AOC_04.

Verilog HDL

Hardware Description Languages (HDL)
Lenguaje de descripción de hardware

- Sirven para representar diagramas lógicos, expresiones booleanas y otros circuitos digitales más complejos.
- Como lenguaje de documentación, un HDL sirve para representar y documentar sistemas digitales en una forma susceptible de ser leída tanto por personas como por computadoras, y es apropiado como lenguaje de intercambio entre diseñadores.
- El contenido en HDL se puede almacenar, recuperar y procesar fácil y eficazmente con software de computador. Hay dos aplicaciones del procesamiento de HDL: simulación y síntesis

- La simulación lógica es la representación de la estructura y el comportamiento de un sistema lógico digital empleando un computador.
- El simulador interpreta la descripción en HDL y produce una salida comprensible, digamos un diagrama de temporización, que predice la forma en que se comportará el hardware antes de que se fabrique físicamente.
- La simulación permite detectar errores funcionales de un diseño sin tener que crear el circuito físico.

- Los errores detectados durante la simulación se corrigen modificando los enunciados HDL apropiados. El estímulo que prueba la funcionalidad del diseño se denomina conjunto de pruebas.
- Así, para simular un sistema digital, el diseño se describe primero en HDL y luego se verifica simulando el diseño y verificándolo con un conjunto de pruebas, que también está escrito en HDL

- Los errores detectados durante la simulación se corrigen modificando los enunciados HDL apropiados.
- El estímulo que prueba la funcionalidad del diseño se denomina conjunto de pruebas.
- Así, para simular un sistema digital, el diseño se describe primero en HDL y luego se verifica simulando el diseño y verificándolo con un conjunto de pruebas, que también está escrito en HDL

La síntesis lógica es el proceso de deducir una lista de componentes y sus interconexiones (lo que se conoce como lista del circuito) a partir del modelo de un sistema digital descrito en HDL.
La lista del circuito en el nivel de compuertas sirve para fabricar un circuito integrado o para diagramar una tarjeta de circuitos impresos.

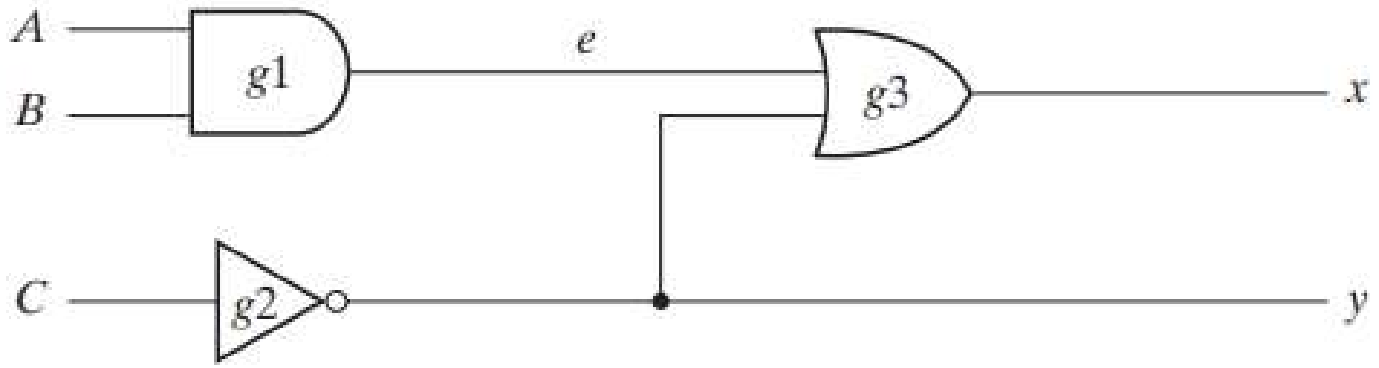
Representación de módulos

Verilog utiliza cerca de 100 palabras clave: identificadores predefinidos, en minúsculas, que definen las construcciones del lenguaje.

- Como ejemplos de palabras clave podemos citar module, endmodule, input, output, wire, and, or, not, etcétera.
- Todo texto comprendido entre dos diagonales (//) y el fin de la línea se interpreta como un comentario.
- Se hace caso omiso de los espacios en blanco y se hace distinción entre mayúsculas y minúsculas. El bloque de construcción en Verilog es el módulo.

ejemplo HDL

```
module circuito_smpl(A,B,C,x,y);
    input A,B,C;
    outputx,y;
    wire e;
    and g1(e,A,B);
    not g2(y, C);
    or g3(x,e,y);
endmodule
```



- La línea que inicia con dos diagonales es un comentario que explica la función del circuito.
- La segunda línea declara el módulo, e incluye un nombre y una lista de puertos.
- El nombre (circuito_smpl en este caso) es un identificador que sirve para referirse al módulo.
- Los identificadores son nombres que se dan a las variables para poder referirse a ellas en el diseño.
- Constan de caracteres alfanuméricos y el carácter de subrayado (_), y hay distinción entre mayúsculas y minúsculas.
- Los identificadores deben iniciar con un carácter alfabético o subrayado; no pueden iniciar con un número.

- En este ejemplo, los puertos son las entradas y salidas del circuito. La lista de puertos se encierra en paréntesis, y se usan comas para separar los elementos de la lista.
- El enunciado termina con un signo de punto y coma (;).
- Todas las palabras clave (que deben estar en minúscula).
- A continuación, las declaraciones input y output definen cuáles puertos son entradas y cuáles son salidas.
- Las conexiones internas se declaran como alambres (wire).
- La lista de puertos es la interfaz a través de la cual el módulo se comunica con su entorno.

- El circuito tiene una conexión interna en la terminal **e**, la cual se declara con la palabra clave **wire**.
- La estructura del circuito se especifica empleando las compuertas primitivas predefinidas como palabras clave.
- Cada declaración de compuerta consiste en un nombre opcional (como g1, g2, etcétera) seguido de la salida y las entradas de la compuerta, separadas por comas y encerradas en paréntesis.

- La salida siempre va primero, seguida de las entradas.
- Por ejemplo, la compuerta OR se llama g3, su salida es x y tiene como entradas e y y.
- La descripción del módulo termina con la palabra clave endmodule.
- Cada enunciado termina con un signo de punto y coma, pero no hay un punto y coma después de endmodule

Retardos de compuerta

- En Verilog, el retardo se especifica en términos de unidades de tiempo y el símbolo #.
- La asociación de la unidad de tiempo con el tiempo físico se efectúa con la directriz de compilador `timescale. (Las directrices al compilador inician con el símbolo `).
- Tales directrices se especifican antes de declarar módulos. Un ejemplo de directriz de escala de tiempo es:
- `timescale 1ns/100ps
- El primer número especifica la unidad de medición de los retardos. El segundo especifica la precisión del redondeo de los retardos, en este caso a 0.1 ns.

- Si no se especifica una escala de tiempo, el simulador utilizará por omisión cierta unidad de tiempo, por lo regular 1 ns
- (1 ns=10^{-9} s).
- En los análisis supondremos la unidad de tiempo por omisión
- El ejemplo se repite la descripción del circuito simple especificando retardos para cada compuerta.
- Las compuertas AND, OR y NOT tienen un retardo de 30, 20 y 10 ns, respectivamente.
- Si se simula el circuito y las entradas cambian de 000 a 111, las salidas cambiarán como se indica en la tabla.
- La salida del inversor en y cambia de 1 a 0 después de un retardo de 10 ns.
- La salida de la compuerta AND en e cambia de 0 a 1 después de un retardo de 30 ns.

```
//Descripción de circuito con retardo
module circuito_con_retardo(A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and #(30) g1(e,A,B);
   or #(20) g3(x,e,y);
   not #(10) g2(y,C);
endmodule
```

Tabla 3-6
Salida de las compuertas después del retardo

	Unidades de tiempo	Entrada	Salida
	(ns)	ABC	*y* e *x*
Inicial	–	000	1 0 1
Cambio	–	111	1 0 1
	10	111	0 0 1
	20	111	0 0 1
	30	111	0 1 0
	40	111	0 1 0
	50	111	0 1 1

- La salida de la compuerta OR en x cambia de 1 a 0 en t=30 ns, y luego vuelve a 1 en t=50 ns.
- En ambos casos, el cambio en la salida de la compuerta OR es resultado de un cambio en sus entradas 20 ns antes.
- Es evidente, por este resultado, que si bien la salida x finalmente se estabiliza en 1 después de los cambios en sus entradas, los retardos de compuerta producen un valor negativo de 20 ns antes de que eso suceda.

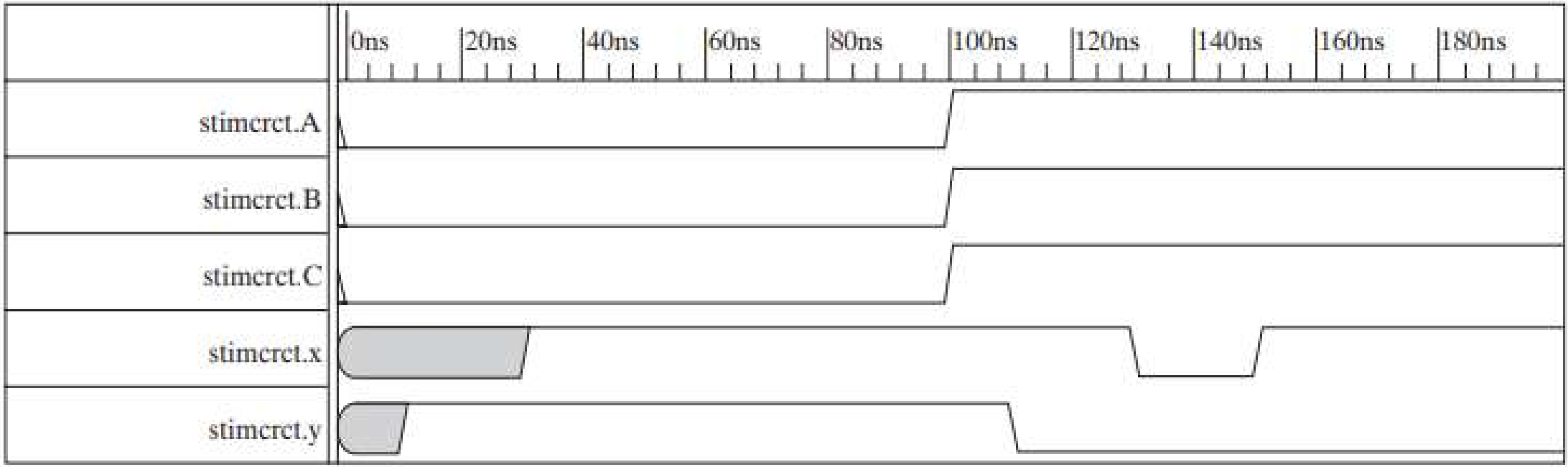
- Al simular un circuito con HDL, es necesario aplicar entradas al circuito para que el simulador genere una respuesta de salida.
- Un conjunto de pruebas es una descripción HDL que suministra el estímulo a un diseño.
- El ejemplo HDL muestra un conjunto de pruebas para estimular el circuito con retardo.
- Se incluyen dos módulos: un módulo de estímulo y el módulo que describe el circuito.
- El módulo de estímulo stimcrct no tiene puertos. Las entradas del circuito se declaran con la palabra clave reg, y las salidas, con la palabra clave wire.

- El enunciado initial especifica las entradas entre las palabras clave begin y end.
- Inicialmente, ABC=000. (A, B y C se ajustan a 1’b0, lo que significa un dígito binario con valor de 0.)
- Después de 100 ns, las entradas cambian a ABC=111.
- Después de otros 100 ns, termina la estimulación ($finish es una tarea del sistema).
- El diagrama de temporización resultado de la simulación se muestra en en la figura.

```
//Estímulo para el circuito simple
module stimcrct;
reg A,B,C;
wire x,y;
circuito_con_retardo ccr(A,B,C,x,y);
initial
   begin
        A = 1'b0; B = 1'b0; C = 1'b0;
     #100
        A = 1'b1; B = 1'b1; C = 1'b1;
     #100  $finish;
   end
endmodule

//Descripción de circuito con retardo
module circuito_con_retardo (A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and #(30) g1(e,A,B);
   or  #(20) g3(x,e,y);
   not #(10) g2(y,C);
endmodule
```

- La simulación tarda 200 ns en total.
- Las entradas A, B y C cambian de 0 a 1 después de 100 ns.
- La salida **y** se desconoce durante los primeros 10 ns, y la **X**, durante los primeros 30 ns.
- La salida y cambia de 1 a 0 a los 110 ns.
- La salida x cambia de 1 a 0 a los 130 ns y regresa a 1 a los 150 ns, tal como se predijo en la tabla anterior


0ns
20ns
40ns
60ns
80ns
100ns
120ns
140ns
160ns
180ns
stimcrct.A
stimcrct.B
stimcrct.C
stimcrct.x
stimcrct.y

Expresiones Booleanas

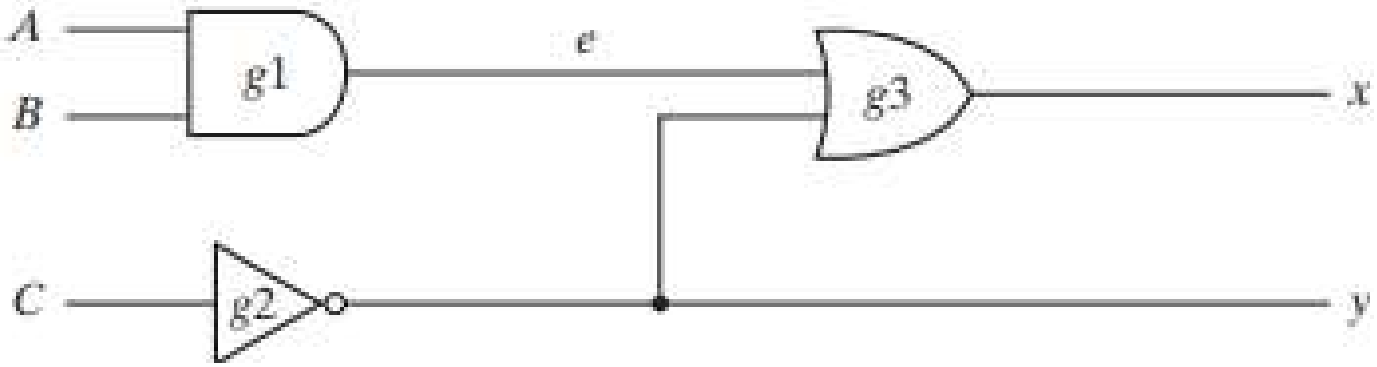
Las expresiones booleanas se especifican en Verilog HDL con un enunciado de asignación continuo que consiste en la palabra clave assign seguida de una expresión booleana.

Para distinguir el más aritmético del OR lógico, Verilog HDL usa los símbolos:

(&) para AND
| para OR
~ para NOT (complemento),

Así pues, para describir el circuito simple tratado en la ppt#7, con una expresión booleana se utiliza el enunciado:

assign x = (A & B) | ~C);



El ejemplo muestra la descripción de un circuito que se especifica con estas dos expresiones booleanas:

x = A + BC + B'D

y = B'C + BC'D'

```
//Circuito especificado con expresiones booleanas
module circuit_bln (x,y,A,B,C,D);
    input A,B,C,D;
    output x,y;
    assign x = A | (B & C) | (~B & D);
    assign y = (~B & C) | (B & ~C & ~D);
endmodule
```

- Las compuertas lógicas que se usan en las descripciones en HDL con las palabras clave and, or, not, etcétera, están definidas por el sistema y se denominan **primitivas del sistema**.
- El usuario puede crear más primitivas definiéndolas de forma tabular.
- Estos tipos de circuitos se llaman primitivas definidas por el usuario (UDP, user-defined primitives).
- Una forma de especificar un circuito digital en forma tabular es con una tabla de verdad. Las descripciones de UDP no utilizan la palabra clave module; se declaran con la palabra clave **primitive.**
- Las reglas generales son:

• Se declara con la palabra clave **primitive** seguida de un nombre y una lista de puertos.

• Sólo puede haber una salida y debe aparecer en primer lugar en la lista de puertos y declararse con la palabra clave output.

- Puede haber cualquier número de entradas.
- El orden en que aparecen en la declaración input debe ser el mismo en el que se les asigna valores en la tabla que sigue.
- La tabla de verdad se encierra entre las palabras clave **table** y **endtable.**
- Los valores de las entradas se dan en orden terminando con un signo de dos puntos (:).

La salida siempre es el último elemento de una fila y va seguida de un punto y coma (;).

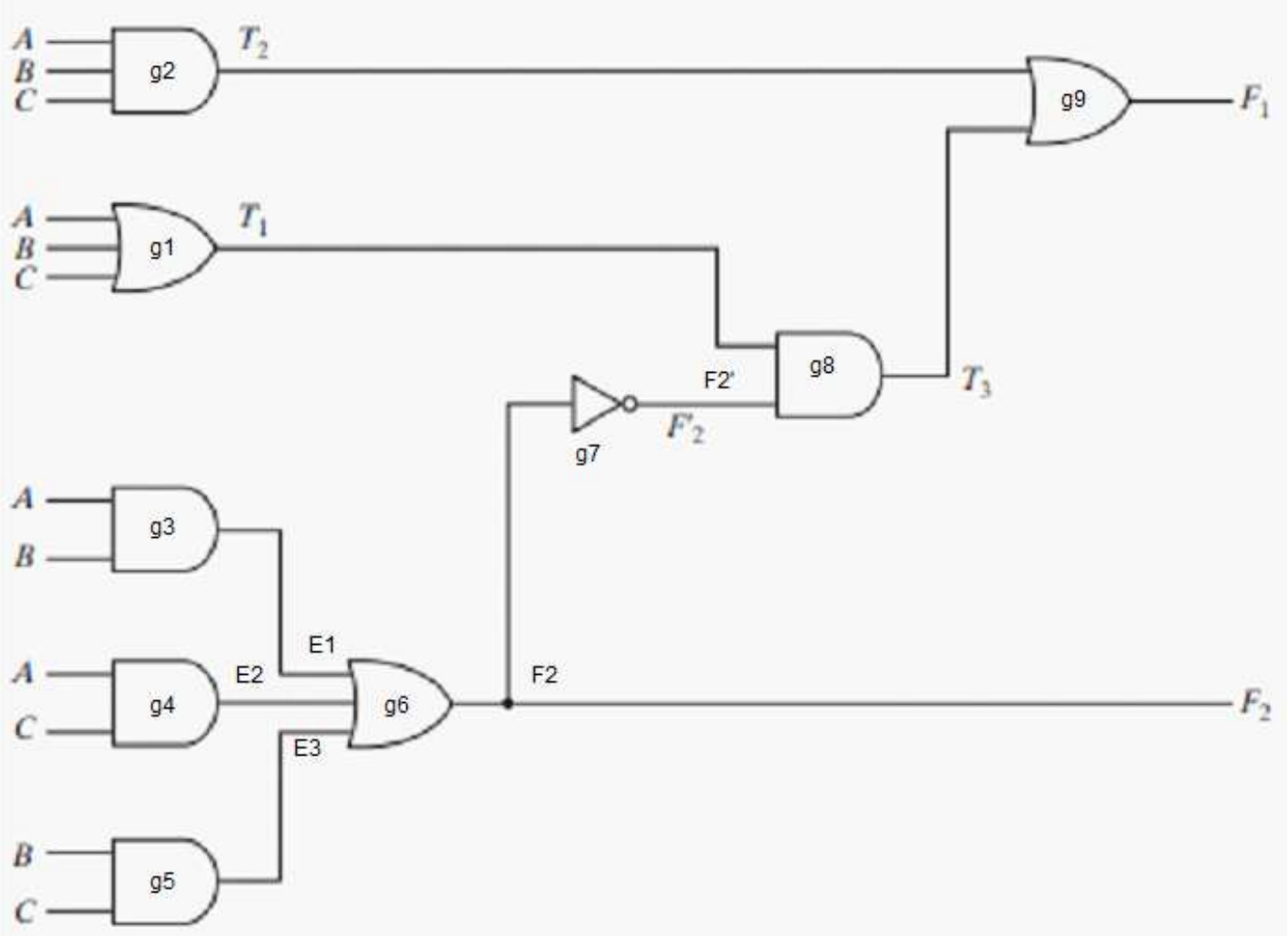
- Termina con la palabra clave **endprimitive**.

Las primitivas definidas por el usuario se utilizan en la construcción de otros circuitos digitales igual que las primitivas del sistema. Por ejemplo, la declaración crctp (w,x,y,z) produce un circuito que implementa w (x, y, z)= (0, 2, 4, 6, 7) con entradas x, y, z y salida w.

```
//Primitiva definida por el usuario (UDP)
primitive crctp (x,A,B,C);
   output x;
   input A,B,C;
//Tabla de verdad para x(A,B,C) = Σ(0,2,4,6,7)
   table
//      A   B   C  :  x   (Esto es sólo un comentario)
        0   0   0  :  1;
        0   0   1  :  0;
        0   1   0  :  1;
        0   1   1  :  0;
        1   0   0  :  1;
        1   0   1  :  0;
        1   1   0  :  1;
        1   1   1  :  1;
   endtable
endprimitive

//Crear una copia de la primitiva
module declare_crctp;
  reg x,y,z;
  wire w;
  crctp (w,x,y,z);
endmodule
```

Ejemplo circuito combinacional varios gates



Fin AOC_04