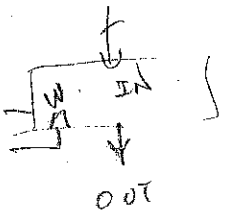
// Declaración módulo tipo A, B, C, D

```
module type1 #(parameter width = 8, initial_value=0)
        (input wire [width-1:0] IN, input wire W, cK,
         output reg [width-1:0] OUT);

        always @ (posedge cK)
            if (W==1)
                OUT <= IN;

endmodule
```



// Declaración módulo tipo MUX

```
module type2 #(parameter width = 8)
        (input wire [width-1:0] IN0, IN1, IN2, IN3, input wire S1,
         output reg [width-1:0] OUT);

        always @ (*)
            case ({S1, S0})
                2'h0 : OUT = IN0;
                2'h1 : OUT = IN1;
                2'h2 : OUT = IN2;
                2'h3 : OUT = IN3;
            endcase
endmodule
```

// Declaración del dec

```
module type3 (input wire d1, d0, EN, output reg [3:0] out);

        always @ (d1, d0, EN) begin
            if (EN==1)
                out = 0;
```

```
else begin
  case ({d1, d0})
    2'h0 : out = 4'b 0001;
    2'h1 : out = 4'b 0010;
    2'h2 : out = 4'b 0100;
    2'h3 : out = 4'b 1000;
  endcase
 end
end
endmodule

// declaración de la unidad de datos

module unidad_datos #(parameter width = 8)
                    (input wire S1, S0, d1, d0, EN, CK);

wire [3:0] w;
wire [width-1:0] OUT [3:0]; wire MUX_OUT;

type1 (width,0) A (MUX_OUT, w[0], CK, OUT[0]);
type1 (width,0) B (MUX_OUT, w[1], CK, OUT[1]);
type 1   "      C    "   w[2]  "  OUT[2]);
type 1   "      D (  "   w[3]  "  OUT[3]));
type 2 (width) miMUX (OUT[0], OUT[1], OUT[2], OUT[3], S1, S0, MUX_OUT);
type 3 miDEC (d1, d0, EN, w);

endmodule
```