Pre-Informe 3
Laboratorio:INF-245

Rodrigo Ignacio Cárdenas Valdivia
201773080-3

Investigación de Conceptos

a) Un lenguaje de descripción de hardware (HDL, Hardware Description Language), es un lenguaje que sirve para describir el comportamiendo de un cicuito electronico (digital), sintetizando asi la logica, diseño y estructura de estos, asi a traves de las descripciones es posible simular el comportamiento del circuito de forma simple con el fin de comprobar que el funcionamiento sea correcto antes de construir el circuito real.
Los lenguajes de programacion se enfocan en la resolucion de problemas y creacion a nivel de software, mientras que los HDL's van dirigidos a la creacion de hardwares (previa a la construcción fisica) e incorporan explicitamente el concepto de tiempo, ademas de lo planteado en el parrafo anterior.

b) *Wire* es un tipo de dato 'red', este se utiliza para variables que representan fisicamente conexiones, siendo util para conectar distintos modulos
reg se utiliza para los registros, con el fin de mantener el valor de las variables guardado.
logic es el tipo de dato mas versatil, ya que puede ser tratado tando como un registro ('blocking'/'nonblocking') o como un tipo de dato de 'red', se usa generalmente como tipo de dato para las variables *input* y *output*.

c) La diferencia esta en que el operador <= es un 'nonblocking assignment' (solo se puede usar en initial y bloques always), es decir que este se ejecutara con las otras lineas de codigo , mientras que = es un operador 'blocking assigment' (se puede usar en initial, bloques always y assign) y a diferencia del anterior se tiene que ejecutar esta linea para que al leer el codigo en funcionamiento se avanze a la siguiente y por último el *assign* asigna la expresión del lado derecho del simbolo '=' continuamente al *wire* del lado izquierdo (se usa fuera de un always).

d) El '*reg*' es la declaracion de un registro, luego el '[15 : 0]' nos dice que el registro sera de 16 bits con indices del 15 al 0, siendo el indice 15 el mas significativo y el indice 0 el menos significativo, la 'g' es el identificador del registro, el '=' nos dice que el valor inicial del registro sera lo que viene tras esto (a su lado derecho), la 'h' nos indica que el valor sera hexadecimal y '$A6B2$' es el numero hexadecimal que sera usado como valor para el registro. Por último el numero hexadecimal '$A6B2$' en binaro equivale a '1010011010110010'.

e) Se declara *first* con el valor 0 y se asgina a *second* el valor 1 (el valor negado de first) previo a la subida del clock (dentro del modulo), tras las subida de este, *first* cambia su valor a 1, pero *second* no varia aún, ya que los cambios fuera del bloque '*always*' no se realizan hasta que se llega al 'end' del mismo, luego se realizan los cambios de '*arr[first]*' y asi el bit en el indice 1 de '*arr*' toma el valor de el bit en el indice 1 de '*arr*',pero negado, es decir, toma el valor de 0 negado, resultando en 1 y por esto '*arr*' queda igual a '0111'.

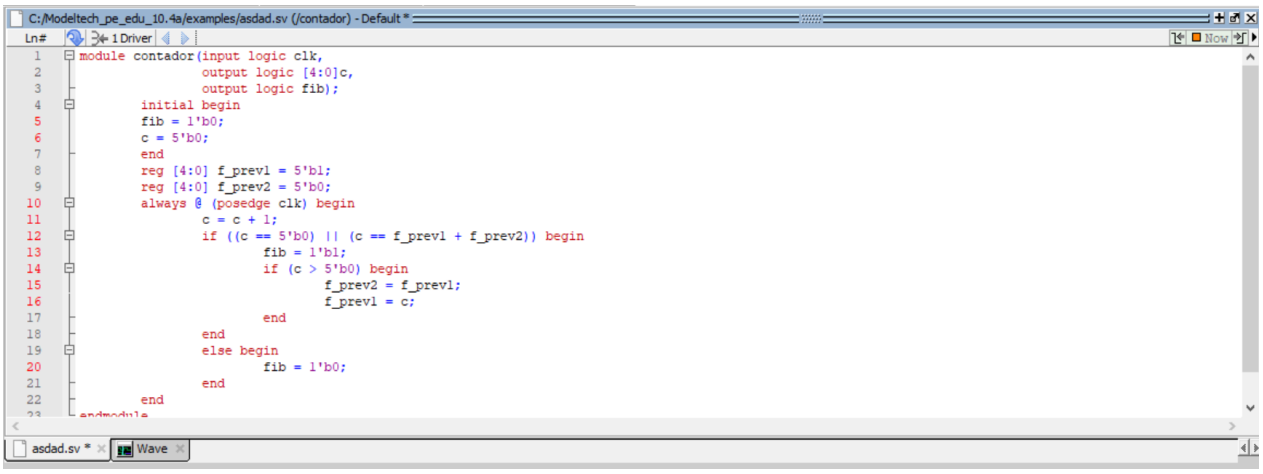
Análisis de Codigo

Con $r = 001$ el resultado del ruteo es el siguiente:

clock	s	z	x	y	m	n
0	0	000	00000000	01110111	001	01110111
1						
	1					
		010	00000000	01110101		
					011	01110101
$0 \longrightarrow 1$						
		011	00000000	01110101		
					100	01110101
$0 \longrightarrow 1$						
		100	00010000	01110101		
					101	10000101
$0 \longrightarrow 1$						
		101	00010000	01100101		
					110	01110101
$0 \longrightarrow 1$						
		110	00010000	01000101		
					111	01010101
$0 \longrightarrow 1$						
		111	00010000	01000101		
					000	01010101
$0 \longrightarrow 1$						
		000	00010001	01000101		
					001	01010110
$0 \longrightarrow 1$						
		001	00010001	01000101		
					010	01010110
$0 \longrightarrow 1$						
		010	00010101	01000101		
					011	01011010
$0 \longrightarrow 1$						
		011	00010101	01000101		
					100	01011010

Tras 10 subidas del clock $n = 01011010$.

Análisis de Codigo




```
1  module contador(input logic clk,
2                  output logic [4:0]c,
3                  output logic fib);
4          initial begin
5          fib = 1'b0;
6          c = 5'b0;
7          end
8          reg [4:0] f_prev1 = 5'b1;
9          reg [4:0] f_prev2 = 5'b0;
10         always @ (posedge clk) begin
11                 c = c + 1;
12                 if ((c == 5'b0) || (c == f_prev1 + f_prev2)) begin
13                         fib = 1'b1;
14                         if (c > 5'b0) begin
15                                 f_prev2 = f_prev1;
16                                 f_prev1 = c;
17                         end
18                 end
19                 else begin
20                         fib = 1'b0;
21                 end
22         end
23 endmodule
```

Figure 1: Código

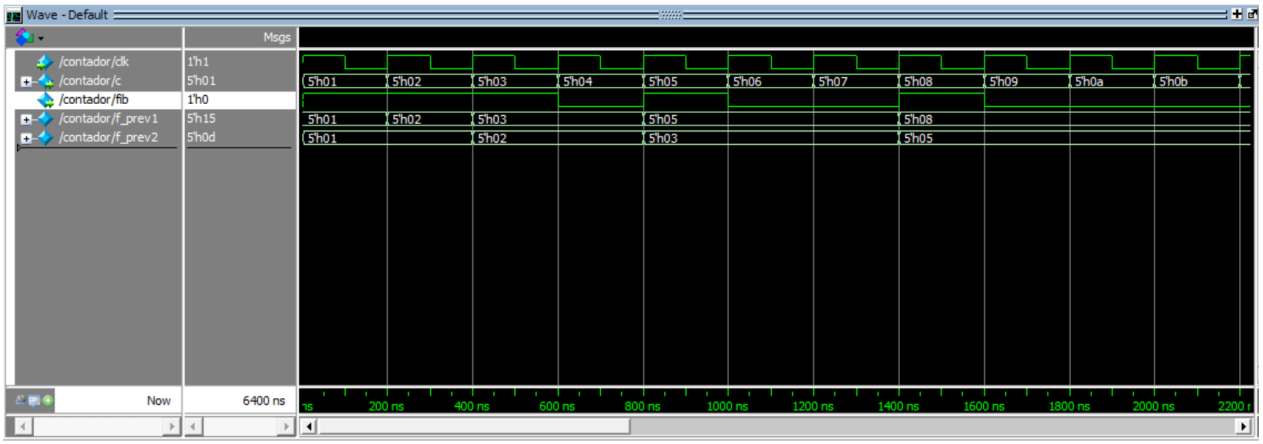


Figure 2: Simulación

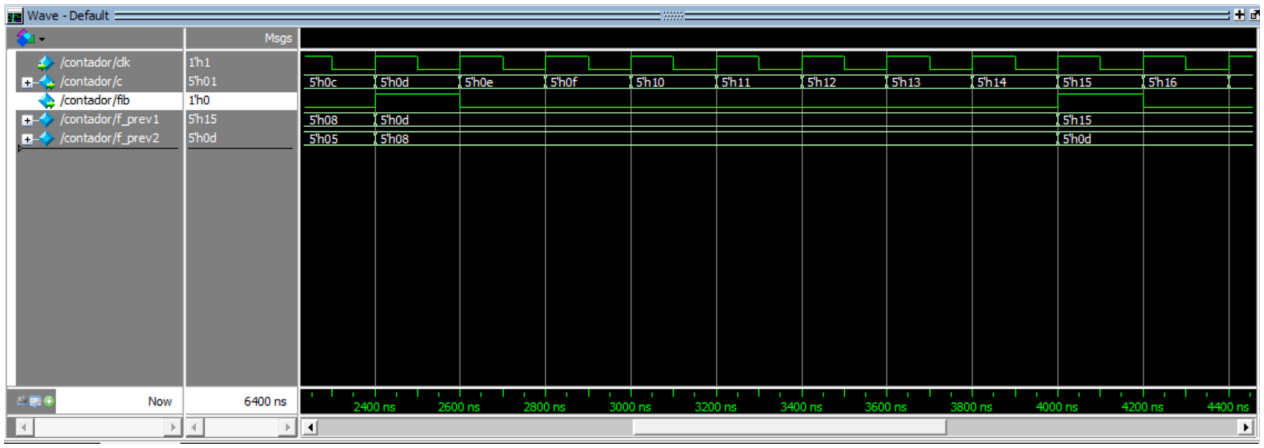


Figure 3: Simulación

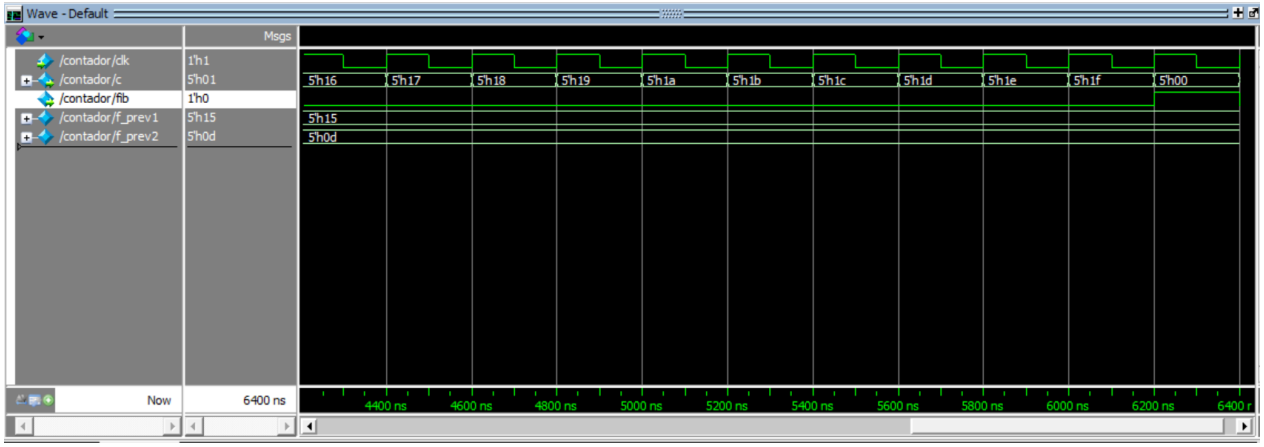


Figure 4: Simulación