

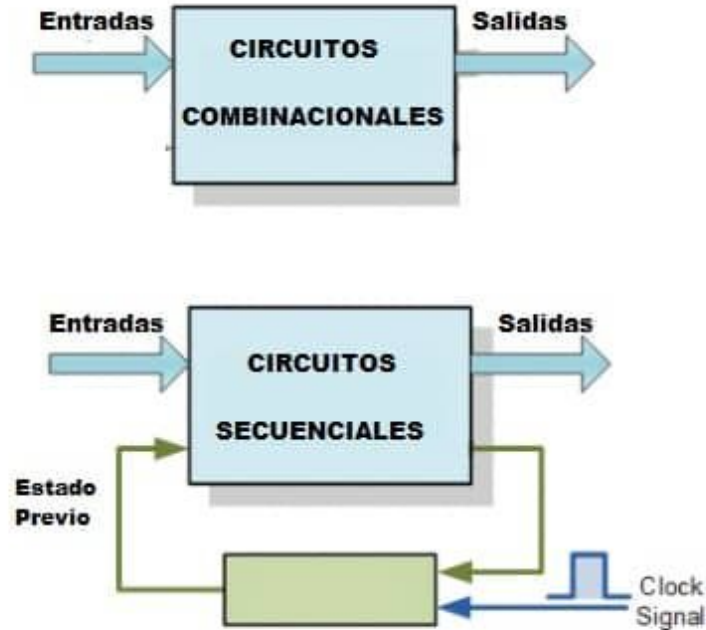


# IE 0323

# Sistemas Digitales I

Introducción a Verilog Conductual  
– Allan Alvarado & Aurelio Córdoba –

# Combinacional vs Secuencial



WWW.AREATECNOLOGIA.COM



# Circuito Combinacional

- Un circuito combinacional es un tipo de circuito digital cuya salida en cada instante depende únicamente de los valores actuales de sus entradas, sin incorporar elementos de memoria ni realimentación interna.
- No existe historial de operación ni estados previos que influyen en el resultado.
- A diferencia de los circuitos secuenciales, un circuito combinacional responde de inmediato a cualquier cambio en sus señales de entrada, ya que carece de elementos de almacenamiento que retarden o condicionen su comportamiento

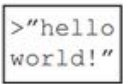


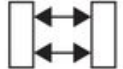
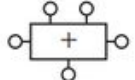
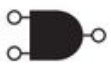
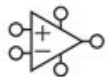




# Ejemplos Circuitos Combinacionales

- **Sumadores:** Realizan operaciones de adición binaria sin memoria.
- **Multiplexores/Demultiplexores:** Seleccionan rutas de señal basadas en líneas de control de entrada.
- **Decodificadores/Encoders:** Convierten códigos de entrada a señales de salida específicas según tablas de verdad.
- **Comparadores:** Determinan igualdad o desigualdad entre vectores de bits en un solo paso lógico.

# Niveles de abstracción

- Se refieren a las distintas capas conceptuales que separan el comportamiento de alto nivel de un sistema de su funcionamiento físico más básico.
- Estos niveles permiten a los diseñadores, programadores e ingenieros enfocarse en diferentes aspectos del sistema sin tener que lidiar con toda su complejidad de una sola vez.

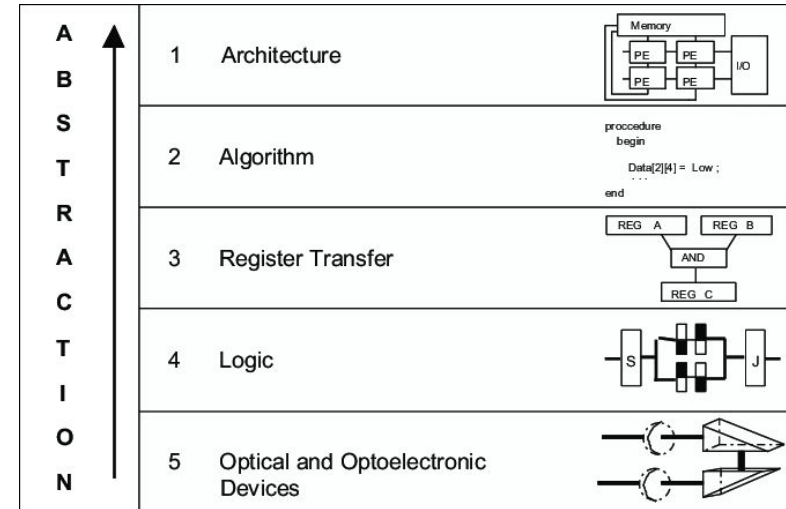
Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons



# Niveles de abstracción: Verilog Conductual



- El verilog conductual está en un nivel de abstracción más alto que el verilog estructural.
- Para pasar de uno a otro se utiliza la síntesis lógica.





# Verilog Conductual

- Estilo de descripción de hardware que se sitúa en el nivel más alto de abstracción dentro de los HDLs.
- Permitiendo describir la funcionalidad de un diseño mediante bloques procedurales y sentencias algorítmicas en lugar de conectar explícitamente puertas o componentes físicos (como se hacía en estructural).
- Enfocado a el “**qué**” se hace, más que en el “**cómo**” se hace físicamente.



# Ventajas del verilog conductual

- Mayor legibilidad y mantenimiento al abstraer detalles de implementación
- Facilita la verificación funcional con testbenches en simuladores como Verilog-XL o ModelSim
- Permite describir algoritmos combinacionales complejos (ej. aritmética, decodificadores, multiplexores) en pocas líneas de código.





# Verilog Conductual

- En hardware existen dos tipos de drivers ( tipo de dato que conduce una carga).
  1. Un driver que almacena el valor (FF), este se denomina “reg”
  2. Un driver que no puede almacenar el valor y solo pasa el dato (cable), este se denomina “wire”
- Los operadores son los mismo que en otros lenguajes de programación. Es muy parecido a C.
- Toman dos valores y los comparan u operan sobre ellos para obtener un nuevo valor.

# Operadores




Verilog Operator	Name	Functional Group
[ ]	bit-select or part-select	
( )	parenthesis	
!	logical negation	logical
~	negation	bit-wise
&	reduction AND	reduction
	reduction OR	reduction
~&	reduction NAND	reduction
~	reduction NOR	reduction
^	reduction XOR	reduction
~^ or ^~	reduction XNOR	reduction
+	unary (sign) plus	arithmetic
-	unary (sign) minus	arithmetic
{ }	concatenation	concatenation
{{ }}	replication	replication
*	multiply	arithmetic
/	divide	arithmetic
%	modulus	arithmetic
+	binary plus	arithmetic
-	binary minus	arithmetic
<<	shift left	shift
>>	shift right	shift
>	greater than	relational
>=	greater than or equal to	relational
<	less than	relational
<=	less than or equal to	relational
==	case equality	equality
!=	case inequality	equality
&	bit-wise AND	bit-wise
^	bit-wise XOR	bit-wise
	bit-wise OR	bit-wise
&&	logical AND	logical
	logical OR	logical
?:	conditional	conditional





# Representación numérica

- En verilog es común utilizar la representación decimal, en binario y en hexadecimal donde cada una de estas tiene una forma de representarse.
- Para la representación en binario y hex se utiliza el símbolo 
- Decimal: V (V = valor en decimal)
- Binario: X'bY (X=cantidad de bits, Y =valor en binario)
- Hexadecimal: X'hZ (X=cantidad de bits, Z=valor en hexadecimal)



# Representación numérica: Ejemplos

```
a = 16'b1010001101010011;  
b = 41811;  
c = 16'hA353;
```

- a, b y c tiene el mismo valor pero en diferente representación



# Variable de más de un bit

- Al definir una variable, se debe especificar la cantidad de bits que tiene esa variable. Se han venido utilizando variables de un bit, pero es posible utilizarlas con más de uno. Para la definición se utiliza los paréntesis [] de la siguiente manera:
- tipo [MSB:LSB] Nombre\_variable
- Ejemplo:

```
wire [15:0] A
```

Esta es una variable tipo wire de 16 bits



# Bitwise

- Bitwise se refiere a operadores que realizan operaciones lógicas bit a bit entre dos operandos de igual ancho, aplicando la misma función booleana a cada par de bits correspondientes de los vectores de entrada
- A diferencia de los operadores lógicos (&&, ||), que evalúan toda la expresión como un único valor booleano, los bitwise conservan el ancho y la posición de cada bit, produciendo un vector de salida donde cada bit es el resultado de la operación sobre los bits de entrada en esa posición.



# Ejemplo bitwise vs lógico

Bitwise:  $4'b0000 \& 4'b0101 = 4'b0000$  (cada bit AND con 0 da 0)

Lógico:  $4'b0000 \&\& 4'b0101 = 0$

Bitwise:  $4'b1010 \& 4'b1100 = 4'b1000$

Lógico:  $4'b1010 \&\& 4'b1100 = 1$  porque ambos vectores son distintos de cero



# Representación numérica: Ejemplo

- Binario: 4'b0010 (4 bits con valor 0010)
- Decimal: 16
- Hexadecimal: 16'hAB (16 bits valor AB)

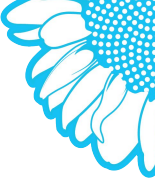




# Variable de más de un bit

- Para utilizar la variable también se utilizan los paréntesis [] pero al lado derecho
- Ejemplo: Si quiero cambiar el primer bit (LSB) de una variable de 16 bits, entonces:

$A[0] = 1;$



# Variable de más de un bit: ejemplo

```
wire [7:0] Hola;
```

```
Hola[0] = 1;
```

```
Hola[1] = 0;
```

```
Hola[2] = 1;
```

```
Hola[3] = 1;
```

```
Hola[4] = 0;
```

```
Hola[5] = 1;
```

```
Hola[6] = 1;
```

```
Hola[7] = 0;
```

La variable Hola tiene el valor 8'b01101101

Nota: Así no se asignan los valores en un código real, es para el ejemplo



# Variable de más de un bit: ejemplo

Otra manera sería:

```
wire [7:0] Hola;
```

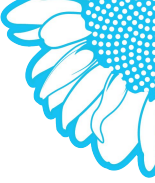
```
Hola = 8'b01101101
```



# Bloque begin-end

- Se utiliza para agrupar múltiples sentencias dentro de estructuras de control como if, case, always, etc.
- Es equivalente al uso de {} en C

```
if (x == 0)begin  
y = 0;  
b = 5;  
end
```



# Bloque “Always”

- Cuando el always detecta un cambio en una señal, se ejecuta el código o instrucciones dadas en la condición de always.
- Al terminar de ejecutar sus instrucciones, vuelve a estar pendiente del mismo cambio en la señal.
- Esta forma es la recomendada para lógica combinacional.



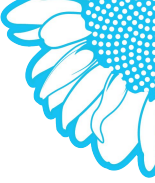
# Bloque “Always”

- El pilar de la descripción conductual de hardware, define procesos que se ejecutan de forma concurrente e indefinida durante la simulación, disparándose en función de eventos especificados o retardos temporales.
- Este bloque es una construcción procedural de Verilog que inicia un flujo de actividad independiente y concurrente con otros bloques, ejecutándose continuamente durante toda la simulación hasta su finalización.
- A diferencia del bloque initial, un always se reevalúa indefinidamente, a menos que se combine con retardos o eventos que permitan avanzar el tiempo de simulación.



## Bloque “Always”

- Mediante `always @(posedge clk)` modela lógica secuencial (flip-flops, registros). Es como decir “haz esto cada vez que el reloj suba su pulso” .
- Con `always @(*)` o listas explícitas describe circuitos combinacionales. En lugar de listar una por una las señales que le interesan, `(*)` le dice “mira todas las que usé dentro de este bloque y ejecuta las instrucciones cuando cambie cualquiera de ellas” .

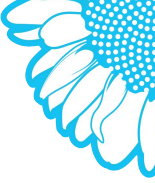


## Ejemplo “Always”

```
module mux2x1_comb (  
    input wire a,      // Primer dato de entrada  
    input wire b,      // Segundo dato de entrada  
    input wire sel,    // Señal selectora  
    output reg y       // Salida combinacional  
);  
    // Siempre que cambie 'a', 'b' o 'sel', reevalúa el bloque  
    always @(*) begin  
        if (sel)  
            y = b; // Si sel=1, y toma el valor de b  
        else  
            y = a; // Si sel=0, y toma el valor de a  
        end  
    endmodule
```



# Condicional if-else



- Es una sentencia condicional que evalúa una expresión booleana y ejecuta distintas ramas de código según el resultado.
- Se utiliza tanto en lógica combinacional como en lógica secuencial, aunque su uso más común en combinacional requiere asegurarse de cubrir todas las posibles ramas para evitar inferir latches.

# Condicional if-else

- Sintaxis

```
if (condicion1) begin
    Evaluacion_Condicion1;
end else if (condicion2) begin
    Evaluacion_Condicion2;
end else begin
    Evaluacion_else;
end
```

- Sintaxis para evaluaciones de una línea

```
if (condicion1)
    Evaluacion_Condicion1;
else if (condicion2)
    Evaluacion_Condicion2;
else
    Evaluacion_else;
```



# Ejemplos if-else

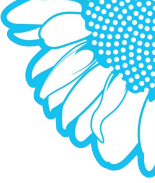
```
if (a > b)
    max = a;
else if (b > c)
    max = b;
else
    max = c;
```

```
if (a == 0) begin
    out = c + d;
    n1 = 5;
    out2 = 0;
end

else if (a >= 20)
    out = 20;

else begin
    n1 = 0;
    out = 0;
    out2 = 0;
end
```





# Loop “for”

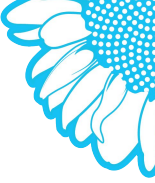
- Se utiliza para repetir una operación hasta que se llegue a una condición de parada utilizando una variable para evaluar
- Sintaxis

```
for (condicion_inicial; condicion_parada; paso) begin
|   Evaluacion
end
```

## Ejemplo “for”

```
for (i = 0; i < 4; i = i + 1) begin  
    suma = suma + i;  
end
```





# Loop “while”

- Se utiliza para repetir una operación mientras se cumpla una condición.
- Sintaxis:

```
while (condicion) begin  
|   Evaluacion  
end
```



## Ejemplo “while”

```
while (temp > 100) begin  
    temp = temp - 1;  
    out = out + 1;  
end
```



# Bloque “case”

- El case es una estructura de selección múltiple que compara una señal de expresión con varios valores literales, ejecutando la rama que coincide
- Es ideal cuando hay que distinguir múltiples combinaciones de bits de una entrada (por ejemplo, un decodificador o una máquina de estados). Para sustituir no utilizar repetidamente el if





# Bloque “case”

- Sintaxis

```
case (variable_a_analizar)
    valor1: evaluacion1;
    valor2: evaluacion2;
    ...
    default: evaluacion_default;
endcase
```



## Ejemplo “case”

```
case (in)
  2'b00: out = 4'b0001;
  2'b01: out = 4'b0010;
  2'b10: out = 4'b0100;
  2'b11: begin
    out = 4'b1000;
    out2 = 1;
  end
  default: out = 4'b0000;
endcase
```



# Assigns

- Se utiliza únicamente para modelar lógica combinacional y se ejecuta continuamente. Es una secuencia de asignación continua.
- Crea una conexión permanente entre la señal de salida (una net) y la evaluación constante de la expresión a su derecha.
- No forma parte de un proceso procedural (always/initial), por lo que debe aparecer fuera de dichos bloques, generalmente, se hace directamente bajo la declaración del módulo.



## Ejemplo “Assign”

`assign <net> = <expresión>;`

`net`: normalmente de tipo `wire` o una concatenación de varios `nets`.

`expresión`: combina variables, operadores (`&`, `|`, `^`, etc.) y constantes.

```
assign S = A + B;
```

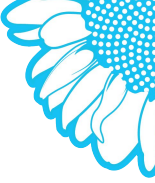


# Ejemplo: Sistema de iluminación

- Sensor de luz ambiental (L): Produce una salida en alto si la iluminación natural es suficiente.
- Sensor de presencia (P): Produce una salida en alto si hay personas en la sala (pacientes o personal médico).
- Control manual (M): Permite activar (salida en alto) o desactivar (salida en bajo) las luces independientemente de las condiciones anteriores.

El sistema debe funcionar bajo las siguientes condiciones:

- Las luces deben encenderse si no hay suficiente luz ambiental y hay personas en la sala.
- Si el control manual está activado, las luces deben encenderse sin importar las condiciones de los sensores.
- Si no hay personas en la sala, las luces deben permanecer apagadas, salvo que el control manual esté activado.



# Ejemplo: Sistema de iluminación

```
module ejemplo_sistema(L, P, M, Z);  
  
    input wire L, P, M;  
    output reg Z;  
  
    always @(*) begin  
        if (M == 1) begin  
            Z = 1;  
        end else if (L == 0 && P == 1) begin  
            Z = 1;  
        end else begin  
            Z = 0;  
        end  
    end  
  
endmodule
```



## Ejemplo: Mux4a1

- Diseñe un multiplexor de 4 a 1 que seleccione una de 4 entradas de 8 bits, dependiendo de una señal de selección llamada “sel” de 2 bits



# Ejemplo: Mux4a1

```
module mux_4a1 (A, B, C, D, sel, Y);  
  
    //Definiciones  
    input wire [7:0] A, B, C, D;    // Entradas de datos  
    input wire [1:0] sel;           // Selector de 2 bits  
    output reg [7:0] Y;             // Salida seleccionada  
  
    always @(*) begin  
        case (sel)  
            2'b00: Y = A;  
            2'b01: Y = B;  
            2'b10: Y = C;  
            2'b11: Y = D;  
            default: Y = 8'b00000000; // Por si acaso  
        endcase  
    end  
  
endmodule
```