Un método simple para pasar de un algoritmo a un modelo en VHDL

Gerardo Muñoz[1]

RESUMEN

El presente artículo describe una metodología para pasar de un modelo algorítmico a un modelo en VHDL sin tener que describir directamente el hardware sobre el que corre el algoritmo. La solución se basa en modelar el algoritmo con una Máquina de Estados Algorítmica (ASM) y posteriormente modelar este ASM funcionalmente en VHDL. Durante el desarrollo del tema también da una introducción a VHDL.

Palabras claves: VHDL, ASM, FSM, Algoritmo

ABSTRACT

The following paper describes a methodology to convert an algorithmic model into a VHDL, without having to describe directly the hardware where the algorithmic runs. The solution is based to model the algorithmic with an Algorithmic State Machine (ASM) and furthermore model this ASM into a functional VHDL. During the development of this subject we are being introduce into VHDL.

Key Words: VHDL, ASM, FSM, Algorithm

I. INTRODUCCIÓN

La mayoría de textos sobre VHDL (VHSIC *Hardware Description Language*) como: [1],[2],[3] y [4]presentan una forma de diseño pensando en el HW (*Hardware*) que se desea construir. Sin embargo, muchas veces la solución de un problema es planteada por medio de un algoritmo y no es evidente el HW que permite ejecutar ese algoritmo. Trabajos anteriores [5] presentan cómo desarrollar un HW a partir de un algoritmo, mediante un trabajo algo dispendioso y que puede ser necesario repetirlo varias veces con el fin de optimizar.

Todo modelo en VHDL se divide en dos partes: la primera es la Entidad y la segunda es la Arquitectura.

El presente artículo presenta una metodología mucho más simple y segura para escribir modelos directamente en VHDL a partir de un algoritmo. Se incluye además una breve descripción de VHDL, para el lector que no este muy familiarizado.

Tomando como ejemplo un multiplicador se introducirá en el capítulo 2 los componentes de VHDL y posteriormente se realizarán varia implementaciones: una combinacional, en el capítulo 3, la cual es muy sencilla pero genera un circuito muy grande; en el capítulo 4 se presenta un algoritmo para multiplicar y las razones por las cuales no se puede implementar directamente. El capítulo 5 muestra un diseño estructural, en el cual se genera un circuito más pequeño pero es mucho más dispendioso su diseño; y el capítulo 6 finalmente presenta el diseño funcional, basada en los diagramas ASM, es más sencilla de implementar que el diseño estructural y genera un circuito más pequeño que el diseño combinacional.

II. COMPONENTES DE VHDL

Todo modelo en VHDL se divide en dos partes: la primera es la Entidad que define las entradas y salidas y la segunda es la Arquitectura que define como se realiza la operación.

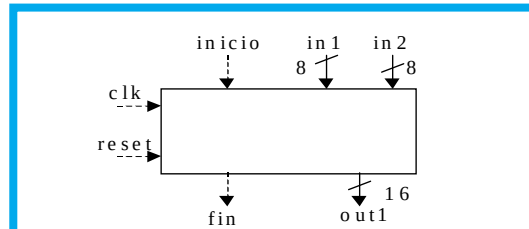


Figura 1. Entradas y salidas del multiplicador

La Figura 1 muestra las entradas y salidas necesarias para un multiplicador. Las entradas "in1" e "in2" son los operandos, cada uno de 8 bits. La entrada "*reset*" de un bit para inicializar los registros recién se enciende el sistema. La entrada "inicio" de un bit, es para indicar que hay una nueva operación y la entrada "*clk*" de un bit, es para sincronizar los registros. La salida "out1" de 16 bits contiene el resultado de la multiplicación y la salida "fin" de un bit, es para indicar que la multiplicación terminó.

El tipo de dato que se utiliza es el STD_LOGIC ya que, entre muchas otras características, sus librerías tienen definidas operaciones tanto aritméticas como lógicas. Para usar este tipo de dato es necesario incluir las siguientes librerías:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

[1] Investigador del grupo LAMIC, Laboratorio de Automática Microelectrónica e inteligencia computacional, Universidad Distrital Francisco José de Caldas.

Artículo recibido en Septiembre 10 de 2002
Aceptado el 13 de Noviembre 2002

Las asignaciones sólo permiten modelar circuitos combinacionales.

Cuando se requiere más de un bit se utiliza el

STD_LOGIC_VECTOR (n-1 DOWNTO 0)

Donde n es el número de bits que se requieren. La entidad quedaría de la siguiente forma:

```
entity MULT is
port(
  CLK,RESET   : in std_logic;
  IN1,IN2     : in std_logic_vector(7 downto 0);
  INICIO      : in std_logic;
  OUT1        : out std_logic_vector(15 downto 0);
  FIN         : out std_logic);
end MULT;
```

La segunda parte de un modelo en VHDL es la arquitectura, en donde se describe cómo debe trabajar el sistema. Esta descripción puede ser de tres tipos: asignación, estructural o funcional.

III. ASIGNACIÓN

Se puede realizar con asignación cuando existe el operador definido, en la ver. 9.23 de MaxPlusII de Altera, sobre la cual se realiza la aplicación del presente artículo, está definido el operador multiplicación con el cual el sintetizador genera un circuito combinacional que hace la multiplicación, por lo tanto este circuito no requiere *clk*. La arquitectura generada es la siguiente:

```
architecture ASIG of MULT is
begin
OUT1<=IN1*IN2;
FIN<=INICIO after 75 NS;
end ASIG
```

Donde ASIG es el nombre que se escogió para la arquitectura y MULT es el nombre dado a la entidad. Como está definida la operación *, entonces la multiplicación se limita a la sentencia:

```
OUT1<=IN1*IN2;
```

Sin embargo, la salida no es inmediata; puede tomar hasta 75 ns (en el dispositivo seleccionado) por eso la señal fin se define con 75 ns después de la señal de inicio. No obstante, la sentencia *after* no es fácilmente implementable y por lo tanto muchos sintetizadores la ignoran.

Esta síntesis combinacional del multiplicador genera un circuito muy grande, requiere 216 LCs y 95 expansores; en algunos casos es preferible realizar otra implementación menos costosas.

IV. ALGORITMO

Una posible implementación consiste en desarrollar los pasos de la multiplicación secuencialmente, similar al proceso que se utiliza convencionalmente con números decimales, como se muestra la Figura 2, solo que en binario únicamente se necesita saber la tabla del cero y del uno.

El primer paso es mirar el bit menos significativo de MN, si es uno se suma MR a PP, de lo contrario PP queda igual. Luego se desplaza MR a la izquierda y MN a la derecha para ver el siguiente bit y se repite el ciclo hasta que MN sea cero.

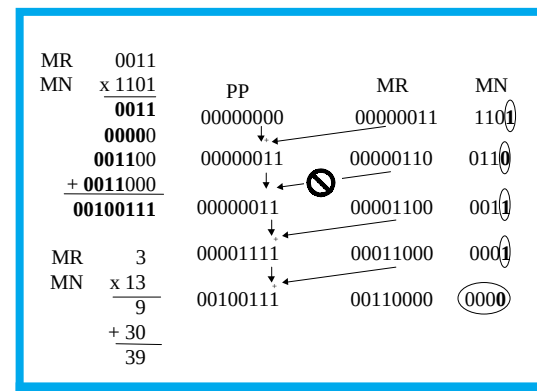


Figura 2. Ejemplo de multiplicación

Para saber cuando comienza la multiplicación es necesario esperar la señal de "inicio" que indica que los datos están listos. El diagrama de flujo se muestra en la Figura 3.

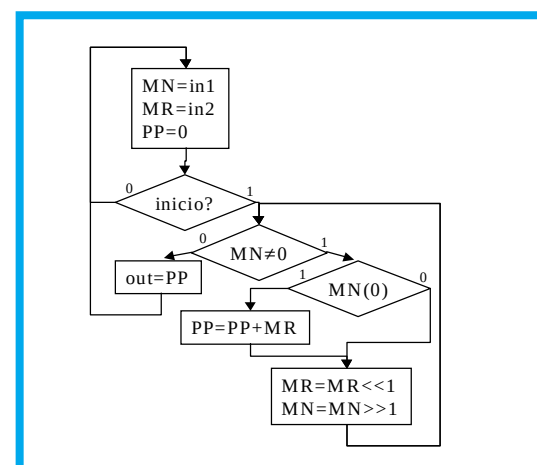


Figura 3. Diagrama de Flujo de Mutliplicación

Este diagrama de flujo se podría pasar directamente a VHDL y quedaría de la siguiente forma:

```
architecture ALG of MULT is
begin
process
variable --define las siguientes variables
  MN,MR,PP : std_logic_vector (15 downto 0);
begin
  loop -- espera señal de inicio
                MR:="00000000" & in1;
                MN:= "00000000" & in2;
                PP:="0000000000000000";
  exit when (inicio='1');
  end loop;
```

```
    while (MN/="0000000000000000") loop
                if (MN(0)='1') then
                        PP:=PP+MR;
                end if;
    MR:=MR(14 downto 0)&'0';
    MN:= '0'&MN(15 downto 1);
    end loop;
    out1<=PP;
end process;
end ALG;
```

A. Algunos comentarios de la sintaxis de VHDL

Cuando in1 se asigna a MR es necesario concatenar ceros a la izquierda para completar el número de bits.

La sentencia EXIT sale del ciclo LOOP - END LOOP cuando se cumple la condición, es decir cuando llega la señal de inicio.

El símbolo "/=" es "diferente de"

Para desplazar usualmente se utiliza el símbolo concatenar "&", cuando es a la izquierda se toman los bits más significativos concatenado con el cero y cuando es a la derecha se concatena un cero con los bits menos significativos.

B. Problemas del Algoritmo

Desafortunadamente esta solución tiene dos problemas:

El primero es que no todos los sintetizadores de VHDL implementan todas las funciones de VHDL y por lo tanto seguramente tendrán problemas de sintaxis, en particular con las funciones FOR o WHILE.

El segundo problema y más grave aún es que en ningún momento se ha utilizado la señal de clk y por lo tanto el sintetizador intentará realizar un circuito asincrónico que realice el algoritmo, lo cual es bastante complejo y generalmente no lo pueden sintetizar.

Al escribir un código en VHDL tenemos que ser consientes que estamos modelando HW y no SW. En caso que el circuito que se desea describir es muy complejo para ser implementado combinacionalmente, es mejor usar sentencias RTL.

C. Descripción de RTL

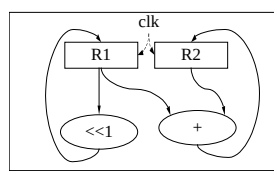


Figura 4 Grafica de sentencias RTL

Una sentencia RTL [6] permite representar en texto un circuito sincrónico, Por ejemplo las sentencias:

R1<=R1<<1

R2<=R1+R2

equivalen al circuito de la Figura 4.

Usualmente en una sentencia RTL se toman los datos de los registros, se operan en un circuito combinacional y el resultado es almacenado en otro registro cuando llegue el pulso de reloj.

D. Máquina de Estados Algorítmica (ASM)

En diseños más complicados, la carga de un registro no se realiza en todos los ciclos de reloj sino dependiendo de ciertas circunstancias. Un diagrama ASM [7] permite modelar fácilmente en qué momento se deben ejecutar las sentencias RTL. En la Figura 5 se presenta el ASM del diagrama de flujo de la multiplicación del algoritmo seleccionado.

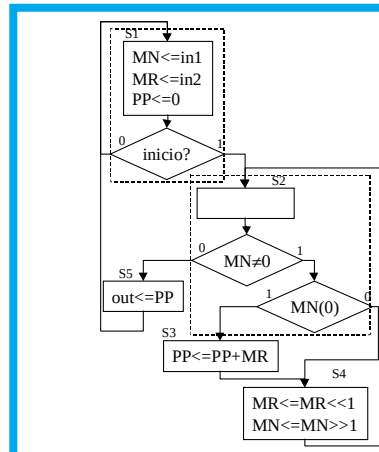


Figura 5. Diagrama ASM de multiplicación

El ASM y el diagrama de flujo son muy similares pero tienen las siguientes diferencias:

- Cada rectángulo del ASM representa un estado cuyo nombre se encuentra al lado.
- Las ecuaciones que hay dentro de cada rectángulo del ASM son sentencias RTL. En cambio, en el algoritmo puede ser cualquier operación. En este caso cada ecuación del algoritmo se puede hacer en una sentencia RTL, pero hay situaciones en que es conveniente separar la ecuación en varias sentencias RTL, por ejemplo a=b+c+d: t1<=b+c y a<=t1+d.
- Todas las sentencias RTL que están dentro de un cuadrado del ASM se ejecutan al mismo tiempo, mientras que en el algoritmo se ejecutan secuencialmente.
- Las decisiones del ASM pertenecen al estado predecesor y se indica con un rectángulo punteado.
- La asignación de la sentencia RTL solo se efectúa hasta que llega la señal de reloj y en el mismo instante toma la decisión y cambia de estado (por lo

Al escribir un código en VHDL tenemos que ser consientes que estamos modelando HW y no SW.

El diseño estructural permite integrar los componentes en varios niveles jerárquicos.

tanto las decisiones son tomadas con los antiguos valores de las variables).

En el estado S1 promete cargar los valores iniciales en los registros cuando llegue el pulso de reloj y al mismo tiempo cambia al siguiente estado, que puede ser el mismo si la entrada "inicio" está en 0, ó S2 si la entrada "inicio" está en 1.

En el estado S2 toma dos decisiones, en una decisión mira si el bit menos significativo de MN es uno y en la otra mira si todos los bits de MN ya son cero; dependiendo de lo anterior puede pasar a S3, S4 ó S5 como se muestra la Figura 5.

En S3 promete sumar PP con MR y pasar a S4. En S4 promete hacer los desplazamientos de MN y MR y repetir el ciclo. Finalmente en S5 promete sacar PP y volver a empezar.

En el estado S2 no se realiza ninguna sentencia RTL pero es necesario colocarlo, ya que si no existiera entonces las decisiones MN≠0 y MN(0) pertenecerían al estado S1 y se presentarían los siguientes problemas:

- El registro MN no se ha alcanzado a cargar con in1, cuando tiene que tomar las decisiones, ya que se carga al mismo tiempo que toma la decisión.
- El estado siguiente de S4 sería S1 y cada vez que repitiera el ciclo borraría el contenido de los registros.

A continuación se explicarán dos formas de implementar el diagrama ASM, la primera es la forma tradicional en la que primero se halla el circuito y después se pasa a VHDL, este diseño se llama estructural. En la segunda forma se pasa directamente del ASM a VHDL, este diseño se llama funcional.

V. DISEÑO ESTRUCTURAL

Se debe diseñar un circuito que permita realizar todas las sentencias RTL del diagrama ASM este circuito se denomina **Camino de Datos**. Además se debe diseñar otro circuito que le dice al Camino de Datos cuales sentencias en particular debe realizar en ese flanco de reloj, este otro se llama Unidad de Control.

El Camino de Datos del multiplicador se presenta en la Figura 6 y se realizó con base en el ASM de la Figura 5 donde cada registro hace todas las operaciones asignadas; por ejemplo MR se carga con in2 en el estado S1 y desplazado a la izquierda en S4.

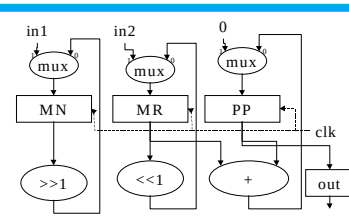


Figura 6 Camino de Datos

La unidad de control, como su nombre lo indica, controla en qué estados debe realizar cada sentencia RTL mediante las líneas de control del camino de datos. Por ejemplo, el *load* de MR (loadMR) o el selector del multiplexor de MR (selMR). La unidad de control es modelada con una máquina de estados finitos (FSM), donde el estado siguiente está determinado por el diagrama del ASM y las salidas están determinadas por las sentencias RTL. Las entradas a la FSM corresponden a las decisiones del ASM.

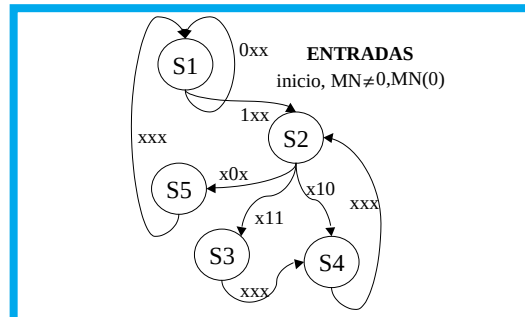


Figura 7. FSM del multiplicador

En la Figura 7 se muestra la FSM del multiplicador y en qué valores deben estar las entradas para cambiar de un estado a otro. Por ejemplo: para pasar de S2 a S4 no importa la señal de inicio, MN debe ser dife-rente de cero y el bit cero de MN debe ser cero.

TABLA I SALIDAS DE FSM DEL MULTIPLICADOR

Estado	sel MN	load MN	sel MR	load MR	sel PP	load PP	load out
S1	1	1	1	1	1	1	0
S2	0	0	0	0	0	0	0
S3	0	0	0	0	0	1	0
S4	0	1	0	1	0	0	0
S5	0	0	0	0	0	0	1

Las salidas de la FSM son las líneas de control del Camino de Datos las cuales se muestran en la Tabla I

Libros como [1], [4] explican cómo modelar en VHDL registros, multiplexores, FSM y en general todos los elementos usados en el camino de datos y la unidad de control. En la Figura 8 se muestra como se integra el camino de datos y la unidad de control:

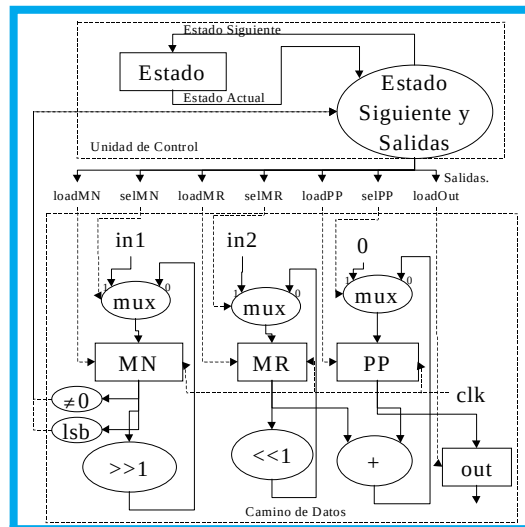


Figura 8 Diseño estructural

No nos detendremos en la explicación del anterior código ya que es la estructura convencional que está debidamente explicada en los libros mencionados anteriormente y concentraremos nuestra antención más adelante en otra forma de síntesis.

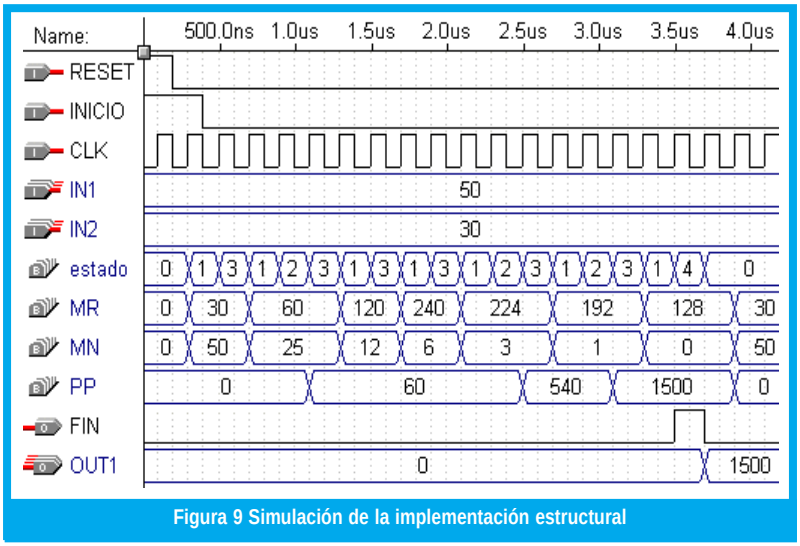


Figura 9 Simulación de la implementación estructural

Aunque el código generado es mucho más extenso, ocupa mucho menos espacio que la versión combinacional. Esta implementación ocupa 87 LCs y 30 expansores, que es tan solo el 40% del circuito combinacional. En la simulación de la Figura 9 se observan los pasos para realizar la multiplicación, tal como se describió en el ASM. Si desea una copia del código en VHDL puede escribir al autor.

Para modelos sencillos como el anterior, el proceso de diseño es aceptable, pero para circuitos más complejos es preferible una metodología de diseño más sencilla.

VI. DISEÑO FUNCIONAL Y METODOLOGÍA PROPUESTA

En este artículo proponemos una metodología intermedia entre escribir directamente el algoritmo y describir estructuralmente el circuito. La metodología consiste en describir directamente el ASM en VHDL

El comportamiento sincrónico del ASM permite ser modelado dentro de una estructura sincrónica de VHDL en la que cada asignación se convierte en una sentencia RTL, la cual es descrita dentro de la estructura de una FSM en VHDL y que se detalla más adelante cuando se explique el código que se muestra a continuación.

En el diseño funcional se pasa directamente del ASM al VHDL, simplificando el proceso de diseño.

```
architecture ASM of MULT is
-- definición de señales internas
signal MN, MR,PP: std_logic_vector(15 downto 0);
type estados is (S1,S2,S3,S4,S5);
signal estado_actual : estados;
begin
process (reset, clk)
begin
if (reset='1') then
 estado_actual<=S1;
 MN<="0000000000000000";
 MR<="0000000000000000";
 PP<="0000000000000000";
 Out1<="0000000000000000";
 fin<='0';
elsif (clk'event and clk='1') then
 case estado_actual is
 when S1 =>
   MN<="00000000"&in1;
   MR<="00000000"&in2;
   PP<="0000000000000000";
   fin<='0';
   if (inicio='0') then
    estado_actual<=S1;
   else
    estado_actual<=S2;
   end if;
 when S2 =>
   if (MN="0000000000000000") then
    estado_actual<=S5;
   else
    if (MN(0)='0') then
    estado_actual<=S4;
    else
    estado_actual<=S3;
    end if;
   end if;
 when S3 =>
   PP<=PP+MR;
   estado_actual<=S4;
 when S4 =>
   MN<='0' & MN(15 downto 1);
   MR<=MR(14 downto 0) & '0';
   estado_actual<=S2;
 when S5 =>
   out1<=PP;
   estado_actual<=S1;
 fin<='1';
 end case;
end if;
end process;
end ASM;;
```

El código generado es análogo al ASM de la Figura 5. Lo primero es definir las variables internas, en este caso son MR, MN y PP, y los estados de la FSM que van desde S1 hasta S5.

Luego dentro del proceso se define una sentencia IF- THEN en la que detecta dos cosas: la primera, si llega una señal de RESET y la segunda si llega un flanco positivo de reloj.

```
process (reset, clk)
begin
if (reset='1') then
...
elsif (clk'event and clk='1') then
...
end if;
end process;
```

Cuando llega la señal de RESET, coloca en cero todos los registros. Cuando el que llega es el flanco de reloj, ejecuta el ASM, que tiene la misma estruc-

Los diseños secuenciales generalmente ahorran recursos aunque pueden ser más demorados que los combinacionales.

tura de una FSM la cual es modelada con una estructura CASE.

```
case ESTADO_ACTUAL is
 when S1 =>
...
when S5 =>
...
end case;
```

Por ejemplo, en el estado S1 debe hacer las siguientes sentencias RTL,

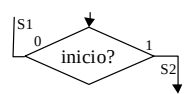
```
MN<=in1
MR<=in2
PP<=0
```

las cuales son escritas directamente en VHDL con la sintaxis adecuada para igualar el número de bits.

```
when S1 =>
  MN<="00000000"&in1;
  MR<="00000000"&in2;
  PP<="0000000000000000";
```

Estas sentencias sólo se realizan cuando llegue el flanco de reloj ya que se encuentran dentro del bloque "elsif (clk'event and clk='1') then". y por lo tanto equivalen a las sentencias RTL del ASM.

Además en el estado S1 debe determinar cual es el siguiente estado.



Esto es modelado en VHDL de la siguiente forma:

```
if (inicio='0') then
 estado_actual<=S1;
else
 estado_actual<=S2;
end if;
```

la síntesis del ASM requirió 85 LCs y 37 expansores. En general es inmediato pasar del ASM a VHDL utilizando esta metodología. En la Figura 10 se presentan los resultados de la simulación.

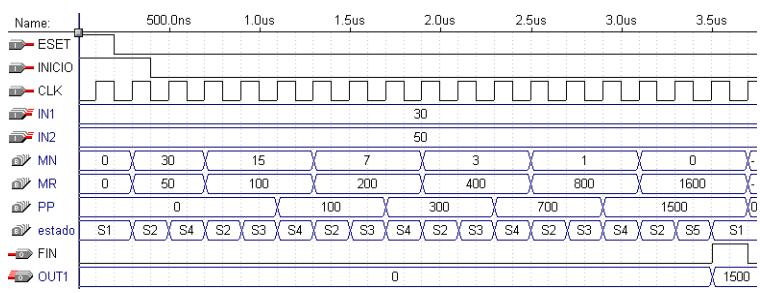


Figura 10 Simulación de la implementación Funcional

VII. CONCLUSIONES

En la Tabla II se presenta un cuadro comparativo entre las diferentes metodologías de diseño. La metodología de Asignación se diferencia de las otras dos en que solo permite diseños combinacionales en los cuales está definido el operador, generando un circuito combinacional, que debe hacer todas las operaciones al tiempo y por lo tanto no se pueden compartir recursos, lo cual implica que el diseño va a ser más rápido pero va a costar más.

TABLA II. COMPARACIÓN ENTRE DISEÑOS

	Diseño	**Costo**	**Vel.**	**Casos**
Asignación	Muy Sencillo	Elevado	Rápido	Simples
Estructural	Complicado	Bajo	Medio	Complejos
Funcional	Sencillo	Bajo	Medio	Complejos

Las otras dos metodologías permiten realizar cualquier tipo de diseño digital, además los resultados son similares. Sin embargo, el diseño funcional es mucho más sencillo de implementar.

REFERENCIAS

[1] Terés L. et al, "VHDL Lenguaje Estándar de Diseño Electrónico", primera edición, Madrid, Mc Graw Hill, 1998.

[2] Chang K. C., "Digital System Design with VHDL and Synthesis, An Integrated Approach", IEEE Computer Society Press, Los Alamitos, California 1999

[3] Chang K. C., "Digital Design and Modeling with VHDL and Synthesis", IEEE Computer Society Press, Los Alamitos, California 1997

[4] Actel, "Actel HDL Coding, Style Guide", USA 1997

[5] Gajski D., "Principios de Diseño Digital", pri-mera edición, Madrid, Prentice Hall Iberia, 1997.

[6] Mano M., "Lógica Digital y Diseño de Compu-tadores" ,Prentice Hall, 1982

[7] Mano M. "Diseño Digital", Primera Edición, Mexico, Prentice Hall, 1987

Gerardo Muñoz
Universidad Distrital FJC. Facultad Ingeniería, Grupo LAMIC
gmunoz@udistrital.edu.co