Introducción General Lenguaje Descriptivo de Hardware Verilog

Vargas Meza Aldo Alexandro, 213495653

Verilog fue inventado por Phil Moorby en 1985 mientras trabajaba en Automated Integrated Design Systems, más tarde renombrada Gateway Design Automation. El objetivo de Verilog era ser un lenguaje de modelado de hardware. Gateway Design Automation fue comprada por Cadence Design Systems en 1990. Cadence ahora tiene todos los derechos sobre los simuladores lógicos de Verilog y Verilog-XL hechos por Gateway.
Eventualmente, Cadence decidió hacer el lenguaje abierto y disponible para estandarización. Verilog se transfirió al dominio público a través de Open Verilog International, actualmente conocida como Accellera. Verilog fue después enviado a la IEEE que lo convirtió en el estándar IEEE 1364-1995, habitualmente referido como Verilog 95.

I. Introducción

Verilog es un lenguaje de descripción de hardware usado para modelar sistemas electrónicos. El lenguaje, soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción.
Los diseñadores de Verilog querían un lenguaje con una sintaxis similar a la del lenguaje de programación C, de tal manera que le resultara familiar a los ingenieros y así fuera rápidamente aceptada.

A diferencia del lenguaje C, Verilog usa Begin/End en lugar de llaves para definir un bloque de código. Por otro lado la definición de constantes en Verilog requiere la longitud de bits con su base. Verilog no tiene estructuras, apuntadores o funciones recursivas. Finalmente el concepto de tiempo que no se encuentra en C.

El lenguaje difiere de los lenguajes de programación convencionales, en que la ejecución de las sentencias no es estrictamente lineal. Un diseño en Verilog consiste de una jerarquía de módulos. Los módulos son definidos con conjuntos de puertos de entrada, salida y bidireccionales. Internamente un módulo contiene una lista de cables y registros. Las sentencias concurrentes y secuenciales definen el comportamiento del módulo, describiendo las relaciones entre los puertos, cables y registros. Las sentencias secuenciales son colocadas dentro de un bloque begin/end y son ejecutadas en orden secuencial, pero todas las sentencias concurrentes y todos los bloques begin/end son ejecutadas en paralelo en el diseño. Un módulo puede contener una o más instancias de otro módulo para definir un sub-comportamiento.

```
File  Edit  Tools  Syntax  Buffers  Window  Help
// Generated by Cadence RTL Compiler (RC) 05.20-p002

module bin_to_therm(bin, therm);
  input [2:0] bin;
  output [6:0] therm;
  wire [2:0] bin;
  wire [6:0] therm;
  wire n_0, n_1, n_2, n_4, n_5, n_10;

  BUF2 I2(.A(bin[2]), .Q(therm[3]));

  NOR20 g138(.A (n_5), .B (n_4), .Q (therm[6]));
  INV0 g139(.A (n_2), .Q (therm[2]));
  AOI210 g140(.A (bin[0]), .B (bin[1]), .C (bin[2]), .Q (n_2));
  INV0 g141(.A (n_10), .Q (therm[0]));
  NOR20 g142(.A (bin[0]), .B (therm[1]), .Q (n_10));
  OAI210 g143(.A (n_5), .B (n_0), .C (n_4), .Q (therm[4]));
  INV0 g144(.A (n_4), .Q (therm[5]));
  NAND20 g145(.A (bin[1]), .B (bin[2]), .Q (n_4));
  INV0 g146(.A (n_1), .Q (therm[1]));
  NOR20 g147(.A (bin[2]), .B (bin[1]), .Q (n_1));
  INV0 g148(.A (bin[2]), .Q (n_0));
  INV0 g149(.A (bin[0]), .Q (n_5));
endmodule
-- INSERT --                                          11,1          50%
```

Imagen 01 “Interfaz programación en lenguaje Verilog”

Un subconjunto de sentencias en el lenguaje es sintetizable. Si los módulos en un diseño contienen sólo sentencias sintetizables, se puede usar software para convertir o sintetizar el diseño en una lista de nodos que describe los componentes básicos y los conectores que deben implementarse en hardware. La lista de nodos puede entonces ser transformada en una forma describiendo las celdas estándar de un circuito integrado, por ejemplo ASIC, o una cadena de bits para un dispositivo de lógica programable (PLD) como puede ser una FPGA o un CPLD.

II. Uso del lenguaje en Verilog

Tipos de datos
Fundamentalmente existen dos tipos de datos: reg y wire. La sintaxis para declarar estas variables es la siguiente.

“<TIPO> [<MSB> : <LSB>] <NOMBRE>;”

Valor	
0	Un valor binario de cero. Corresponde a cero voltios.
1	Un valor binario de 1. Dependiendo de la tecnología
x	Un valor cualquiera. los valores x no son ni 0 o 1
z	Un valor de alta impedancia de un bufer de tres estados

Imagen 02 “Tipos de valores en Verilog”

Los diferentes tipos de variables son:
reg: Representan variables con capacidad de almacenar información.
wire: Representan conexiones estructurales entre componentes. No tienen capacidad de almacenamiento.
integer: Registro de 32 bits.
real Registro capaz de almacenar números en coma flotante
time: Registro sin signo de 64 bits.
MSB y LSB: Por defecto estas las variables son de un sólo bit, aunque es posible definir vectores de bits.

```
reg[5:0] data;
wire outA;
integer numA;
reg[31:0] numB;
```

Imagen 3 "Ejemplo declaración de variables"

III. TIPOS DE OPERADORES

Binarios Aritméticos
+ (Suma o signo positivo): Sirve para indicar una suma entre dos números.
- (Resta o signo negativo): Sirve para indicar la resta entre dos números. Si va delante de una expresión modifica el signo de la expresión.
*** (Multiplicación):** Multiplica dos números de cualquier tipo.
/ (División): Divide dos números de cualquier tipo.
% (resto): Obtiene el resto de la división de dos números de cualquier tipo.

Igualdades
Permiten comparar dos operandos, retornando 1 ó 0, verdadero o falso respectivamente.
==, != (igualdad): El primero devuelve verdadero si los operando son iguales y falso en caso contrario. El segundo indica desigualdad, funcionando al revés que el anterior.
===, !== (igualdad): Su funcionalidad es idéntica a la anterior, pero difiere en tambien se comparan los valores indefinidos ('X') o de alta impedancia ('Z').

Relacionales
Permiten comparar dos operandos, retornando 1 ó 0, verdadero o falso respectivamente.
>, >=, <, <= (menor mayor): Poseen el significado habitual (mayor que, mayor o igual que, menor que, menor o igual que, respectivamente).

Lógicos
Aparece entre dos operandos lógicos y proporciona un valor lógico (verdadero o falso).
! (not lógica): Cambia el valor lógico del operando que va justo detrás del operador.
&& (and lógica): El resultado será la combinación de los dos operandos lógicos. Es decir, para que el valor sea verdadero, ambos operandos deben serlo, en caso contrario el resultado será falso.
|| (or logica): El resultado será la combinación de los dos operandos lógicos. Para que el resultado sea verdadero, bastará con que uno de los operandos lo sea.

Lógica de bit
Permite efectuar operaciones lógicas con los bits de los operandos.
~ (negación): Negación bit a bit.
& (AND): AND bit a bit.
|| (OR): OR bit a bit.
^ (XOR): XOR bit a bit.
~& (NAND): NAND bit a bit.
~| (NOR): NOR bit a bit.
~^ (NOT XOR): NOT XOR bit a bit. También puede ser ^~.

Lógica de reducción
El resultado de aplicar este operando al único argumento es un sólo bit.
& (AND): Se realiza un AND de todos los bits.
| (OR): Se realiza un OR de cada uno de los bits del operando.
^ (XOR): Se realiza un XOR de cada bit.
~& (NAND): Se realiza un NAND de todos los bits.
~| (OR): Se realiza un NOR de cada uno de los bits del operando.
~^ (XOR): Se realiza un NOT XOR de cada bit. También puede ser ^~.

Otros
{,} (Concatenación): Concatenación de dos operandos.
<< (Desplazamiento izquierda): Desplaza bits a la izquierda, añadiendo ceros.
>> (Desplazamiento derecha): Desplaza bits a la derecha, añadiendo ceros.
?: (Condicional): Dependiendo del resultado lógico (verdadero o falso) se devolverá un valor u otro.
Los comentarios, cuya información hasta el final de la línea es ignorado, van precedidos de los caracteres //.
También, pueden ir entre /* y */ en esta segunda puede haber más de una línea.

IV. Programa Ejemplo

Programa ejemplo

```
 1 module compuertas_logicas();
 2
 3 initial begin
 4   // AND
 5   $display ("1'b1 && 1'b1 = %b", (1'b1 && 1'b1));
 6   $display ("1'b1 && 1'b0 = %b", (1'b1 && 1'b0));
 7   $display ("1'b1 && 1'bx = %b", (1'b1 && 1'bx));
 8   // OR
 9   $display ("1'b1 || 1'b0 = %b", (1'b1 || 1'b0));
10   $display ("1'b0 || 1'b0 = %b", (1'b0 || 1'b0));
11   $display ("1'b0 || 1'bx = %b", (1'b0 || 1'bx));
12   //NOT
13   $display ("! 1'b1       = %b", ( !   1'b1));
14   $display ("! 1'b0       = %b", ( !   1'b0));
15   //NAND
16   $display (" ~& 4'b1001 = %b", (~& 4'b1001));
17   $display (" ~& 4'bx001 = %b", (~& 4'bx001));
18   $display (" ~& 4'bz001 = %b", (~& 4'bz001));
19   #10  $finish;
20 end
21
22 endmodule
```

V. Bibliografía

Asic World. (s.f.). Obtenido de http://www.asic-world.com/verilog/operators1.html

ECCE. COLORADO. (s.f.). Obtenido de Colorado Education: http://ecee.colorado.edu/~ecen5837/cadence/problem_images/synth_code2.JPG

IUMA oficial. (s.f.). Obtenido de http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf

Tamu.EDu. (s.f.). Obtenido de http://students.cs.tamu.edu/tanzir/csce350/reference/verilog.html

Verilog Oficial Site. (s.f.). Obtenido de http://www.verilog.com/

Wikibooks. (s.f.). Obtenido de https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog/