Machete Verilog v1.1 (corregida)

Resumen práctico

0) ¿Qué es un módulo y cómo se ve "por dentro"?

Un **módulo** es una "cajita" que recibe señales (**entradas**) y entrega señales (**salidas**). Adentro tiene **cables/variables** y **lógica** que hace el trabajo.

Estructura general (partes):

1. **Inicio**: `module Nombre (lista de puertos);`
2. **Cuerpo**: señales internas y la lógica (combinacional y/o secuencial)
3. **Fin**: `endmodule`

1) Tres mini-módulos muy simples (para ver las partes)

1A) Combinacional mínimo (sin reloj) — con `assign`

```
module EjemploAssign (
  input  a, b,          // ENTRADAS: dos bits
  output y              // SALIDA: un bit
);
  // CUERPO: l gica combinacional con assign
  assign y = a ^ b;     // y = a XOR b
endmodule
```

1B) Combinacional (sin reloj) — con `always @*` y `case`

```
module EjemploCase (
  input  [1:0] sel,     // ENTRADAS
  input        a, b,
```

```
  output       y
);
  reg r;                // variable interna (se asigna en
     always)
  assign y = r;

  always @* begin
    case (sel)
      2'b00: r = 1'b0;
      2'b01: r = a & b;
      2'b10: r = a | b;
      2'b11: r = a ^ b;
      default: r = 1'b0;    // evita memorias accidentales (
         latches)
    endcase
  end
endmodule
```

1C) Secuencial mínimo (con reloj) — FF-D con reset

```
module EjemploDFF (
  input clk,            // reloj
  input rst,            // reset (activo alto)
  input D,              // dato de entrada
  output Q              // salida almacenada
);
  reg q;                // almac n (registro)
  assign Q = q;

  always @(posedge clk or posedge rst) begin
    if (rst) q <= 1'b0;  // en reset, guardo 0
    else     q <= D;     // copio D en q en el flanco de clk
  end
endmodule
```

2) "Ancho" de un bus, MSB y LSB (explicado simple)

- **Ancho** = cuántos bits tiene una señal. Ej.: `input [7:0] a;` significa 8 bits.
- **MSB** (Most Significant Bit) = bit más grande (izquierda, índice mayor).
- **LSB** (Least Significant Bit) = bit más chico (derecha, índice menor).

Mini-ejemplo:

```
wire [7:0] dato;            // 8 bits: dato[7] ... dato[0]
wire       msb = dato[7];
wire       lsb = dato[0];
wire [3:0] alto = dato[7:4];  // mitad alta
wire [3:0] bajo = dato[3:0];  // mitad baja
```

3) `wire` vs `reg` (cuándo usar cada uno) — con ejemplos

Idea sencilla:

- `wire` = *cable*. Se usa con `assign` o para conectar módulos.
- `reg` = *variable* que se asigna dentro de un `always`.

Ejemplo A — `wire` con `assign`:

```
wire y;
assign y = a & b;   // y es wire, calculado con assign
```

Ejemplo B — `reg` dentro de `always @*`:

```
reg y;              // y es reg porque se asigna en always
always @* begin
  if (sel) y = a;
  else     y = b;
end
```

Nota sobre salidas: si una salida se asigna en `always`, puede ser `output reg`.

```
module EjemploSalidaReg (input a, b, sel, output reg y);
  always @* begin
    y = sel ? a : b;
  end
endmodule
```

4) Lógica combinacional (sin reloj)

Cuando la salida depende **sólo de entradas actuales**.

4A) Opción 1 — con `assign`

```
assign y = (a & b) | (~a & c);
```

4B) Opción 2 — con `always @*`

```
reg y;
always @* begin
  // IMPORTANTE: dar SIEMPRE un valor en todas las ramas (
     evita latches)
  if (sel) y = a;
  else     y = b;
end
```

Checklist combinacional:

- Usar `assign` o `always @*`.
- Cubrir todas las ramas (`else` / `default`).
- Cuidar anchos (coincidencias de bits).

5) Lógica secuencial (con reloj)

Cuando la salida depende de **entradas + estado previo**.

5A) Reset síncrono

```
always @(posedge clk) begin
  if (rst) q <= 1'b0;
  else     q <= d;
end
```

5B) Reset asíncrono

```
always @(posedge clk or posedge rst) begin
  if (rst) q <= 1'b0;
  else     q <= d;
end
```

Reglas de oro en secuencial:

- Usar `<=` (no bloqueante) dentro de `always @(posedge clk ...)`.
- Aclarar tipo de reset (síncrono o asíncrono).

6) Lista de sensibilidad y bloqueante/no bloqueante

6A) Lista de sensibilidad (cuándo "se dispara" el bloque)

- `always @*`: recomputa cuando cambia algo usado adentro (combinacional).
- `always @(posedge clk)`: ejecuta en cada flanco de subida (secuencial).
- `always @(posedge clk or posedge rst)`: idem, o cuando `rst` sube (reset asíncrono).

6B) = (bloqueante) vs <= (no bloqueante)

- = bloqueante: cambia al instante dentro del bloque (usar en combinacional procedural).
- <= no bloqueante: programa el cambio para el final del flanco (usar en secuencial).

Mini-demostración (conceptual):

```
// Combinacional con '='
reg x, y, z;
always @* begin
  x = a;       // x toma a
  y = x;       // y toma el NUEVO x (a)
  z = y;       // z toma el NUEVO y (a)
end
// Resultado: x=y=z=a

// Secuencial con '<='
always @(posedge clk) begin
  x <= a;      // se actualizan al final del flanco
  y <= x;      // y toma el x ANTERIOR
  z <= y;      // z toma el y ANTERIOR
end
```

7) Estructuras de control comunes

- `if / else`: elegir entre dos caminos.

- `case`: elegir entre varios (como un MUX). Siempre poner `default`.

Ejemplo `case` como MUX 4→1:

```
always @* begin
  case (sel)
    2'b00: y = d0;
    2'b01: y = d1;
    2'b10: y = d2;
    2'b11: y = d3;
    default: y = 1'b0;
  endcase
end
```

8) Literales en Verilog (reemplaza al antiguo bloque 8)

Formato general:

`<ancho>'<base><valor>`

Partes:

- **ancho**: cuántos bits ocupa el número.
- **base**: `b` (binario), `d` (decimal), `h` (hexadecimal).
- **valor**: el número en esa base.

Ejemplos útiles:

- `1'b0` → 1 bit, valor 0 en binario.
- `1'b1` → 1 bit, valor 1 en binario.
- `2'b00` → 2 bits, valor 0.
- `2'b11` → 2 bits, valor 3.
- `4'd9` → 4 bits, valor 9 decimal.
- `8'hA5` → 8 bits, valor A5 hex (165 decimal).

Por qué usarlos: fijan el *ancho* exacto y evitan truncados/expansiones inesperadas.

9) ¿Qué uso según lo que me pidan? (guía rápida)

Si dice COMBINACIONAL:

- Usar `assign` o `always @*`.
- Usar `=` adentro de `always @*`.
- Cubrir todas las ramas (`else` / `default`).
- Nada de `clk` ni `<=`.

Si dice SECUENCIAL:

- Usar `always @(posedge clk [or posedge rst])`.
- Usar `<=` (no bloqueante).
- Definir comportamiento de **reset**.

Casos típicos:

- MUX/decoder/ALU chiquita → combinacional con `case`.
- Registro/contador/shift → secuencial con reloj.

10) Checklist antes de entregar

- ¿Combinacional bien cerrado? (`assign` o `always @*`)
- ¿Secuencial con `<=` y reloj?
- ¿Declaré bien anchos `[MSB:LSB]`?
- ¿Mis literales tienen ancho correcto (`N'baseValor`)?
- ¿Cubrí todas las ramas con `else` / `default`?