Ejemplos de códigos de FPGA

Compuertas lógicas (Operaciones bitwise)

- **Ejemplo 1**

```
module compuertas(
   input a,       //Declaramos la entrada a
   input b,       //Declaramos la entrada b
   output z0,     //Declaramos la salida z0
   output z1,     //Declaramos la salida z1
   output z2,     //Declaramos la salida z2
   output z3,     //Declaramos la salida z3
   output z4,     //Declaramos la salida z4
   output z5      //Declaramos la salida z5
   );

assign z0=a|b;       //Asignamos la operacion OR a la salida z0
assign z1=a&b;       //Asignamos la operacion AND a la salida z1
assign z2=a^b;       //Asignamos la operacion XOR a la salida z2
assign z3=!(a|b);    //Asignamos la operacion NOR a la salida z3
assign z4=!(a&b);    //Asignamos la operacion NAND a la salida z4
assign z5=!(a^b);    //Asignamos la operacion XNOR a la salida z5

endmodule
```

- **Ejemplo 2**

```
module compuertas(
   input [1:0]i,       //Declaramos el bus de entrada i
   output [5:0]z       //Declaramos el bus de salida z
   );

assign z[0]=|i;       //Asignamos la operacion OR a la salida z0
assign z[1]=&i;       //Asignamos la operacion AND a la salida z1
assign z[2]=^i;       //Asignamos la operacion XOR a la salida z2
assign z[3]=~|i;    //Asignamos la operacion NOR a la salida z3
assign z[4]=~&i;    //Asignamos la operacion NAND a la salida z4
assign z[5]=~^i;    //Asignamos la operacion XNOR a la salida z5

endmodule
```

Conexiones internas auxiliares

```
module codigo_wire(
   input a,
   input b,
   input c,
   output z
   );
wire aux;          //Declaramos la conexion aux de tipo wire

assign aux=a&b;    //asignamos a aux la salida de la compuerta AND entre a y b
assign z=aux|c;    //asignamos a z la salida de la compuerta OR entre aux y c.

endmodule
```

Multiplexores

- **Ejemplo 1 – MUX 2x1**

```
module mux2_1(
   input I0,              //Declaramos la entrada I0
   input I1,              //Delaramos la entrada I1
   input sel,             //Declaramos la entrada sel
   output out             //Declaramos la salida out
   );

assign out=sel?I1:I0;  //Si sel es igual a 1 entonces out sera igual a I1
                       //De lo contrario out sera igual a I0
endmodule
```

- **Ejemplo 2 – MUX 4x1**

```
module mux4_1(
   input [3:0]I,       //Declaramos el bus de entrada I
   input [1:0]Sel,    //Declaramos el bus de entrada S
   output out          //Declaramos la salida out
   );

assign out= I[Sel];    //Out sera igual a la entrada correspondiente a la seleccion.
                       //por ejemplo si sel=1 entonces out=I[1]
endmodule
```

Convertidor hexadecimal a 7 segmentos

```
// Autor: Luis Alberto Vargas Tijerino.
// Create Date:    22:21:03 10/24/2010
// Module Name:    hex7seg
```

```
// Pais:  Nicaragua
module hex7seg(
  input [3:0] in,      //Declaramos el bus de entrada in
  output [6:0] a_to_g, //Declaramos el bus de salida a_to_g, para controlar los katodos.
  output [3:0] an      //Declaramos el bus de salida an, para controlar los andos.
  );

//Tabla para realizar la decodificacion de hex a 7-seg.
assign a_to_g= (sw==0)? 7'b000_0001:
            (sw==1)? 7'b100_1111:
            (sw==2)? 7'b001_0010:
            (sw==3)? 7'b000_0110:
            (sw==4)? 7'b100_1100:
            (sw==5)? 7'b010_0100:
            (sw==6)? 7'b010_0000:
            (sw==7)? 7'b000_1111:
            (sw==8)? 7'b000_0000:
            (sw==9)? 7'b000_0100:
            (sw=='hA)? 7'b000_1000:
            (sw=='hB)? 7'b110_0000:
            (sw=='hC)? 7'b011_0001:
            (sw=='hD)? 7'b100_0010:
            (sw=='hE)? 7'b011_0000:
                  7'b011_1000;
assign an=0;  //se encenderan todos los display.
endmodule
```

Máquinas de estados secuenciales

1. **Contador de corrida libre**

- **Ejemplo 1**

```
module contador(
   input clk,         //Declaramos la entrada clk
   input clr,         //Declaramos la entrada clr
   output reg [3:0] Q  //Declaramos el bus de salida Q.
   );

always @(posedge clk or posedge clr)
   if(clr)Q<=0;       //Si se activa clr el siguiente valor de Q sera 0
   else Q<=Q+1;       //De lo contrario el siguiente valor de Q sera Q+1.

endmodule
```

- **Ejemplo 2**

```
module contador#(
   parameter N=4)(       //Declaramos el parametro N=4
   input clk,            //Declaramos la entrada clk
   input clr,            //Declaramos la entrada clr
   output reg [N-1:0] Q  //Declaramos el bus de salida Q.
   );

always @(posedge clk or posedge clr)
   if(clr)Q<=0;        //Si se activa clr el siguiente valor de Q sera 0
   else Q<=Q+1;        //De lo contrario el siguiente valor de Q sera Q+1.

endmodule
```

2. **Contador modulable**

```
module cont_modN#(
  parameter N=3,       //Declaramos el Parametro N igual a 3.
  parameter Mod=9      //Declaramos el Parametro Mod igual a 9.
  )(
  input clk,           //Declaramos la entrada clk
  input clr,           //Declaramos la entrada clr
  output reg [N-1:0] Q //Declaramos el bus de salida tipo registro Q.
  );

always @(posedge clk or posedge clr) begin
  if(clr) Q<=0;          //Si se presiona clr el proximo valor de Q sera 0 de forma
asincrona.
  else if(Q==Mod) Q<=0;  //De lo contrario Si Q es igual a Mod (en este caso 9) el
proximo valor de Q sera 0.
  else Q<=Q+1;           //De lo contrario el proximo valor de Q sera 0.
end

endmodule
```

3. **Contador de Anillo**

```
module cont_despN #(
   parameter N=4)(
   input clk,
   input clr,
   output reg [N-1:0] Q
   );

always @(posedge clk or posedge clr)
```

```
  if(clr) Q <= 4'b0001;     //Si se Activa clr Q sera igual a 4'b0001
  else Q <= {Q[N-2:0],Q[N-1]};          //De lo contrario Q sera igual al valor de Q
dezplazado un bit hacia la izquierda.

endmodule
```

4. **Divisor de reloj**

```
module clkdiv(
    input mclk,
    input clr,
    output clk190,
    output clk25
    );

reg [17:0] q;

//contador de 18-bit
always @(posedge mclk or posedge clr)
    if(clr) q <= 0;        //Si se activa clr el proximo valor de q sera 0
    else    q <= q + 1;    //de lo contrario el siguiente valor de q sera q+1

assign clk25  = q[0];    //50MHz/2^(0+1)=25MHz
assign clk190 = q[17];   //50MHz/2^(0+17)=190Hz

endmodule
```

Controlador de los displays de 7 segmentos completos

```
module hex7seg(
   input clr,
   input clk,
   input [15:0] x,      //Declaramos el bus de entrada in
   output [6:0] a_to_g, //Declaramos el bus de salida a_to_g, para controlar los katodos.
   output [3:0] an      //Declaramos el bus de salida an, para controlar los andos.
   );

wire [3:0]in;  //Declaramos el bus auxiliar tipo wire de 4 bits in.
reg [1:0]sel;  //Declaramos el bus auxiliar tipo reg de 2 bits sel.

//Contador de corrida libre de 2 bits.
always @(posedge clk or posedge clr)   //Siempre que ocurra el flanco positivo de clk o
clr.
   if(clr)sel<=0;    //Si se activa clr el siguiente valor de sel sera 0.
   else sel<=sel+1;  //De lo contrario el siguiente valor de sel sera sel+1.
```

```
//Multiplexor Quadruple de 4 entradas 1 salidas.
assign in=x[sel*4+:4]; //Si sel=2 entonces in=x[2*4+3+:4] o in=x[11:8].
                       //Por lo tanto se seleccionan las 4 entradas correspondientes al display
a encender.

//Decodificador de 2 a 4. Selecciona el bit de salida correspondiente.
assign an=(sel==0)?4'b1110:   //Si sel es igual a 0 an sera igual a 4'b1110
        (sel==1)?4'b1101:   //Si sel es igual a 1 an sera igual a 4'b1101
        (sel==2)?4'b1011:   //Si sel es igual a 2 an sera igual a 4'b1011
                4'b0111;   //Si sel es igual a 3 an sera igual a 4'b0111

//Tabla para realizar la conversion de hex a 7-seg.
assign a_to_g= (in==0)? 7'b000_0001:
                (in==1)? 7'b100_1111:
                (in==2)? 7'b001_0010:
                (in==3)? 7'b000_0110:
                (in==4)? 7'b100_1100:
                (in==5)? 7'b010_0100:
                (in==6)? 7'b010_0000:
                (in==7)? 7'b000_1111:
                (in==8)? 7'b000_0000:
                (in==9)? 7'b000_0100:
                (in=='hA)? 7'b000_1000:
                (in=='hB)? 7'b110_0000:
                (in=='hC)? 7'b011_0001:
                (in=='hD)? 7'b100_0010:
                (in=='hE)? 7'b011_0000:
                        7'b011_1000;

endmodule
```

Contador BCD

- **Ejemplo 1**

```
module Tap4x4_16(
   input [3:0] x15_12,
   input [3:0] x11_8,
   input [3:0] x7_4,
   input [3:0] x3_0,
   output [15:0] x
   );

assign x={x15_12,x11_8,x7_4,x3_0};
```

```
endmodule
```

- **Ejemplo 2**

```
module Tap4_1w3(
   input [3:0] x,
   output x3
   );

assign x3=x[3];

endmodule
```

Circuito anti-rebotes

```
module debounce4(
  input [3:0] inp,     //Declaramos la entrada tipo bus de 4 bits inp
  input cclk,          //Declaramos la entrada clk
  input clr,           //Declaramos la entrada clr
  output [3:0] outp    //Declaramos la salida tipo bus de 4 bits outp
  );

reg [3:0] delay1;      //Declaramos la conexion auxiliar tipo reg delay1
reg [3:0] delay2;      //Declaramos la conexion auxiliar tipo reg delay2
reg [3:0] delay3;      //Declaramos la conexion auxiliar tipo reg delay3

always @(posedge cclk or posedge clr) begin
  if(clr) begin        //Si se activa clr
    delay1<=0;         //el siguiente valor de delay1 sera 0
    delay2<=0;         //el siguiente valor de delay2 sera 0
    delay3<=0;         //el siguiente valor de delay3 sera 0
  end
  else begin           //De lo contrario
    delay1<=inp;       //el siguiente valor de delay1 sera inp
    delay2<=delay1;    //el siguiente valor de delay2 sera delay1
    delay3<=delay2;    //el siguiente valor de delay3 sera delay2
  end
end

//Si la entrada inp es 1 durante 3 ciclos de cclk  outp sera 1.
assign outp = delay1 & delay2 & delay3;   //outp sera 1 si delay1, delay2 y delay3 son 1.

endmodule
```

Generador de pulsos de reloj

```
module clock_pulse(
  input inp,       //Declaramos la entrada inp
  input cclk,      //Declaramos la entrada cclk
  input clr,       //Declaramos la entrada
  output outp      //Declaramos la salida outp
  );

reg [2:0] delay;    //Declaramos el registro auxiliar de 3 bits delay

always @(posedge cclk or posedge clr)
  if(clr) delay<=0;                 //Si se activa clr el proximo valor de delay sera 0.
  else delay[2:0]<={delay[1:0],inp};  //de lo contrario el proximo valor de delay[2] sera
delay[1]
                                    //el proximo valor de delay[1] sera delay[0] y
                                    //el proximo valor de delay[0] sera imp.

//Si imp permanece activo durante 2 ciclos de reloj outp se activara y en el siguente
ciclo se desactivara
assign outp = delay[0] & delay[1] & ~delay[2];//outp sera 1 si delay[0] y delay[1] son 1
y delay[2] es 0.

endmodule
```

Máquinas de estados aleatorios

```
module maquinaestados(
   input clk,
   input clr,
   output [1:0] present_state
   );
reg [1:0] next_state;              //Declaramos la variable auxiliar next_state.
assign present_state=next_state;    //Declaramos que next_state se convertira en
present_state.

always@(posedge clk or posedge clr) //Siempre que ocurra el flanco positivo de clk o
clr.
  if(clr)next_state=0;             //Si se activa clr el proximo estado sera 0.
  else                          //De lo contrario
    case(present_state)          //En dependecia del estado presente.
      0:next_state=1;           //Si el estado presente es 0 el proximo Estado sera 1.
      1:next_state=3;           //Si el estado presente es 1 el proximo Estado sera 3.
      2:next_state=1;           //Si el estado presente es 2 el proximo Estado sera 1.
      3:next_state=2;           //Si el estado presente es 3 el proximo Estado sera 2.
```

```
        default:next_state=0;       //Por defecto el proximo estado sera 0.
    endcase

endmodule
```