTI – I basa cualquier temporización en la elección de un reloj predeterminado que sea una fracción de la señal principal de reloj. Sin embargo, esto último es aplicable sólo a los sistemas denominados ***Mk-II***. Los primeros sistemas de tipo ***I*** de clases A y B, son íntegramente seguidores de la ***Idea Primaria***. Esto se debió a que inicialmente se deseaba corroborar los diseños previos con el objetivo de volverlos bloques operativos que puedan ser reutilizados cuantas veces sean necesarios. En los apartados siguientes se tratará sobre las particularidades de diseño de cada clase.

3.1 SASTI – IA

El SASTI – IA usa la idea de permitir el control de los dispositivos de salida mediante el ADD (Accionamiento por Dato Directo). Esto quiere decir que los bits de Instrucción se convierten en realidad en bits de control que sirven para la activación o desactivación del actuador; y en forma extensiva, puede indicar sentido de actuación, aunque no magnitud de la misma. Esta facilidad permite que su diseño sea bastante reducido, requiriendo un número mínimo de elementos lógicos para una implementación satisfactoria; siendo el único bloque de relativa complejidad el relacionado a la Memoria de Sistema.

3.1.1 Accionamiento y tiempos

El control de los actuadores (cuyo número y bits de control dependerán del caso), se realiza manteniendo en un estado estable de salida los bits correspondientes al actuador que se desee controlar. Esto implica que si se desean controlar varios actuadores, éstos pueden ser activados en forma independiente o en forma colectiva. Esta capacidad de activación múltiple es la que da origen a la idea del MultiPort.

Considerando n actuadores con m bits de control por motor, se tiene que el tamaño de la Palabra de Memoria se puede expresar como:

$$\mathbf{P_M = n.m}$$

Lo anterior es válido si es que se consideran que los actuadores y sus bits de control son de carácter uniforme. Sin embargo, se puede dar el caso que todos los actuadores son de tipo distinto y por ende los bits de control son distintos entre sí. Es por ello que considerando un número ***n*** de actuadores y $\boldsymbol{m_n}$ bits de control, se puede generalizar la ecuación de extensión como:

$$\mathbf{P_M = \sum_{i=1}^{n} m_i} \qquad (1)$$

Por ejemplo, se consideran 3 actuadores: un ***relay*** (Control On/Off, 1 bit), un Motor DC (Control On/Off y Sentido de Giro, 2 bits) y un Motor de Pasos (Control de Parada,

Giro y Sentido; 4 bits; magnitud de rotación relacionada a la frecuencia del $\boldsymbol{R_T}$). Así, se tiene que:

$$\mathbf{P_M = m_1 + m_2 + m_3}$$

$$\mathbf{P_M = 7\ bits}$$

Una vez definida la ***Palabra de Memoria***, se debe considerar cuántas palabras se deben usar para la secuencia de movimiento esperado. Bajo el uso exclusivo de la ***Idea Primaria*** esto presenta un inconveniente: es la misma frecuencia de reloj la limitante en cuanto al mínimo valor de tiempo de ejecución y en cuanto al máximo número de instrucciones necesarias para lograrlo. Esto último debido a que cada $\boldsymbol{P_M}$ sólo permanecerá activando a los actuadores durante el tiempo de transición entre pulsos de reloj (desde el momento en que son aceptados los cambios).

El reloj de Sistema ($\boldsymbol{M_C}$, ***MainClock***) consta de un tiempo de ejecución efectivo para las $\boldsymbol{P_M}$ constante equivalente a $\boldsymbol{t_P}$ en segundos. Si se considera que un cierto actuador $\boldsymbol{A_C}$ necesita un tiempo en particular para actuar $\boldsymbol{t_{A_C}}$, y siendo $\boldsymbol{N_{P_M}}$ el número de ***Palabras de Memoria*** necesarias para tal fin; la ecuación generalizada sería:

$$t_{A_C} = \frac{n}{t_P} = N_{P_M} \qquad (2)$$

Como ejemplo, si un actuador cualquiera necesita, para realizar una mínima actuación, de 3 segundos de operación y tenemos que el reloj de sistema está configurado para tener un tiempo de efectivo de ejecución de cada $\boldsymbol{P_M}$ de 0.5 segundos, se tiene:

$$\frac{3}{0.5} = 6 = N_{P_M}$$

Así, para poder tener un tiempo mínimo de ejecución, se requieren de 6 posiciones de memoria. Esto ya permite dar un esbozo del inconveniente que en forma empírica se pudo descubrir al experimentar el primer tipo: se requiere una gran capacidad de memoria para realizar las acciones mientras más pequeño sea el periodo del Reloj del Sistema.

3.1.2 Tiempos y posiciones de memoria necesarias

Para dar mayores luces sobre lo anterior, se considera un número ***n*** de actuadores actuando en forma global un tiempo determinado ***k*** en segundos y con un mínimo tiempo de actuación ***q*** en segundos. Si consideramos que los actuadores se pueden manejar en forma paralela, es posible dejar de usar el factor ***n*** en la ecuación y centrarnos en los tiempos del actuador que requiera menos tiempo en una acción conjunta. Sin embargo, se debe considerar que no necesariamente todos los actuadores actuarán en forma paralela y que existirán algunos que pueden actuar en forma independiente. Así, se tiene que definitivamente existirá un ***tg*** global en el cual los actuadores estarán actuando en solitario o bajo la premisa de ejercer una predominancia en el tiempo de actuación.

Ejemplo de una tabla de Temporización de Actuadores

Actuador	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
M_1									
M_2									
Act_1									
Act_2									

EL T_T de actuación es

$$T_T = \sum_{i=0}^{9} t_i$$

Se observa en la tabla cómo cada actuador se encuentra activo un determinado tiempo. El mínimo tiempo de actuación es usado como referencia para definir t_i.

En éste caso, es un sistema con nueve posiciones de memoria.

Simplificando las consideraciones, se puede llegar a esbozar una relación como:

$$N_{P_M} = \sum_{i=1}^{t_g} \frac{k_i}{q_i} \qquad y \qquad q_i = x_i\, t_P \qquad (3)$$

Es fácil darse cuenta (aún sin mencionar un ejemplo práctico), que mientras mayor tiempo de actuación global se tenga y menor sea el tiempo de ejecución efectiva, mayor será el número de posiciones de memoria usadas para satisfacer el proceso de actuación completo.

Caso: Necesidad de tiempos más finos.

Actuador	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
M_1														
M_2														
Act_1														
Act_2														

Usando el sistema anterior, la tabla se expande hasta t_{14}; sin embargo, al existir un tiempo mínimo de actuación que es distinto al tiempo mínimo usado, los actuadores no pueden accionarse en el tiempo correcto. Bajo éstas nuevas condiciones queda sólo una opción: variar la base de tiempos. Esto implicaría que en vez de 14 posiciones de memoria, ahora serían necesarias 28.

3.1.3 Instrucciones

El diseño de instrucciones es, para ésta etapa de la arquitectura, casi irrelevante ya que por defecto la instrucción natural del sistema es la de *Activación/Desactivación*. En realidad, la única capacidad que tiene éste sistema inicial es la de manejar bits para que un actuador se active, permanezca así, o se desactive. Tratar de realizar cualquier otra acción choca inmediatamente con el hecho que la extensión del bus es para uso exclusivo de los bits de control de los actuadores; si por algún motivo esos bits son usados, se puede llegar a tener problemas en la ejecución de las acciones debido a que el propio sistema de hecho ejecutaría ambas acciones al unísono. Es por ello que lo más conveniente si es que se desea usar bits del bus de datos para implementar instrucciones complementarias, es usar los *bits muertos*.

Los *Bits Muertos* (*DB, Dead Bits*) son definidos para ésta arquitectura como aquellos bits que tienen un significado real bajo una determinada combinación; sin embargo, para una cierta combinación, no significan nada para el actuador que los recibe. Ésta combinación de bits son los que realmente se les denomina *DB*. No siempre es posible encontrar *DB* para poder implementar instrucciones adicionales. Primeramente, nunca existirían para el caso de 1 bit (lo cual es lógico ya que sus estados excluyen cualquier

posibilidad adicional). A partir de 2 bits puede existir una combinación que tenga *DB*; pero, ésta tiene que estar completamente bien definida para evitar cualquier tipo de confusión. Por lo general, es posible encontrar *DB* cuando existen combinaciones de bits de tipo irrelevante o que presenten una duplicidad de ejecución. Sólo en éstos casos será posible implementar cualquier otra instrucción aprovechando los *DB*.

Si se desea evitar el problema de buscar una combinación adecuada de *Dead Bits* la solución más práctica radica en usar bits adicionales para poder implementar instrucciones adicionales. Eso haría que la ecuación *(1)* se viera modificada como:

$$\mathbf{P_M} = \left(\sum_{i=1}^{n} \mathbf{m_i}\right) + \mathbf{Z}$$

Donde *Z* es el número de bits adicionales necesarios para implementar otras instrucciones. Se comprenderá que bajo el esquema existente, la implementación de instrucciones adicionales choca directamente con la simplicidad del diseño. Literalmente, la somera cantidad de elementos no permite elaborar acciones complejas para esta arquitectura inicial; por lo que la única instrucción práctica en ésta etapa es la de *Reset*. Debido a que el sistema no ha sido diseñado pensando en el manejo de datos numéricos sino en el manejo de señales de control, instrucciones como aquellas relacionadas a ejecutar saltos serían casi imposibles de implementar; esto principalmente al hecho que dichas instrucciones implican la necesidad del manejo de datos numéricos, tales como valores de posición, para lo cual el sistema no fue diseñado. Una posibilidad es adicionar una etapa especializada que reciba un dato numérico externo y bajo ésta condición, el sistema sea capaz de realizar un salto; sin embargo, ésta posibilidad no se considera para ésta etapa debido a las propias características de la *Idea Primaria* no permitirían en forma real un óptimo aprovechamiento de ésta posibilidad. Así, a lo máximo para ésta etapa, la ecuación *(1)* extendida sería

$$\mathbf{P_M} = \left(\sum_{i=1}^{n} \mathbf{m_i}\right) + \mathbf{1} \qquad (1.1)$$

Esta ecuación es válida sí y sólo sí no sea posible el uso de *Dead Bits* para implementar el *Reset*.

El formato general de la *Instrucción* estaría dado, en su máxima expresión, por las siguientes:

3.1.3.1 Instrucción de activación y sentido

Esta instrucción es de tipo múltiple, lo cual quiere decir que esta instrucción permite el manejo en forma paralela de varios actuadores. Así, considerando que existen *n* actuadores que pueden efectuar una *sumatoria de acciones* (esto es resaltado para poner hincapié que no se trata de una suma aritmética), se tiene que es posible representar la instrucción como:

$$\sum_{i=1}^{n} (M_i \ , \ Acc_i) \qquad (4)$$

Donde Acc_i indica una acción que dicho actuador tomará, ya sea activarse, desactivarse o realizar una acción en un sentido u otro. Debido a que la extensión de la propia instrucción variará dependiendo del número de actuadores existentes, es recomendable usar un gráfico de accionamiento para poder determinar qué acción se tomará en qué momento y por cuánto tiempo. Esto será una ayuda al momento de programar el sistema.

3.1.3.2 Instrucción de *Reinicio* o *Reset.*

La instrucción de Reset es totalmente independiente de la concepción original y sólo se implementará si es que realmente es necesaria. Cuanta mayor capacidad de memoria se tenga y preferentemente en casos de usar una memoria regrabable por usuario, ésta instrucción puede ser considerada. Para aplicaciones pequeñas, implementarla sería un malgasto de recursos. Ésta instrucción actuaría directamente en el bloque *Contador de Programa*, haciendo que éste reinicie su cuenta una vez que se active ésta opción. Teniendo en cuenta que al reiniciar el sistema, la instrucción presente a ejecutar será distinta a la instrucción de *Reset*, el sistema volverá a ejecutar las instrucciones memorizadas en un bucle permanente.

3.2 SASTI – IB

La idea para desarrollar el *SASTI – IB* nace de la necesidad de mantener un valor mínimo en la *Palabra de Memoria* a pesar que el número de actuadores sea mayor. Realizar estructuras de memoria de muchos bits de datos no es algo extraño en los tiempos actuales; pero, siempre y cuando éste número de bits sea justificable. *SASTI – IA* podría ser aceptable en su manejo de bits mientras los valores de éstos sean relativamente pequeños; sin embargo, cuando el número de actuadores empieza a ser mayor, el crecimiento descomunal de la Palabra crea problemas de implementación. La solución para implementar un manejo de varios actuadores con un tamaño de palabra relativamente menor es usando la idea del *Decodificador de Instrucciones*. Esto puede mejorar en algo el desempeño de los sistemas basados en la arquitectura del *SASTI – IA*; aunque la mejora sustancial de estos sistemas se cristaliza cuando se les adiciona el *MultiClock* y se vuelven *Mk-II*.

EL diseño del *SASTI – IB* trata de solucionar el problema del número de bits de la *Palabra de Memoria* mediante la idea de la *Instrucción Decodificada*. Esto permite que se pueda implementar una memoria no basada en el número de bits necesarios para controlar un actuador, sino en el número de instrucciones necesarias para poder controlarlo. Esto implica necesariamente que se deben de conocer cuántos actuadores se van a usar, cuántas acciones se pueden desarrollar con ellos y el número de combinaciones necesarias para la interacción de los mismos. Esto último, puede acarrear problemas.

3.2.1 Tiempos

El manejo de tiempos en un *SASTI – IB* es el mismo del *SASTI – IA*. Esto es: usa el reloj principal para temporizar sus instrucciones (R_T). Es por ello que se tomarán como válidas para este modelo las consideraciones del anterior. La diferencia quizás se dé en que no se hablará ahora de una *Actuación* sino de una *Acción* conjunta por vez.

3.2.2 Instrucciones y acciones

La idea de rebajar el tamaño de la *Palabra de Memoria* usando un *Decodificador de instrucciones* es a primera vista una solución bastante aceptable. Se podría decir que en forma efectiva se puede mantener el tamaño de la Palabra en un nivel pequeño así sean muchos los actuadores involucrados; sin embargo, esto no es tan simple ya que se debe tener en cuenta que la idea de usar un *Decodificador de Instrucciones* implica que se debe saber a ciencia cierta cuántas instrucciones son necesarias para poder realizar una *Acción de Programa*. Pero, primeramente se debe definir qué es considerada una *Acción de Programa* y cómo se diferenciará de una *Actuación*.

Bajo los lineamientos iniciales, se hablaba de *Actuación* bajo la premisa que al controlar directamente los bits mediante el programa, lo que se lograba era que un actuador realizara la acción para la cual se le había diseñado (a lo cual se le denominaba *Actuación*). Este control era directo, por lo que bastaba una instrucción de tipo compuesto para solucionar el problema de programar. Teniendo una relación de qué bits realizan las acciones y cuál era su combinación para las mismas, se puede expresar esto en forma sencilla como:

$$\sum_{i=1}^{n} M_i \;,\; Acc_i$$

Esto funciona fácilmente con el SASTI – IA que es de aplicación directa de los bits al actuador. Sin embargo, esto ya no es tan fácil de aplicar al *SASTI – IB* desde el primer momento que usa una *Instrucción Decodificada*. Para poder realizar las mismas acciones de control que le daban flexibilidad al primer SASTI, se debe considerar cuántos actuadores van a existir y cuántas formas de actuar (es decir, cuántas actuaciones) van a desarrollar. Primero, se consideran que existen n actuadores, cada uno con un número m de formas de actuar (por ejemplo, detenerse, avanzar, abrirse, cerrarse, etc.). Si consideramos a todos uniformes, el número de actuaciones posibles puede ser definido en forma gráfica como sigue:

Ejemplo de combinación de Actuadores

Considerando los Actuadores ***A***, ***B*** y ***C***, cada uno con sólo dos posibles estados: ***ON*** y ***OFF***, la relación de combinaciones para cada uno de ellos puede verse en forma gráfica como:

A (ON)		B (ON)		C (ON)
				C (OFF)
		B (OFF)		C (ON)
				C (OFF)
A (OFF)		B (ON)		C (ON)
				C (OFF)
		B (OFF)		C (ON)
				C (OFF)

Se puede observar cómo se crea un árbol de relaciones con respecto a cada estado de los actuadores con respecto al actuador siguiente. A mayor número de estados, mayor es la ramificación. Se puede observar la relación multiplicadora entre ellos.

Se observará que cada actuador tiene un número finito de formas de actuar. Una vez que una forma se escoge para el primer actuador, ésta sólo podrá tener una forma de actuar relacionada con una de las formas de actuar con el siguiente actuador. Una vez que estas dos formas son seleccionadas, sólo podrán tener una forma de actuar relacionada con una de las formas de actuar del siguiente actuador. Esta cadena se continúa hasta llegar al último actuador y a sus formas de actuar. Todas estas formas de actuar son las que realmente configuran el número de instrucciones que se pueden tener disponibles y que son independientes del número de bits involucrados en dichas actuaciones.

Si se tienen ***n*** actuadores y ***m*** formas de actuar ecuación del ***Número de Instrucciones*** se puede expresar como:

$$N_I = m \times m \times m \times \ldots . \times m \qquad (n \ veces)$$

$$N_I = m^n$$

Considerando que en definitiva no todos los ***n*** actuadores tendrán un número igual de formas de actuar y que más bien éstas están en relación al actuador siendo un número de formas de actuar m_n, se puede generalizar la ecuación como:

$$N_I = \prod_{i=1}^{n} m_n$$

Esto definitivamente crea un problema a la arquitectura. Dependiendo de cuántos actuadores, cuántos bits y cuántas formas de actuar, el uso de una arquitectura de ***Instrucción Decodificada*** puede presentar algunos inconvenientes.

Como ejemplo: Se tienen 3 actuadores controlables con 1 bit (control On/Off). Esto quiere decir que los actuadores sólo tienen cada uno 2 formas de actuar: encendido o apagado. Siguiendo los lineamientos de ***SASTI – IA***, se tiene que el tamaño de la ***Palabra de Memoria*** se vuelve 3 bits. Considerando el planteamiento de ***SASTI – IB***, con 2 modos por cada uno de los 3 actuadores, el número de ***Instrucciones*** es de 8. Para lograr 8 instrucciones se requieren 3 bits, lo cual define la ***Palabra de Memoria*** para este tipo. Aparentemente, ambos permiten obtener lo mismo. Sin embargo, se podría tener el caso que los actuadores tengan un tamaño distinto de bits de control. Por ejemplo, que tengan 1 bit, 2 bits y 3 bits para control sus acciones respectivamente; y sólo se les manejará con dos opciones para cada uno: encendido o apagado. Así, para el caso de del primer SASTI, la Palabra es de 6 bits, mientras que el segundo SASTI mantiene la Palabra en 3 bits al ser sólo 8 las posibilidades de instrucción. Si se tomara la máxima capacidad de actuación para cada uno: 2, 4 y 8; se obtienen 64 formas de actuar. Esto implica que el tamaño de la ***Palabra de Memoria*** para el segundo SASTI es de 6 bits. Esto permite definir que el número de bits de la ***Palabra de Memoria*** para el ***SASTI – IB*** estará dado por la relación:

$$P_M \leq N_b$$

Donde N_b es el número máximo de bits involucrados en el control de todos los actuadores. Con éste panorama, la pregunta que queda es ¿realmente importa

implementar la idea de ***Instrucción Decodificada***? Para ciertos casos podría funcionar; pero indudablemente tendería a igualarse. Es por ello que el acercamiento mediante la ***Actuación*** se ha cambiado mediante la idea de la ***Acción de Programa***.

La ***Acción de Programa*** se refiere al funcionamiento de los actuadores en conjunto para desempeñar una determinada acción. Si por ejemplo, se tiene un móvil que usa 2 actuadores que usan 2 bits para controlar su actuación. Se desea que el móvil realice las siguientes acciones: Avanzar, Detenerse, Girar a la derecha y Girar a la izquierda. Si se tomara en cuenta el concepto del ***SASTI – IA***, sería necesario el uso de 4 bits para desarrollar las acciones. Sin embargo, mediante el concepto del ***SASI – IB***, al ser sólo 4 ***Acciones de Programa***, sólo son necesarios 2 bits.

Mediante la definición de las ***Acciones de Programa*** es posible determinar las combinaciones de acciones necesarias para el caso que se trata y así evitar todas las posibilidades combinatorias de actuación no usadas que pueden existir.

Así, la forma más correcta de expresar éste hecho es mediante:

$$2^{P_M} = N_{AccP} \qquad (5)$$

Donde N_{AccP} representa el número de ***Acciones de Programa*** o ***Instrucciones*** disponibles y P_M es el número de bits requeridos en la ***Palabra de Memoria*** para satisfacer ésta necesidad.

Si bien es cierto en el esquema puro del ***SASTI – IB*** no es tan necesario un ***Reset*** (recordando al primer SASTI), debido al simple hecho que es un esquema de ***Instrucción Decodificada***, es posible implementarlo usando un bit adicional con la ventaja sustancial que se ganarían más instrucciones. Si se considera al bit de reset como un bit exclusivo, las ***Acciones de Programa*** disponibles serían:

$$2^{P_M - 1} + 1 = N_{AccP}$$

Si no se considera al bit de reset como un bit exclusivo y forma parte de la ***Palabra de Memoria*** a decodificar, salvo la instrucción de reset, se puede obtener que:

$$N_{TAccP} = 2N_{AccP} - 1$$

Finalmente, se debe de tener en cuenta que no siempre los valores de $\mathbf{P_M}$ serán exactos para satisfacer el N_{AccP}, por lo que es posible tener ***Instrucciones Libres***. Dentro de estas instrucciones es posible implementar un ***Reset***, con lo que la ecuación ***(5)*** sería suficiente y P_M no sufriría variaciones

3.2.3 Tamaño de la memoria y Temporización

Como en el caso del ***SASTI – IA***, en esta variante se tiene que el tamaño de la memoria, independiente de la P_M, dependerá del tiempo de la acción. Usa las mismas consideraciones que ***SASTI – IA.***

3.2.4 Decodificador de Instrucciones y Bus de Salida

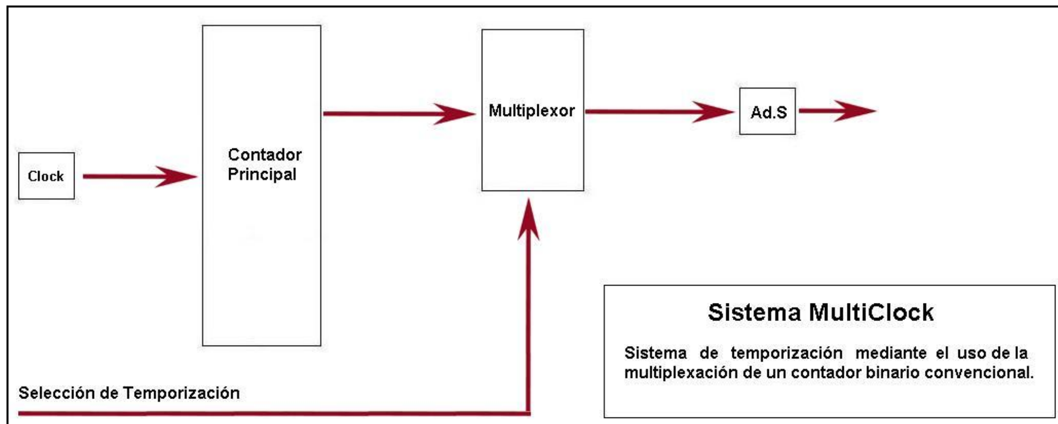
Algo nuevo que diferencia al ***SASTI – IB*** del ***SASTI – IA*** es la existencia del ***Bus de Salida.*** Este bus recibe los bits de control directamente desde el ***Decodificador de Instrucciones***. Por líneas generales, éste bus tiene una longitud equivalente a los m bits de control de cada uno de los n actuadores que se tenga y se adiciona un bit por la opción de ***Reset***. Así, se puede expresar como:

$$T_{B_S} = \sum_{i=0}^{n} m_i + 1$$

La implementación física del ***Decodificador de Instrucciones*** puede efectuarse ya sea mediante una combinación lógica o mediante una memoria.

3.3 El subsistema *MultiClock*

El ***MultiClock*** es una forma de realizar un control por temporización usando como base la propia frecuencia de reloj que ingresa y generando salidas seleccionables con distintos grados de temporización. La idea fundamental del ***MultiClock*** se basa en la premisa básica de la ***Idea Primaria***: el uso del reloj como base de temporización (R_T).



Ya que para poder desarrollar una determinada acción, se deben tener una serie de ***bits de control*** en un determinado estado en un determinado monto de tiempo para lograr un determinado efecto, esto implica que si adicionalmente se requirieran un determinado grupo de bits bajo la forma de ***bits de datos*** para poder modificar el accionar de un determinado sistema, éstos también necesariamente deberían permanecer un determinado tiempo bajo este estado para que los datos sean leídos y puedan así ser usados. En ambos casos, es necesario un cierto tiempo en el cual los bits estarán presentes; ***sin que eso quiera decir que esos tiempos son iguales para ambos tipos***. Bajo ciertas condiciones, los bits de control pueden ser lentos, mientras que los bits de datos pueden ser enviados en forma más rápida (e inclusive, podría ser al contrario). Por ello, es necesario que el ***Contador de Programa*** sólo realice una transición de una posición a otra cuando el tiempo necesario para cada acción se cumpla (***Temporización Independiente Efectiva, TIE***). La señal para que el contador pueda realizar dicha transición se denomina ***Señal de Aceptación de Temporización*** (***SAdT***). Una ***SadT*** puede venir de dos fuentes: una ***Externa*** o una ***Interna***. Usar una señal externa implicaría tener un sistema totalmente pasivo. Desde el primer momento en que se ha planteado la realización de un sistema autómata secuencial, ésta opción como criterio de diseño ha sido por el momento dejada de lado. Eso no implica que su propia implementación sea problemática (de hecho, es bastante sencilla), sino que ésta

posibilidad no va a ser implementada por el momento. Así, la fuente de aceptación para el sistema se vuelve uno de carácter interno. Se espera en trabajos futuros incorporar la opción de ***Estímulo Externo*** (***E***^{2}) como parte integral de la arquitectura y con instrucciones propias. Esto permitirá al sistema darle una flexibilidad adicional ya que no sólo será capaz de recibir datos externos por medios convencionales (desde un puerto de lectura) sino que agregará una función particular con la cual no necesitaría otras consideraciones de programación para lograr retardos variados. ***E***^{2} es una opción interesante; pero, será dejada para futuras ampliaciones.

Ya que la premisa es diseñar un sistema que funcione más que nada usando la temporización como base, se debe de considerar que dicha señal de cambio debe tener por sí una temporización, de preferencia de carácter estable. Una señal de características estable ideal presenta un ciclo de trabajo (***Duty Cycle***) regular en forma de onda cuadrada de niveles lógicos adecuados (ya sean ***TTL***, ***CMOS***, etc.) y de una frecuencia estable. Una señal como la antes mencionada, puede hacer que un estado se mantenga estable durante un determinado tiempo definido por el periodo de la señal, realizando los cambios en forma contínua durante toda la extensión de datos disponibles; es decir, durante todo el tamaño de programa establecido. Debido a que por naturaleza un actuador cualquiera requiere un tiempo relativamente grande para realizar una acción (comparándolo con los tiempos de respuesta de los sistemas digitales), es posible definir un reloj que sea “lento” para realizar las acciones deseadas. Sin embargo, se debe tener en cuenta que si se requirieran usar elementos más rápidos que otros o con tiempos menores de acción, ***se debería ajustar el reloj*** y es ahí donde, como una forma de solucionar este procedimiento, ***MultiClock*** aparece.

3.3.1 Diseño del MultiClock

MultiClock se basa en una premisa sencilla: para poder obtener una temporización aceptable es menester usar una frecuencia estable a la cual se procederá a dividir en múltiplos de la misma. Así, el sistema usa una frecuencia base, la cual será ingresada a divisores de frecuencia para obtener múltiplos estables de la frecuencia original. Ya que

en la etapa experimental no será necesario el uso de un divisor muy específico, se usará un divisor sencillo de potencia de 2 para obtener el efecto deseado.

Como base del ***MultiClock*** se usará un bloque contador binario convencional cuyas salidas serán seleccionadas de acuerdo a la frecuencia que se desee obtener. La premisa por la cual se usará un contador binario es sencilla: si se revisa el accionar de un contador binario sencillo, es fácil descubrir que un contador binario es por sí un divisor de frecuencia en potencias de 2: la frecuencia de sus salidas son divisiones en potencias de 2 de la frecuencia de entrada. El hecho de conformar múltiples relojes de frecuencias distintas y seleccionables, es lo que le da el nombre a este subsistema.

Mientras más bits de salida se tengan, mayor será la división de la frecuencia final con respecto a la frecuencia original. Es por ello que inicialmente se ha considerado que una relativamente buena opción en cuanto a flexibilidad de operación es el uso de un contador de 16 bits como base del sistema. Esto se debe a un hecho simple: 16 salidas implican el uso de un multiplexor de 16 a 1, el cual puede ser gobernado por 4 bits, los cuales serán los bits que comandará el sistema ***SASTI***.

Como se puede ver, el sistema ***MultiClock*** es funcional y sencillo de implementar y si bien es cierto, cabe la posibilidad de una falla en la temporización debido a la posibilidad de un cambio en el pulso fuera de fase, para fines de experimentación, es posible considerarlo suficiente. Cabe anotar que es posible observar que existe un desfase de inicio entre las señales. Si bien es cierto estos dos puntos son posibles de dejar pasar por alto para efectos de experimentos preliminares, idealmente no se puede dejar de considerar otras opciones.

3.3.1.1 MultiClock con autoreset

El ***MultiClock*** con autoreset trata de eliminar los problemas de pérdida de sincronía usando un recurso sencillo: realiza un reset cada vez que existe un pulso de salida. Esto implica que el sistema estará enviando una señal de reinicio de cuenta que permitirá al sistema automáticamente situarse en una posición inicial cualquiera que sea la salida seleccionada. Esto permitirá que no exista la posibilidad de una pérdida de tiempos debido a que el propio sistema estará reiniciando la cuenta total, con lo cual siempre se contará desde el mismo punto de referencia.

Con esto, el ***MultiClock*** puede actuar con un mejor control de tiempos y de hecho, regula bastante bien el funcionamiento global del sistema.

3.3.2 Diseño del bloque *MultiClock*

El sistema es posible de implementar usando los bloques correspondientes a circuitos conocidos. Ésta ha sido siempre la pauta para el diseño del ***SASTI***; y a pesar de ello, en éste punto se ha preferido realizar un bloque en ***VHDL*** para tener la comodidad de un solo bloque.

Como todo bloque en ***VHDL***, se empieza definiendo las librerías a usar y la entidad:

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter2 is
```

Como se pretende usar un número de forma referencial, se opta por:

```
generic(n: natural :=16);
```

El uso de éste tipo sirve para prefijar una variable relacionada al número de salidas que se espera tener. En el caso del ***MultiClock***, se ha considerado un multiplexor de 4 bits de selección, por lo que el número de salidas del contador es de 16.

A continuación, se definen los elementos de entrada y salida (***Ports***) y luego la ***Entidad*** se finaliza. La configuración queda de la siguiente manera:

```
port(   clock:  in std_logic;
        clear:  in std_logic;
        count: in std_logic;
        Q:      out std_logic_vector(n-1 downto 0)
```

```
);
end counter2;
```

El siguiente paso es definir la arquitectura general, la cual se plantea como:

```
architecture behv of counter2 is
        signal Pre_Q: std_logic_vector(n-1 downto 0);
```

Debido a que se requiere un valor cambiante dentro de la misma arquitectura, se define un tipo ***Signal*** (Señal) ***Prev_Q*** como variable para éste vector. El uso de la variable predefinida ***n*** permite una flexibilidad al momento de modificar el tamaño del vector. Luego, se procede a definir el proceso completo (***Process***).

```
begin
process(clock, count, clear)
```

El proceso en éste caso es dependiente de la señal de reloj (***Clock***), un habilitador de conteo (***Count***) y una señal de reset (***Clear***); por lo que se les define como tales. Luego, el proceso en sí empieza:

```
      begin
            if clear = '1' then
Pre_Q <= Pre_Q - Pre_Q;
```

Lo anterior, define un ***reset*** de conteo mediante una instrucción condicional en función del estado de ***clear***. Este mismo estamento condicional permite definir la opción de

conteo real. Para definir el tipo de flanco de conteo se usa la opción de ***clock'event***, la cual sólo tomará una decisión sobre el estado de la señal ***clock*** siempre y cuando exista una transición (***event***) de dicha señal.; en este caso, sólo se tomará en cuenta si dicha transición ha sido a un 1. Así, las instrucciones quedan:

```
elsif (clock='1' and clock'event) then
if count = '1' then
    Pre_Q <= Pre_Q + 1;
end if;
end if;
end process;
```

Estas instrucciones son las que definen el conteo general; pero, faltaría todavía el envío de la información manipulada por el bloque a una salida externa. Es por ello que se definió ***Q*** como variable de salida. Ésta variable recibe el valor cambiante de ***Pre_Q*** (sobre el cual se ha trabajado íntegramente) y la envía al exterior.

```
Q <= Pre_Q;
end behv;
```

3.4 S.A.ST.I. serie Mk-II

La serie ***Mk-II*** es la mejora final de la serie ***SASTI – I.*** La razón de implementar esta serie es la de solucionar el problema del ineficiente uso de la memoria y la rigidez del R_T. En líneas generales, permite usar relojes más rápidos y aún así obtener una buena temporización. Esta adición a la arquitectura básica es la que da origen en realidad a todo lo que es la arquitectura actual.

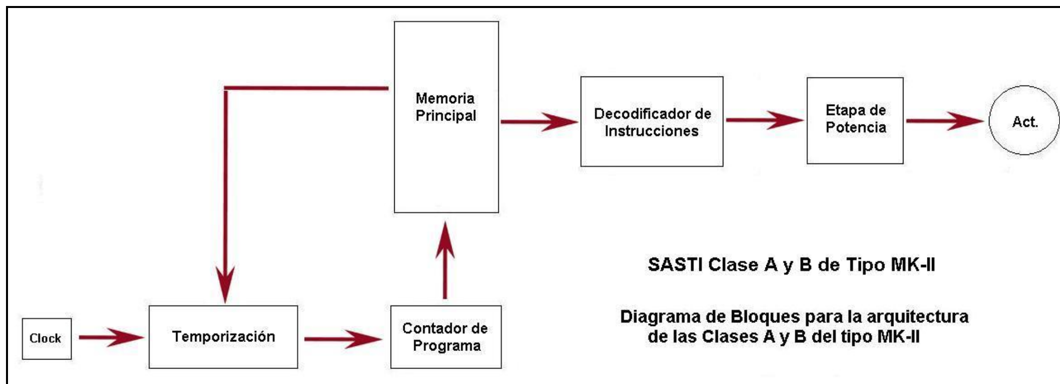


Diagrama general del tipo MK-II. Se observa cómo del bloque de Memoria Principal parte información a un bus especializado para el control de temporización.

3.4.1 SASTI – IA Mk-II

El SASTI – IA es por sí un sistema muy limitado. Su manejo de memoria es bastante ineficiente, sólo justificable en el caso que realmente no importe desperdiciar la capacidad de memoria existente para alcanzar el objetivo de controlar los actuadores disponibles. Sin embargo, esto no es lo más deseable. Un manejo de memoria eficiente permite darle una mayor capacidad al sistema para realizar acciones complejas. Es por ello que se realiza una mejora al sistema inicial *SASTI - IA* adicionando el subsistema *MultiClock*.

3.4.1.1 Accionamiento y tiempos

El accionamiento y los tiempos son parecidos a los usados en el SASTI – IA , inclusive compartiendo la capacidad de manejo paralelo tipo *MultiPort*.

La misma ecuación básica sirve de origen al diseño del tamaño de la *Palabra de Memoria* para los sistemas *Mk-II*. Hay que anotar que el diseño va a necesitar obligatoriamente la implementación de *Reset*.

Se tiene que la ecuación *(1)* se podía expresar como:

$$P_M = \sum_{i=1}^{n} m_i$$

Esta ecuación es válida si se usa un *Reset* implementado usando los *Dead Bits*; de lo contrario, se debería usar la ecuación *(1.1)*:

$$P_M = \left(\sum_{i=1}^{n} m_i\right) + 1$$

Cuando se adiciona el sistema *MultiClock*, se adicionan a los bits existentes el número de bits relacionados a éste sistema; es decir, se adicionan *Mc* bits a la ecuación, la cual quedaría como:

$$P_M = \left(\sum_{i=1}^{n} m_i\right) + M_c$$

En el caso de no usar los *Dead Bits*, quedaría como:

$$P_M = \left(\sum_{i=1}^{n} m_i\right) + M_c + 1$$

Por ejemplo, si se adiciona un sistema *MultiClock16*, el número de bits adicionados por M_C en cualquiera de las dos variantes es 4.

En cuanto a los tiempos, éstos no dejan de tener relación a los tiempos mínimos de activación que pueda tener un actuador; sin embargo, estos tiempos se vuelven sólo una consideración de retardo que debe ser especificada configurando adecuadamente el *MultiClock*.

Se hace notar que ahora el número de posiciones de memoria para esta arquitectura mejorada es a grosso modo independiente del tiempo de actuación. No se menciona que sea totalmente independiente debido a la posibilidad que la capacidad total de tiempo que en forma efectiva pueda manejar la memoria con cualquier variante de *MultiClock* pueda ser rebalsada. Esto último es difícil que pase debido a que siempre es posible implementar un *Multiclock* lo suficientemente poderoso como para poder solucionar la mayoría de problemas; pero, desde que existe una posibilidad, no debe de descartarse.

En forma efectiva, existirá un *tg* o tiempo global de ejecución total. Éste tiempo está subdividido en secciones correspondientes al mínimo tiempo de activación de un actuador dentro de un grupo de actuadores. La estructuración del tiempo dependerá del mínimo tiempo de actuación dentro de un grupo, siempre y cuando dicho tiempo no sobrepase la máxima temporización que permita el *MultiClock*; en caso que esto último pasara, se deberá continuar ejecutando la acción cuantas veces sea necesario.

En forma general, la estructura de tiempos será:

$$T_T = \left(\sum_{i=1}^{t_g} t_{Accmin_i} \right)$$

Donde:

t_{Accmin_i} Es el tiempo mínimo de actuación dentro de un grupo de actuadores involucrados en una acción conjunta. En valor, es menor o igual a la máxima temporización del *MultiClock* usado:

$t_{Accmin} \leq T_{M_C}$ Adicionalmente, se debe tener en cuenta que el tiempo de actuación invariablemente se mantiene como:

$t_{A_C} = n\, t_P$ Y que el tiempo mínimo de actuación siempre será un número h de veces el tiempo de actuación básico; es decir:

$t_{Accmin} = h.t_{A_C}$

t_g Es el tiempo global, que es el número de veces que el T_T es subdividido en tiempos de actuación independientes.

T_T Es el tiempo total efectivo de actuación

3.4.1.2 Posiciones de memoria y tiempos

En el caso de los *SASTI – IA Mk-II* el cálculo de las posiciones de memoria necesarias es equivalente al número de subdivisiones de $t_{A_{ccmin_i}}$ por lo que ya no es necesaria una ecuación sino simplemente considerar el número de acciones que se van a llevar a cabo en forma mínima. Este valor resultante es el mínimo necesario para poder desarrollar la acción que se desee en un determinado momento. Es debido a esto, que el tamaño de la memoria en éste tipo de sistemas suele ser más pequeño, lo cual le permite tener una memoria pequeña y aun así poder realizar acciones de mayor duración. Es posible, a partir de este tipo de sistemas, considerar memorias de mayor capacidad que pueden ser regrabadas para poder alojar secuencias distintas de duración variable. Es por ello que la relación del número de posiciones de memoria necesarias sirve sólo como una referencia para una determinada acción. El manejo de tiempos mejorado al usar el *MultiClock* permite considerar un diseño basado en los tiempos mínimos y máximos que dicho sistema permite, para poner a un determinado sistema dentro de un rango de tiempos. Así, considerando que un *MultiClock* tiene un tiempo mínimo de t_{MC} y un tiempo máximo de T_{MC}, y adicionalmente se desea que dicho sistema tenga B posiciones de memoria, se tiene que el *SASTI – IA Mk-II* tiene un rango de acción de:

$$B.t_{MC} \leq T_T \leq B.T_{MC}$$

Como se puede desprender de todo lo anterior, el número de posiciones de memoria necesarias se vuelve un valor que puede ser escogido en forma arbitraria siempre y cuando se considere el rango de tiempos en el cual el sistema trabajará.

3.4.1.3 Instrucciones

Las instrucciones disponibles siguen siendo las mismas que se mencionaron para el caso del *SASTI – IA*. Lo único que cambia es que ahora se vuelve necesario el uso de al menos la instrucción de *Reset* en forma obligatoria. Así, se tiene que las instrucciones serían:

a) Instrucción de activación y sentido

   Es la misma instrucción de tipo múltiple vista anteriormente: igualmente realiza una *sumatoria de acciones*. El cambio radica en que ahora la instrucción

contará adicionalmente de una relación temporal dada por el *MultiClock*, por lo que toma la siguiente forma:

$$DEL_{MC} \; : \; \sum_{i=1}^{n} M_i \;,\; Acc_i$$

Donde DEL_{MC} indica el valor de temporización relativa que realiza el *MultiClock*.

b) Instrucción de *Reinicio* o *Reset.*

Como el sistema original, afectará al *Contador de Programa* haciendo que éste reinicie su cuenta una vez que se active ésta opción.

3.4.2 SASTI – IB Mk-II

El *SASTI – IB Mk-II* es sólo la adición de un bloque *MultiClock* al sistema *SASTI - IB* convencional. Comparte las mismas ecuaciones de diseño.

3.4.2.1 Memoria

En el caso de la memoria, la novedad es la adición de bits correspondientes al *MultiClock.* Así, se puede redefinir como:

$$2^{P_M} = N_{AccP} \quad ; \qquad P_{M_T} = P_M + M_C$$

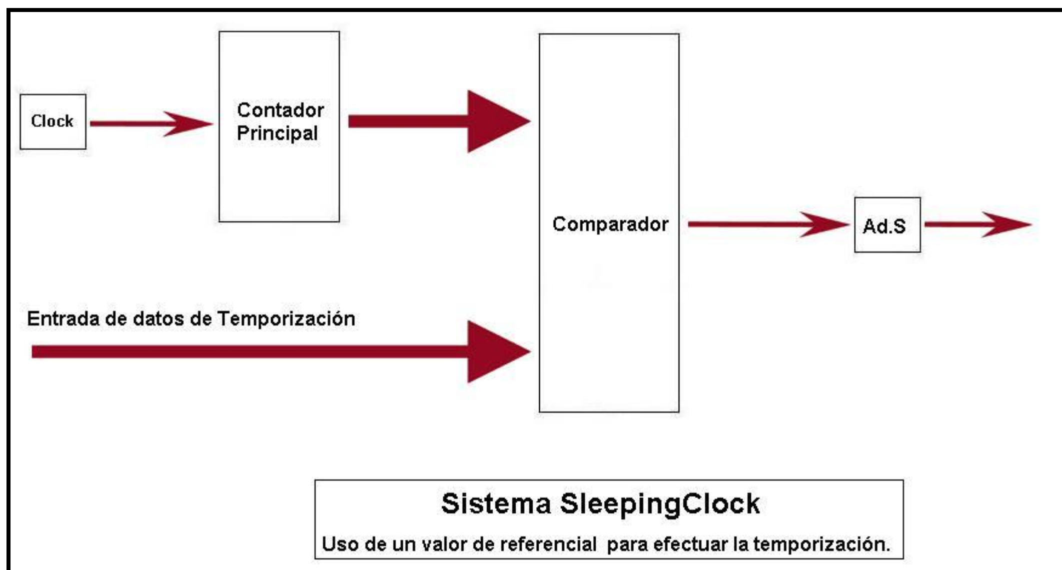
Donde M_C corresponde al número de bits que tenga el *MultiClock*; y *PMT* es el tamaño real de la *Palabra de Memoria* que se usará.

3.4.2.2 Decodificador de Instrucciones y Bus de Salida

Ambos elementos son considerados comunes al *SASTI - IB* convencional. Es por ello que no existe una variación en ésta parte.

3.4.3 SASTI – IC y SASTI - ID

Cronológicamente, el desarrollo de las opciones de tipo *I* clases *C* y *D* fueron dejadas de lado para concentrarse en el desarrollo de las clases *C* y *D* del tipo *II*. Esto se debe principalmente por una razón: la *Modularidad del Sistema*. Desde un principio se planteó realizar el sistema de forma modular, de modo que se pudieran ir desarrollando los bloques de subsistemas necesarios para cada etapa de experimentación. Como resultado, el sistema es altamente modular, permitiendo intercambiar bloques sin ningún contratiempo. Esto permitió que los experimentos en ambos tipos de arquitectura pudieran realizarse sin dificultades. Debido a ello, los *SASTI – IC* y *SASTI – ID* no han sido implementados físicamente. Se ha preferido dejar la implementación en los tipo *II*. Los gráficos que se mostrarán sobre ellos son sólo para evidenciar su implementación y lo sencillo de su modularidad. Los datos de operación y resultados se discutirán en apartados posteriores.



El funcionamiento del ***SleepingClock*** es sencillo. Primeramente, la señal del reloj de referencia llega a un contador con ***Autoreset***. Éste empieza un conteo que es comparado. El ***Valor de Temporización*** (V_T) es ingresado a un bloque comparador

junto al valor numérico alcanzado por el contador; este comparador se encarga de definir (por defecto) si tanto la cuenta como el valor de $\boldsymbol{V_T}$ son iguales, tras lo cual envía un pulso que actúa sobre el ***Contador de Programa*** y que a su vez actúa sobre el mismo contador generando un reset.

En forma casi instantánea (los retardos dependen de los propios bloques, aunque para un sistema lento esto no es importante), la posición de memoria del sistema cambia a la siguiente, el contador del ***SleepingClock*** se reinicia y un nuevo valor de $\boldsymbol{V_T}$ está listo para ser comparado. Este ciclo se repite una y otra vez.

CAPÍTULO IV

EL *S.A.S.T.I.* TIPO *II*

Si el *SASTI* tipo *I* es completamente operativo, ¿cuál es la razón de implementar un tipo *II*? La respuesta es sencilla: el tipo *II* es un acercamiento distinto para solucionar el problema de control de temporización por programa de corte distinto al sistema *MultiClock*. El *SASTI - II* es la evolución de la idea de SASTI usando el subsistema *SleepingClock* para realizar la temporización. A diferencia del tipo *I*, este sistema no manipula la frecuencia del reloj de temporización (R_T) que hacía necesario la selección de frecuencia dividida bajo la ayuda del subsistema *MultiClock*, sino que usa el R_T como referencia para producir un retardo de acuerdo a la acción que deba realizar e inmediatamente realizar un cambio de estado hacia la nueva posición de memoria a ejecutar. Así, la temporización se convierte en una función del R_T; siendo el *Reloj de Sistema* (R_S) una señal distinta del R_T pero dependiente tanto de ésta como de un *Valor de Temporización* (V_T).

El Diseño del *SASTI-IIA* y *SASTI-IIB* se diferencian del *SASTI-IA Mk-II* y del *SASTI-IB Mk-II* sólo en la elección de bits para la *Palabra de Memoria* y en el uso del *SleepingClock* en vez del *MultiClock*.

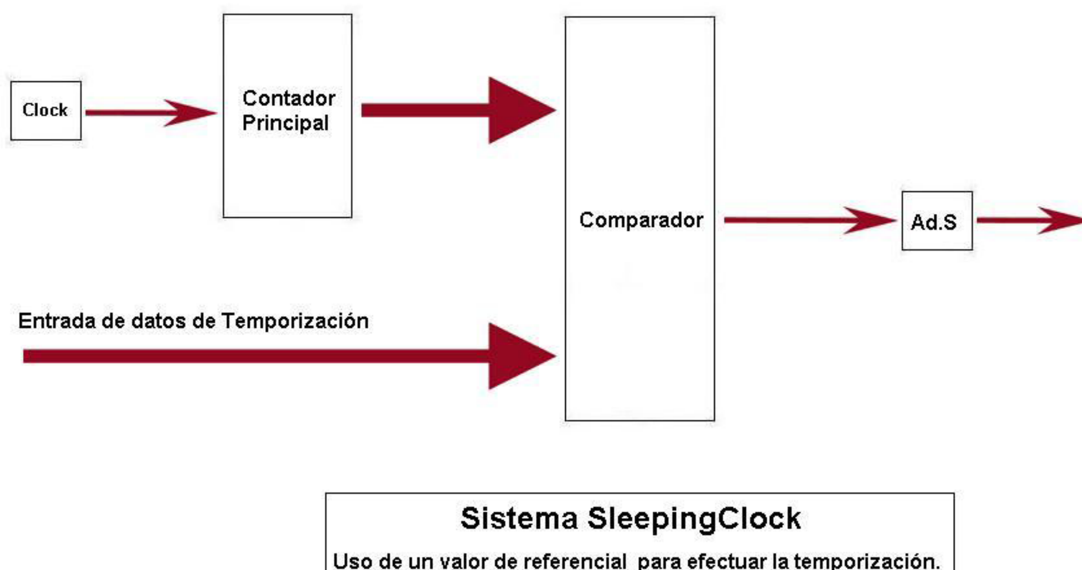
4.1 SleepingClock

La idea del *SleepingClock* se basa en un hecho: reducir el número de elementos funcionando a la frecuencia de reloj principal. Desde el primer momento que se necesita una señal estable de una determinada frecuencia y ésta se encargue de hacer funcionar un elemento, dicho elemento estará permanentemente consumiendo energía (salvo que se controle de alguna forma su funcionamiento). Ya que es primordial que esa señal no sea interrumpida como parte de una base de tiempos que ella establece, el consumo relacionado a la sección del mismo siempre existirá. Así que buscar la forma de minimizar dicho consumo es siempre adecuado.

También, se tiene en cuenta otro factor: la división que realiza el *MultiClock* es demasiado rígida. No da opciones para un control fino del accionar de un actuador. Es debido a estos factores que se empieza a analizar una forma más efectiva para realizar un control de tiempos y es el bloque de sistema *SleepingClock* la solución planteada.

4.1.1 Funcionamiento General

SleepingClock recibe su denominación por la idea de tener un *"Reloj Durmiente"*. En el caso del *MultiClock*, todas las señales de reloj estaban activas, siendo el sistema el que decidía qué señal tomaba cada vez. Si bien es cierto que ninguno de ellos es por sí un generador activo, se deseaba tratar de resaltar la idea principal. En vez de usar las salidas de un contador como frecuencia de salida, se realiza una temporización a modo de conteo, la cual permite obtener un cierto control adicional al que se tenía con sólo depender de un valor fijo de división. Así, el pulso de la señal de reloj de sistema se encuentra en un estado "*Durmiente"* hasta que sea activado por la igualdad en el valor de referencia para temporizar, momento en el cual se activará y reiniciará el proceso. Así, el *SleepingClock* conforma una señal totalmente irregular (*ChaosClock*) que depende únicamente de los valores de temporización que se desee obtener.



Sistema SleepingClock
Uso de un valor de referencial para efectuar la temporización.

El funcionamiento del ***SleepingClock*** es sencillo. Primeramente, la señal del reloj de referencia llega a un contador con ***Autoreset***. Éste empieza un conteo que es comparado. El ***Valor de Temporización*** (V_T) es ingresado a un bloque comparador junto al valor numérico alcanzado por el contador; este comparador se encarga de definir (por

defecto) si tanto la cuenta como el valor de V_T son iguales, tras lo cual envía un pulso que actúa sobre el ***Contador de Programa*** y que a su vez actúa sobre el mismo contador generando un reset.

En forma casi instantánea (los retardos dependen de los propios bloques, aunque para un sistema lento esto no es importante), la posición de memoria del sistema cambia a la siguiente, el contador del ***SleepingClock*** se reinicia y un nuevo valor de V_T está listo para ser comparado. Este ciclo se repite una y otra vez.

4.1.2 La Condición de *Igualdad Absoluta*

La llamada condición de ***Igualdad Absoluta*** es aquella por la cual el sistema ***SleepingClock*** original se paralizaba. Dada una condición consecutiva de valores que produzcan una igualdad en el comparador, la señal de salida del sistema permanecerá inamovible, por lo cual no existe transición de estado que defina un reloj que permita un cambio de estado en el ***Contador de Programa***, por lo cual el sistema se paraliza en su totalidad en la instrucción desencadenante. Ésta condición configuraba en forma efectiva una virtual instrucción de ***Halt***. Como es llamada por condición, ni siquiera se tendrá que implementar en el sistema.

Sin embargo, por problemas de inestabilidad observados bajo el pulso de reset del sistema durante las simulaciones, se optó por reconfigurar el ***SleepingClock*** proporcionándole un retardo adicional correspondiente a un pulso de R_S como ancho y un reset después de ello. Esto permitió salvar el problema de estabilidad; sin embargo, como consecuencia de la modificación, el sistema se hizo inmune a la condición de ***Igualdad Absoluta*** y presentó características ligeramente distintas a las esperadas en temporización. Sin embargo, esto no significó que la implementación fracasara; sólo algunas condiciones cambiaron.

Para aplicaciones con los tipos ***IIA*** y ***IIB,*** se descubrió que el ***SleepingClock*** original era funcional y sin problemas. Es por ello que estos sistemas pueden tener la opción de ***Halt*** tácito, mientras que en los demás tipos se tendría que implementar como parte de una instrucción (esto último, dejado de lado por ahora). En el futuro, se espera

experimentar nuevas opciones para modificar el subsistema con el objetivo de estabilizar mejor el pulso de reinicio y volver a tener un ***Halt*** tácito.

4.1.3 Temporización

La temporización en el sistema ***SASTI – II*** se realiza mediante un conteo ascendente hasta lograr una igualdad con un valor de referencia. El Sistema básico para lograr esto es el subsistema ***SleepingClock***, cuya estructura básica se muestra a continuación:

El R_T tiene una frecuencia de trabajo fija y un ciclo de trabajo del 50%. Esto garantiza un periodo definido y una señal estable y simétrica. Dicha señal tendrá una temporización efectiva de:

$$T = \frac{1}{R_T}$$

El Sistema envía un valor numérico binario que conforma el ***Valor de Temporización*** (V_T), el cual será comparado con un valor dado por un contador binario reiniciable. Dado que el contador binario empieza su cuenta en 0 y que la activación de la señal del ***SleepingClock*** se realiza cuando el valor es igual a V_T, se tiene que el contador realiza un conteo efectivo desde **0** hasta ($V_T - 1$). Considerando que dicho conteo cuenta con n elementos, y que cada uno de ellos permanece un tiempo T, el tiempo total de temporización es de:

$$T_T = n.T$$

Ya que dicho número de elementos es igual al V_T, se puede simplificar la ecuación a sólo:

$$T_T = V_T.T$$

De ésta forma sencilla, el sistema logra temporizaciones relativamente más finas y de una manera más eficiente que las logradas con ***MultiClock***. Debido a la modificación del sistema adicionando un retardo adicional para estabilizar el pulso de reinicio,

algunos valores han visto modificados sus prestaciones. A pesar de ello, el sistema es funcional y el tiempo de retardo global es casi equivalente.

4.1.4 Tamaño de la Memoria

Para poder diseñar una memoria adecuada para el ***SASTI – II*** se deben seguir los lineamientos generales iniciales para el desarrollo de los sistemas ***SASTI-IA*** y ***SASTI-IB***, El número de bits adicionales que se debe tener para implementar la ***Palabra de Memoria*** estarán relacionados a la máxima temporización que se desee tener. Considerando que la máxima temporización es de:

$$M_T = 2^{N_b}$$

Donde M_T es la máxima temporización y N_b es el número de bits para lograr dicho valor. Como se desprende de la propia ecuación, M_T es un valor en potencia de 2.

Dada una ***Palabra de Memoria*** (P_M) diseñada sólo con la consideración de actuadores o instrucciones a implementar, para lograr el tamaño total de la Palabra, se debe considerar:

$$T_{P_M} = N_b + P_M$$

Se observará que el tamaño real de la ***Palabra de Memoria*** es bastante grande. El incremento del tamaño de la Palabra implica un punto a tomar en cuenta al momento de diseñar este tipo de sistemas. Dependiendo de la aplicación, es posible que este tipo de sistemas tenga valores típicos de Palabra del orden de los 16 bits (considerando un ***SleepingClock*** típico de 8 bits). Mientras más fino sea el control esperado, sería lógico considerar una frecuencia más alta de trabajo y un mayor número de bits de ***VT*** para poder realizar el control. Esto, si bien es cierto mejora su desempeño, implicaría tamaños de Palabra muy grandes. Los bits de temporización serían inclusive mucho mayores a los necesarios para realizar las acciones esperadas.

Esto es fácilmente observable con un ejemplo: si se diseña un ***SleepingClock*** de 8 bits, para sólo manejar 2 motores en modo ***On/Off*** con control de sentido de giro, se

necesitarían 4 bits de control de actuadores y 8 bits de temporización, teniendo la ***Palabra de Memoria*** un tamaño de 12 bits.

Aunque se puede decir que existiría una desproporción en cuanto a los bits útiles en comparación a los bits de temporización, esto podría ser relativo. Principalmente, si se considera que la idea de realizar la acción con su respectiva temporización es ahorrar espacio en memoria.

4.1.5 Equivalente con sistemas SASTI de tipo I

En cuanto a consideraciones de diseño, usan prácticamente las mismas relacionadas a los sistemas ***SASTI*** de tipo ***Mk-II***. La única variante es la relacionada al tamaño de bits de temporización a considerar dentro del diseño.

4.1.6 El uso del concepto de Puerto

Hasta ahora, se había considerado la extensión de los buses de salida como una función de las necesidades operativas: estaban supeditados a la cantidad de actuadores que se requerían y a los bits necesarios para realizar las acciones. Se trató de solucionar el problema del aumento desmedido de la ***Palabra de Memoria*** resultante con el uso de un ***Decodificador de Instrucciones***; sin embargo, se pudo observar que esto era una solución limitada y puntual: no podía ir más allá de "podar" el árbol de acciones posibles para intentar tener sistemas con ***Palabras de Memoria*** relativamente pequeñas. Es así que un replanteamiento de la situación llevó a considerar una idea usada en otros sistemas: la idea de Puerto.

Un Puerto es en líneas generales un elemento que almacena un determinado dato y lo envía fuera del sistema; es decir, lo envía a su parte externa: el mundo real. Esta función fue pensada originalmente sólo para bits del ***Bus de Salida***; sin embargo, el uso de puertos permite flexibilizar aún más el desempeño de sistema y permite adicionalmente limitar la extensión de la propia ***Palabra de Memoria***. En primer lugar, el campo de ***Datos de Temporización***, como campo especializado, puede ser tratado de forma independiente en su propio subsistema (tal como se ha observado en todos los prototipos). Luego, los ***Datos Numéricos*** y el campo de ***Instrucciones*** pueden ser

definidos con sólo los bits necesarios. Esto permite poder obtener un manejo de pocos elementos, los cuales a su vez podrían servir para poder accionar muchos otros elementos más. Así, la extensión del número de bits de un puerto es máximo el mismo número de bits de Datos. Fundamentalmente, los puertos serán de dos tipos: ***Puertos de Entrada*** (***IN Port***) y ***Puertos de Salida*** (***PortLatch***). La diferencia con los sistemas convencionales es la forma de trabajar con ambos tipos.

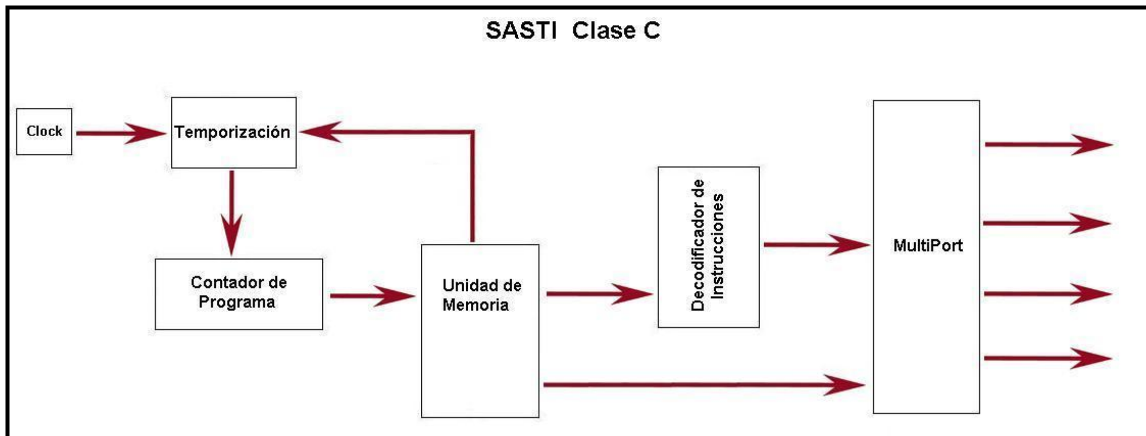
OutPort será implementado mediante un ***Latch*** triestado. La idea es poder implementar en un futuro una opción de ***"Shutdown"*** por la cual ante alguna dificultad un subsistema aísle los puertos usando la opción de habilitación de salida triestada. Si bien esto no va a ser implementado, queda como una opción a futuro. En líneas generales, cualquier elemento dentro de la arquitectura es considerado un ***OutPort***. Así, cualquier registro (de comparación, vector de saltos, etc.) es ***OutPort*** que deberá ser cargado con un valor. Todos los puertos de salida están conectados a un bus ***independiente*** del ***Bus de Datos*** de la Memoria Interna (***Palabra de Memoria***); dicho bus es manejado por un registro especial: el ***BusLatch***.

El ***BusLatch*** es parecido a un registro Acumulador en los sistemas convencionales, con la diferencia que los valores que tenga se mantienen mientras no hallan otros valores a enviar. Esto permite, junto a la idea del ***MultiPort***, enviar el mismo dato al mismo tiempo a todos los puertos y registros que se encuentren asociados al ***BusLatch***. Esto permite realizar una limpieza inicial de puertos instantánea o una duplicidad de secuencias sin necesidad de pasar por enviar datos a cada puerto por vez.

IN Port será igualmente implementado mediante un ***Latch***; sin embargo, la diferencia es que los datos que reciba n serán derivados a bloques específicos. Si bien es posible realizar un equivalente a la dupla ***BusLatch/OutPort*** para los datos de entrada, se ha considerado por el momento dejar ésta posibilidad como un desarrollo futuro. Inclusive, el objetivo máximo nominal es el de manejo de puertos, por lo que la opción de ***IN Port*** será vista sólo en forma superficial. La posibilidad de una opción de realimentación permitirá al ***SASTI*** ser más que un simple autómata al permitirle modificar su accionar mediante condiciones definidas por programa. Como propuesta, ***IN Port*** estará ligada a un bloque comparador que permitirá al sistema discernir entre un valor igual, mayor o menor a una referencia; ésta decisión estará unida a un salto (conformando ***Saltos Condicionales***), los cuales le darán mayores opciones de desempeño.

La selección del puerto que sacará los datos estará configurada mediante el registro especializado ***PortLatch***. Este registro definirá cuántos puertos de salida se activarán para sacar los datos del ***BusLatch***. Esta capacidad de poder definir cuántos puertos al unísono pueden recibir datos del sistema conforma la característica de ***MultiPort***.

4.2 El SASTI-IIC y el SASTI-IC



Aunque parezca extraño, el desarrollo del ***SASTI-IIC*** es anterior al desarrollo del ***SASTI-IC***. Esto debido a un hecho fundamental: los diseños coincidentes.

Al haber sido desde el principio pensados de forma modular, los sistemas ***SASTI*** pueden variar su desempeño variando algunos elementos puntuales. El principal de ellos: el subsistema de temporización. Ya sea que usen ***MultiClock*** o ***SleepingClock***, los elementos de sistema de manejo de actuadores es en la práctica el mismo. Es por ello que al momento de pensar en una solución para la implementación de Puertos, se realizó directamente bajo el concepto del tipo ***II*** ya que era fácilmente configurable para el tipo ***I***. Es por ello que los mismos datos generales de diseño serán compartidos.

4.2.1 Número de puertos

El número de puertos a disposición en realidad dependerá de cuántos bits de ***PortLatch*** se implementarán. Considerando que existen D_b bits de ***Datos***, el número de bits de ***PortLach*** estará limitado por:

$$D_b \geq \#_b PortLatch \quad y \quad \#Puertos = \#_b PortLatch$$

Donde $\#_b\mathbf{PortLatch}$ es el número de bits del ***PortLatch***. La razón de definir el número de bits del ***PortLatch*** de ésta manera es debido a la característica de ***MultiPort*** que se desea implementar; de esta forma, se pueden activar varios puertos a la vez.

Ya que $\#_b\mathbf{BustLatch}$ (número de bits de ***BusLatch***) es igual al número de bits de ***Datos***, de éste elemento sale un bus que une a todos los ***OutPort*** en sus entradas de datos formando un bus dedicado. A su vez, los habilitadores serán comandados por el ***PortLatch***. Esto culminaría toda la etapa de diseño de puertos. Un punto a tomar en cuenta: es posible definir el número de puertos como el máximo número de bits de ***Datos***, dejando libres los bits sin usar para así conformar una reserva para expansión.

4.2.2 Decodificador de Instrucciones

El decodificador de instrucciones estará definido por el número de instrucciones que se deseen implementar. La relación está dala por la ecuación.

$$2^{(\#_b I)} = \#Inst$$

Donde $\#_b I$ es el número de bits de la instrucción y **#Inst** es el número de instrucciones que se plantea implementar.

El número de instrucciones a implementar puede ser arbitrario. Como mínimo se saben que existirán las siguientes instrucciones:

<DEL>; BusLatch, Data: Instrucción de envío de datos al ***BusLatch.***

<DEL>; PortLatch, Data: Instrucción de envío de datos al ***PortLatch.***

Reset: reinicio de operaciones.

Bajo este esquema básico, sólo son necesarios 2 bits para cumplir con las instrucciones mínimas necesarias. Elegir más bits para éstos sistemas será válido siempre y cuando estén bajo la idea de futuras expansiones a implementar.

4.2.3 Temporización

Para la temporización del sistema en su versión de tipo ***IIC*** se usa un ***SleepingClock*** de tipo modificado; es decir, incluye un adecuador de pulso. El número de bits para este sistema dependerá del máximo retardo que se desee obtener. Por estándar, se recomienda un ***SleepingClock*** de 8 bits para tener una opción de temporización media.

Para la temporización del sistema en su versión de tipo ***IC*** se usa un ***MultiClock***. El número de bits para este sistema dependerá del máximo retardo que se desee obtener. Por estándar, se recomienda un ***MultiClock*** de 4 bits para tener una opción de gran temporización.

4.2.4 Memoria y Contador de Programa (C_P)

En general y tal como los casos anteriores, la memoria dependerá de cuántas acciones mínimas se llevarán a cabo. Eso determina la ***Extensión*** de la memoria. Para determinar la ***Magnitud*** de la ***Palabra de memoria*** (P_M), se deben de considerar el número de bits de retardo (***Delay***, D_E), el número de bits de Instrucción (I) y el número de bits de ***Datos*** (D_A). Así, se puede expresar la relación de ***Magnitud*** como:

$$P_M = D_E + I + D_A$$

En la mayoría de casos, la ***Extensión*** de la memoria será arbitraria. Se espera considerar un número relativamente alto de posiciones de memoria como reserva en caso de necesitar programas de distinto tamaño. En caso de una aplicación específica y de carácter no modificable, se puede optar por un diseño ajustado, como el visto en secciones anteriores.

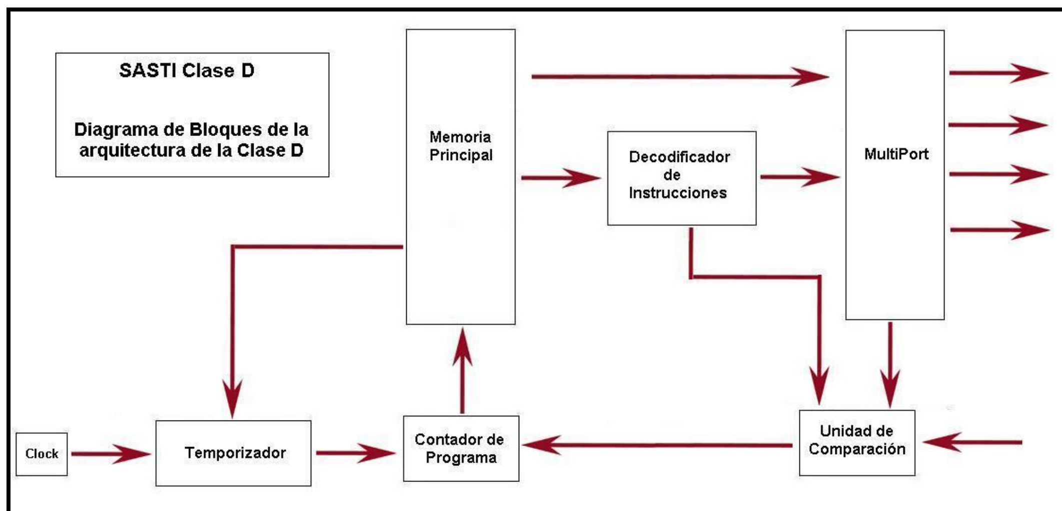
El C_P se diseñará dependiendo de la máxima extensión de memoria que se vaya a utilizar. Debe tener un pin para reset. Así, el contador debe tener los bits de salida necesarios para poder hacer un recorrido de todas las posiciones de memoria de la ***Memoria de Sistema***. Considerando P posiciones de memoria la ***Extensión*** de la ***Memoria de Sistema*** y sean C_b el número de bits del contador, se debe considerar la siguiente relación:

$$2^{(C_b)} \geq P$$

El hecho de asumir la posibilidad de un número mayor es por cuestiones prácticas: a veces los valores de posiciones no se ajustan al valor de la potencia de 2.

No se consideran otras opciones tales como carga de datos a menos que ésta se implemente. Para ello, es necesario que los puertos necesarios sean usados para guardar la posición del vector de salto y que la señal para salto esté incluida dentro de las instrucciones. Una variante implicaría que el vector de salto sea una señal externa (un ***IN Port***) que sería accedida por programa. Sin embargo, ninguna de estas opciones se implementará en ésta etapa. Sólo quedan como una referencia de las posibilidades de expansión de la arquitectura.

4.3 El SASTI –IID y SASTI – ID



Como en el caso anterior, el desarrollo para el tipo ***IID*** y el tipo ***ID*** es prácticamente el mismo, salvo en el ***Sistema de Temporización***; salvo éste punto, en la práctica usan los mismos elementos. La novedad es ésta arquitectura es que aparte de usar la idea de puertos, aquí se aplica la idea de ***Registros Especializados*** (R_E) y los saltos condicionales. Se ha querido limitar otros elementos como recepción de ***Estímulos Externos***, unidades aritméticas, entre otros. Estos subsistemas serán dejados para implementarlos en futuros trabajos.

En líneas generales, es muy parecido a los sistemas anteriores. De hecho, usas las mismas consideraciones para definir bloques generales de subsistemas.

4.3.1 Los Registros Especializados (R_E)

La idea de usar los R_E se basan en la necesidad de brindarle al sistema básico la posibilidad de tomas de decisiones básicas y manipulación de datos externos. Si bien la idea original de ***S.A.S.T.I.*** no contemplaba ningún tipo de manipulación de datos, ésta capacidad es necesaria para ciertas aplicaciones en las cuales requeriría una cierta recopilación de condiciones para trabajar. Esto abre muchas posibilidades: implementación de bloques aritméticos, implementación de registros de almacenamiento temporal (o una memoria interna de variables operativas), implementación de saltos, toma de decisiones condicionales, entre otros.

Dada la idea primordial de la separación de la memoria mediante un registro que conforma un bus unidireccional de datos generales de salida, los R_E son considerados virtualmente como puertos; y su carga de datos dependerá de las instrucciones de carga de puertos. Las salidas de dichos registros irán conectadas a los subsistemas que lo requieran; tales como la ***Unidad de Comparación***, la ***Unidad de Saltos***, entre otros. Cada subsistema tendrá sus elementos primordiales que les permitirán afectar las condiciones de ejecución o permitirán manipular datos.

4.3.2 La Unidad de Comparación (U_{Comp})

La ***Unidad de Comparación***, como su nombre lo indica, indica al sistema si dos valores comparten una relación, ya sea de igualdad, mayoría o minoría. Estas condiciones básicas permitirán al sistema tomar decisiones ya sean para variar sus condiciones de salida o las condiciones de datos. Básicamente, está conformada por un bloque comparador simple o un bloque de ***Multicomparación***.

Un bloque de comparación simple está formada por un R_E, un ***IN Port***, un bloque comparador y tres salidas. El tamaño del R_E y del ***IN Port*** dependerán de la magnitud de datos que maneje (4 bits mínimo). El sistema ingresará un ***Valor de Comparación*** (V_C) al R_E, y cuando lo considere necesario ingresará un valor externo al ***IN Port***.. Las

salidas de ambos registros estarán ligadas directamente al bloque comparador, el cual dará como resultado tres bits de referencia (equivalentes a los ***flags*** en los procesadores convencionales): $A=B$, $A<B$, y $A>B$; ***A*** indicará el valor del R_E y ***B*** indicará el valor del ***IN Port***. Tanto los valores almacenados como los bits de salida son de tipo ***Estables*** mientras no se varíen los valores que contengan. Es decir, si en el caso se quisiera comparar varias veces un mismo valor con un valor externo, simplemente se setea la variable interna (casi como un ***Set Point***) y se puede comparar con la variable externa.

Un bloque de ***MultiComparación*** es una unidad más especializada. Tiene un número de ***n*** registros internos y puede recibir ***m*** puertos externos. Tiene entradas de configuración que permiten identificar qué valores van a ser comparados. Sus salidas son 3 bits de referencia. Esta unidad en sí es una matriz seleccionable que permite almacenar muchos valores de comparación para poderlos comparar con muchas entradas. Esto le da al sistema una mayor flexibilidad en cuanto al manejo de datos internos y externos.

En la presente tesis, sólo se ha considerado abordar el sistema primario: un bloque de comparación simple. La experimentación sobre éste bloque y sus relación con otros subsistemas permitirán sacar experiencias que se cristalizarán en futuros trabajos de expansión de ésta arquitectura.

4.3.3 Unidad de Saltos (U_S)

La ***Unidad de Saltos*** actúa modificando el comportamiento del C_P al variar el conteo regular ascendente haciendo que reinicie el conteo en una posición arbitraria definida por su programa. La U_S realiza 4 tipos de salto:

a) Salto Incondicional

Salta a la Posición de Memoria apuntada por el ***Vector de Salto*** (V_S). Esta acción se realiza sin que medie ningún tipo de condición previa a cumplir.

b) Salto por Igualdad

Salta a la Posición de Memoria apuntada por el (V_S) siempre que previamente se cumpla la condición de igualdad en la U_{Comp}.

c) Salto por Mayoría

Salta a la Posición de Memoria apuntada por el ($\boldsymbol{V_S}$) siempre que previamente se cumpla la condición de mayoría en la $\boldsymbol{U_{Comp}}$.

d) Salto por Minoría

Salta a la Posición de Memoria apuntada por el ($\boldsymbol{V_S}$) siempre que previamente se cumpla la condición de minoría en la $\boldsymbol{U_{Comp}}$.

4.3.4 Contador de Programa ($\boldsymbol{C_P}$)

El diseño del $\boldsymbol{C_P}$ es idéntico a los diseños anteriores. La diferencia principal radica en que en el diseño del contador se debe considerar que sea pre configurable. Esto permite cargar un valor (el $\boldsymbol{V_S}$) a partir del cual empezará a contar. El $\boldsymbol{C_P}$ estará ligado al subsistema de saltos quien se encargará de modificar el funcionamiento del mismo.

CAPÍTILO V

PRUEBAS PRÁCTICAS

Las pruebas prácticas se realizaron usando el módulo educativo ***UP2*** de ***Altera***, el programa ***MAX+PlusII V.10.2*** *y el programa* ***Quartus II V.9.0***. Las pruebas definitivas están implementadas en placas de uso general y unidades móviles.

5.1 Metodología general

El proceso experimental se regirá mediante una serie de pasos flexibles que permitirán probar los bloques operativos del sistema para asegurar su funcionamiento, depurar fallas y rediseñar de ser necesario. Los pasos principalmente serán:

a) Esquematización de la idea general

   El primer paso consiste en el diseño en papel de los sistemas que se deseen tener y las interconexiones entre los mismos. Son realizados a mano alzada para facilitar el diseño, permitiendo distintas variables que pueden ser depuradas hasta obtener un diseño definitivo.

b) Esquematización del bloque requerido

   Una vez que los bloques han sido definidos, se definen los bloques en forma independiente teniendo en cuenta qué elementos deberán tener. Si hay necesidad de variantes, éstas se consignarán para modificar los diseños.

c) Implementación y pruebas de los bloques

   Una vez que los bloques individuales están esquematizados, se procede a probar cada bloque en forma independiente. En líneas generales, los bloques fueron sometidos a pruebas físicas reales; es decir, fueron cada uno implementado y probadas sus entradas, salidas y controles. Para estas pruebas se usaban circuitos adicionales al bloque de ser necesarios. Una vez que el bloque está probado y su

funcionamiento era el esperado, se procede a guardar el bloque convirtiéndolo en un elemento simbólico que puede ser usado en cualquier diseño con sólo llamarlo desde el menú respectivo.

d) Interconexión de bloques

Los bloques individuales son unidos y probados para verificar que las etapas son compatibles y funcionales. Se empiezan desde elementos generales (como contadores) y se va montando cada bloque adicional hasta verificar el funcionamiento.

e) Pruebas generales de bloques

Los bloques unidos son puestos a prueba mediante programas. Se pone énfasis que los resultados esperados se den sin ninguna contrariedad. Ésta prueba y las pruebas anteriores son ejecutadas exclusivamente en la ***FPGA***. Al ser regrabable sin restricciones (aunque volátil), permite depurar los diseños y realizar pruebas completas.

f) Acomodo de bloque experimental en unidad definitiva.

Una vez que los bloques experimentales se encuentran depurados, se procede a elegir la ***CPLD*** donde se alojarán los prototipos. Esto es posible realizarse mediante el propio programa de diseño, el cual permite probar si un diseño cabe dentro de un determinado modelo o puede autoseleccionar alguno que calce (la mejor opción según el programa).

Cuando las pruebas sean culminadas en forma satisfactoria, se pasará a la etapa de diseño y elaboración de placas impresas y unidades operativas. Con esto se culmina todo el proceso de diseño y se podrá tener un modelo operativo.

5.2 Prototipo Experimental de *SASTI – IA*

El Prototipo Experimental de ***SASTI – IA*** se ha diseñado teniendo en cuenta las consideraciones de diseño expuestas en el capítulo correspondiente. El sistema está pensado para emular al ***Amauta MCh-1*** en cuanto a capacidades operativas, mas no en capacidad de programación por usuario. Se usará una ***PseudoROM*** para emular una RAM 7489 ya programada para realizar una secuencia finita de movimientos repetitivos.

5.2.1 Palabra de Memoria

Para esbozar el primer Prototipo Experimental, se deberá considerar el tamaño de la ***Palabra de Memoria***. El modelo original tenía una RAM de 4 bits, la cual era suficiente para los actuadores que usaba. El prototipo, que pretende emularlo, también deberá tener 4 bits; pero, como un ejemplo para corroborar las ecuaciones propuestas, éstas serán desarrolladas. Para el prototipo se usarán 2 motores DC con 2 bits de control cada uno (Control On/Off con sentido de giro); así, usando la primera ecuación se tiene que:

$$\mathbf{P_M = m_1 + m_2} \quad \text{y} \quad \mathbf{P_M = 4}$$

Así, se considera que el tamaño de la ***Palabra de Memoria*** es de 4 bits; lo cual corrobora que el diseño del ***MCh-1*** estaba en lo correcto.

5.2.2 Tamaño de Memoria

A continuación, se debe considerar la extensión de la memoria. En éste caso, ambos actuadores tienen las mismas características y al actuar en paralelo, se considerará sólo el tiempo de actuación de uno de ellos. Sin embargo, por consideraciones experimentales, se ha tomado en forma arbitraria que t_P sea igual a un segundo.

Para estar de acorde al diseño original, se ha decidido dotar al Modelo experimental de SASTI – IA con una memoria de 16 posiciones, con lo que se estaría hablando de un tiempo total efectivo de 16 segundos de ejecución. Si bien éste tiempo puede parecer

corto, para fines demostrativos es suficiente. Adicionalmente, esto permite usar un bloque contador ya existente con lo cual se simplifica el diseño primario. Así, el tamaño final de Memoria será de 64 bits (16 x 4 bits), suficiente para esta etapa.

5.2.3 Instrucciones

Las instrucciones que manejará son simplemente las relacionadas a la activación o desactivación de sus actuadores, con el adicional de poder efectuar un cambio de sentido de giro. Así, la única instrucción disponible estará dada mediante el siguiente formato:

M1 , < Acc > : M2 , < Acc >

Donde **M1** y **M2** son los actuadores a usar (en éste caso, motores DC) y **Acc** es la acción a llevar: giro a la derecha, giro a la izquierda o apagado. Este formato de ***Instrucción*** se ha preferido al clásico ***Adelante***, ***Atrás***, ***Derecha*** e ***Izquierda***, ya que permite un manejo más especializado del sistema. Avanzar y retroceder son acciones indiscutibles; sin embargo, los giros a derecha o izquierda pueden ser realizados mediante giros sobre su propio eje o pivoteando con una de sus ruedas, por lo que el giro sería en arco.

A	B	C	D		Acción
0	0	0	0		Detenido
0	0	0	1		Motor 0 derecha
0	0	1	0		Motor 0 izquierda
0	0	1	1		Detenido
0	1	0	0		Motor 1 derecha
0	1	0	1		Motor 0 y 1 derecha
0	1	1	0		Motor 0 izquierda, Motor 1 derecha
0	1	1	1		Motor 1 derecha
1	0	0	0		Motor 1 izquierda
1	0	0	1		Motor 0 derecha, Motor 1 izquierda
1	0	1	0		Motor 0 y 1 derecha
1	0	1	1		Motor 1 izquierda
1	1	0	0		Detenido
1	1	0	1		Motor 0 derecha
1	1	1	0		Motor 0 izquierda
1	1	1	1		Detenido

Tabla de Instrucciones. Por efecto del controlador de motores, se tienen posiciones repetidas que implican la posibilidad de implementación de *DB*

En Resumen, el Sistema maneja 2 actuadores mediante un bus de datos de 4 bits.

La capacidad total de memoria es de 64 bits (16 x 4 bits). Tiene un tiempo de ejecución máxima de 16 segundos. Su única instrucción, de tipo compuesta. No cuenta con instrucciones de reinicio de secuencia, por lo que debe usar toda su capacidad de memoria antes de reiniciar el proceso.

5.2.4 Contador de Programa (C_P)

El C_P estará implementado mediante un bloque contador en base a un circuito contador ***TTL*** 74193 de 4 bits. Si bien es posible desarrollar un contador específico usando ***VHDL***, se ha preferido la opción de usar el bloque ya existente con miras de futuras alteraciones. El contador en sí permite carga de datos de pre configuración de conteo, posibilidad de conteo ascendente y descendente, reset, salidas de que indican cuando la cuenta se rebalsa. Dado este abanico de posibilidades, se dejarán habilitadas las que son convenientes (por ahora sólo conteo ascendente y reset) y las demás se dejarán en forma latente para futuras pruebas. Ya que el programa suele depurar las etapas que se detectan no funcionan, no hay muchos problemas de recursos.

5.2.5 Sistema de Reset

El sistema de reset estará conformado por un simple interruptor; de preferencia, ***SPST*** (***Single Pole Single Throw***, un polo, una posición) con los elementos necesarios para mantener estados y eliminar el problema de rebote.

5.2.6 Conclusiones

Para aplicaciones que requieran grandes tiempos de ejecución, el sistema se vuelve absolutamente inviable: el tamaño de la memoria necesaria lo vuelve absolutamente impráctico.

Mientras mayor sea el número de actuadores involucrados y mayor sea el número de bits necesarios por actuador, es muy probable que el tamaño de la ***Palabra de Memoria***

sea tan descomunal que para fines operativos sea en definitiva absolutamente impráctica.

El SASTI – IA puede llegar a ser útil para aplicaciones muy pequeñas en las cuales no se requieran tiempos de ejecución grandes, que use pocos actuadores y tenga tiempos globales pequeños. Sólo bajo éstas consideraciones se podría tener ésta arquitectura inicial como aceptable.

5.3 Prototipo Experimental SASTI – IB

El Prototipo Experimental del ***SASTI – IB*** fue diseñado pensando en un móvil que tenga como actuadores 2 motores DC, use 2 bits de control y que pueda realizar las siguientes acciones como mínimo: Avanzar, Retroceder, Detenerse, Giro a Derecha y Giro a Izquierda.

5.3.1 Determinación de la *Palabra de Memoria* e Instrucciones

Primeramente se debe considerar que el número de ***Instrucciones*** o ***Acciones de Programa*** son en total 5. Para tener 5 instrucciones son necesarios 3 bits, con lo cual se tendrían 3 ***Instrucciones Libres.*** Si una de ellas se usa como ***Reset***, se tendrían 2 instrucciones sobrantes. Para fines de uso, se configuran como 2 salidas de tipo ***On/Off*** genéricas disponibles.

Con esto, se determina que el sistema experimental cuenta con una memoria cuya ***Palabra de Memoria*** es de 3 bits y puede ejecutar 8 ***Acciones de Programa***.

Por cuestiones de comodidad, se usará un bloque estándar de 4 bits de datos.

5.3.2 Tamaño de la Memoria

En el caso experimental, no se ha tenido en cuenta alguna consideración en especial para determinar la extensión de memoria. La misma frecuencia de reloj se ha colocado en 1 Hz como comodidad. Se ha tomado un bloque contador estándar de 8 bits en forma

arbitraria, lo cual garantiza unos 4 minutos y 16 segundos de tiempo máximo de ejecución.

5.3.3 Tamaño del Bus de Salida

El ***Bus de Salida*** soportará 2 bits por cada actuador, 2 bits adicionales para elementos genéricos y 1 bit para reset. Es decir.

$$T_{B_S} = 2 + 2 + 2 + 1 \quad ; \quad T_{B_S} = 7\, bits$$

En Resumen, el Sistema maneja 2 actuadores mediante 2 bits de control. La capacidad total de memoria es de 1 Kb (1024 bits, 256 x 4 bits). Tiempo Máximo de Ejecución: 4 minutos y 16 segundos. Posee 8 Instrucciones efectivas de Acción. Capacidad de reinicio, lo que le permite programas de tamaño variable ejecutables en forma cíclica.

5.3.4 Contador de Programa (C_P)

El C_P para ésta etapa va a ser modificado. Se crea un bloque que internamente manejará dos contadores 74193 en cascada para lograr un contador de 8 bits con características de conteo pre configurable, conteo ascendente o descendente y reset. Este bloque será grabado como un elemento que podrá ser llamado para aplicaciones posteriores. En ésta etapa, se considerará sólo el conteo ascendente y el reset.

5.3.5 Sistema de Reset

El sistema de ***Reset*** estará implementado de dos formas. La primera es un ***Reset por Software*** (Sof_R) y un reset manual. Mediante una compuerta se permitirá a cualquiera de las opciones generar un reset al sistema y reiniciar su operación desde el principio. El Sof_R estará implementado directamente desde el ***Decodificador de Instrucciones***; mientras que el reset manual estará implementado mediante un reset y una red anti rebote sencilla.

5.3.6 Conclusiones

Mejora el aspecto relacionado al tamaño de la ***Palabra de memoria***. El tamaño del ***Bus de Salida*** es independiente de ésta y sólo dependerá del número de bits requeridos para la acción.

Al estar referido a la arquitectura inicial, mantiene el problema de manejo ineficiente de la capacidad de memoria: está ligada a la velocidad de reloj.

Por cuestiones de tamaño de bloque de memoria, se ha usado uno que tiene 4 bits de bus de datos. Por ello, se puede decir que el sistema experimental hace uso parcial de la capacidad de la memoria.

Al igual que en el ***SASTI – IA***, su uso está condicionado para pequeñas aplicaciones en las cuales se pueda considerar aceptable el manejo de memoria que realiza.

5.4 Prototipos Experimentales de clase *Mk-II*

Los sistemas ***SASTI – IA Mk-II*** y ***SASTI – IB Mk-II*** serán implementados simplemente al adicionar un ***MultiClock*** a cada uno de los modelos convencionales. Esto implica hacer unas ligeras modificaciones.

5.4.1 Caso del tipo IA Mk-II

El prototipo experimental es en líneas generales el mismo usado en el ***SASI – IA***; pero, con la adición de un sistema ***MultiClock***. El tipo ***IA Mk II*** tendrá una memoria de mayor ***Magnitud***, aunque la ***Extensión*** de la misma no será modificada. Adicionalmente, contará con una red de compuertas para generar el reset mediante ***DB***. Las características generales son:

El Sistema maneja 2 actuadores mediante un bus de datos de 8 bits. 4 bits se encargan del control efectivo de los actuadores y 4 bits se encargan de la temporización.

La capacidad total de memoria es de 128 bits (16 x 8 bits). Éste valor ha sido determinado en forma arbitraria para poder usar la misma estructura del prototipo original adecuando la memoria al aumento de bits de su palabra.

Debido al uso de un ***MultiClock***, se debe considerar en primer lugar la frecuencia de trabajo entrante y los valores de temporización máximos y mínimos con relación a ella.

Presenta 2 instrucciones. La primera es una instrucción convencional de Activación de tipo compuesta y de tipo Temporizada. La segunda es una instrucción de reinicio de secuencia o ***Reset***.

5.4.1.1 Conclusiones

El Sistema ***Mk-II*** soluciona el problema de tiempos de ejecución largos, permitiendo tener una capacidad de memoria bastante adecuada. Esto permite usar la misma estructura del sistema para programas de largo tiempo de ejecución o para programas de tiempos más pequeños. Esto le da una flexibilidad inconcebible en el sistema original.

El tamaño de la ***Palabra de Memoria*** es un factor que no es posible solucionar. Al igual que el sistema original, si el número de actuadores crece al igual que sus bits de control, así lo hará la Palabra. Es por ello que el peligro de volverse impráctico queda latente.

El ***SASTI – IA Mk-II*** puede llegar a ser útil para aplicaciones en las cuales el número de actuadores no sea muy grande o que los bits de control de dichos elementos no sean demasiados. Salvo éste inconveniente, su manejo de tiempos es bastante aceptable y puede considerarse el sistema como apto.

5.4.2. Caso del tipo *IB Mk-II*

El Prototipo Experimental se basa en el mismo diseño del ***SASTI – IB*** convencional añadiéndole el sistema ***MultiClock*** y aumentando los bits necesarios para la Memoria que se use. Esto permite usar las mismas consideraciones de la ***Idea Primaria*** para esta

serie, con sólo una adición de un bloque funcional y aumento en el tamaño de la ***Palabra de Memoria***.

En Resumen:

El Sistema maneja 2 actuadores mediante 2 bits de control.

La capacidad total de memoria es de 2 Kb (2048 bits, 256 x 8 bits).

8 Instrucciones efectivas de Acción.

Capacidad de reinicio, ejecución cíclica.

5.4.2.1 Conclusiones

El Sistema mejora notablemente su desempeño adicionando el sistema ***Multiclock*** para temporizar sus acciones. Conserva las ventajas del tipo ***IB*** normal con las flexibilidad de temporización, permitiendo aprovechar mejor la memoria y obtener por la misma capacidad de memoria la posibilidad de programas más complejos.

Aún conserva el problema el problema del tamaño de bits para la memoria. Desde el primer momento que se indica que no es posible realizar el control total de todas las posibilidades de interacción de los actuadores debido a que conllevaría a la duplicidad de configuración con el ***SASTI IA*** (cuyo manejo lo podía volver bastante ineficiente). Esto no quiere decir que para ciertas aplicaciones específicas, se pueda considerar el uso de este sistema.

La idea de tratar de conservar el tamaño del bus de datos de la memoria en un tamaño relativamente pequeño en comparación al bus de salida ha demostrado que es un enfoque más eficiente en cuanto al diseño de memoria y a la flexibilidad de las salidas, por lo cual se ahondará en este punto para los diseños posteriores.

5.5 Prototipos Experimentales del tipo *IIC y IC*

Los sistemas ***SASTI – IC*** y ***SASTI – IIC*** sólo se diferencian en el hecho que el primero usa un subsistema temporizador basado en el ***MultiClock*** y el segundo usa uno basado en el ***SleepingClock***. Los bloques son comunes y los esquemas son comunes, por lo que se decidió implementar el tipo ***IIC*** y dejar al tipo ***IC*** como esquema preparado; aunque no implementado.

5.5.1 Elementos estándares

Los elementos estándares que se consideran en éste diseño están relacionados a bloques que serán usados por defecto en forma arbitraria con la finalidad de tener un sistema bastante manejable de propósito general. Así, se ha limitado el contador a un conteo máximo de 256 posiciones de memoria, lo cual indica que el sistema es un contador de 8 bits; adicionalmente, se han habilitado sólo sus opciones de conteo ascendente y reset (manual y Sof_R). Igualmente, se usará un bloque estándar ***MultiClock*** de 4 bits. Por último, se ha especificado un ***Decodificador de Instrucciones*** de tipo genérico que tendrá una máxima capacidad de 8 instrucciones. Sin embargo, sólo se usarán 4 operaciones decodificadas y las demás se mantienen como ***Instrucciones Reservadas***.

La idea detrás de usar bloques arbitrarios (sobre todo en el caso del ***Decodificador de Instrucciones***) se justifica por razones de desarrollos futuros. Prácticamente todos los diseños son considerados bancos de prueba y como tales, se ha preferido prevenir futuras mejoras dejando latentes algunas posibilidades.

5.5.2 Diseño de la memoria

La memoria, por condiciones del C_P, se ha limitado a 256 posiciones. Hay que considerar que las condiciones de la memoria cambiarán dependiendo si se opta por usar un ***MultiClock*** o si se implementa un ***SleepingClock***.

En el caso del tipo ***IC***, el bus para la P_M se obtiene de sumar los bits tanto del ***MultiClock***, los bits para el número de instrucciones y el bus de ***Datos*** que se usarán. Es decir:

$$\#_b MultiClock + \#_b Instrucciones + \#_b Datos = P_M$$

$$4 + 3 + 8 = 15$$

Nominalmente, la P_M tendría 15 bits; pero por motivos de estandarización (en múltiplos de 2) se ha optado por designar una memoria con 16 bits. Si bien es cierto esto incluso hace que 1 bit se pierda, esto puede ser relativo. De hecho, si se considera que el sistema expuesto ha sido especialmente desarrollado con miras a futuros experimentos, éste bit queda sólo en un estado ***latente***. Este bit trabaja independientemente de las instrucciones, por lo que podría conformar una función especial. Esta posibilidad será estudiada en desarrollos futuros.

En el caso del tipo ***IIC***, el bus para la P_M se obtiene de sumar los bits tanto del ***SleepingClock***, los bits para el número de instrucciones y el bus de ***Datos*** que se usarán. Es decir:

$$\#_b SleepigClock + \#_b Instrucciones + \#_b Datos = P_M$$

$$8 + 3 + 8 = 19$$

Nominalmente, la P_M tendría 19 bits; pero por motivos de estandarización (en múltiplos de 2) se ha optado por designar una memoria con 20 bits. Igual al caso anterior, aparentemente 1 bit se pierde; aunque de hecho, igual al caso anterior, éste queda en un estado ***latente***.

5.5.3 Sistema de Puertos

El ***Sistema de Puertos*** es exclusivamente de salida. En forma arbitraria se ha considerado usar sólo 4 puertos de los 8 disponibles. Esto no quiere decir que en un

futuro pueda trabajar a toda capacidad. Como base de pruebas, puede ser modificada en cualquier momento que se desee. Cada bit del ***PortLatch*** selecciona un puerto.

5.6 Prototipo Experimental *SASTI – IID*

El sistema ***SASTI – IID*** es la culminación de la arquitectura en cuanto a los objetivos iniciales planteados. Inclusive, va más allá al permitir que este sistema autómata tenga una opción de decisión propia. Al ser un sistema adicional a los originalmente pensados, su tiempo de experimentación ha sido poca; y sin embargo, los resultados han sido satisfactorios. Se ha limitado sus decisiones sólo a saltos por condiciones ya sean de igualdad, mayoría o minoría.

5.6.1 Elementos estándares

En éste tipo, los elementos estándares son en la práctica casi todos los vistos en el prototipo anterior. La única diferencia es en el bloque de puertos (limitados a sólo 2 de salidas) y sus unidades especializadas.

5.6.2 Unidad de Comparación (U_{Comp})

La U_{Comp} es del tipo simple. Sólo va a comparar 2 valores. Uno desde un R_E especializado al que se denomina ***CompLatch*** y desde un único puerto de entrada ***IN Port***. El resultado de la U_{Comp} será llevado a una red lógica para su uso interno.

5.6.3 La Unidad de Saltos

La ***Unidad de Saltos*** está conformada por una red lógica que decodifica las condiciones de la U_{Comp} y las señales del ***Decodificador de Instrucciones***. Estará relacionado a un R_E que será el ***Vector de Salto*** y al que se denomina ***JumpLatch***. Éste registro deberá ser cargado con el valor adecuado cada vez que se desee realizar un salto. Ya que es de tipo estable, sólo cambiará de valor cuando así se le indique; siendo esto un detalle a considerar cuando se intente programar el sistema.

5.6.4 La Instrucción-Dato

Bajo los lineamientos de la ***Instrucción-Dato***, cada instrucción indicará una acción general a ejecutar, ya sea carga al ***BusLatch***, carga al ***PortLatch*** o Salto. El caso del Salto es particular por el hecho que será una instrucción que dependerá del dato asociado para ejecutar un determinado salto. El valor del dato asociado permitirá indicar que deberá efectuar un salto de un determinado tipo. Esto permite ahorrar instrucciones y genera la posibilidad de con sólo una instrucción, se puedan tener múltiples condiciones. En forma extendida, si se implementara toda la capacidad de saltos, por diseño se pueden considerar hasta 8 razones de salto.

CAPÍTULO VI

OBSERVACIONES Y CONCLUSIONES

Éste último capítulo se ha dejado reservado para algunas de las conclusiones generales. Toma algunas conclusiones previas vistas en capítulos anteriores y adiciona conclusiones globales y observaciones de algunos hechos acontecidos al momento de realizar los experimentos. Se espera que éstos sirvan de ayuda a cualquiera que desee experimentar lo expuesto.

6.1 Observaciones y Recomendaciones

a) Algunas de las configuraciones más sencillas (los de clase ***A*** y ***B***, ***Mk-II***) pueden ser implementadas físicamente en placas impresas usando circuitos integrados convencionales (principalmente de lógica ***TTL***) para fines demostrativos o didácticos. Si bien no se recomienda para aplicaciones prácticas debido al tamaño en placa, para los fines antes citados puede ser suficiente. En este caso se debería usar de preferencia una memoria ***NVRAM***, aunque en apariencia esto sea oneroso; la idea de una ***NVRAM*** radica en su capacidad de grabado y conservación de los datos almacenas.

b) La memoria interna ha sido una ***PseudoROM*** sólo por cuestiones prácticas. El uso de una ***RAM*** es posible, para lo cual un sistema ***Programador*** deberá ser implementado. Es posible implementar un programador integrado al propio sistema siempre que el dispositivo sobre al cual se configure tenga los elementos y las salidas necesarias para soportarlo. En caso del uso de una ***FPGA***, la cantidad de elementos lógicos disponibles y el número de salidas (dependiendo siempre del modelo elegido) permite un desarrollo con un sistema programador integrado; aunque haría necesario que el dispositivo tuviera adicionado una ***Memoria de Configuración*** (de preferencia regrabable) para que pudiera configurarse cuando se encienda. En el caso de usar una ***CPLD*** el único

inconveniente es elegir el modelo que permita integrar la mayoría de la lógica. Es recomendable el uso de una memoria ***RAM*** externa al propio sistema ***SASTI*** (de preferencia ***NVRAM***) y un dispositivo decodificador de teclado independiente (existentes en el mercado, como el ***74C922***). La base de tiempos externa, al no ser de valor crítico, puede ser a base de un timer 555 convencional.

c) Se han usado en el diseño bloques que simulan el comportamiento de circuitos integrados existentes comercialmente por la comodidad al momento de diseño y a la posibilidad de ampliación e implementación física. La implementación mediante bloques programados en ***VHDL*** son posibles de implementar. Inclusive, es posible reducir todos los sistemas para diseñar un ajustado exclusivamente a las necesidades de un determinado diseño. Pero, debido a lo antes mencionado, se han escrito y usado sólo algunos bloques en ***VHDL.***

d) Se ha preferido el uso de un archivo de manejo de bloques y diseños esquemáticos en vez de usar un formato exclusivo en texto (es decir, en vez de usar un ***Lenguaje Descriptor de Hardware*** en forma exclusiva). La razón es que un formato exclusivo en texto tiene la tendencia a ser tedioso y poco amigable al usuario, comparándolo a la flexibilidad de un entorno gráfico. Esto queda evidenciado, por ejemplo, en la evolución de la interface de usuario de los ***Sistemas Operativos***: ***DOS***© era un formato más que nada textual, mientras que ***Windows***© es un formato mucho más gráfico; no en vano las interfaces gráficas se han impuesto notablemente sobre las basadas sólo en texto.

e) Originalmente se diseñó un ***Subsistema Temporizador*** de salida directa. Este sistema no presentó problemas para los ***Tipo I*** y ***II***, clases ***A*** y ***B***, subclase ***Mk-II***; igualmente, las pruebas preliminares para los ***Tipo C*** no demostraron problemas apreciables. Sin embargo, para los ***Tipos I*** y ***II***, clase ***D*** se presentó una serie de problemas posiblemente relacionados a ***Metaestabilidad***. El ancho del pulso de salida es estrecho y aparentemente esto provocaba problemas con diseños mucho más elaborados. Entre las fallas figuraban:

- Ejecución de instrucciones no programadas.
- Aparición de datos fantasmas constantes.

- Variación de condiciones programadas.
- Fallas aleatorias por tamaño de programa.
- Fallas aleatorias por adición de elementos no relacionados al propio diseño.
- Fallas por configuración de alta frecuencia o baja frecuencia en forma aleatoria.

Este comportamiento extraño tenía algo en ***común***: la programación por zonas de la ***FPGA***. Se descubrió que para algunos cambios en el diseño, la ***FPGA*** escogía unas zonas de sus elementos internos para configurar el prototipo; siendo la zona en la cual se configuraba dependiente de los elementos adicionales que se establecieran. Se descubrió que algunas configuraciones tendían a trabajar a una frecuencia y no en otras, sólo por la configuración del dispositivo. Este comportamiento errático e indeseable fue solucionado al agregar una etapa de retardo en serie a la salida, la cual permite obtener un pulso de carácter periódico. Esto presenta el inconveniente que agrega un pequeño lapso de tiempo permanentemente, el cual deberá ser considerado en el diseño. Futuros diseños van a estudiar éstos inconvenientes para buscar alternativas de solución.

f) El manejo de circuitos programables presenta detalles críticos al momento de implementar cualquier tipo de diseño, muy en especial con elementos como las ***FPGA*** y ***CPLD***. Trabajar con componentes de más de 40 pines presenta inconvenientes propios que van desde encapsulados distintos a los comúnmente usados en proyectos a nivel estudiantil (máximo los encapsulados ***DIP***), hasta configuraciones propias de pines que dificultan el diseño. La obtención de información técnica es crítica y el cuidadoso diseño de entradas y salidas es necesario para realizar una buena implementación. Es por ello recomendable que se consideren las opciones de diseño disponibles. El diseño multipropósito, la cual configura todos los pines para uso del experimentador, debe ser considerado en el caso de implementar una unidad didáctica; dicha unidad necesitará en alguna oportunidad el uso de cualquiera de las salidas independientemente de su ubicación. Dependiendo si es ***CPLD*** o ***FPGA*** se deberá considerar el número máximo de regrabaciones (caso ***CPLD***) o si requiere de una memoria de configuración (caso ***FPGA***, con lo cual se le incluirá en el diseño); adicionalmente, por comodidad y siempre que el propio dispositivo lo permita, es recomendable el uso de ***sockets para integrados*** para

facilitar los cambios. Por otro lado, si se trata de un diseño dedicado, sólo es necesario habilitar los pines usados; de todas maneras, el uso de un socket también se recomienda en éste caso.

6.2 Conclusiones del Proyecto

***a) Se pudo construir un Sistema Alternativo*.**

El ***S.A.S.T.I.*** como concepto ha resultado operativo. Ha cumplido con la idea inicial de diseñar un sistema cuya forma de manejo en general es distinta a los procesadores convencionales. Su frecuencia de trabajo, irregular por diseño, le permite ejecutar instrucciones a velocidades altas o lentas (se experimentó con frecuencias que podían llegar a 1 KHz o 10 mHz). Puede responder a condiciones de igualdad o proporcionalidad básicas (mayor o menor). A diferencia de los sistemas ***Self-Timed***, éste diseño asíncrono controla el retardo de sus instrucciones por programa sin la necesidad de líneas especiales para ello.

b) Su diseño modular le permite una alta flexibilidad de diseño.

Se ha tenido especial énfasis en realizar los elementos de la forma más modular posible: los bloques definidos pueden ser modificados y ensamblados sin muchos problemas. Esto permite hacer modificaciones y mejoras puntuales sin que el diseño en general se vea necesariamente afectado. Lo único a considerar son las entradas y salidas necesarias para el control de los bloques.

c) Para aplicaciones sencillas se puede implementar la arquitectura inclusive con circuitos integrados de lógica convencional.

Si bien es cierto que se prefiere el uso elementos programables para reducir el tamaño general del dispositivo, en casos particulares el tipo de arquitectura permite la implementación mediante circuitos de lógica comerciales de sistemas completos. Sus tipos y clases le permiten una variedad de configuraciones dependiendo de la aplicación, por lo que con pocos elementos es posible

implementar un sistema bastante aceptable. Para el campo de aplicación propuesto (el campo del entretenimiento), existen aplicaciones que no son críticas; así, el diseño sencillo del sistema propuesto es una alternativa.

d) Su potencial real debe ser explorado.

La presente tesis ha pretendido sólo explorar algunas de las capacidades básicas de la ***Arquitectura S.A.S.T.I*** con fines experimentales y para verificar su funcionamiento y robustez. Se ha tratado de buscar los problemas que pueda tener y se han explorado algunas formas de solucionarlos. Sin embargo, hay bastantes opciones aún no experimentadas tales como ***Respuesta a Estímulo Externo***, ***Memoria Interna***, ***Capacidad Aritmética***, ***Manejo de datos análogos***, entre otras opciones consideradas inicialmente. Ya que es un tipo particular de configuración, es necesario explorar sus posibilidades si se desea expandir sus aplicaciones a campos distintos

GLOSARIO

A

Acción de Programa

Es una ***Sumatoria de Acciones*** que en conjunto desempeñan una acción más compleja para efectos de programación. Por ejemplo, ***Avanzar*** es una acción de programa que puede conformarse mediante la sumatoria de acciones de 2 motores girando en un sentido determinado.

ACP, Aislamiento Circuital por Programación

Es un procedimiento mediante el cual la etapa de potencia o de actuación de un sistema es aislado totalmente del sistema controlador principal mediante un bloque triestado (alta impedancia).

ADD*, *Accionamiento por Dato Directo

Implica el uso de bits no como datos numéricos sino como datos para activación de actuadores. Esto es independiente a la existencia de bits de ***Magnitud de Actuación***. ***ADD*** implica usar tantos bits sean necesarios para el actuador.

B

BusLatch

Es un registro especializado que cumple la función de almacenamiento temporal de datos internos que deban ser enviados a subsistemas internos o a registros externos

(puertos). Mantendrá el dato en forma indefinida hasta que otro dato sea cargado. Forma un bus de datos distinto al de memoria.

C

Contador de Programa, C_P

El un subsistema encargado de definir la posición de memoria a ejecutar y realizar los saltos de programa.

D

Dead Bits, DB, Bits Muertos

Son una combinación binaria de dos o más bits que realmente no conforman una ***ADD*** para el actuador asociado a ellas, o tienen una duplicidad de acción con otra combinación; por lo general, no están relacionados a un actuar sino a una detención. Suele usárseles en sistemas pequeños para implementar instrucciones adicionales a las ***ADD*** básicas.

Decodificador de Instrucciones

Es un bloque que permite que las instrucciones contenidas en la memoria del sistema puedan ser ejecutadas, dando activación a los actuadores asociados a la misma. Su existencia real como subsistema se implementa sólo a partir de los ***SASTI*** tipos ***I*** y ***II*** a partir de la clase ***B***. La clase ***A*** no tiene en forma real un Decodificador de Instrucciones; a lo máximo, un circuito que sirve para aislar la memoria de los circuitos de control de potencia.

E

***Estímulo Externo* (*E^2*)**

Es un tipo de señal que el sistema espera recibir para poder reanudar su funcionamiento habitual; y puede ser por la activación de uno o más bits. Aunque inicialmente no se le ha implementado, es parte de las ideas generales. Para permitir el reinicio normal de actividades del sistema, el ***E^2*** debe ser comparado con una condición que el sistema espera obtener, la cual no necesariamente implica que deba ser una igualdad.

Extensión de Capacidad

Es el número de posiciones de memoria disponibles para el uso del sistema principal. La capacidad está relacionada a las instrucciones que se deseen realizar; aunque, puede ser de un valor arbitrario alto con la finalidad de tener una reserva para modificar el programa.

I

IDCIC*, *Implementación Directa con Circuitos Integrados Conocidos

Este concepto está relacionado a sistemas pequeños (tipos ***I*** y ***II***, clases ***A*** y ***B***, y subclase ***Mk-II***). Se basa en la idea que en casos específicos la implementación de una ***Arquitectura SASTI*** es posible mediante el uso de circuitos lógicos convencionales, en vez de usar circuitos programables (tales como ***CPLD*** o ***FPGA***). Los circuitos a usar dependerán de la lógica a usar (***TTL, CMOS***)

Idea Primaria

Se le llama así al concepto inicial de ***SASTI***: el uso de los bits no como datos numéricos sino como elementos de activación de actuadores. Su tiempo de permanencia era controlado mediante el cambio de posición del C_P gobernado por la señal de reloj externa. Ésta idea evolucionó para añadir le temporización por programa y subsistemas especializados.

Instrucción-Dato (***I-D***)

Forma una versión particular de formato de posición de memoria largo. No es propiamente una ***VLIW*** ya que no comparte el mismo concepto de operación. Una ***I-D*** implica una ejecución al unísono de 3 campos: ***Temporización***, ***Instrucción*** y ***Datos***. La conjunción de sus acciones conforman la instrucción.

Instrucción Decodificada

Es un formato de instrucción para realizar una ***Acción de Programa***. Este tipo de instrucción especifica qué actuadores deben de operar y en qué forma, para lograr la acción deseada.

Instrucciones Libres

Son códigos de instrucción que por alguna razón no han sido todavía implementados. Las razones para esta falta de implementación pueden ser de dos tipos: la primera, es debido a que el número de instrucciones diseñadas es menor al número de combinaciones posibles con los bits de la instrucción; la segunda, es por definir un número arbitrario de bits de instrucciones con la premisa de tener instrucciones de reserva para implementarse en cualquier momento.

Igualdad Absoluta

Es una condición típica de la primera versión del subsistema de temporización de los ***SASTI*** tipo ***II***. Bajo ésta condición, si en forma consecutiva se encuentra un valor de igualdad (para el sistema, ***0***) el sistema no puede ejecutar la emisión de un pulso de reloj ocasionando que el sistema entre en una condición de ***Halt*** (detenido). Esta condición ha sido subsanada en la segunda versión del subsistema de temporización; aunque queda en estudio la posibilidad de reimplantarla para efectos de ganar una instrucción.

M

Magnitud de Actuación

Se refiere a un valor relacionado a la acción que realiza un actuador. Por ejemplo, si es rotatorio la magnitud implica que tan rápido rotara.

Magnitud de Palabra

Se refiere al número de bits que conforman una ***Palabra de Memoria***.

Modularidad del Sistema

Concepto por el cual se realiza un diseño mediante la interacción de bloques independientes, para en conjunto realizar una acción determinada. Cada bloque es independiente, siendo sólo relevantes sus entradas y salidas. Por ello, el diseño interno puede ser variado sin afectar las relaciones con los demás. Igualmente, se refiere al hecho que todos los elementos se diseñarán y probarán por separado, formando una librería de bloques, los cuales pueden ser llamados y unidos para verificar su desempeño.

MultiClock

Subsistema de temporización basado en los retardos ocasionados por un circuito contador trabajando en modo de divisor de frecuencia.

MultiPort.

Configuración de acceso de puertos de salida que permite la salida de datos iguales al mismo tiempo.

P

Palabra de Memoria

Nombre genérico usado para referirse a los bits que conforman el bus de datos de una memoria.

PortLatch

Es el nombre genérico por el cual se denomina a todo registro independiente o que forme parte de un subsistema y que se encuentre unido al bus de datos del ***BusLatch***. Tomará otro nombre sólo para fines de identificación detallada.

PseudoROM

Es un bloque de memoria de sólo lectura (***ROM***) mediante una lógica combinacional. Este mismo concepto fue usado en 1948 para la ***Modified ENIAC***, un precursor de los computadores actuales. Se le implementa mediante una programación de bloque en ***VHDL***.

R

Registros Especializados (*R_E*)

Nombre genérico por el cual se denomina a todo registro unido al bus de datos del ***BusLatch*** y que no sirva para entregar datos fuera del sistema. Por lo general, forman parte de subsistemas especializados y pueden tener nombre particulares dependiendo del subsistema asociado.

Reloj de Temporización, R_T

Este es el ***Reloj Principal*** (***M_C***) del sistema ***SASTI***. Es de tipo irregular y su salida de pulso dependerá de la programación efectuada.

S

Scrapyard

Es una denominación genérica usada para referirse al conjunto de componentes reciclados, equipos para canibalizar y placas circuitales de diverso tipo usados como elementos de reserva para experimentos y proyectos electrónicos. Puede ser desde una simple caja con los elementos antes mencionados, hasta una habitación completa.

Sentido de Actuación

Es un término genérico usado para referirse a la forma en la cual un actuador realiza una acción. Originalmente se usó con la idea de circunscribirlo a actuadores rotacionales (de allí el término ***sentido***). Posteriormente, se generalizó a los demás tipos, conservando el término.

Señal de Aceptación de Temporización (*SAdT*)

Señal que nace de un subsistema de temporización por la cual el $\boldsymbol{C_P}$ realiza una transición desde una posición a otra. Es sinónimo para $\boldsymbol{R_T}$.

Sleeping Clock

Subsistema de temporización que utiliza una comparación de magnitudes para realizar la temporización del sistema.

Sof_R, Software Reset

Reinicio del $\boldsymbol{C_P}$ a sus condiciones iniciales de ejecución mediante una instrucción de programa.

Sumatoria de Acciones

Concepto que define una acción compleja mediante el accionar de varios actuadores en conjunto. Pueden ser todos al mismo tiempo, en un lapso de tiempo específico; o por grupos en lapsos de tiempo propios. No es una sumatoria aritmética. Es sólo una conceptualización de la idea.

T

TIE, Temporización Independiente Efectiva

Tiempo real durante el cual una señal de salida se mantendrá estable y sin cambios, con la finalidad de realizar una determinada acción.

U

Unidad de Comparación (*U_{Comp}*)

Es un subsistema que se encarga de recibir datos externos y obtener de ellos una relación con respecto a un valor base predefinido por programa.

Unidad de Saltos (*U_S*)

Es un subsistema que se encarga de indicar al ***CP*** si debe realizar un salto a una determinada posición de memoria indicada mediante un ***Vector de Salto***. Verifica si las condiciones para realizar el salto se cumplen y de ser así, ejecuta el salto; pero, si no se cumplen las condiciones, realiza una temporización y continúa con la siguiente ***I-D***.

V

Valor de Temporización (***V_T***).

Nombre genérico para indicar la magnitud de temporización. Dependiendo del subsistema usado puede indicar un valor de división o un valor de conteo.

Vector de Salto

Es un valor numérico que se refiere a una posición de memoria física hacia la cual el ***C_P*** deberá cambiar su valor de conteo, con la finalidad de ejecutar instrucciones específicas.

BIBLIOGRAFÍA

Fuentes Bibliográficas

ASHENDEN, Peter J.

VHDL Cookbook. University of Adelaide, Dept. Computer Science, South Australia. 1990. 111 p.

Ebook en: http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/

VHDL-Cookbook.pdf

GARCÍA VILLARREAL, **Jorge R. y GARCÍA VILLARREAL, Juan R.**

CIRCUITOS DIGITALES, Lima: Consorcio Integrado de Electrónica e Informática (CIEI), 1990, 91 p.

GONZÁLEZ, Felipe.

CURSO BÁSICO DE MICROPROCESADORES NIVEL 1, Pereira: Cekít S.A. 1988.

226 p. ISBN 958-9108-23-7 ISBN 958-9108-23-7

LOGSDON, Tom.

ROBOTS: UNA REVOLUCIÓN. Buenos Aires: Ediciones Orbis, S.A. 1987. 224 p.

ISBN 950-614-688-8

mi computer. Editorial Artemisa, S.A. de C.V. 1984, n°79. Puebla (Pue): Editorial Artemisa, S.A de C.V. 1984. 20 p. ISBN 968-22-0114-4

MORRIS MANO, M

Arquitectura de Computadoras, Prentice Hall Hispanoamérica S.A., Enrique Jacob 20, Col Conde 53500 Naucalpan de Juárez, Edo. México.

Nurmi, Jari (ed)

Processor Design: System-on-Chip Computing for ASIC's and FPGA. Tempere University of Technology, Finland, Springer, P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

EBook en: http://www.rapidshareindex.com/Processor-Design-System-On-Chip-Computing-for-ASICs-and-FPGAs_21355.html

Referencias virtuales

ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/ Excepts_ A_ Conversation_with_Gordon_Moore.pdf

http://arantxa.ii.uam.es/~mcts/papers/6)%2059_ortega.pdf

http://datasheets.chipdb.org/Intel/MCS-4/datashts/intel-4004.pdf

http://eelinux.ee.usm.maine.edu/courses//ele498/Lecture%20Material/MEMS-Overview.PDF

http://esd.cs.ucr.edu/labs/tutorial/

http://novella.mhhe.com/sites/dl/free/8448156366/507530/Cap_Muest_Barrientos_ 8448156366.pdf

http://en.wikipedia.org/wiki/Actuator

http://en.wikipedia.org/wiki/Analytical_engine

http://en.wikipedia.org/wiki/Complex_programmable_logic_device

http://en.wikipedia.org/wiki/Delay_line_memory

http://en.wikipedia.org/wiki/Disjunctive_normal_form

http://en.wikipedia.org/wiki/Ferranti_Mark_I

http://en.wikipedia.org/wiki/Fpga

http://en.wikipedia.org/wiki/History_of_computing_hardware

http://en.wikipedia.org/wiki/Hybrid_computer

http://en.wikipedia.org/wiki/Hybrid-core_computing

http://en.wikipedia.org/wiki/Integrated_circuit

http://en.wikipedia.org/wiki/Intel_4004

http://en.wikipedia.org/wiki/Konrad_Zuse

http://en.wikipedia.org/wiki/Manchester_Small-Scale_Experimental_Machine

http://es.wikipedia.org/wiki/Matem%C3%A1ticas

http://en.wikipedia.org/wiki/Plankalkül

http://en.wikipedia.org/wiki/Systolic_array

http://en.wikipedia.org/wiki/Transistor

http://en.wikipedia.org/wiki/VLIW

http://en.wikipedia.org/wiki/Williams_tubes

http://en.wikipedia.org/wiki/Zuse_Z3

http://www.computersciencelab.com/ComputerHistory/History.htm

http://www.cs.ualberta.ca/~database/MEMS/sma_mems/mems.html

http://www.cs.virginia.edu/~robins/Computing_Without_Clocks.pdf

http://www.intel.com/museum/archives/4004.htm

http://www.miky.com.ar/fpga_2004.pdf

http://www.muslimheritage.com/uploads/Automation_Robotics_in_Muslim
%20Heritage.pdf

http://www.tauzero.org/2009/06/computacion-cuantica-un-nuevo-paradigma/

Artículos, Clases y Exposiciones:

CARMEN MOLINA, **Mª y SÁNCHEZ-ÉLEZ, Marcos**

intvhdl.pdf. Universidad Complutense de Madrid, Facultad de Informática.

En: http://www.dacya.ucm.es/marcos/intvhdl.pdf

GSyC

Teoría_Actuadores.pdf, Grupo de Sistemas y Comunicaciones

En: http://www.zerobots.net/manuales/Teoria_Actuadores.pdf

Nota: Al momento de presentar la presente Tesis, el sitio ya no está disponible.

PINO GORDO, Silvia Mª. del

historia.pdf. Universidad Complutense de Madrid, Facultad de Informática.

En: http://www.fdi.ucm.es/profesor/sdelpino/ETC/historia.pdf

MOORE, Simon

Introduction to Computing Without Clocks, University of Cambridge, Computer Laboratory.

En: http://www.cl.cam.ac.uk/teaching/Lectures/compwoclocks/selftimed.4up.pdf

VICENTE RODRÍGUEZ, Antonio José de.

VHDL.pdf. Universidad de Alcalá. Laboratorio de Arquitectura de Computadores.

En: http://atc2.aut.uah.es/~avicente/asignaturas/lac/pdf/VHDL.pdf

WOLF, Edward L.

Nanophysics and Nanotechnology: An Introduction to Modern concepts in Nanoscience

En:

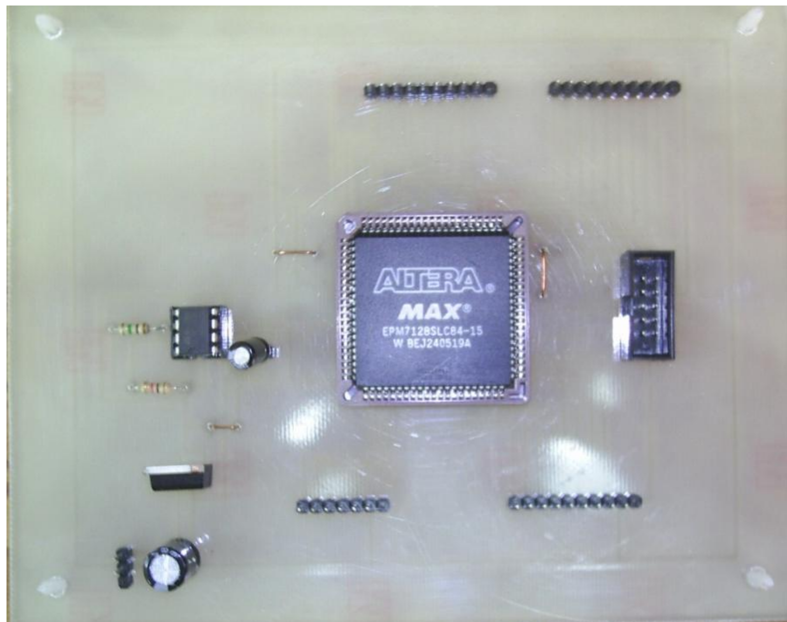
http://books.google.com/books?id=v69g3naaTPEC&pg=PP1&dq=WOLF,++Edward+L.+%2BNanophysics+and+Nanotechnology:+An+Introduction+to+Modern+concepts+in+Nanoscience&lr=&hl=es&cd=1#v=onepage&q=&f=false

ANEXOS

Modulo Experimental para el SASTI IID

1) Elementos físicos

Imagen general del Módulo Experimental para SASTI IID:



Características básicas:

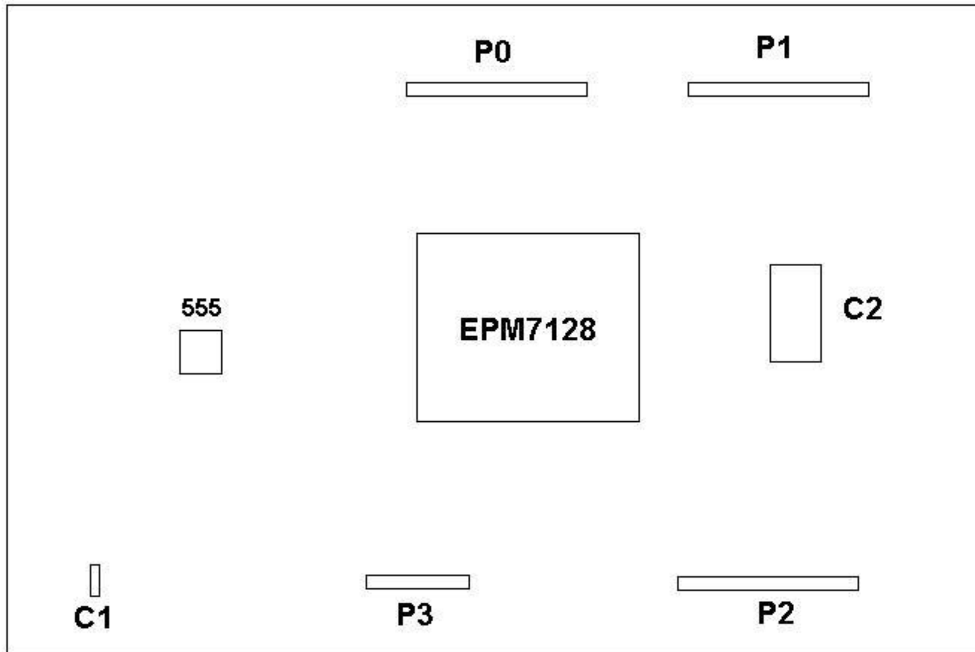
Entrada para fuente de 9 a 12 voltios.

Generador de base de tiempos configurable por hardware mediante un condensador.

Conector dedicado para programación mediante cable ***ByteBlaster***.

3 puertos de 8 bits y 1 puerto de 5 bits con tierra.

Imagen esquematizada del módulo con su relación de partes:



Sus elementos son:

C1: Conector de Fuente. Pin 1: Vcc (superior). Pin2: GND (inferior)

C2: Conector de Programación estándar.

P0: Puerto 0. Conteo de izquierda a derecha. Pines de 0 a 7: datos. Último pin: GND.

P1: Puerto 1. Conteo de izquierda a derecha. Pines de 0 a 7: datos. Último pin: GND.

P2: Puerto 2. Conteo de izquierda a derecha. Pines de 0 a 7: datos. Último pin: GND.

P3: Puerto 3. Conteo de izquierda a derecha. Pines de 0 a 4: datos. Último pin: GND.

EPM7128: CPLD ***EPM 7128SLC84 – 15*** en socket ***PLCC84***. Compatible con CPLD EPM7064SLC84.

555: Generador de tiempos en base a un ***Timer 555*** en modo astable. Configuración de resistencias para ciclo de trabajo de 50%. Configurable por condensador.

2) Configuración de pines de puertos

Las presentes tablas están relacionadas a los pines internos de la CPLD usada y su equivalente de salida en los puertos mencionados anteriormente.

Bit de Puerto	Pin de CPLD
P0 – 0	18
P0 – 1	16
P0 – 2	12
P0 – 3	11
P0 – 4	10
P0 – 5	9
P0 – 6	6
P0 - 7	5

Bit de Puerto	Pin de CPLD
P1 – 0	4
P1 – 1	81
P1 – 2	79
P1 – 3	80
P1 – 4	77
P1 – 5	75
P1 – 6	76
P1 - 7	74

Bit de Puerto	Pin de CPLD
P2 – 0	39
P2 – 1	41
P2 – 2	45
P2 – 3	51
P2 – 4	55
P2 – 5	54
P2 – 6	57
P2 - 7	56

Bit de Puerto	Pin de CPLD
P3 – 0	33
P3 – 1	34
P3 – 2	35
P3 – 3	36
P3 – 4	37

Elemento	Pin de CPLD
Clock	24

3) Conjunto de Instrucciones del SASTI IID

Para poder comprender el funcionamiento de la ***Instrucción Temporizada***, se debe considerar que ésta tiene 2 campos fundamentales: el campo de ***Temporización*** y el campo de ***Instrucción y Datos***.

El campo de Temporización consta de 8 bits que deben ser seleccionado dependiendo de la velocidad a la cual se estime que la instrucción deba ejecutarse. Se les denominará para fines de visualización sólo como TT.

El campo de Instrucción y Datos dependerá bastante de qué instrucción o datos se van a utilizar. Como regla general se denominarán los datos como DD.

A continuación, el listado de instrucciones con la ***Palabra de Memoria*** completa:

a) **BusLatch, < Data >**

Recibe un dato de 8 bits y lo traslada al bus de trabajo ***BusLatch***.

Código de instrucción hexadecimal: TT1DD

b) **IN Portlatch , < Port >**

Envía el dato del ***BusLatch*** al puerto < ***Port*** >.

Código de instrucción en hexadecimal: TT2DD

c) NOP – Delay

Técnicamente, ***NOP - Delay*** está constituido por cualquier valor de instrucción desconocido para el procesador (el sistema ***nunca se colgará ante una instrucción desconocida, sólo la ignorará y no hará nada durante la temporización programada***).

Código de instrucción en hexadecimal: TT<X> DD

d) IN JPLatch

Ingresa el dato del ***BusLatch*** al ***JPLatch***. Este dato se convierte en el ***Vector de Salto***, apuntando a una dirección específica. Debe ser configurado antes de cualquier salto.

Código de instrucción en hexadecimal: TT204

e) IN CompLatch

Ingresa el dato del ***BusLatch*** al registro ***CompLatch*** del a ***Unidad de Comparación***. El dato es comparado con el dato de ingreso del registro ***IN PORT*** para así tomar una decisión de salto.

Código de instrucción en hexadecimal: TT208

f) IN PORT, < Port >

Ingresa un dato de 8 bits al registro ***IN PORT***

Código de instrucción en hexadecimal: TT210

g) JP, < Cond >

Realiza un salto hacia la posición apuntada mediante el ***JPLatch*** si se cumple alguna de las condiciones de comparación.

Código de instrucción en hexadecimal: TT401 (Salto Incondicional)

Código de instrucción en hexadecimal: TT402 (Salto si Igual)

Código de instrucción en hexadecimal: TT404 (Salto si Mayor)

Código de instrucción en hexadecimal: TT406 (Salto si Igual o Mayor)

Código de instrucción en hexadecimal: TT408 (Salto si Menor)

Código de instrucción en hexadecimal: TT40A (Salto si Menor o Igual)

Código de instrucción en hexadecimal: TT40C (Salto si Mayor o Menor)

h) RESET

Realiza un reset del ***Contador de Programa*** haciendo que regrese al principio del programa a ejecutar.

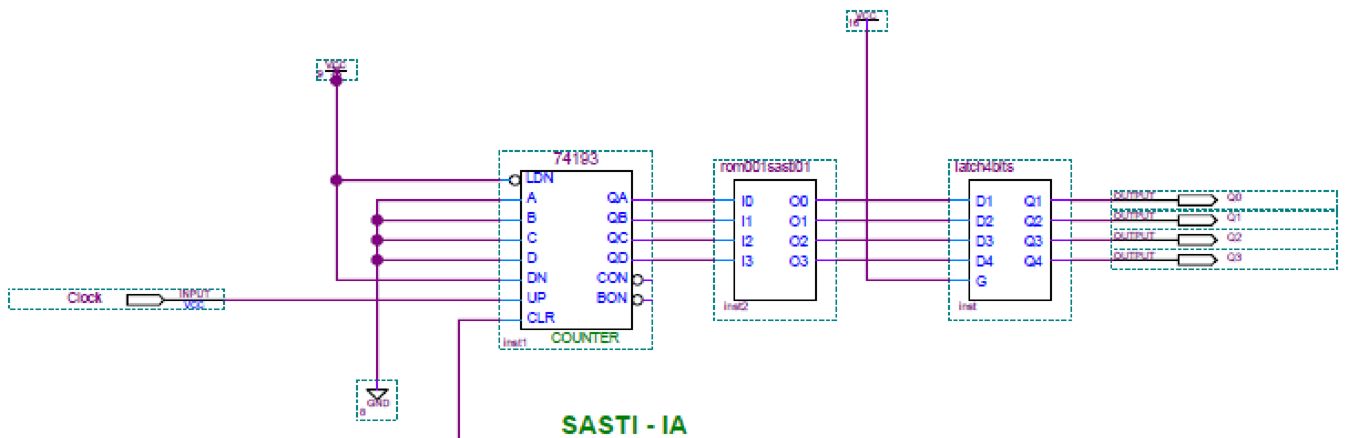
Código de instrucción en hexadecimal: TT700

5) Algunas recomendaciones al programar

- Las instrucciones de carácter interno pueden ser realizadas a la máxima velocidad. Las instrucciones de manejo externo y las instrucciones de ingreso de datos pueden ser manejadas con una temporización (por ejemplo, para tener un ***debouncing***).
- Tener presente que cualquier decisión condicional de salto debe contar antes con la configuración del ***Vector de Salto***.
- Usar la capacidad de ***MultiPort*** para configurar puertos y registros al unísono. Igualmente, para acciones repetidas en puertos distintos.

S.A.S.T.I – IA

- Diagrama General



- Programa de PseudoROM

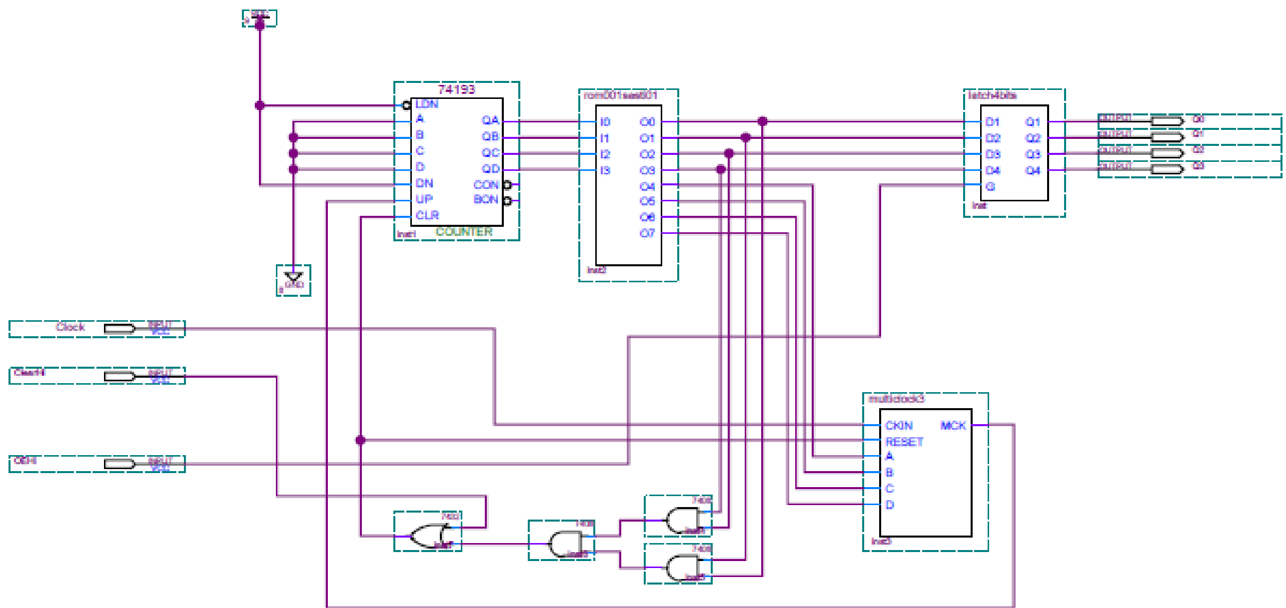
library ieee;

```
use ieee.std_logic_1164.all;

entity sasti01arom001 is

port( I: in std_logic_vector(3 downto 0);

O: out std_logic_vector(3 downto 0)

);

end sasti01arom001;

architecture DatosMem of sasti01arom001 is

begin

process (I)

begin

case I is

when "0000" => O <= "0000"; -- Detenido

when "0001" => O <= "0001"; -- Giro atrás, izquierda

when "0010" => O <= "1000"; -- giro atrás, derecha

when "0011" => O <= "0110"; -- Adelante

when "0100" => O <= "0110"; -- Adelante

when "0101" => O <= "0110"; -- Adelante

when "0110" => O <= "0101"; -- Giro derecha

when "0111" => O <= "1001"; -- Atrás

when "1000" => O <= "1001"; -- Atrás

when "1001" => O <= "1010"; -- Giro Izquierda
```

```
when "1010" => O <= "0110"; -- Adelante

when "1011" => O <= "0110"; -- Adelante

when "1100" => O <= "0110"; -- Adelante

when "1101" => O <= "0101"; -- Giro derecha

when "1110" => O <= "0101"; -- Giro derecha

when "1111" => O <= "1001"; -- Atrás

when others => O <= "0000"; -- Detenido

end case;

end process;

end DatosMem;
```

S.A.S.T.I – IA Mk II

- Diagrama General



SASTI - IA-MkII

Sistema Básico:

Contador de Programa:	Capacidad para 16 Instrucciones, autoreset
Capacidad de Memoria:	16 x 8 bits (128 bits)
Tipo de memoria	PseudoROM (Para el Prototipo)
Accionamiento:	Dato Directo
Temporización:	Uso de Multiclock16
Instrucciones:	Habilitado el Reset por Software
Adicional:	Latch de salida habilitable

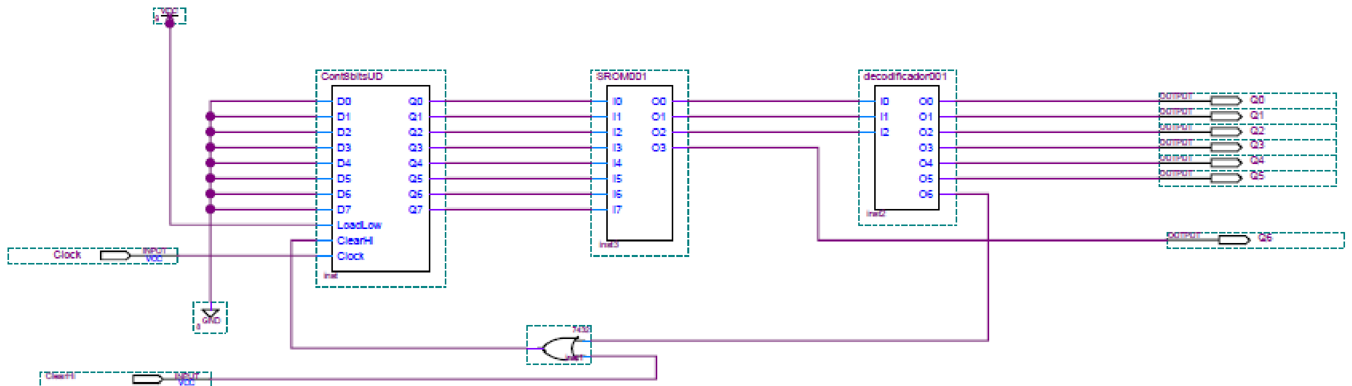
- Programa de PseudoROM

```
library ieee;

use ieee.std_logic_1164.all;

entity sasti01arom001 is

port( I: in std_logic_vector(3 downto 0);

O: out std_logic_vector(7 downto 0)

);

end sasti01arom001;

architecture DatosMem of sasti01arom001 is

begin
```

```
process (I)

begin

-- Formato:

-- [Clock] + [Data/Instrucción]

case I is

when "0000" => O <= "00110000"; -- 00 : 30 : Delayclock3, Detenido

when "0001" => O <= "00110001"; -- 01 : 31 : Giro atrás, izquierda

when "0010" => O <= "00111000"; -- 02 : 38 : Giro atrás, derecha

when "0011" => O <= "01110110"; -- 03 : 76 : Adelante

when "0100" => O <= "00110101"; -- 04 : 35 : Giro Derecha

when "0101" => O <= "01111001"; -- 05 : 79 : Atrás

when "0110" => O <= "00111010"; -- 06 : 3A : Giro Izquierda

when "0111" => O <= "01110110"; -- 07 : 76 : Adelante

when "1000" => O <= "00110101"; -- 08 : 35 : Giro Derecha

when "1001" => O <= "01111001"; -- 09 : 79 : Atrás

when "1010" => O <= "00111111"; -- 0A : 3F : RESET

when others => O <= "00111111"; -- RESET

end case;

end process;

end DatosMem;
```

S.A.S.T.I – IB

- Diagrama General



SASTI - IB

Sistema Básico:

Contador de Programa:	Capacidad para 256 Instrucciones, autoreset
Capacidad de Memoria:	256 x 4 bits (1024 bits)
Tipo de memoria	PseudoROM (Para el Prototipo)
Accionamiento:	Dato Decodificado
Temporización:	Dependiente de la frecuencia de reloj
Instrucciones:	Accionamientos y Reset por Software
Adicional:	Memoria usada parcialmente: 3 bits efectivos

- Programa de PseudoROM

```
library ieee;

use ieee.std_logic_1164.all;

entity sasti01brom001 is

port( I: in std_logic_vector(7 downto 0);

O: out std_logic_vector(3 downto 0)

);

end sasti01brom001;

architecture DatosMem of sasti01brom001 is

begin

process (I)

begin
```

```
case I is

when "00000000" => O <= "0000"; -- Detenido

when "00000001" => O <= "0001"; -- Adelante

when "00000010" => O <= "0001"; -- Adelante

when "00000011" => O <= "0001"; -- Adelante

when "00000100" => O <= "0001"; -- Adelante

when "00000101" => O <= "0000"; -- Detenido

when "00000110" => O <= "0011"; -- Giro derecha

when "00000111" => O <= "0011"; -- Giro Derecha

when "00001000" => O <= "0000"; -- Detenido

when "00001001" => O <= "0001"; -- Adelante

when "00001010" => O <= "0001"; -- Adelante

when "00001011" => O <= "0000"; -- Detenido

when "00001100" => O <= "0100"; -- Giro Izquierda

when "00001101" => O <= "0100"; -- Giro Izquierda

when "00001110" => O <= "0000"; -- Detenido

when "00001111" => O <= "0010"; -- Atrás

when "00010000" => O <= "0010"; -- Atrás

when "00010001" => O <= "0010"; -- Atrás

when "00010010" => O <= "0010"; -- Atrás

when "00010011" => O <= "0111"; -- Reset
```

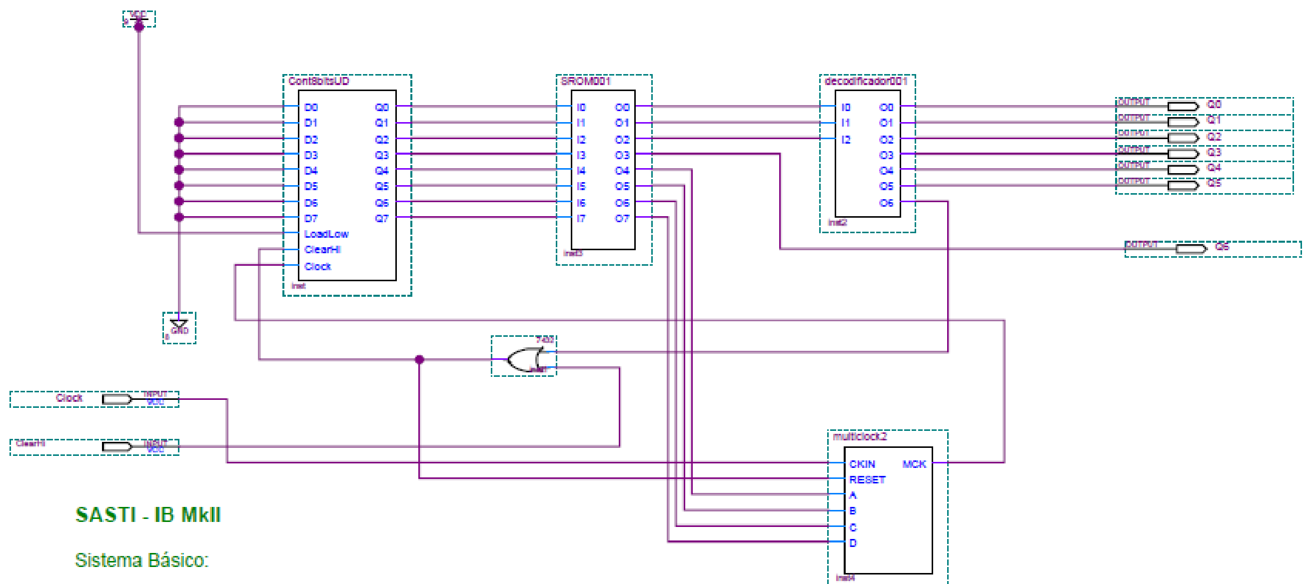
```
when "00010100" => O <= "0110"; -- Adelante

when "00010101" => O <= "0110"; -- Adelante

when "00010110" => O <= "0101"; -- Giro derecha

when "00010111" => O <= "1001"; -- Atrás

when "00011000" => O <= "1001"; -- Atrás

when "00011001" => O <= "1010"; -- Giro Izquierda

when "00011010" => O <= "0110"; -- Adelante

when "00011011" => O <= "0110"; -- Adelante

when "00011100" => O <= "0110"; -- Adelante

when "00011101" => O <= "0101"; -- Giro derecha

when "00011110" => O <= "0101"; -- Giro derecha

when "00011111" => O <= "1001"; -- Atrás

when "00100000" => O <= "0000"; -- Detenido

when "00100001" => O <= "0001"; -- Giro atrás, izquierda

when "00100010" => O <= "1000"; -- giro atrás, derecha

when "00100011" => O <= "0110"; -- Adelante

when "00100100" => O <= "0110"; -- Adelante

when "00100101" => O <= "0110"; -- Adelante

when "00100110" => O <= "0101"; -- Giro derecha

when "00100111" => O <= "1001"; -- Atrás

when "00101000" => O <= "1001"; -- Atrás
```

```
when "00101001" => O <= "1010"; -- Giro Izquierda
when "00101010" => O <= "0110"; -- Adelante
when "00101011" => O <= "0110"; -- Adelante
when "00101100" => O <= "0110"; -- Adelante
when "00101101" => O <= "0101"; -- Giro derecha
when "00101110" => O <= "0101"; -- Giro derecha
when "00101111" => O <= "1001"; -- Atrás
when "00110000" => O <= "0000"; -- Detenido
when "00110001" => O <= "0001"; -- Giro atrás, izquierda
when "00110010" => O <= "1000"; -- giro atrás, derecha
when "00110011" => O <= "0110"; -- Adelante
when "00110100" => O <= "0110"; -- Adelante
when "00110101" => O <= "0110"; -- Adelante
when "00110110" => O <= "0101"; -- Giro derecha
when "00110111" => O <= "1001"; -- Atrás
when "00111000" => O <= "1001"; -- Atrás
when "00111001" => O <= "1010"; -- Giro Izquierda
when "00111010" => O <= "0110"; -- Adelante
when "00111011" => O <= "0110"; -- Adelante
when "00111100" => O <= "0110"; -- Adelante
when "00111101" => O <= "0101"; -- Giro derecha
```

```
        when "00111110" => O <= "0101"; -- Giro derecha

        when "00111111" => O <= "1001"; -- Atrás

        when others => O <= "0000"; -- Detenido

        end case;

    end process;
```

S.A.S.T.I – IB Mk II

- Diagrama General



SASTI - IB MkII

Sistema Básico:

Contador de Programa: Capacidad para 256 Instrucciones, autoreset
Capacidad de Memoria: 256 x 8 bits (2048 bits)
Tipo de memoria PseudoROM (Para el Prototipo)
Accionamiento: Dato Decodificado
Temporización: Uso de Multiclock16
Instrucciones: Accionamientos y Reset por Software
Adicional: Memoria usada parcialmente: 7 bits efectivos

- Programa de PseudoROM

```
library ieee;

use ieee.std_logic_1164.all;

entity sasti01brom001 is
```

```
port( I: in std_logic_vector(7 downto 0);

O: out std_logic_vector(7 downto 0)

);

end sasti01brom001;

architecture DatosMem of sasti01brom001 is

begin

process (I)

begin

case I is

when "00000000" => O <= "01000000"; -- 00 : 40 : Detenido

when "00000001" => O <= "01110001"; -- 01 : 71 : Adelante

when "00000010" => O <= "01000000"; -- 02 : 40 : Detenido

when "00000011" => O <= "01000011"; -- 03 : 43 : Giro Derecha

when "00000100" => O <= "01000000"; -- 04 : 40 : Detenido

when "00000101" => O <= "01110001"; -- 05 : 41 : Adelante

when "00000110" => O <= "01000000"; -- 06 : 40 : Detenido

when "00000111" => O <= "01000100"; -- 07 : 44 : Giro Izquierda

when "00001000" => O <= "01000000"; -- 08 : 40 : Detenido

when "00001001" => O <= "01110010"; -- 09 : 42 : Atrás

when "00001010" => O <= "01000111"; -- 0A : 47 : RESET

when others => O <= "01000111"; -- RESET
```

```
end case;
end process;
end DatosMem;
```

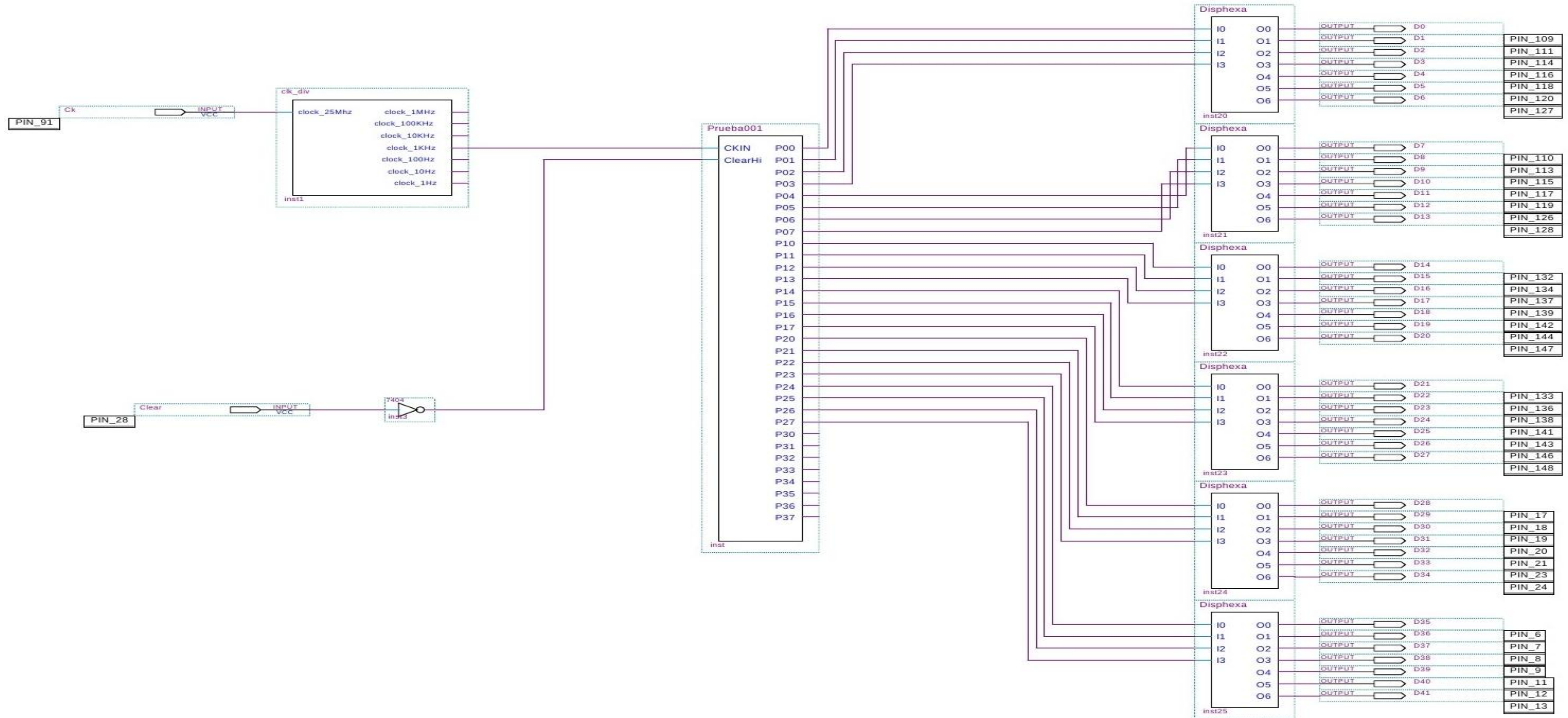
- Programa de Decodificador de Instrucciones

```
library ieee;
use ieee.std_logic_1164.all;
entity decodinst001 is
port( I: in std_logic_vector(2 downto 0);
O: out std_logic_vector(6 downto 0)
);
end decodinst001;
architecture DatosMem of decodinst001 is
begin
process (I)
begin
case I is
when "000" => O <= "0000000"; -- Detenido
when "001" => O <= "0000110"; -- Adelante
when "010" => O <= "0001001"; -- Atrás
when "011" => O <= "0001010"; -- Derecha
```

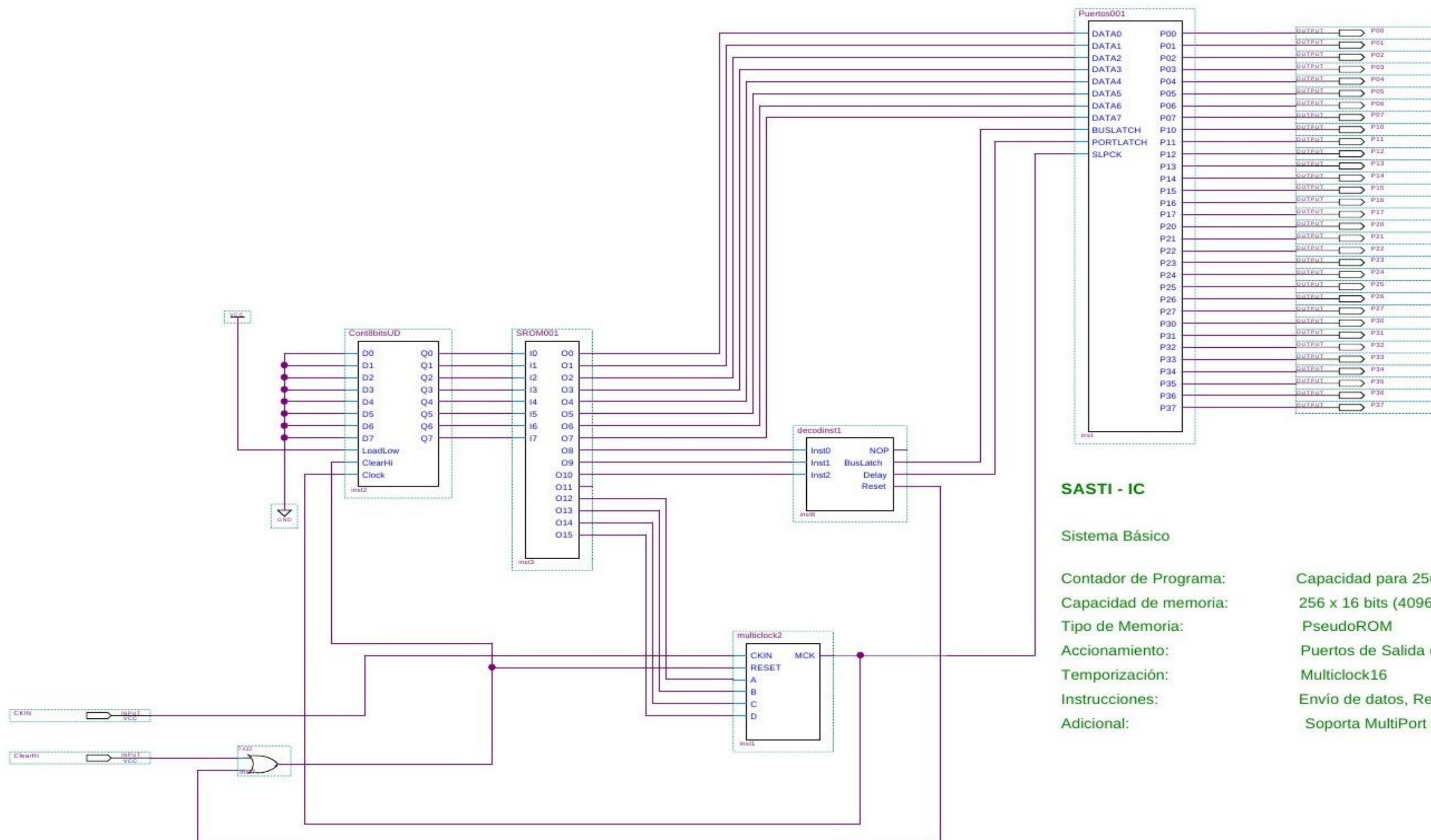
```
when "100" => O <= "0000101"; -- izquierda

when "101" => O <= "0010000"; -- Act1

when "110" => O <= "0100000"; -- Act2

when "111" => O <= "1000000"; -- Reset

when others => O <= "0000000"; -- Detenido

end case;

end process;

end DatosMem;
```

S.A.S.T.I Tipo C

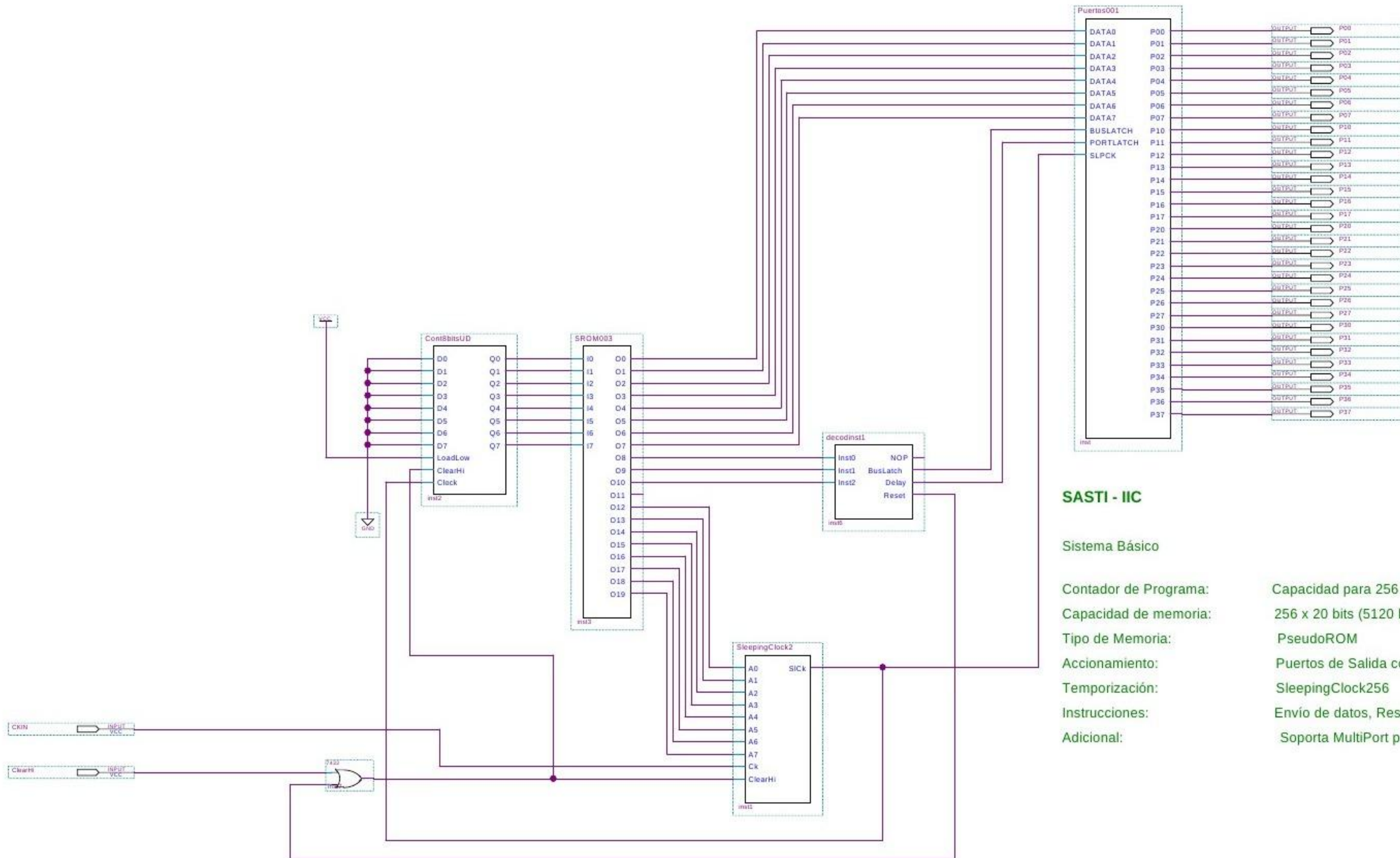
- Circuito General de Pruebas



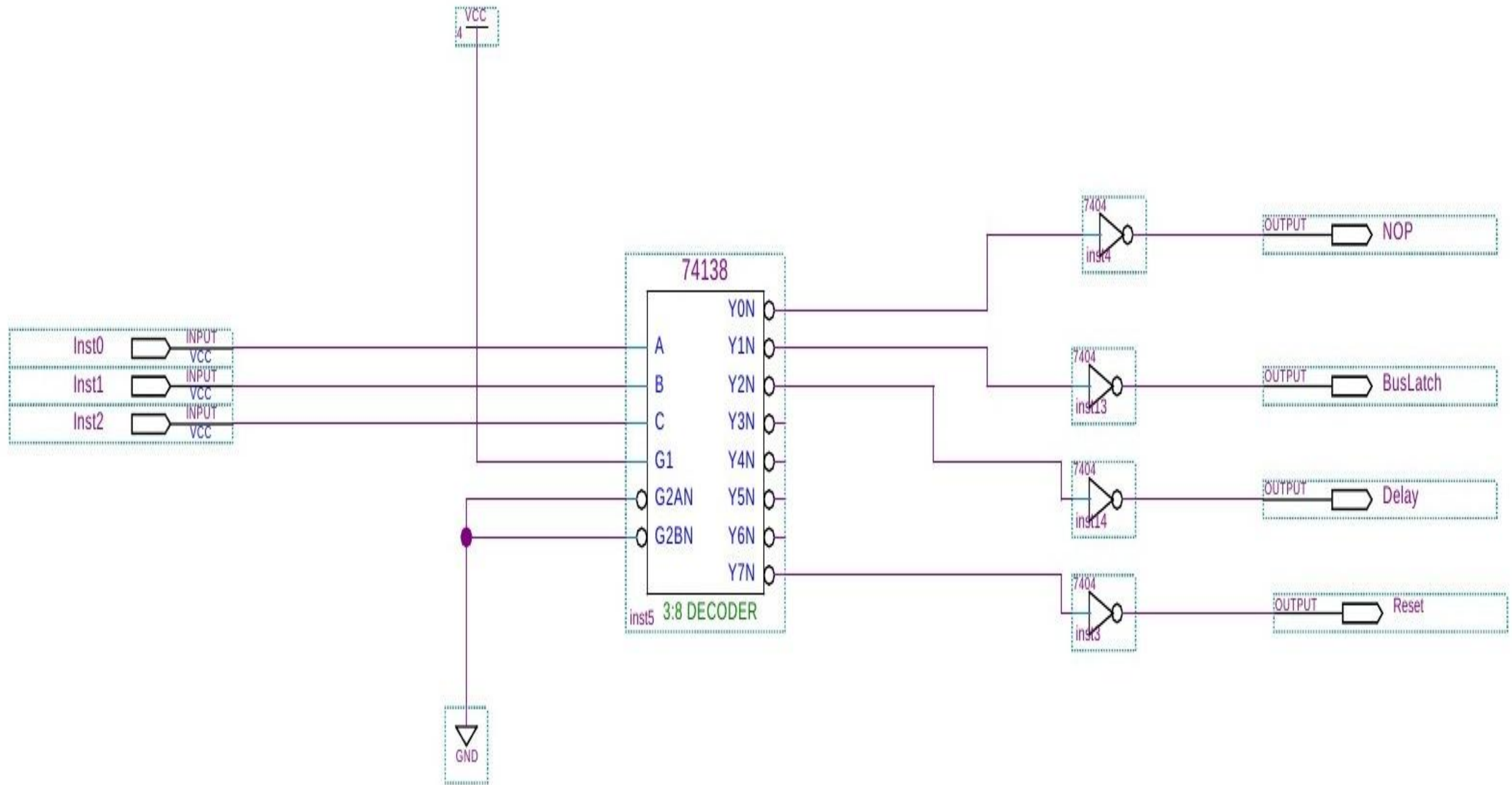
- Diagrama del Tipo IC



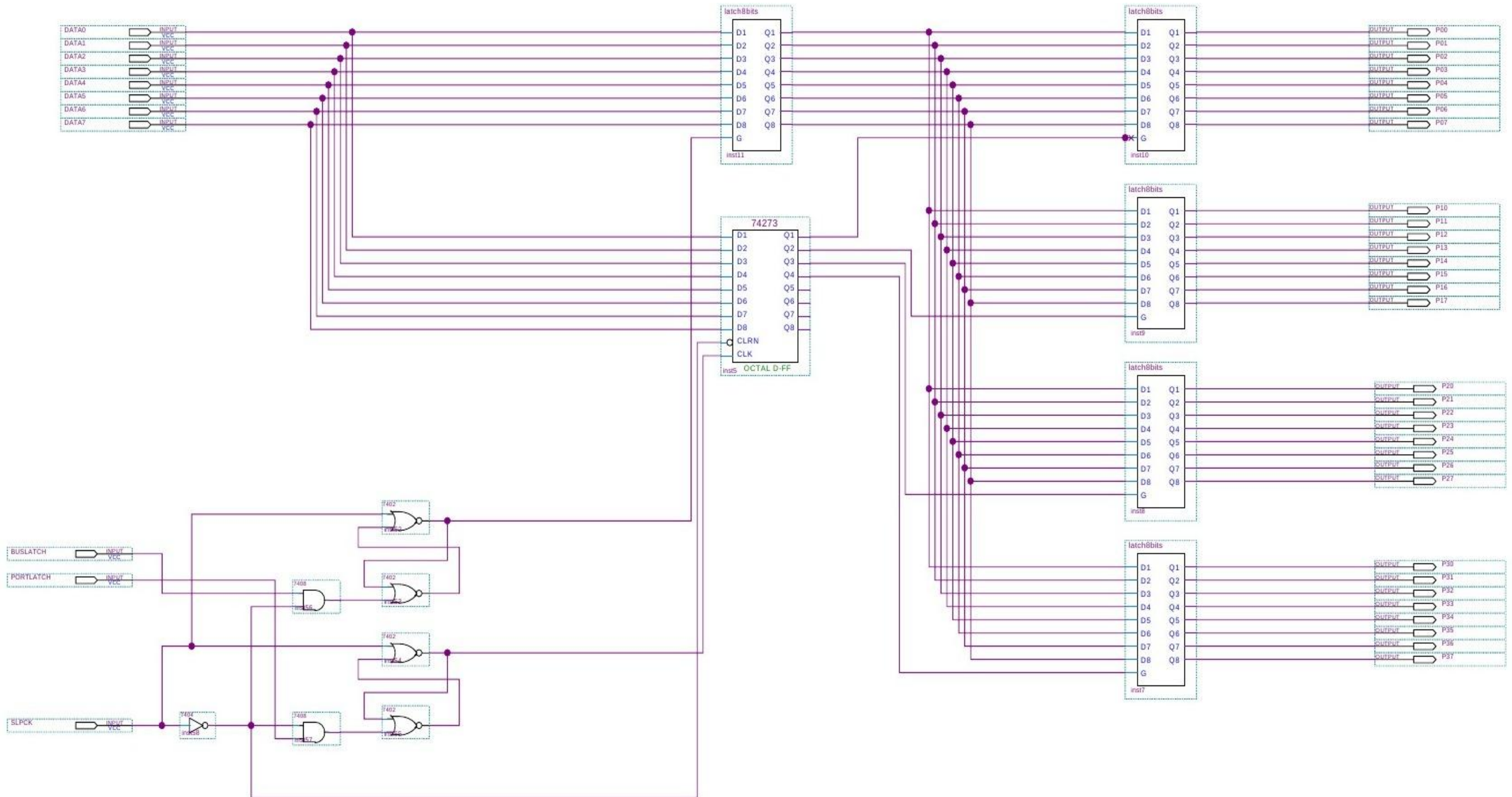
- Diagrama del Tipo IIC



- Diagrama del Decodificador de Instrucciones



- Diagrama de la Unidad de Puertos



- Programa de PseudoROM

```
library ieee;
use ieee.std_logic_1164.all;
entity ROM003 is
port( I: in std_logic_vector(7 downto 0);
O: out std_logic_vector(19 downto 0)
);
end ROM003;
architecture DatosMem of ROM003 is
begin
process (I)
begin
case I is
when "00000000" => O <= "00001010000100000000"; -- 00 02100 : BusLatch, 00H
when "00000001" => O <= "11111110001000001111"; -- 01 FF20F : PortLatch, 0FH
when "00000010" => O <= "11111110000100000001"; -- 02 FE101 : BusLatch, 01H
when "00000011" => O <= "10000000001000000011"; -- 03 80203 : PortLatch, 03H
```

when "00000100" => O <= "00000010000100000011"; -- 04 02100 : BusLatch, 03H

when "00000101" => O <= "10000000001000000110"; -- 05 80206 : Portlatch, 06H

when "00000110" => O <= "00000010000100000111"; -- 06 02107 : BusLatch, 07H

when "00000111" => O <= "10000000001000000110"; -- 07 80202 : Portlatch, 02H

when "00001000" => O <= "00000010000100001111"; -- 08 02104 : BusLatch, 0FH

when "00001001" => O <= "10000000001000000110"; -- 09 80202 : Portlatch, 02H

when "00001010" => O <= "00000010000100011111"; -- 0A 0211F : BusLatch, 1FH

when "00001011" => O <= "10000000001000000110"; -- 0B 80202 : Portlatch, 02H

when "00001100" => O <= "00000010000100111111"; -- 0C 0212F : BusLatch, 3FH

when "00001101" => O <= "10000000001000000110"; -- 0D 80202 : Portlatch, 02H

when "00001110" => O <= "00000010000101111111"; -- 0E 0213F : BusLatch, 7FH

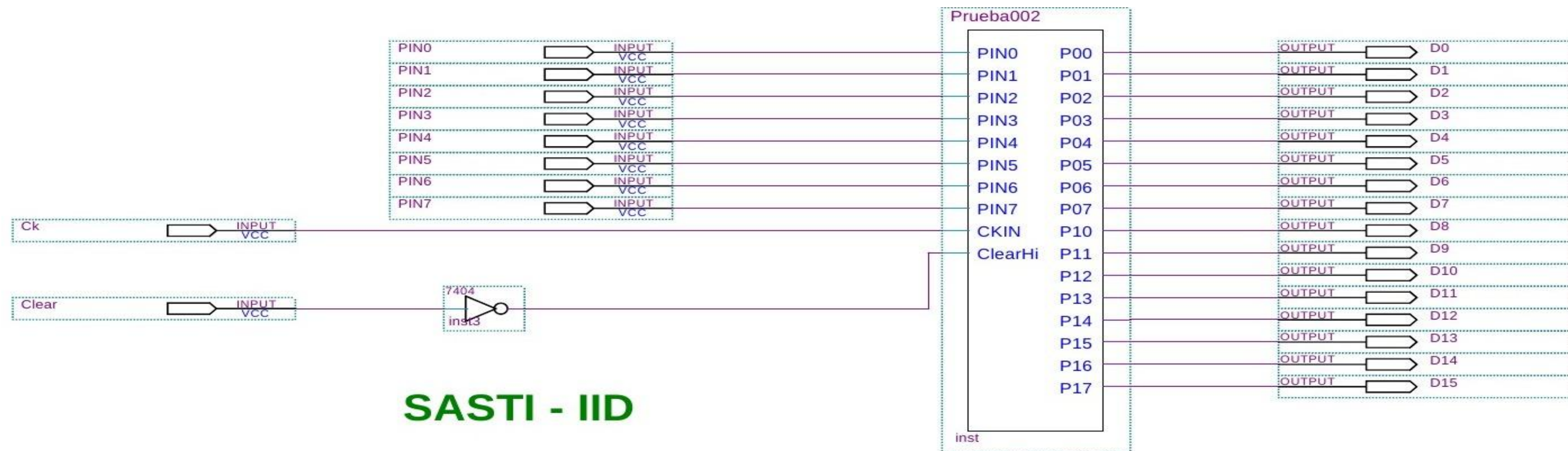
when "00001111" => O <= "10000000001000000110"; -- 0F 80202 : PortLatch, 02H

when "00010000" => O <= "00000010000111111111"; -- 10 021FF : Buslatch, FFH

```
when "00010001" => O <= "10000000001000000110"; -- 11 80202 : PortLatch, 02H

when "00010010" => O <= "00000010000100000000"; -- 12 02100 : BusLatch, 00H

when "00010011" => O <= "11111111001000000111"; -- 13 FE203 : PortLatch, 03H

when "00010100" => O <= "01001100011100000010"; -- 14 4C702 : RESET

when others => O <= "00000000000000000000"; -- Detenido

end case;

end process;

end DatosMem;
```

S.A.S.T.I – IID

- Diagrama General



Sistema Básico

Contador de programa:	**Capacidad para 256 instrucciones, Autoreset**
Capacidad de memoria:	**256 x 20 bits (5120 bits)**
Tipo de Memoria:	**PseudoROM (Para prototipo)**
Accionamiento:	**Instrucción compleja**
Temporización:	**SleepingClock256**
Instrucciones:	**Carga a BusLatch, PortLatch, JPLatch CompLatch, INPORT, HALT, Saltos**

- Diagrama del Procesador

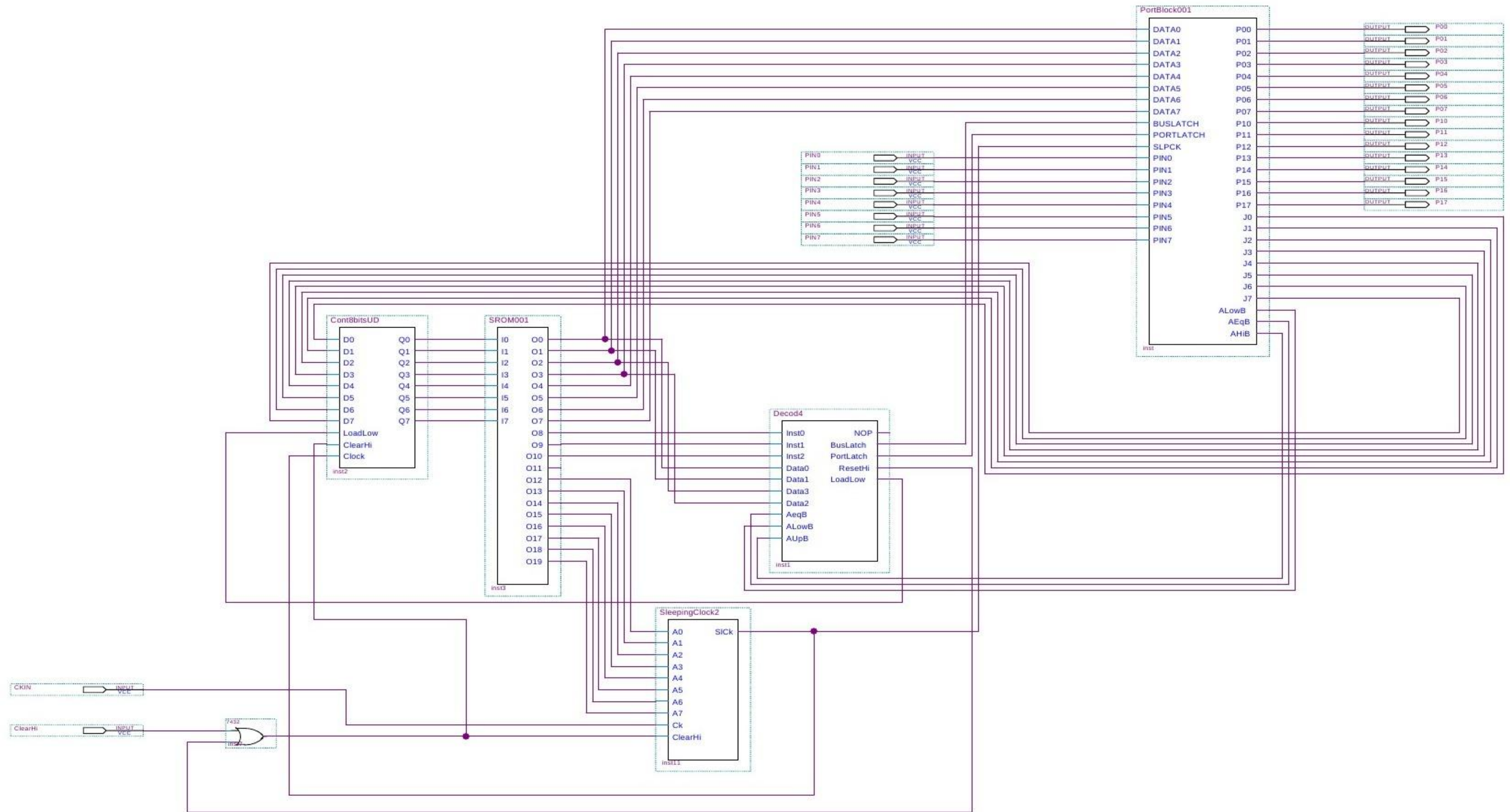
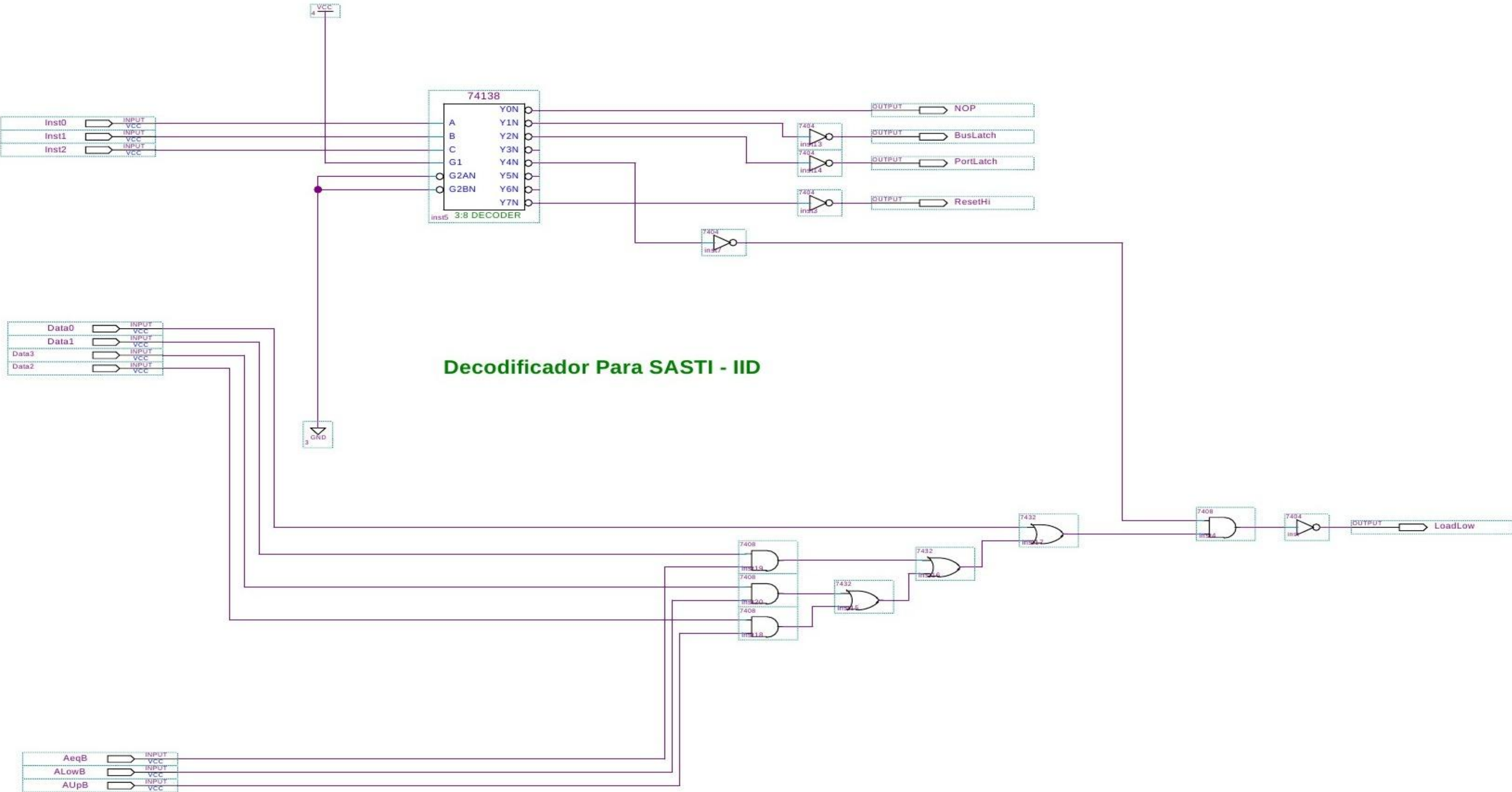
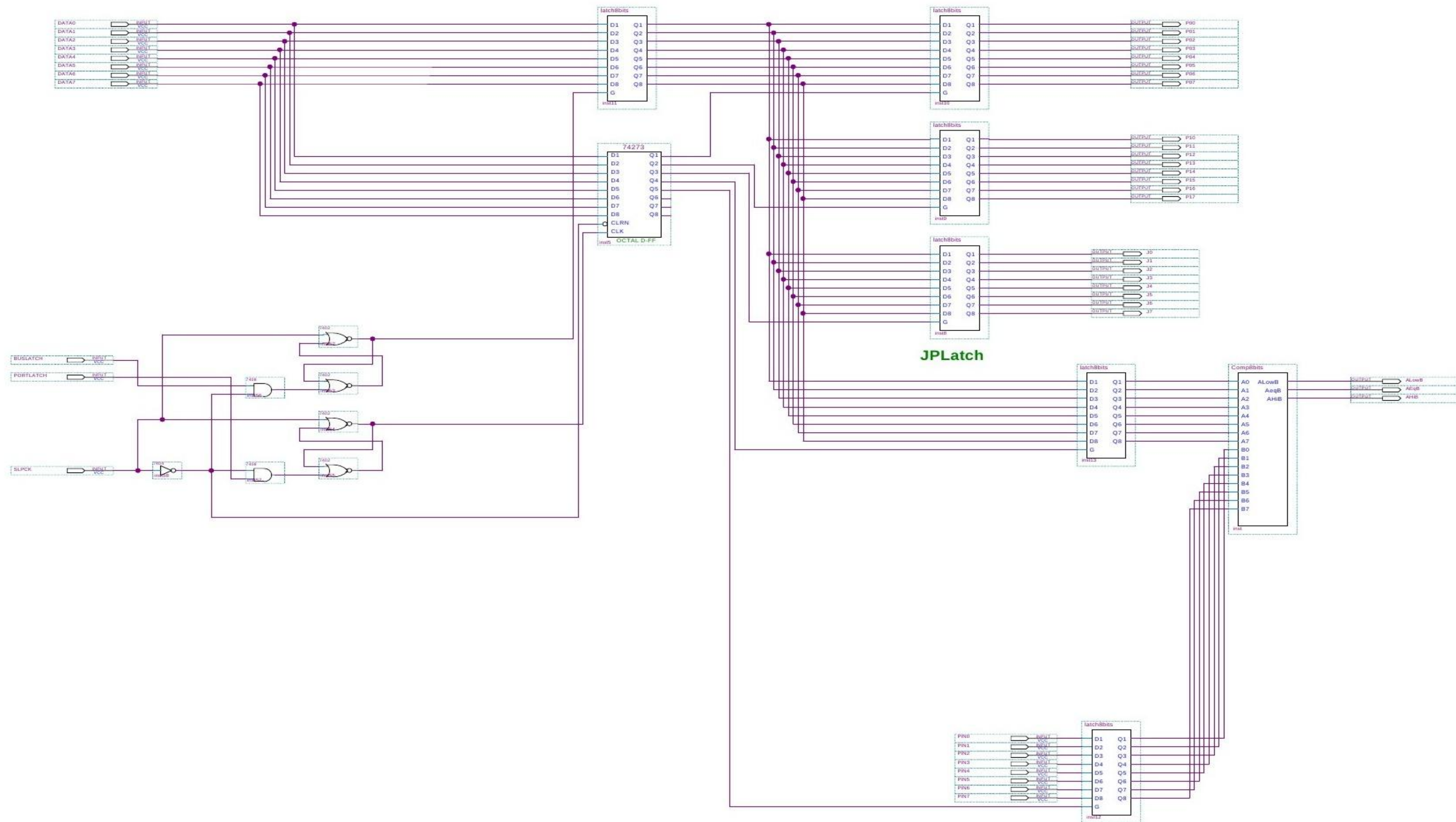


Diagrama del Decodificador de Instrucciones



- Diagrama de la Unidad de Puertos y Decisión



- Programa de PseudoROM

```
library ieee;

use ieee.std_logic_1164.all;

entity ROM002 is

port( I: in std_logic_vector(7 downto 0);

O: out std_logic_vector(19 downto 0)

);

end ROM002;

architecture DatosMem of ROM002 is

begin

process (I)

begin

case I is

when "00000000" => O <= "00000010000100000000"; -- 00 02100 : BusLatch,
00H

when "00000001" => O <= "00000010001000001111"; -- 01 0220F : PortLatch,
0FH

when "00000010" => O <= "01001100000100000001"; -- 02 4C101 : BusLatch,
01H

when "00000011" => O <= "01001100001000000001"; -- 03 4C201 : PortLatch,
01H
```

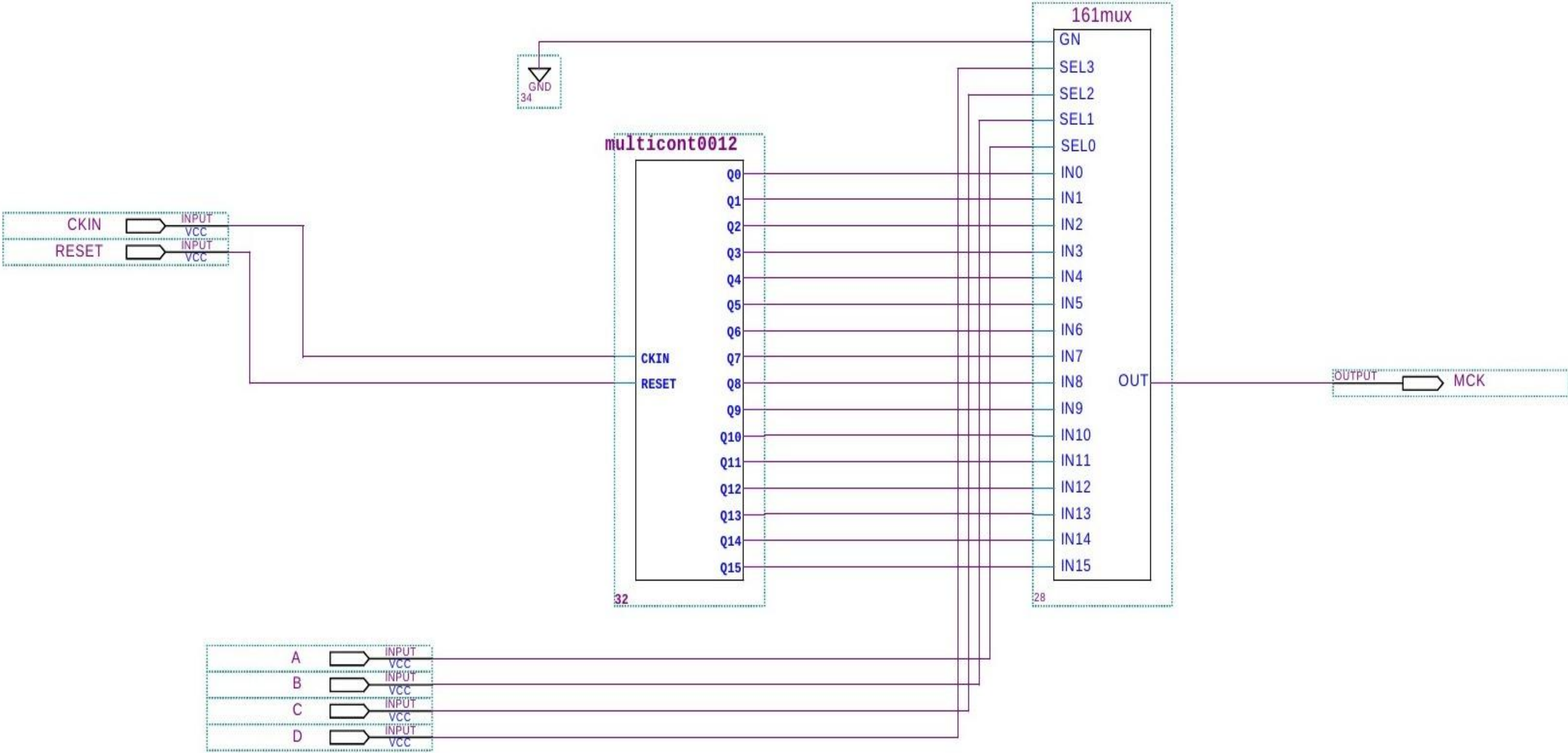
```
when "00000100" => O <= "01001100000100000010"; -- 04 4C102 : BusLatch, 02H

when "00000101" => O <= "01001100001000000001"; -- 05 4C201 : Portlatch, 01H

when "00000110" => O <= "01001100000100000011"; -- 06 4C103 : BusLatch, 03H

when "00000111" => O <= "01001100001000000001"; -- 07 4C201 : Portlatch, 01H

when "00001000" => O <= "01001100000100000100"; -- 08 4C104 : BusLatch, 04H

when "00001001" => O <= "01001100001000000001"; -- 09 4C221 : Portlatch, 01H

when "00001010" => O <= "01001100000100000101"; -- 0A 4C100 : BusLatch, 05H

when "00001011" => O <= "01001100001000000001"; -- 0B 4C202 : Portlatch, 01H

when "00001100" => O <= "01001100000000000000"; -- 0C 4C000 : NOP

when "00001101" => O <= "01001100000100000001"; -- 0D 4C101 : Buslatch, 01H

when "00001110" => O <= "01001100001000000010"; -- 0E 4C202 : PortLatch, 02H

when "00001111" => O <= "01001100000100000010"; -- 0F 4C102 : BusLatch, 02H

when "00010000" => O <= "01001100001000000010"; -- 10 4C202 : Portlatch, 02H
```

```
when "00010001" => O <= "01001100000100000011"; -- 11 4C103 : Buslatch, 03H

when "00010010" => O <= "01001100001000000010"; -- 12 4C202 : Portlatch, 02H

when "00010011" => O <= "01001100000100000100"; -- 13 4C104 : Buslacth, 04H

when "00010100" => O <= "01001100001000000010"; -- 14 4C202 : Portlatch, 02H

when "00010101" => O <= "01001100000100000101"; -- 15 4C105 : Buslatch, 05H

when "00010110" => O <= "01001100001000000010"; -- 16 4C202 : Portlatch, 02H

when "00010111" => O <= "01001100000000000000"; -- 17 4C000 : NOP

--when "00011000" => O <= "01001100011100000000"; -- 18 4C700 : RESET

when "00011000" => O <= "00000100000100000100"; -- 18 4C104 : BusLatch, 04H

when "00011001" => O <= "00000100001000000100"; -- 19 4C204 : Portlatch, 04H

when "00011010" => O <= "00000100000100001000"; -- 1A 4C108 : BusLatch, 08H

when "00011011" => O <= "00000100001000001000"; -- 1B 4C208 : PortLatch, 08H

when "00011100" => O <= "00000100001000010000"; -- 1C 4C210 : PortLatch, 10H

when "00011101" => O <= "00000100010000000100"; -- 1D 4C400 : JP, INC

when "00011110" => O <= "00001000011100000000"; -- 1E 4C700 : RESET
```

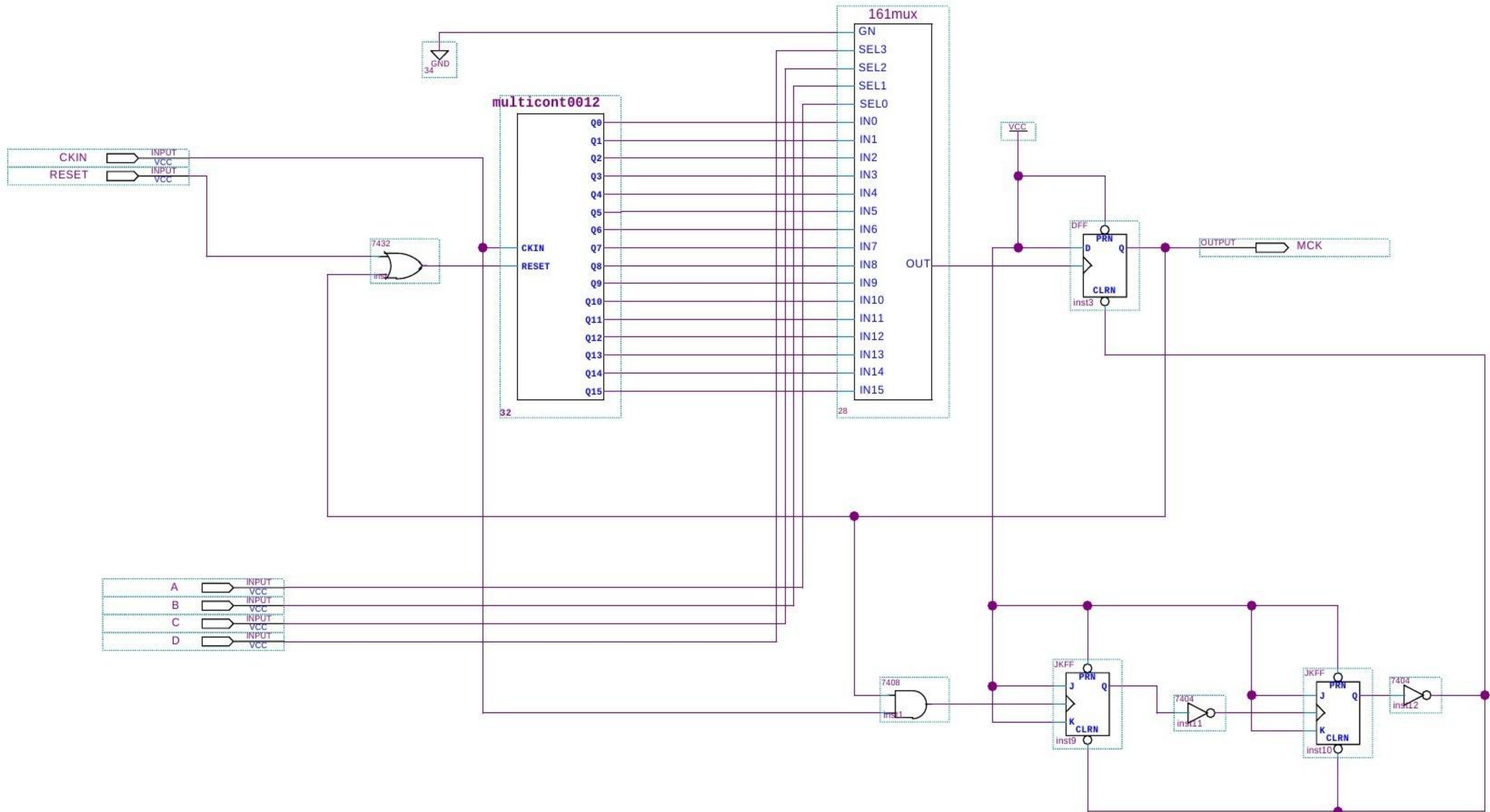
```
when others => O <= "00000000000000000000"; -- Detenido

end case;

end process;

end DatosMem;
```

Unidad MultiClock

- MultiClock original

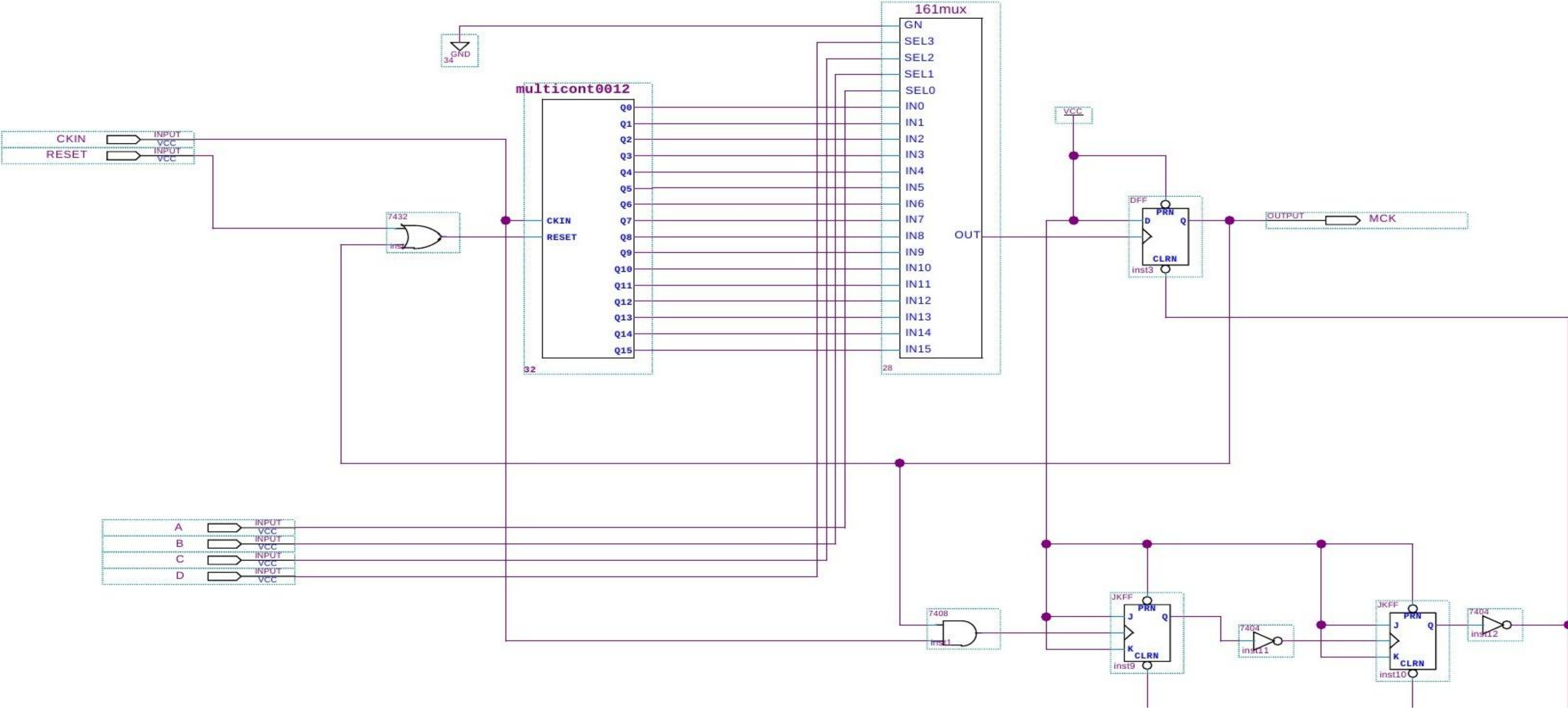


- MultiClock con autoreset



Unidad SleepingClock

- SleepingClock original



- SleepingClock con autoreset