

Electrónica Digital Práctica

2025

Temario

1- Sintaxis II

2- Diseño de circuitos Secuenciales

2.1- Biestables

2.2- Registros

3- Simulación II

Sintaxis II

1.1- Constantes

1.2- Concatenación

1.3- Lista de sensibilidad con detección de flancos

1.4- Asignaciones Bloqueantes / No Bloqueantes

Sintaxis II - 1.1 Constantes

Dentro de un módulo se pueden definir constantes utilizando `parameter`

Ejemplos:

```
parameter DURATION = 100;
```

```
parameter uno = 4'b0001, ultimo = 4'b1111;
```

```
reg[3:0] x;
```

```
x = ultimo;
```

Sintaxis II - 1.2 Concatenación

Se utiliza para agrupar señales formando un arreglo

- Sintaxis: {señalX, señalY,}

```
module concatena(  
    input a,b,c,  
    output reg igual_a_3  
);  
  
always @(*)  
    case({a,b,c})  
        3'b011: igual_a_3 = 1;  
        default: igual_a_3 = 0;  
    endcase  
endmodule
```

Sintaxis II - 1.3 Lista de sensibilidad - Detección de flancos

Sirve para que un proceso sólo se ejecute en determinados flancos de reloj o de otras señales de entrada.

- Se indica en la lista de sensibilidad de un proceso mediante un prefijo a la señal:

- El prefijo **posedge** detecta el flanco de subida

- El prefijo **negedge** detecta el flanco de bajada

```
module detector_flanco(  
    input clk,  
    output reg z);  
  
    always @(posedge clk)  
        ....  
endmodule
```

Sintaxis II - 1.4 Asignación Bloqueante

- Si en un proceso always se desea que la salida cambie inmediatamente, se debe utilizar una asignación bloqueante.
- Modelan, sobretodo, salidas combinacionales.
- Importa el orden en que se efectúan las asignaciones bloqueantes puesto que las acciones en un proceso se ejecutan secuencialmente, una detrás de otra, en el orden en que aparecen

```
module bloqueante(  
    input a,clk,  
    output reg z2);  
  
    reg q;  
  
    always @(posedge clk)  
        begin  
            q = a;  
            z2 = q;  
            /* equivalen a z2=a */  
        end  
endmodule
```

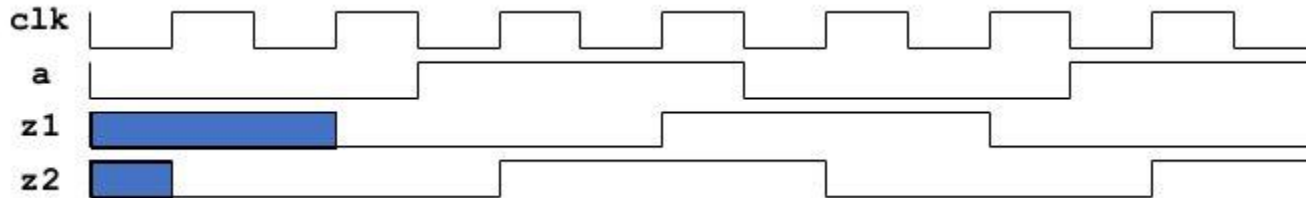
Sintaxis II - 1.4 Asignación No Bloqueante

- Modela los cambios de estados en flip-flops.
- Define “el próximo estado”
- Se calculan primero los valores de la derecha de la asignación \leq ; tras esto, se asignan todas simultáneamente.
- Cuando se tienen varias asignaciones no bloqueantes, no importa el orden en que son escritas.

Comparación: asignación No bloqueante - asignación bloqueante

```
module no_bloqueante(input a,clk,  
    output reg z1);  
    reg q;  
    always @(posedge clk)  
        begin  
            q <= a;  
            z1 <= q; //Cambia en prox ciclo  
        end  
endmodule
```

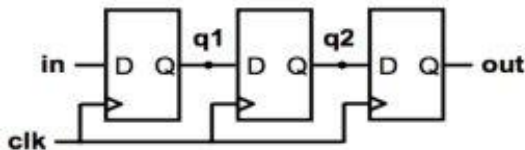
```
module bloqueante(input a,clk,  
    output reg z2);  
    reg q;  
    always @(posedge clk)  
        begin  
            q = a;  
            z2 = q;  
        end  
endmodule
```



Comparación: asignación No bloqueante - asignación bloqueante

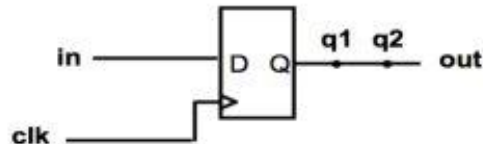
```
module no_bloqueante(input in,clk,  
    output reg out);  
    reg q1, q2;  
    always @(posedge clk)  
        begin  
            q1 <= in;  
            q2 <= q1;  
            out <= q2;  
        end  
endmodule
```

“At each rising clock edge, *q1*, *q2*, and *out* simultaneously receive the old values of *in*, *q1*, and *q2*.”



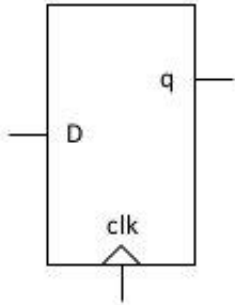
```
module bloqueante(input in,clk,  
    output reg out);  
    reg q1, q2;  
    always @(posedge clk)  
        begin  
            q1 = in;  
            q2 = q1;  
            out = q2;  
        end  
endmodule
```

“At each rising clock edge, *q1* = *in*.
After that, *q2* = *q1* = *in*.
After that, *out* = *q2* = *q1* = *in*.
Therefore *out* = *in*.”



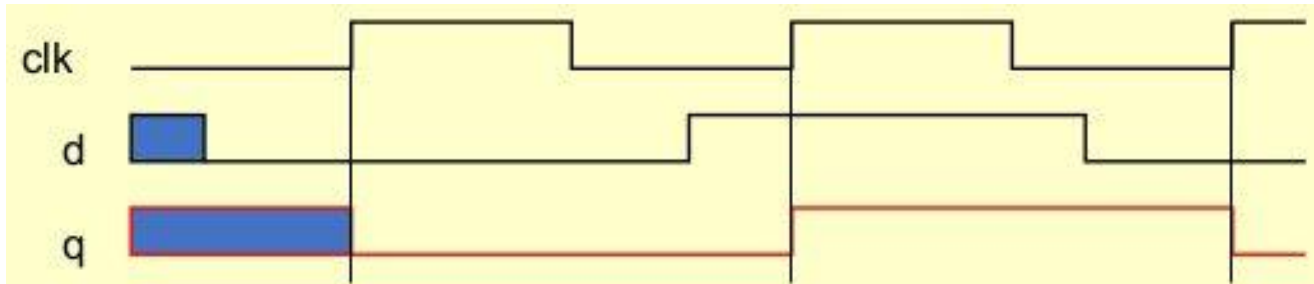
2.1 Biestables

Ejemplo FF D



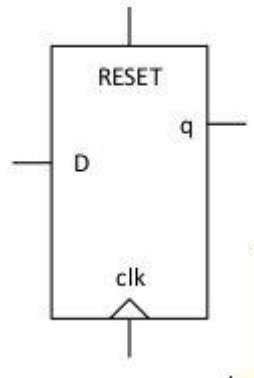
```
module biestable_d(  
    input clk,d,  
    output reg q);
```

```
    always @ (posedge clk)  
        q <= d;  
endmodule
```



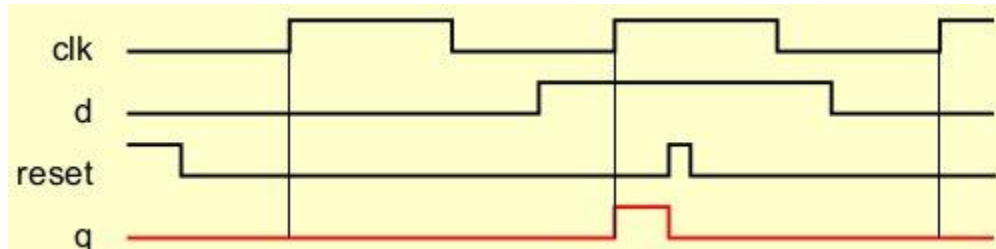
2.1 Biestables

Ejemplo FF D con reset asíncrono



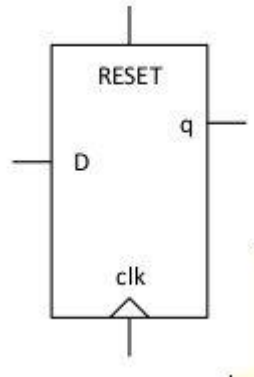
```
module biestable_d(  
    input clk, reset, d,  
    output reg q);
```

```
    always @ (posedge clk or posedge reset)  
        if (reset)  
            q <= 1'b0;  
        else  
            q <= d;  
    endmodule
```

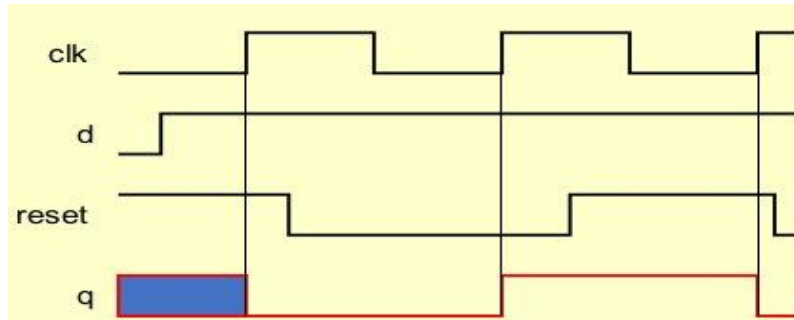


2.1 Biestables

Ejemplo FF D con reset síncrono

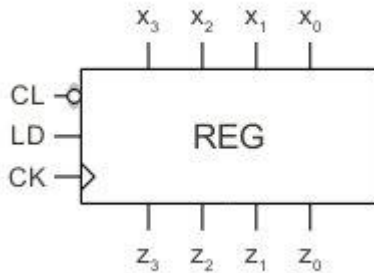


```
module biestable_d(  
    input clk,d,reset,  
    output reg q);  
always @ (posedge clk)  
    if (reset)  
        q <= 1'b0;  
    else  
        q <= d;  
endmodule
```



2.2 Registros

Ejemplo Registro de carga en paralelo con clear asíncrono

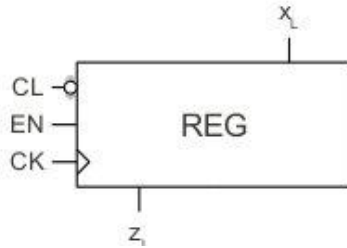


CL, LD	Operation	Type
0x	$q \leftarrow 0$	async.
11	$q \leftarrow x$	sync.
10	$q \leftarrow q$	sync.

```
module registro(  
    input ck, cl, ld,  
    input [3:0] x,  
    output [3:0] z);  
    reg [3:0] q;  
    always @(posedge ck, negedge cl)  
        if (cl == 0) q <= 4'b0;  
        else if (ld == 1) q <= x;  
    assign z = q;  
endmodule
```

2.2 Registros

Ejemplo Registro de desplazamiento



CL, EN	Operation	Type
0x	$q \leftarrow 0$	async.
11	$q \leftarrow \text{SHL}(q)$	sync.
10	$q \leftarrow q$	sync.

```
module reg_shl(  
    input ck, cl, en, xl,  
    output zl);  
    reg [3:0] q;  
    always @(posedge ck, negedge cl)  
        if (cl == 0) q <= 0;  
        else if (en == 1) q <= {q[2:0], xl};  
    assign zl = q[3];  
endmodule
```

3 Simulación de módulos secuenciales

- Tenemos que dar valores a las entradas de datos (X), de reloj (clk) y de inicialización (clear, reset...) para comprobar que el módulo responde adecuadamente al diagrama de estados diseñado
- La señal clk se modela fácilmente en un bloque `always`:

Ej. `always #5 clk = !clk; // clk T=10ns`

- La secuencia de valores de entrada se hace en el bloque `initial` y podemos:
- asignar valor a variables:
- Esperar un tiempo
- Esperar a que llegue un flanco de la señal de reloj:
 - `@(posedge clk)` ; espera a que llegue un flanco de subida
 - `@(negedge clk)`; espera a que llegue un flanco de bajada

3 Simulación de módulos secuenciales

```
`default_nettype none
`define DUMPSTR(x) `"x.vcd`"
`timescale 100 ns / 10 ns
```

```
module biestable_d_tb();
parameter DURATION = 100;
reg sd;
wire sq;
```

```
    biestable_d UUT(
        .clk(sclk),
        .d(sd),
        .q(sq)
    );
```

```
reg sclk = 0;
always #0.5 sclk = ~sclk;
```

```
initial begin
    $dumpfile(`DUMPSTR(`VCD_OUTPUT));
    $dumpvars(0, biestable_d_tb);

    #10
    sd = 1'b0;
    #25
    sd = 1'b1;
    #25
    sd = 1'b0;
    ...

    #(DURATION) $display("End of simulation");
    $finish;
end
```

Hands On !