DISEÑO DIGITAL 1. BLOQUE TEMÁTICO 1

TÍTULO DE LA ACTIVIDAD: **Tutorial ModelSim: realización de *Test-Benches* de circuitos secuenciales**	**CÓDIGO:** **BT1.P3**

FECHA:			
NOMBRE:		APELLIDOS:	

MODALIDAD:	Tutorial. Individual	TIPO:	Presencial	DURACIÓN:	60 minutos

CALENDARIO:	Sesión síncrona P3.	REQUISITOS:	

CRITERIO DE ÉXITO:	

COMENTARIOS E INCIDENCIAS:

TIEMPO DEDICADO:	Minutos	AUTOEVALUACIÓN: [entre 0 y 10 puntos]	No procede

PARTE I. Introducción: Test-Bench de Sistemas Secuenciales Síncronos simples

En esta actividad se van a abordar cuestiones relacionadas con la realización de *Test-Benches* de modelos de sistemas secuenciales síncronos simples. Se tratarán, en concreto, los siguientes asuntos:

- Definición del estímulo de reloj del TB: Se mostrará el modo de definir un reloj de periodo parametrizable mediante el valor de una constante VHDL.
- Sincronización de estímulos síncronos en los flancos activos de reloj: Se explicará cómo sincronizar los cambios de nivel de las entradas síncronas con los flancos de reloj utilizando sentencias WAIT FOR y WAIT UNTIL.
- Estructura de las pruebas de simulación de sistemas secuenciales simples: En esta parte del tutorial se presentará un conjunto de reglas que deben cumplirse a la hora de secuenciar las pruebas.

Todas estas cuestiones se ilustrarán tomando como ejemplo la realización del *Test-Bench* de un registro de 16 bits con habilitación de reloj y reset síncrono.

Instrucciones para la realización del tutorial

Durante la realización del tutorial:

1. Debe ejecutar todas las operaciones que se le indican mediante el predicado "**Realice las siguientes operaciones"**.

2. El texto en cursiva contiene explicaciones relativas a elementos del lenguaje VHDL o reglas que debe aplicar en la realización de código. Léalas con atención y, si tiene cualquier duda o curiosidad sobre ellas, no dude en preguntar a su profesor.

3. Comunique a su profesor cualquier problema o incidencia con que se encuentre durante la realización del tutorial.

4. Cuando termine de realizar el tutorial comuníqueselo a su profesor.

PARTE II. Tutorial

Realice las siguientes operaciones:

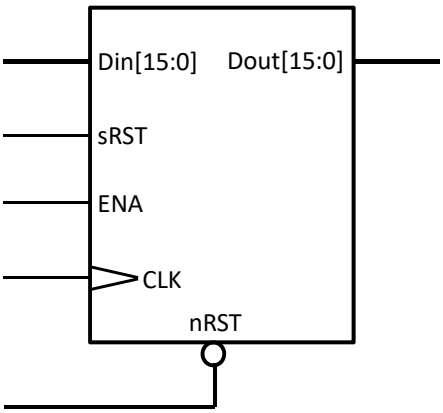
1. Arranque el simulador ModelSim (P1 T.1).

2. Cree un nuevo proyecto y denomínelo *Proy_P3* (P1 T.2).

3. Cree un fichero añadido al proyecto, de nombre *reg16b_ena_rst.vhd* (P2 T.1) y ábralo para editarlo.

El sistema secuencial que sirve de base a este tutorial es el registro de 16 bits con entrada de habilitación y entrada de reset síncrono, cuyo modelo lógico muestra la figura 1.

INTERFAZ



FUNCIONAMIENTO

nRST	sRST	ENA	CLK	Dout
0	X	X	X	**Todos 0**
1	1	X	↑	**Todos 0**
1	0	0	↑	**Dout**
1	0	1	↑	**Din**

Figura 1.- Modelo lógico del registro

Modelo VHDL del registro

Como ya sabe, la Declaración de Entidad que modela la interfaz de un sistema secuencial se construye siguiendo las mismas reglas que en el caso de los sistemas combinacionales:

1. La Declaración de Entidad va precedida por las cláusulas de visibilidad que permiten utilizar los tipos std_logic y std_logic_vector y debe tener un nombre derivado de la función que realiza el sistema.

2. Los puertos deben tener un nombre relacionado con su función, dirección IN (si son de entrada) o BUFFER (si son de salida) y, por último, su tipo de datos debe ser std_logic (si son puertos de 1 bit) o std_logic_vector (si son puertos de varios bits).

En la figura 2 se muestra el código VHDL que modela la interfaz del registro. Observe que los nombres de los puertos se corresponden con los del símbolo de la figura 1 y que el nombre del modelo es reg16b_ena_rst.

```
1   -- Modelo de un registro de 16 bits, con entrada
2   -- de reset y habilitación
3
4
5
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9
10
11  entity reg16b_ena_rst is
12  port(
13        clk:  in std_logic;
14        nRST: in std_logic;
15        sRst: in std_logic;
16        ena:  in std_logic;
17        Din:  in std_logic_vector(15 downto 0);
18        Dout: buffer std_logic_vector(15 downto 0)
19        );
20  end entity;
21
```

Figura 2.- Declaración de entidad del registro

Realice la siguiente operación:

1. Añada al fichero *reg16b_ena_rst.vhd* el código comprendido entre las líneas 1 y 20 de la figura 2.

Los registros tienen una arquitectura de Moore que presenta la peculiaridad de que la salida es la propia memoria de estado del sistema. El modelo de funcionamiento de este tipo de sistemas puede comprimirse, en VHDL, en un solo proceso. En la figura 3 se muestra el código VHDL de la arquitectura que modela el funcionamiento del registro.

```
22
23  architecture rtl of reg16b_ena_rst is
24  begin
25
26    -- Proceso que modela el funcionamiento del registro
27    process(clk, nRST)
28    begin
29      if nRST = '0' then
30        Dout <= (others => '0');
31
32      elsif clk'event and clk = '1' then
33        if sRst = '1' then
34          Dout <= (others => '0');
35
36        elsif ena = '1' then
37          Dout <= Din;
38
39        end if;
40      end if;
41    end process;
42  end rtl;
43
```

Figura 3.- Cuerpo de Arquitectura del registro

Como ya sabe:

1. El proceso es sensible únicamente al reloj y a la entrada de inicialización asíncrona –no es sensible a ninguna de las entradas síncronas (*sRst*, *ena* y *Din*), porque los cambios de valor de estas entradas no pueden producir cambios en las salidas del registro: la salida de un sistema secuencial síncrono sólo puede cambiar de valor por la activación de la entrada de inicialización asíncrona o por la ocurrencia de un flanco activo de reloj.

```
27    process(clk, nRST)
```

2. El código del proceso emplea una sentencia IF para evaluar el estado de la entrada de inicialización asíncrona –si está activa (a '0') se asigna el valor inicial a la salida del registro- y comprobar, en caso de que dicha entrada no esté activa, si la ejecución del proceso se debe a la ocurrencia de un flanco de subida del reloj –ocurrencia que lleva asociado un código que asigna el valor de salida que determinen las entradas síncronas del registro (*sRst*, *ena* y *Din*).

```
29      if nRST = '0' then
30        Dout <= (others => '0');
31
32      elsif clk'event and clk = '1' then
33        if sRst = '1' then
34          Dout <= (others => '0');
35
36        elsif ena = '1' then
37          Dout <= Din;
38
39        end if;
```

Observe que el código que modela el funcionamiento síncrono describe fielmente lo especificado por el modelo lógico del registro representado en la figura 1:

- Si la entrada de reset síncrono está activa (a '1'), el registro se pone a cero.

```
33          if sRst = '1' then
34            Dout <= (others => '0');
```

- Si el reset síncrono no está activo y el registro está habilitado (*ena* a '1'), se carga el dato de entrada.

```
33          if sRst = '1' then
34            Dout <= (others => '0');
35
36          elsif ena = '1' then
37            Dout <= Din;
```

Observe, también, que no es necesario codificar el caso en que el registro mantiene el valor almacenado cuando ni la entrada de reset síncrono ni la de habilitación están activas: en este caso el código del proceso no ejecuta ninguna sentencia de asignación de valor y, efectivamente, la salida del registro no cambia.

Realice las siguientes operaciones:

1. Añada al fichero *reg16b_ena_rst.vhd* el código comprendido entre las líneas 23 y 42 de la figura 3.

2. Salve el fichero (P1 T.7) y compílelo (P1 T.9). Si se detecta algún error, corríjalo (P1 T.13).

Codificación de Test-Benches de sistemas secuenciales simples

Al igual que en el caso de los TB de los sistemas combinacionales, el desarrollo del código del TB de un sistema secuencial puede dividirse en dos partes: la primera, de naturaleza sistemática, consiste en la creación de una Declaración de Entidad sin puertos y de un Cuerpo de Arquitectura donde se declaran señales y se emplaza el modelo del sistema secuencial a verificar; a continuación, en la segunda parte, se codifican las pruebas de simulación que se hayan diseñado para el test del sistema.

En la figura 4 se muestra el código del TB del registro que puede generarse de manera mecánica: la Declaración de Entidad, la declaración de señales y el emplazamiento del modelo del registro.

```
1   -- Test-Bench del modelo de un registro de 16 bits
2
3
4
5
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9
10  entity reg16b_ena_rst_TB is
11  end entity;
12
13  architecture Test of reg16b_ena_rst_TB is
14    signal clk:  std_logic;
15    signal nRST: std_logic;
16    signal sRst: std_logic;
17    signal ena:  std_logic;
18    signal Din:  std_logic_vector(15 downto 0);
19    signal Dout: std_logic_vector(15 downto 0);
20
21  begin
22
23  dut: entity Work.reg16b_ena_rst(rtl)
24       port map(clk  => clk,
25                nRST => nRST,
26                sRst => sRst,
27                ena  => ena,
28                Din  => Din,
29                Dout => Dout);
```

Figura 4.- TB del registro

Realice las siguientes operaciones:

1. Cree un fichero añadido al proyecto, de nombre *reg16b_ena_rst_TB.vhd* (P2 T.1) y ábralo para editarlo.
2. Añada al fichero *reg16b_ena_rst_TB.vhd* el código comprendido entre las líneas 1 y 29 de la figura 4.

Proceso para la generación del reloj

La entrada de reloj de un sistema secuencial síncrono está manejada por una señal periódica que sirve para definir el discurrir del tiempo: la señal de reloj. Los parámetros que definen una señal de reloj son:

1. *La frecuencia: es la inversa del periodo e indica el número de instantes de tiempo, por segundo, que la señal de reloj define en el sistema al que está conectado –esto es así puesto que para un sistema secuencial síncrono cada periodo de la señal de reloj es, por medio de la ocurrencia de un flanco activo, un nuevo instante de tiempo.*

2. *El ciclo de trabajo: Es la fracción de tiempo que la señal de reloj está a nivel alto durante un periodo. Las señales de reloj de los sistemas digitales suelen tener un ciclo de reloj del 50%.*

El modelado de la señal de reloj en un TB VHDL puede realizarse con un código muy sencillo, aprovechando el carácter cíclico de la ejecución de los procesos VHDL que contienen sentencias WAIT. Recordemos el modo de ejecución de este tipo de procesos:

1. *El proceso se ejecuta por primera vez en T = 0. Las sentencias contenidas en el proceso se ejecutan secuencialmente, empezando por la primera, hasta que se alcance una sentencia WAIT. La ejecución de esta sentencia suspende la ejecución del proceso.*

2. *Cuando el proceso se reactiva, continúa la ejecución secuencial de las sentencias hasta que se encuentre una sentencia WAIT y el proceso vuelva a suspenderse. Si se alcanza a ejecutar la última sentencia del proceso, la ejecución continúa a partir de la primera –a este mecanismo es al que hacemos referencia cuando hablamos de ejecución cíclica del proceso. Esta rutina de suspensión y reanudación de la ejecución del proceso se repite indefinidamente mientras no se ejecute una sentencia de suspensión incondicional del proceso (WAIT;).*

Aplicando estas reglas, resulta fácil entender que un código tan simple como el que se muestra en la figura 5 genere una señal de reloj como la representada en el cronograma adjunto.

```
process
begin
  clk <= '0';
  wait for 50 ns;
  clk <= '1';
  wait for 50 ns;
end process;
```

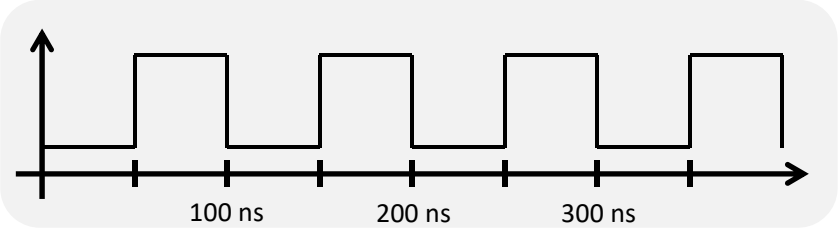


Figura 5.- Señal de reloj

Como puede verse, la ejecución del código del proceso supone la asignación de un nivel bajo a la señal de reloj y su mantenimiento durante 50 ns y, seguidamente, la asignación de un nivel alto y su mantenimiento durante otros 50 ns. La repetición indefinida de esta secuencia de asignaciones, derivada de la ejecución cíclica del proceso, da lugar a la generación de una señal periódica. En este ejemplo la señal de reloj tiene un periodo de 100 ns (una frecuencia de 10 MHz) y un ciclo de trabajo del 50%.

Cambiando el retardo asociado a las sentencias WAIT del código de la figura 5 resulta muy sencillo generar señales de reloj de cualquier frecuencia:

1. *El proceso de generación de señal de reloj tiene siempre la misma estructura:*

```
process
begin
  clk <= '0';
  wait for T_CLK/2;
  clk <= '1';
  wait for T_CLK/2;
end process;
```

Donde T_CLK/2 es la mitad del valor del periodo.

2. *Dada la frecuencia de la señal de reloj (F), se calcula su periodo (T = 1/F), se divide por dos y se obtiene el valor del retardo que debe especificarse en las sentencias WAIT.*

Constantes VHDL

Una alternativa muy recomendable para la generación de la señal de reloj consiste en definir su periodo mediante una constante. Las constantes son objetos VHDL similares –idénticos- a los disponibles en los lenguajes de programación. Las constantes VHDL pueden declararse en diferentes zonas, dependiendo del ámbito en que se desea que sean visibles: si, como en el caso que nos ocupa, deseamos que se pueda utilizar en cualquier proceso que pueda haber en el Cuerpo de Arquitectura, la constante debe declararse en la zona de declaración del mismo –es decir, donde se declaran las señales que se conectan al modelo emplazado en el TB. Para declarar una constante hay que:

1. *Indicar que el objeto declarado es una constante empleando la palabra clave CONSTANT*

2. *Indicar su nombre y tipo de datos, con una sintaxis similar a la de la sentencia de declaración de señal*

3. *Indicar, en la propia declaración, el valor de la constante*

Por ejemplo, para declarar una constante, de nombre T_CLK y de valor 100 ns, debe realizarse la siguiente declaración:

```
constant T_CLK: time:= 100 ns;
```

El tipo de datos de la constante T_CLK es TIME, un tipo del lenguaje VHDL que permite especificar valores de tiempo -un valor de tipo TIME está compuesto por un número (entero o real) y un factor de escala (fs, ps, ns, us, ms, sec, min y hr)

Observe como el valor de la constante se indica en la declaración, a continuación del tipo de datos, mediante la asignación con ":=".

Contando con esta declaración de constante, el siguiente código equivaldría al de la figura 5:

```
process
begin
  clk <= '0';
  wait for T_CLK/2;
  clk <= '1';
  wait for T_CLK/2;
end process;
```

Esta manera de definir la señal de reloj facilita la definición de lapsos de tiempo como múltiplos del periodo de reloj del sistema. Por ejemplo, si se desea suspender un proceso durante 9123 periodos de reloj, puede recurrirse a la fórmula:

```
wait for 9123*T_CLK;
```

La especificación de los tiempos que secuencian los estímulos en base a un periodo del reloj definido por una constante permite que, aunque se cambie la frecuencia de reloj (modificando el valor de la constante), se mantenga la integridad de la pruebas codificadas, ya que los tiempos que definen la secuencia de estímulos se escalan automáticamente sin necesidad de tener que modificar el código del TB.

Realice las siguientes operaciones:

1. Añada la siguiente constante en la zona de declaración del Cuerpo de Arquitectura del TB –a continuación, por ejemplo, de la declaración de las señales del TB.

```
20
21      constant T_CLK: time:= 100 ns;
22
```

2. Añada al TB el código que define la señal de reloj:

```
33    process
34    begin
35      clk <= '0';
36      wait for T_CLK/2;
37      clk <= '1';
38      wait for T_CLK/2;
39    end process;
40
```

El proceso para la generación de reloj forma parte de todos los TB de sistemas secuenciales y, por tanto, puede considerarse como otra *sección* del código que puede generarse de manera sistemática. En la figura 6 se muestra todo el código editado hasta ahora, el que puede generarse de un modo *mecánico*.

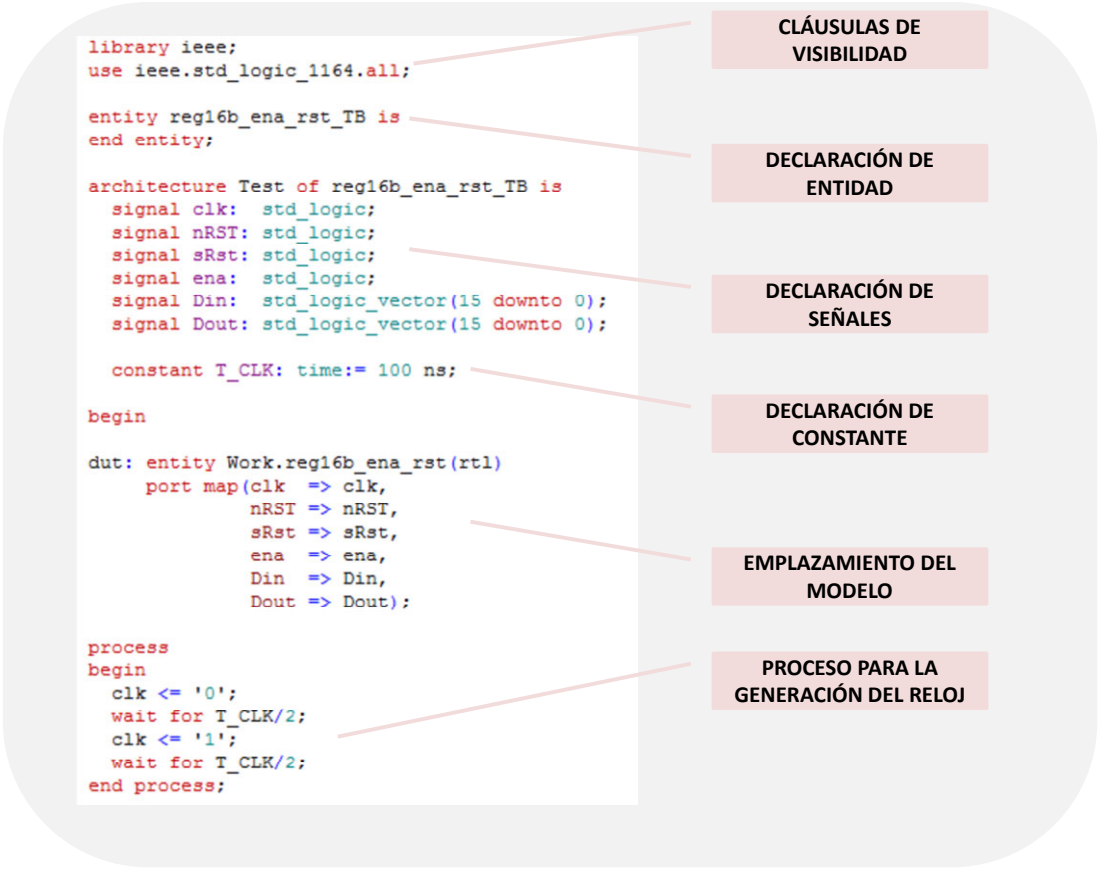


Figura 6.- Código del TB

Para completar el código del TB de un sistema secuencial hay que añadir al TB el código de las pruebas que verifican el funcionamiento. Para ello debe partirse de una especificación de dichas pruebas.

Diseño de pruebas de simulación de sistemas secuenciales simples

A la hora de diseñar las pruebas de test de un sistema secuencial simple no es posible, salvo en casos muy contados, plantearse la opción de realizar un test exhaustivo. Por ello hay que realizar una selección de un conjunto mínimo de pruebas que permitan certificar el correcto funcionamiento del sistema. Este conjunto de pruebas debe permitir verificar siempre que:

1. *Cuando las entradas asíncronas se activan, producen el efecto previsto en el sistema –y que cuando están inactivas no afectan a su funcionamiento.*

2. *La respuesta síncrona del sistema para todas las combinaciones de las entradas de control síncronas es la esperada.*

Aplicando estas reglas al diseño del test del registro que se está utilizando como ejemplo en este tutorial, obtendríamos la siguiente serie de pruebas:

1. *Activación y desactivación del reset asíncrono*
2. *Prueba de carga de datos (reset síncrono inactivo y habilitación activa)*
3. *Prueba de reset síncrono con la habilitación activa*
4. *Prueba de reset síncrono con la habilitación inactiva*
5. *Prueba de mantenimiento del dato almacenado (habilitación y reset síncrono inactivos)*

Una vez definidas las pruebas de simulación de un sistema, deben combinarse para construir la secuencia de test. Al hacerlo debe tenerse en cuenta que:

1. *Al secuenciar las pruebas puede resultar conveniente elegir un determinado orden para facilitar la propia ejecución de las pruebas.*

   Por ejemplo, la primera prueba que se hace siempre en el test de un sistema secuencial es la activación de la entrada de inicialización asíncrona, porque inicializa la memoria del sistema y lo deja preparado para la ejecución de las pruebas de funcionamiento síncrono. Otro ejemplo: en el test del registro que se está utilizando en este tutorial debe realizarse una prueba de carga de datos antes de probar la operación de reset síncrono, para que pueda apreciarse positivamente el efecto de éste –si se prueba la operación de reset síncrono inmediatamente después de haber probado la inicialización asíncrona, se resetea el registro cuando está a 0: es mejor realizar una prueba en la que quede patente el cambio de valor del registro por efecto del reset y, por eso, es mejor que antes se haya realizado la carga de un dato.

2. *Debe minimizarse, en la medida de lo posible, la longitud del test.*

   Cuanto más reducido sea el test, menor será el esfuerzo que habrá que realizar para codificarlo, y menor el tiempo requerido para comprobar los resultados de las simulaciones; además, el simulador tardará menos tiempo en ejecutar la simulación –si bien es verdad que esta última ventaja sólo resulta apreciable cuando se simulan sistemas digitales complejos.

   Para reducir la longitud del test hay que evitar, en la medida de lo posible, la realización de pruebas redundantes, y ejecutar cada prueba síncrona con el número de ciclos de reloj estrictamente necesarios.

El cronograma de la figura 7 muestra un diseño del test del registro que contiene todas las pruebas especificadas para la verificación de su correcto funcionamiento. El orden y duración de las pruebas es el siguiente:

1. En primer lugar se realiza un reset asíncrono que debe inicializar el registro a 0.

2. En el primer flanco de reloj (2, en la figura) en el que la entrada de inicialización asíncrona está desactivada, se carga un dato (xAAAA) en el registro.

3. En el siguiente flanco (3) se realiza un reset síncrono con el registro habilitado que pone el registro a cero.

4. A continuación, se carga (4) un nuevo dato (x5555) en el registro –esta prueba es redundante, pero se realiza porque aún falta por probar la activación del reset síncrono cuando la habilitación está desactivada.

5. En el siguiente flanco (5) se prueba que el registro mantiene el dato almacenado cuando no está habilitado y el reset síncrono está inactivo.

6. En el último flanco activo (6) se prueba la operación del reset síncrono cuando la habilitación está inactiva.

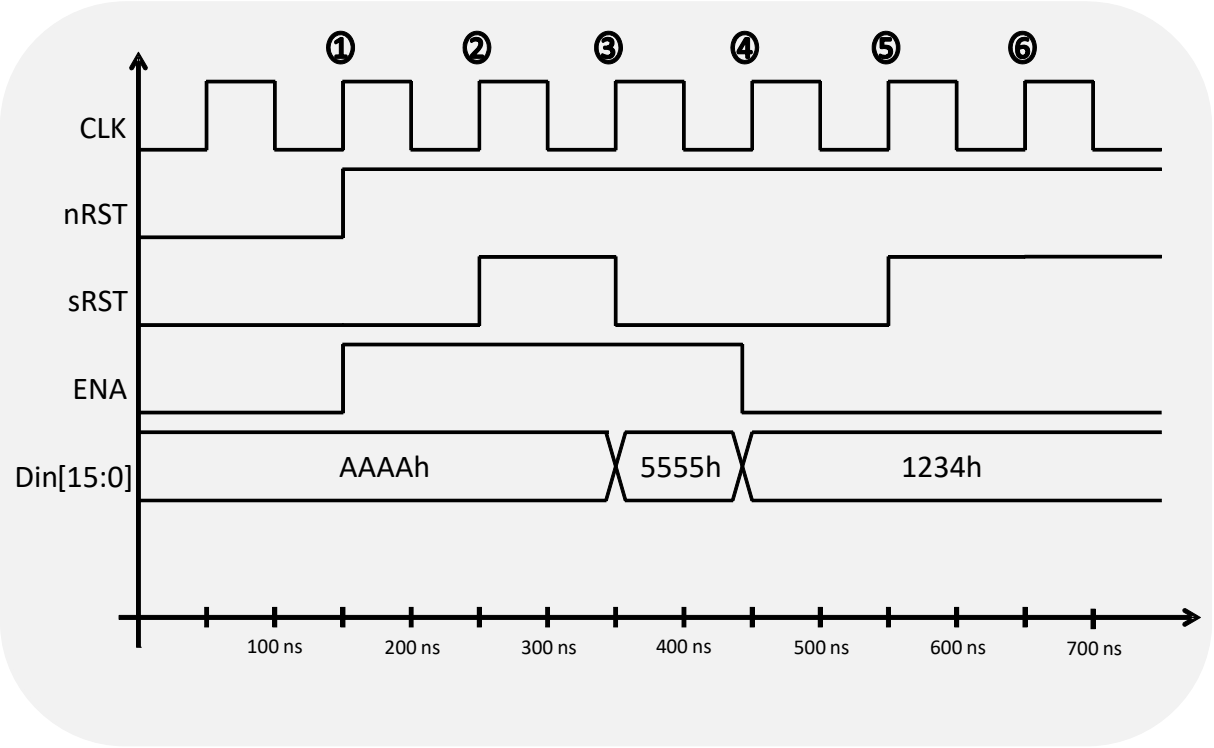


Figura 7.- Test del registro

La secuencia de la figura 7 comprueba todas las combinaciones de las entradas síncronas sRST y ENA, pero no las de la entrada de datos Din, ya que al ser Din una entrada de 16 bits, podrían formarse hasta 65536 combinaciones diferentes y no tiene sentido testear la carga (o mantenimiento) del valor en el registro para cada una de estas combinaciones; se prueban únicamente 3 valores de Din, que se han elegido con los siguientes criterios:

1. *Que en todos los bits del registro se carguen unos y ceros.*

   Es interesante comprobar la conmutación a '0' y '1' de todos los bits de salida. Por este motivo en las dos cargas de datos que se realizan se emplean combinaciones complementarias (los bits de una combinación tienen los niveles lógicos complementarios de la otra.

2. *Que los patrones de ceros y unos que se emplean resulten fáciles de recordar.*

   El uso en el test de constantes fácilmente reconocibles x5555 (010101….01), xAAAA (101010…10) ó x1234 facilita la inspección de los resultados de la simulación. El ejemplo que nos ocupa es muy sencillo y no permite que se aprecie esta ventaja, pero en el test de sistemas de mayor complejidad resulta de gran ayuda poder retener en la memoria los valores con los que se estimula el modelo bajo prueba o los valores que se esperan como respuestas.

Codificación de las pruebas de simulación

El código que define los estímulos correspondientes a las entradas síncronas y asíncronas de un sistema secuencial debe realizarse atendiendo a los siguientes criterios:

1. *Los estímulos deben codificarse con el menor número posible de procesos.*

   En los test de sistemas secuenciales simples y de mediana complejidad suele bastar un proceso para codificar eficazmente la secuencia completa de pruebas a realizar.

2. *La secuencia de pruebas debe temporizarse con sentencias WAIT, de modo que los estados de todas las entradas síncronas dure un número entero de ciclos de reloj del sistema.*

   La generación de la secuencia de señales que componen las pruebas del test debe realizarse con sentencias WAIT. Aunque pueden emplearse sentencias WAIT FOR, la idónea para este cometido es la sentencia WAIT UNTIL.

Una sentencia WAIT UNTIL incluye una condición lógica (una expresión que puede evaluarse como TRUE o FALSE). Cuando durante la ejecución de un proceso se alcanza una sentencia de este tipo, se suspende su ejecución hasta que la condición asociada a la sentencia WAIT sea TRUE. Por ejemplo, la ejecución de la sentencia WAIT UNTIL S = '0', suspende el proceso y difiere la reanudación de su ejecución hasta que S tome el valor '0'.

Un detalle importante a tener en cuenta sobre el mecanismo de operación de la sentencia WAIT UNTIL es que suspende la ejecución del proceso aunque la condición que contiene sea TRUE en el instante en que se alcance la sentencia. Por ejemplo, si S vale '0' y se ejecuta WAIT UNTIL S = '0', el proceso se suspende y sólo se reanudará cuando S vuelva a valer '0' en un instante posterior de la simulación.

La convención que rige en la interpretación de un cronograma sin retardos de un sistema secuencial es que, cuando una entrada conmuta en un flanco activo de reloj, el nuevo valor al que conmuta se considera aplicado un tiempo infinitésimamente pequeño después del flanco: los estímulos cambian, por tanto, ***después*** *de los flancos de subida del reloj. Para asignar un valor inmediatamente después de un flanco activo –en este ejemplo, un flanco de subida- se utiliza la siguiente fórmula:*

```
wait until clk'event and clk = '1';
Asignación
```

La expresión que forma parte de la sentencia WAIT UNTIL ya es conocida, toma el valor TRUE en los instantes del tiempo de simulación en los que ocurren flancos de subida de la señal de reloj, y, por tanto, la ejecución de esta sentencia WAIT suspende un proceso hasta que, en el transcurso de la simulación, se dé la ocurrencia de un flanco de subida del reloj. Tras el flanco de reloj se reanuda el proceso y se ejecuta la asignación, que ocurre justo en el flanco, pero tras él.

Empleando sentencias WAIT UNTIL, el test definido en la figura 7 podría codificarse en un proceso VHDL de la siguiente manera:

1. En T = 0 (primera sentencia del proceso) se activa la entrada de reset asíncrono y se inicializan las entradas síncronas.

```
process
begin
   nRST <= '0';      -- Inicialización asíncrona
   sRst <= '0';      -- Valores válidos en entradas síncronas
   ena <= '0';
   Din <= X"0000";
```

Para que la señal de reset se mantenga activa al menos un periodo de reloj, se establece la ocurrencia de dos flancos activos antes de desactivar la entrada de reset asíncrono.

```
process
begin
   nRST <= '0';      -- Inicialización asíncrona
   sRst <= '0';      -- Valores válidos en entradas síncronas
   ena <= '0';
   Din <= X"0000";

  wait until clk'event and clk = '1';
  wait until clk'event and clk = '1';
   nRST <= '1';      -- Desactivación de la inicialización asíncrona
```

Como el primer flanco de subida del reloj ocurre cuando ha transcurrido un tiempo de simulación igual a medio periodo de reloj, la entrada de reset asíncrono permanece activa durante ciclo y medio de reloj; se desactiva tras el segundo flanco de subida de éste.

2. Coincidiendo con el flanco de reloj en el que se desactiva el reset, se ordena la carga del dato xAAAA en el registro:

   - La entrada síncrona ENA se pone a ‘1’ (estaba a ‘0’), a la entrada síncrona Din se le asigna el valor x”AAAA”; como la entrada sRST debe valer ‘0’ y ese es su valor actual, no se le asigna valor. Esta combinación de valores de las entradas síncronas se mantiene hasta después del siguiente flanco activo de reloj.

```
process
begin
   nRST <= '0';      -- Inicialización asíncrona
   sRst <= '0';      -- Valores válidos en entradas síncronas
   ena <= '0';
   Din <= X"0000";

  wait until clk'event and clk = '1';
  wait until clk'event and clk = '1';
   nRST <= '1';      -- Desactivación de la inicialización asíncrona
   ena <= '1';       -- Carga de datos
   Din <= X"AAAA";

  wait until clk'event and clk = '1';
```

3. Tras el siguiente flanco de subida, se realiza un reset síncrono con el registro habilitado.

   - Solo se cambia el valor de sRst (se le asigna un '1') y se mantiene la combinación hasta después del siguiente flanco de subida del reloj.

```
process
begin
   nRST <= '0';      -- Inicialización asíncrona
   sRst <= '0';      -- Valores válidos en entradas síncronas
   ena <= '0';
   Din <= X"0000";

  wait until clk'event and clk = '1';
  wait until clk'event and clk = '1';
   nRST <= '1';      -- Desactivación de la inicialización asíncrona
   ena <= '1';       -- Carga de datos
   Din <= X"AAAA";

  wait until clk'event and clk = '1';
   sRst <= '1';      -- Activación del reset síncrono con ena = '1'

  wait until clk'event and clk = '1';
```

4. Después del siguiente flanco se carga x5555 en el registro.

   - Como ENA ya estaba a '1', se desactiva el reset síncrono y se asigna a Din el valor x5555. (a continuación se muestra sólo el código correspondiente a esta prueba)

```
wait until clk'event and clk = '1';
 sRst <= '0';      -- Desactivación del reset síncrono
 Din <= X"5555";   -- Carga de dato

wait until clk'event and clk = '1';
```

5. A continuación, tras el siguiente flanco, se prueba que el registro mantiene el dato almacenado cuando no está habilitado y el reset síncrono está inactivo.

   - Para ello se desactiva ENA (se pone a '0') y se cambia el valor de Din (a x1234) para comprobar que no se carga; la entrada de reset síncrono se deja desactivada.

```
wait until clk'event and clk = '1';
 ena <= '0';       -- Desactivación de carga de datos
 Din <= X"1234";

wait until clk'event and clk = '1';
```

6. Por último, se prueba la operación del reset síncrono cuando la habilitación está inactiva, manteniendo la entrada de reset síncrono activa durante un ciclo de reloj, y se finaliza la secuencia de pruebas con una sentencia WAIT.

```
wait until clk'event and clk = '1';
 sRst <= '1';      -- Activación del reset síncrono con ena = '0'

wait until clk'event and clk = '1';
 wait;             -- Fin del test
```

En la figura 8 se muestra el código completo del proceso.

```
41  process
42  begin
43     nRST <= '0';      -- Inicialización asíncrona
44     sRst <= '0';      -- Valores válidos en entradas síncronas
45     ena <= '0';
46     Din <= X"0000";
47
48    wait until clk'event and clk = '1';
49    wait until clk'event and clk = '1';
50     nRST <= '1';      -- Desactivación de la inicialización asíncrona
51     ena <= '1';       -- Carga de datos
52     Din <= X"AAAA";
53
54    wait until clk'event and clk = '1';
55     sRst <= '1';      -- Activación del reset síncrono con ena = '1'
56
57    wait until clk'event and clk = '1';
58     sRst <= '0';      -- Desactivación del reset síncrono
59     Din <= X"5555";   -- Carga de dato
60
61    wait until clk'event and clk = '1';
62     ena <= '0';       -- Desactivación de carga de datos
63     Din <= X"1234";
64
65    wait until clk'event and clk = '1';
66     sRst <= '1';      -- Activación del reset síncrono con ena = '0'
67
68    wait until clk'event and clk = '1';
69     wait;             -- Fin del test
70  end process;
```

Figura 8.- Secuencia de pruebas

Realice las siguientes operaciones:

1. Añada al fichero *reg16b_ena_rst_TB.vhd* el código comprendido entre las líneas 41 y 70 de la figura 8.
2. Complete el código del TB añadiendo al final de la arquitectura la sentencia de cierre:

```
end Test;
```

3. Salve el fichero (P1.T7) y compílelo (P1 T.9). Si se detecta algún error, corríjalo (P1 T.13).
4. Arranque la simulación (P2 T.2) y prepare el visor de formas de onda (P2 T.4 y T.5) para visualizar todas las entradas y salidas del registro (figura 9).

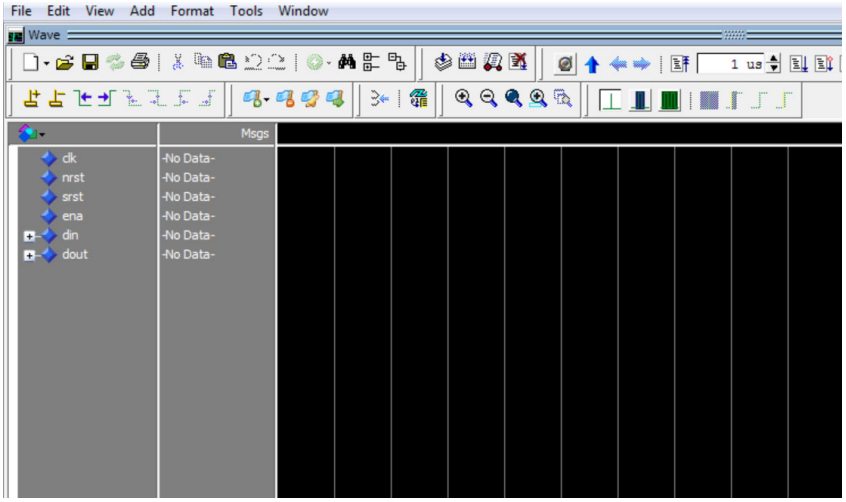


Figura 9.- Visor

5. Defina un tiempo de simulación de 750 ns (P2 T.7) y ejecute una simulación (P2 T.8).

En la figura 10 se muestra la respuesta esperada para las pruebas de test con un funcionamiento correcto del modelo del registro.

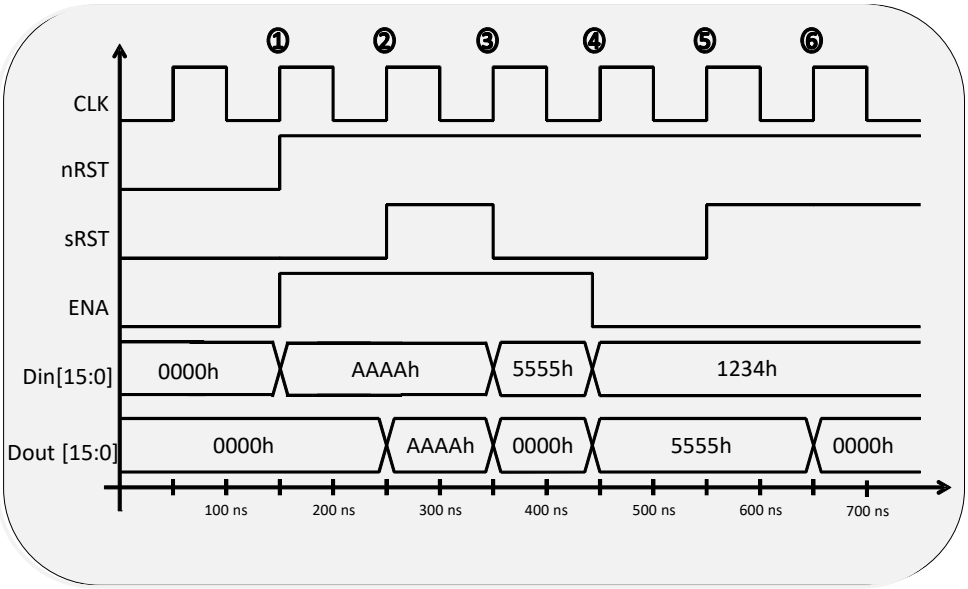


Figura 10.- Funcionamiento del registro

Realice las siguientes operaciones:

1. Compruebe que los resultados de la simulación coinciden con los esperados y finalice la simulación.

Uso de sentencias WAIT FOR en los Test-Benches de sistemas secuenciales

En el TB que se ha realizado, se han utilizado sentencias WAIT UNTIL para pautar la secuencia de estímulos. Como se ha comprobado en la simulación realizada, las sentencias WAIT UNTIL han supuesto la suspensión del proceso que controla los estímulos durante un ciclo de reloj, desde un flanco de subida hasta el siguiente. Podría pensarse, por tanto, que una vez alineados al flanco

activo de subida de reloj, una sentencia WAIT UNTIL CLK'EVENT AND CLK = '1' podría sustituirse por una sentencia WAIT FOR T_CLK (siendo T_CLK el periodo de reloj) consiguiéndose el mismo efecto a la hora de definir la secuencia de estímulos. Pero no es así.

*La diferencia estriba en que la sentencia WAIT UNTIL suspende el proceso durante un ciclo de reloj, de flanco a flanco, pero hasta **después** del siguiente flanco (un **después** infinitesimalmente pequeño), mientras que una sentencia WAIT FOR que suspende el proceso durante un ciclo de reloj (WAIT FOR T_CLK) y está alineada con los flancos de reloj, lo suspende hasta un instante infinitesimal **antes** de la ocurrencia del flanco.*

Para entender el porqué de dicha diferencia hay que recordar cómo discurre, en un determinado instante de tiempo, la ejecución de un ciclo de simulación en VHDL:

1. *En la primera parte del ciclo de simulación se ejecutan los procesos suspendidos por sentencias WAIT FOR. La ejecución de estos procesos da lugar a que se proyecten asignaciones de valor a señales.*

2. *En la segunda parte del ciclo de simulación se actualizan las señales para las que se han proyectado asignaciones de valores. Si como consecuencia de estas actualizaciones se activan procesos, se ejecuta un nuevo ciclo de simulación sin avance de tiempo (un ciclo delta)*

Veamos por qué la sentencia WAIT UNTIL CLK'EVENT AND CLK = '1' suspende la ejecución del proceso hasta después de que se da la ocurrencia de un flanco de subida del reloj:

1. *Cuando, transcurrido el tiempo de espera especificado en la sentencia WAIT FOR del proceso generador del reloj, dicho proceso reanuda su ejecución y proyecta la asignación de un '1' a la señal de reloj, entonces, en la segunda parte del ciclo de simulación, la señal de reloj se actualiza a '1'.*

2. *Como se ha actualizado la señal de reloj y hay procesos sensibles a dicha señal y procesos que contienen sentencias WAIT con condiciones de las que forma parte dicha señal (también estos procesos son sensibles a la señal de reloj), se desencadena un ciclo delta en el que CLK'EVENT es TRUE (porque la señal de reloj ha conmutado).*

3. *En la primera parte del ciclo delta de simulación se ejecutan concurrentemente los procesos que modelan el hardware (el que modela el registro en nuestro ejemplo) y el suspendido por la sentencia WAIT UNTIL; como las asignaciones de valor que se proyectan en este último proceso se actualizan (en la segunda parte del ciclo delta de simulación) cuando ya se ha ejecutado el proceso que modela el hardware (el registro) por causa de la ocurrencia del flanco, son asignaciones, por tanto, posteriores al flanco de reloj –tendrán vigor para el proceso que modela el hardware en el siguiente flanco activo de reloj.*

Una sentencia WAIT FOR que pauta la asignación de valor a los estímulos en coincidencia con la asignación de valor al reloj provoca, en cambio, que las asignaciones sean efectivas antes del flanco, porque:

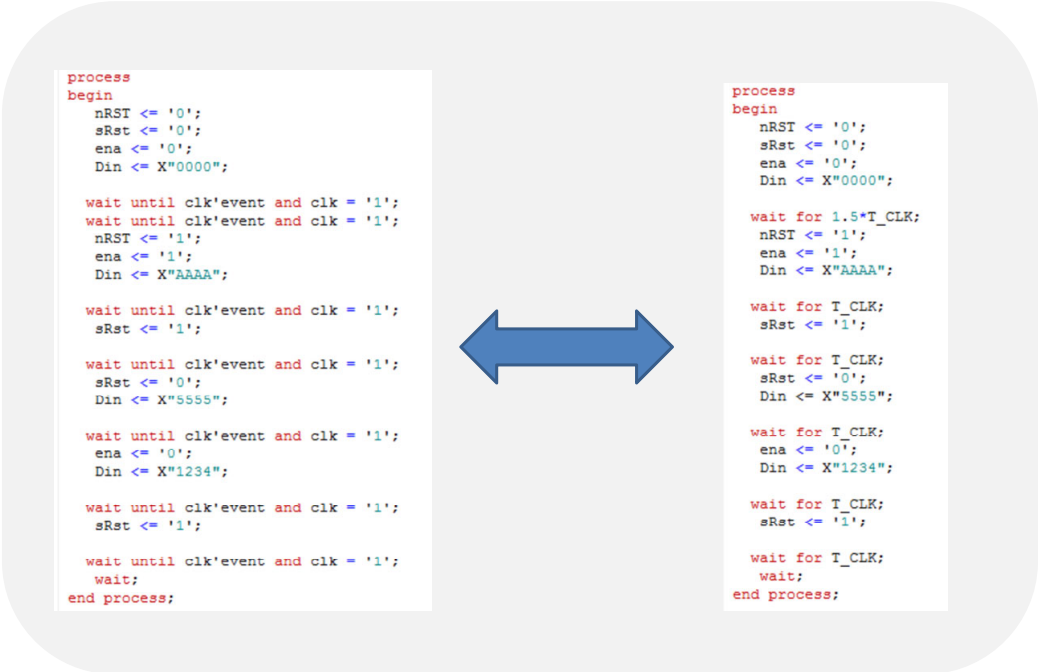
1. *El proceso generador del reloj reanuda su ejecución concurrentemente con el de generación de estímulos –ambos "despiertan" en el mismo instante del tiempo de simulación. Tras la*

ejecución de ambos procesos en la primera parte del ciclo de simulación, las asignaciones que proyectan (al reloj y resto de estímulos) se efectúan en la segunda parte de dicho ciclo.

2. *Como se ha actualizado la señal de reloj y hay procesos sensibles a dicha señal se desencadena un ciclo delta en el que CLK'EVENT es TRUE (porque la señal de reloj ha conmutado).*

3. *En la primera parte del ciclo delta de simulación los procesos que modelan el hardware evalúan la respuesta en el flanco con los valores de los estímulos de simulación ya actualizados, luego el efecto equivale a una actualización anterior al flanco activo de reloj.*

En el ejemplo que está realizando es muy sencillo comprobar esta diferencia de comportamiento, basta con cambiar las sentencias WAIT UNTIL por sentencias WAIT FOR, tal y como se muestra en la figura 11.



```
process
begin
   nRST <= '0';
   sRst <= '0';
   ena <= '0';
   Din <= X"0000";

  wait until clk'event and clk = '1';
  wait until clk'event and clk = '1';
   nRST <= '1';
   ena <= '1';
   Din <= X"AAAA";

  wait until clk'event and clk = '1';
   sRst <= '1';

  wait until clk'event and clk = '1';
   sRst <= '0';
   Din <= X"5555";

  wait until clk'event and clk = '1';
   ena <= '0';
   Din <= X"1234";

  wait until clk'event and clk = '1';
   sRst <= '1';

  wait until clk'event and clk = '1';
   wait;
end process;
```

```
process
begin
   nRST <= '0';
   sRst <= '0';
   ena <= '0';
   Din <= X"0000";

  wait for 1.5*T_CLK;
   nRST <= '1';
   ena <= '1';
   Din <= X"AAAA";

  wait for T_CLK;
   sRst <= '1';

  wait for T_CLK;
   sRst <= '0';
   Din <= X"5555";

  wait for T_CLK;
   ena <= '0';
   Din <= X"1234";

  wait for T_CLK;
   sRst <= '1';

  wait for T_CLK;
   wait;
end process;
```

Figura 11.- Codificación con sentencias WAIT FOR

Aunque la representación gráfica de los estímulos sería la misma para ambas simulaciones, la respuesta obtenida para la codificación con sentencias WAIT FOR interpreta que el valor al que cambian las entradas en los flancos es previo a los flancos (figura 12).

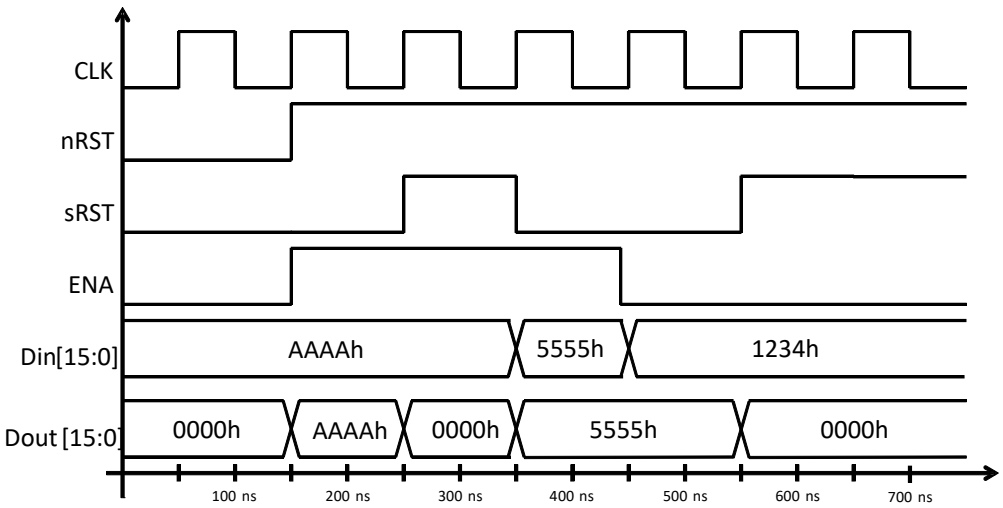


Figura 12.- Respuesta a sentencias WAIT FOR

Este comportamiento contradice la interpretación convencional de representación de cronogramas ideales de sistemas secuenciales y, por ello, debe evitarse este modo de definir las secuencias de estímulos.

Realice las siguientes operaciones:

1. En el fichero *reg16b_ena_rst_TB.vhd*, sustituya el código del proceso de definición de estímulos por el código comprendido entre las líneas 74 y 102 de la figura 14.

```
74   process
75   begin
76      nRST <= '0';      -- Inicialización asíncrona
77      sRst <= '0';      -- Valores válidos en entradas síncronas
78      ena <= '0';
79      Din <= X"0000";
80
81     wait for 1.5*T_CLK;
82      nRST <= '1';      -- Desactivación de la inicialización asíncrona
83      ena <= '1';       -- Carga de datos
84      Din <= X"AAAA";
85
86     wait for T_CLK;
87      sRst <= '1';      -- Activación del reset síncrono con ena = '1'
88
89     wait for T_CLK;
90      sRst <= '0';      -- Desactivación del reset síncrono con ena = '1'
91      Din <= X"5555";   -- Carga de dato
92
93     wait for T_CLK;
94      ena <= '0';       -- Desactivación de carga de datos
95      Din <= X"1234";
96
97     wait for T_CLK;
98      sRst <= '1';      -- Activación del reset síncrono con ena = '0'
99
100    wait for T_CLK;
101     wait;             -- Fin del test
102  end process;
```

Figura 14

2. Salve y compile el fichero y realice una simulación para comprobar que los resultados obtenidos coinciden con los del cronograma de la figura 12.

Uso de las sentencias WAIT FOR en Test-Benches de Sistemas Secuenciales

El uso de sentencias WAIT FOR es, sin embargo, recomendable en un caso concreto: cuando se desea mantener una combinación de valores para los estímulos durante gran número de ciclos de reloj. Pongamos por caso que se desea establecer una combinación de valores durante 10.000 ciclos de reloj. Para definir este periodo de tiempo con sentencias WAIT UNTIL habría que ejecutar 10.000 sentencias WAIT UNTIL CLK'EVENT AND CLK = '1' seguidas en el proceso. Semejante secuencia podría generarse con la ayuda de un bucle FOR. La sintaxis de estos bucles en VHDL incluye la definición implícita de una variable entera cuyo recorrido dentro de un rango define el número de iteraciones del bucle. El siguiente código generaría la ejecución de las 10000 sentencias WAIT mencionadas anteriormente.

```
for I in 0 to 9999 loop
  wait until clk'event and clk = '1';
end loop;
```

Esta forma de definir un retardo de 10.000 periodos de reloj supone la ejecución de 10.000 sentencias –por lo que el proceso tiene que reanudar su ejecución 10.000 veces para volver a suspenderse sin más- y es, obviamente, mucho menos eficiente que la ejecución de una única sentencia WAIT FOR que genere el mismo retardo:

```
wait for 10000*T_CLK;
```

Pero, como ya sabe, si esta sentencia se ejecuta en un flanco activo de reloj, el proceso se reanuda, diez mil ciclos de reloj después, un instante de tiempo infinitesimalmente pequeño antes del flanco activo de reloj. Para corregir los problemas asociados a esta situación (anteriormente descritos) basta con añadir una sentencia WAIT UNTIL que provoque una espera que haga transcurrir ese instante. La secuencia de las siguientes sentencias WAIT equivale a la ejecución de las diez mil sentencias WAIT UNTIL del bucle:

```
wait for 10000*T_CLK;
wait until clk'event and clk = '1';
```

Esta es la fórmula de codificación que debe utilizar para la definición de retardos cuya duración es un gran número de ciclos de reloj: es la más ventajosa en cuanto al esfuerzo exigido al simulador para la ejecución de una simulación y tiene una sintaxis muy simple.