```
module addbit(
  a,
  b,
  ci,
  sum,
  co
);
```

DEclare ports

```
  input a;
  input b;
  input ci;

  output sum;
  output co;

  wire a;
  wire b;
  wire ci;
  wire sum;
  wire co;

  assign {co,sum} = a + b + ci;

endmodule~~;~~
```

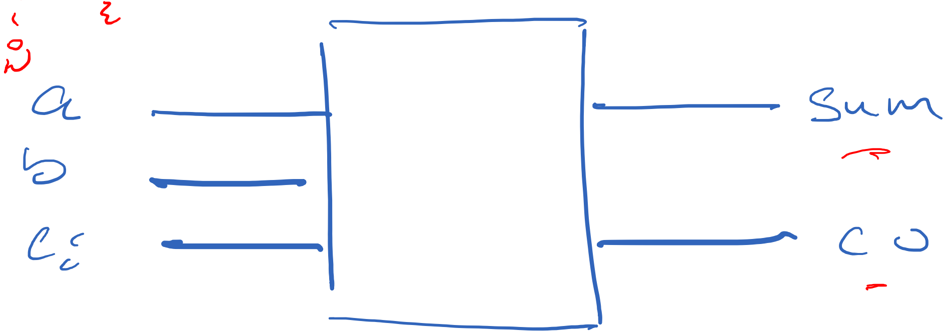
HDL

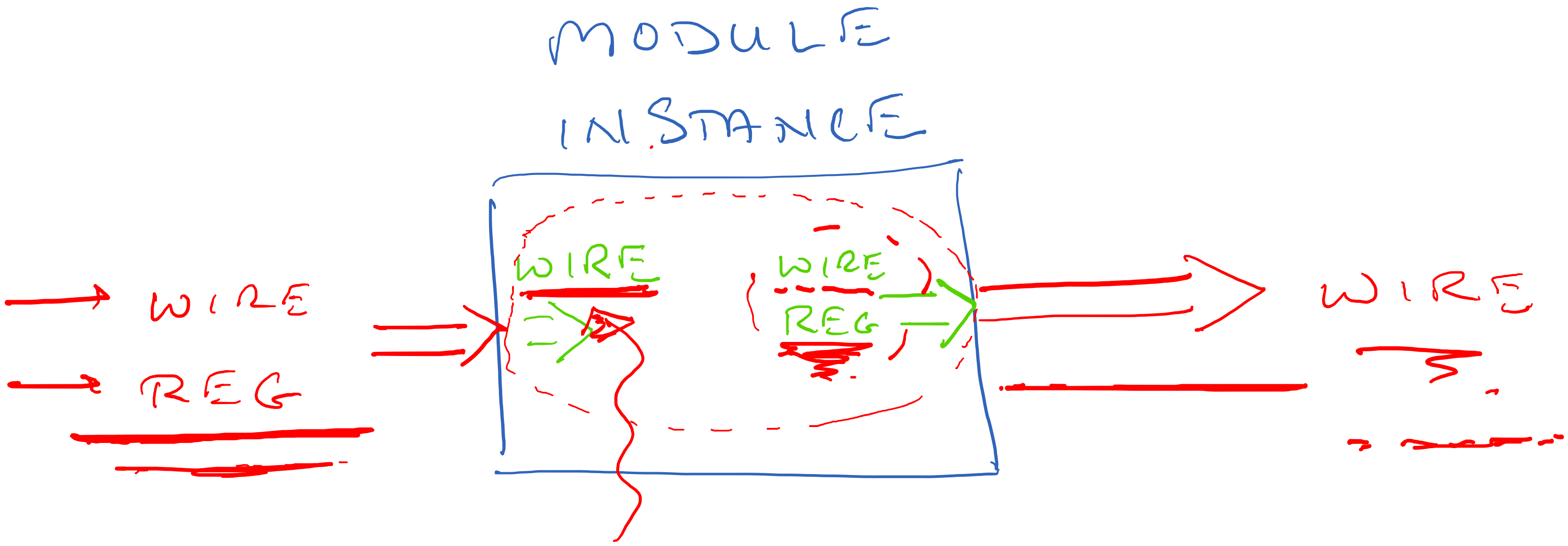
HARDWARE DESCRIPTION LANGUAGE

→ VHDL (EUROPE)

→ VERILOG (US)

FULL ADDER

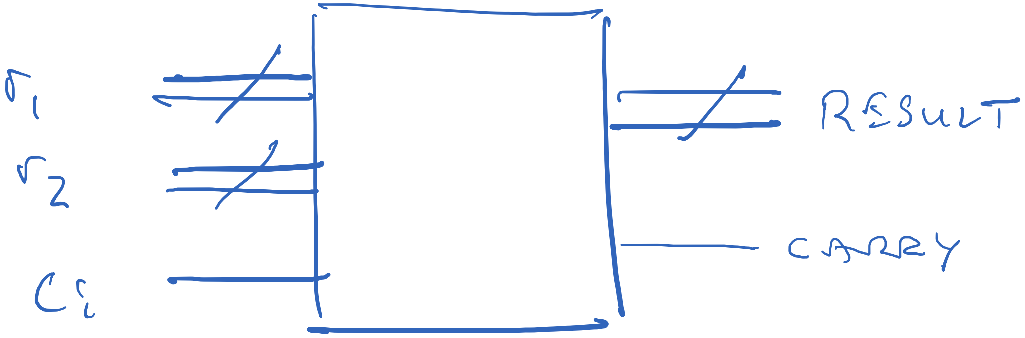


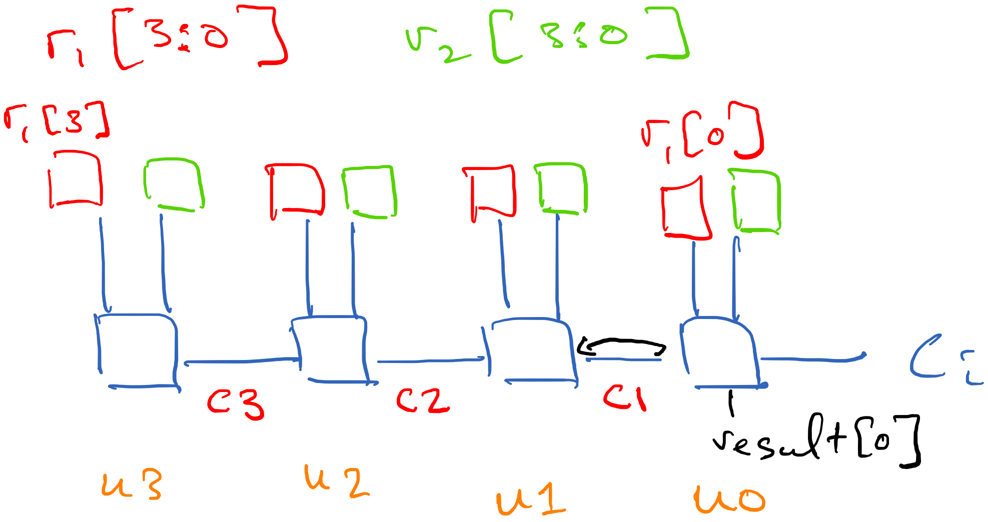

MODULE
INSTANCE
WIRE
REG
WIRE
WIRE
REG
WIRE

```
module adder(
  result,
  carry,
  r1,
  r2,
  ci
);

  input [3:0] r1;
  input [3:0] r2;
  input ci;

  output [3:0] result;
  output carry;

  wire [3:0] r1;
  wire [3:0] r2;
  wire ci;
  wire [3:0] result;
  wire carry;

  wire c1;
  wire c2;
  wire c3;

  addbit u0(r1[0],r2[0],ci,result[0],c1);
  addbit u1(r1[1],r2[1],c1,result[1],c2);
  addbit u2(r1[2],r2[2],c2,result[2],c3);
  addbit u3(r1[3],r2[3],c3,result[3],carry);

endmodule
```

INTERNAL VARIABLES

IMPLICIT INSTANTIATION





```
module tb();

 reg [3:0] r1, r2;
 reg ci;
 wire [3:0] result;
 wire carry;

 initial begin

     // Dump waves
   $dumpfile("dump.vcd");
   $dumpvars(1);

   $display("time\t r1 r2 ci result carry");

   $monitor("%g %b %b %b %b %b",
        $time, r1, r2, ci, result, carry);

   r1 = 0;
   r2 = 0;
   ci = 0;
   #10 r1 = 10;
   #10 r2 = 2;
   #10 ci = 1;
   #10
   $finish;
 end


 adder U(result,carry,r1,r2,ci);

endmodule
```

TESTBENCH

edaplayground.com

r1 = 4'b0000

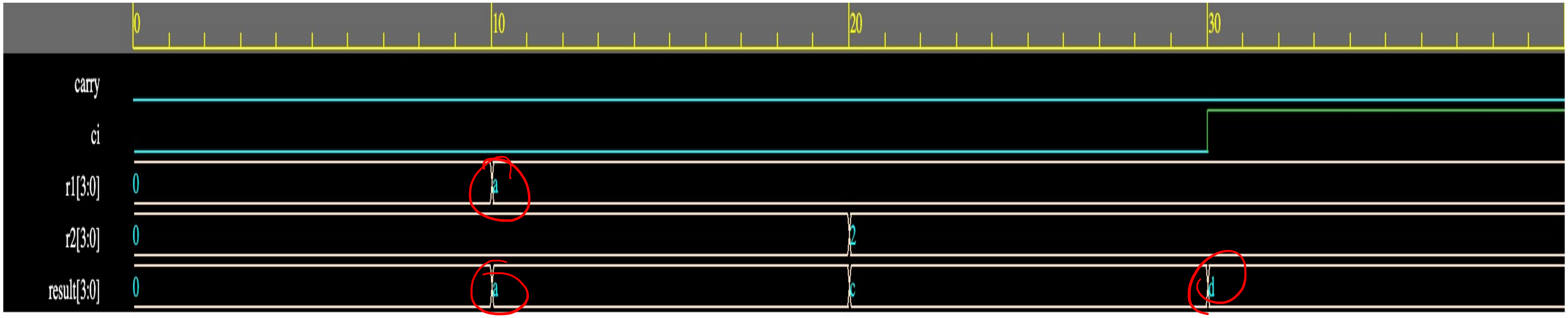
#10

0010

1010

r1 = 4'b1010

r2 = 4'b0010



```
time r1 r2 ci result carry
 0 0000 0000 0 0000 0
10 1010 0000 0 1010 0
20 1010 0010 0 1100 0
30 1010 0010 1 1101 0
```

$r_1 \rightarrow 10$ "a"

$r_2 = 0$

$r_1 + r_2 = a$

$r_1 = a$

$r_2 = 2$

$a + 2 = c$

$a + 2 + 1 = d$

```
time r1 r2 ci result carry

 0 0000 0000 0 0000 0
10 1010 0000 0 1010 0
20 1010 0110 0 0000 1
30 1010 0110 1 0001 1
```

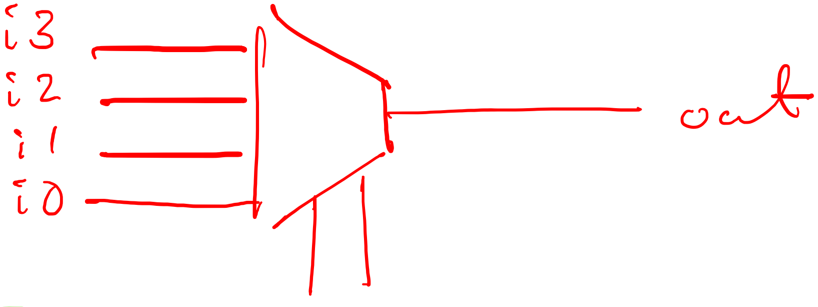
```
module mux_4_to_1(
  out,
  i0,
  i1,
  i2,
  i3,
  s0,
  s1
);

  output out;
  input i0, i1, i2, i3;
  input s0, s1;

  wire out;
  wire i0, i1, i2, i3;
  wire s0, s1;

  assign out = s1 ? (s0 ? i3 : i2) : (s0 ? i1 : i0);

endmodule
```

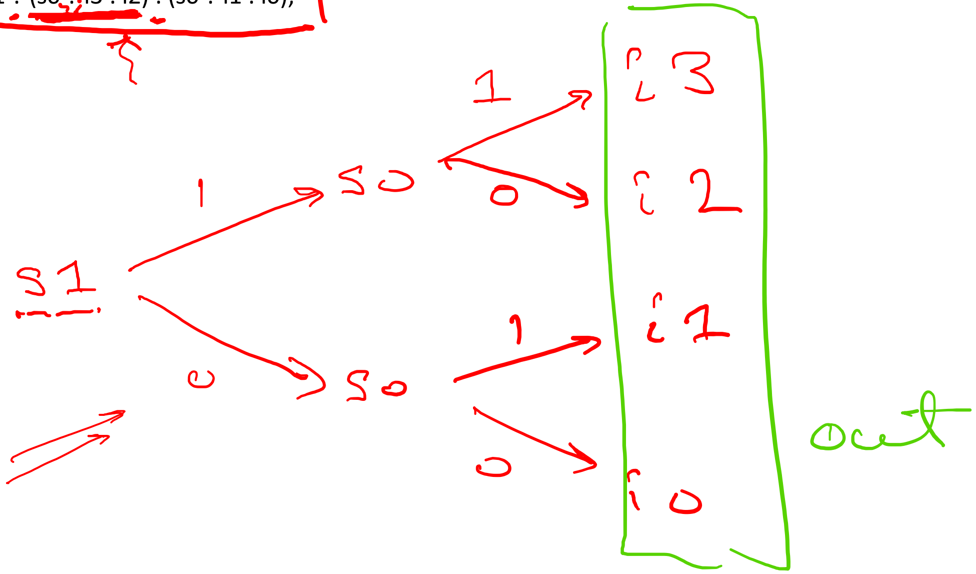
(out, i, s);

input [3:0] i;

input [1:0] s;

4 TO 1 MULTIPLEXER





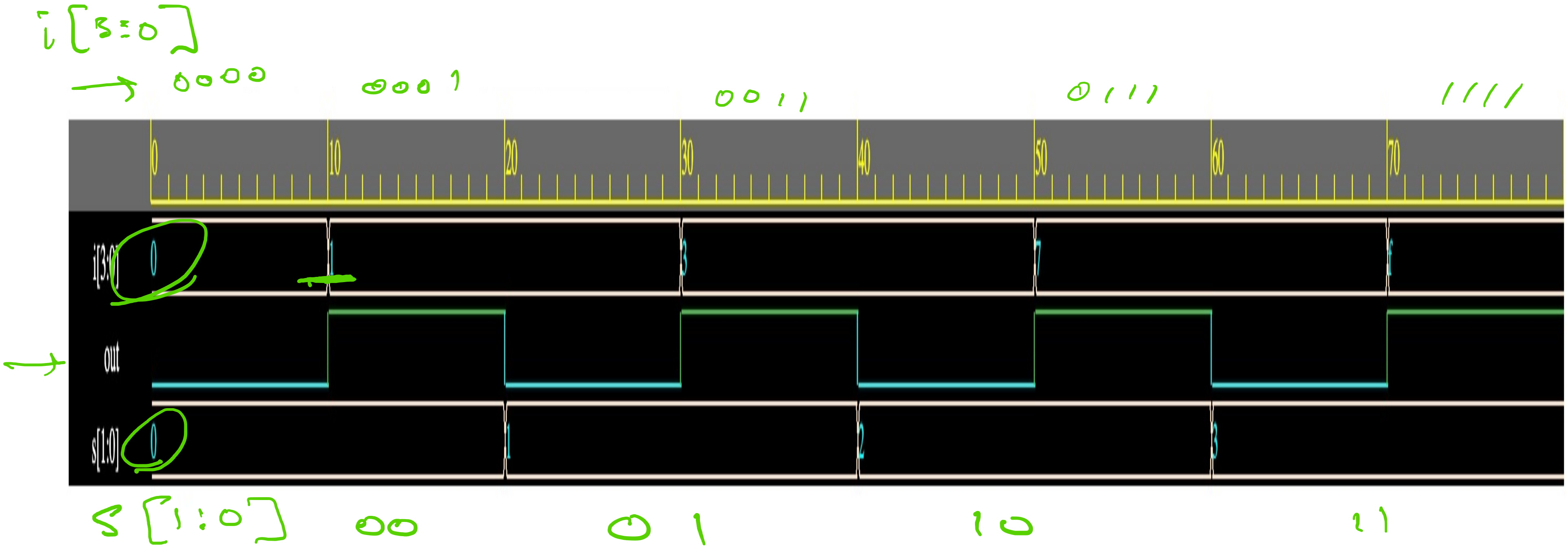
NESTED CONDITIONAL

s = 2'b11 out ← i3

2'b10 out ← i2

⋮

```
module tb_mux();

  reg [3:0] i;
  reg [1:0] s;
  wire out;

  initial begin

      // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(1);

    $display("time\t i s result");

    $monitor("%g %b %b %b",
          $time, i, s, out);

    i = 4’b0000;
    s = 2’b00;
    #10 i = 4’b0001;
    #10 s = 2’b01;
    #10 i = 4’b0011;
    #10 s = 2’b10;
    #10 i = 4’b0111;
    #10 s = 2’b11;
    #10 i = 4’b1111;
    #10
    $finish;
  end

  mux_4_to_1 U(out,i[0],i[1],i[2],i[3],s[0],s[1]);

endmodule
```


i[3:0]
0000
0001
0011
0111
1111
0
10
20
30
40
50
60
70
i[3:0]
0
1
3
7
f
out
s[1:0]
0
1
2
3
s[1:0]
00
01
10
11

```
module FJKRSE(J,K,Clk,R,S,CE,Qout);

  input J,K;  //inputs
  input Clk;  //Clock
  input R;    //synchronous reset (R)
  input S; //synchronous set (S)
  input CE; //clock enable (CE)
  output Qout;  //data output (Q)

  //Internal variable
  reg Qout;

  always@ (posedge(Clk))  //Everything is synchronous to positive edge of clock
  begin
    if(R == 1) //reset has highest priority.
      Qout = 0;
    else
      if(S == 1)  //set has next priority
        Qout = 1;
      else
        if(CE == 1) //J,K values are considered only when CE is ON.
          if(J == 0 && K == 0)
            Qout = Qout; //no change
          else if(J == 0 && K == 1)
            Qout = 0;  //reset
          else if(J == 1 && K == 0)
            Qout = 1;  //set
          else
            Qout = ~Qout;  //toggle
        else
          Qout = Qout; //no change
  end

endmodule
```

SYNCHRONOUS RESET

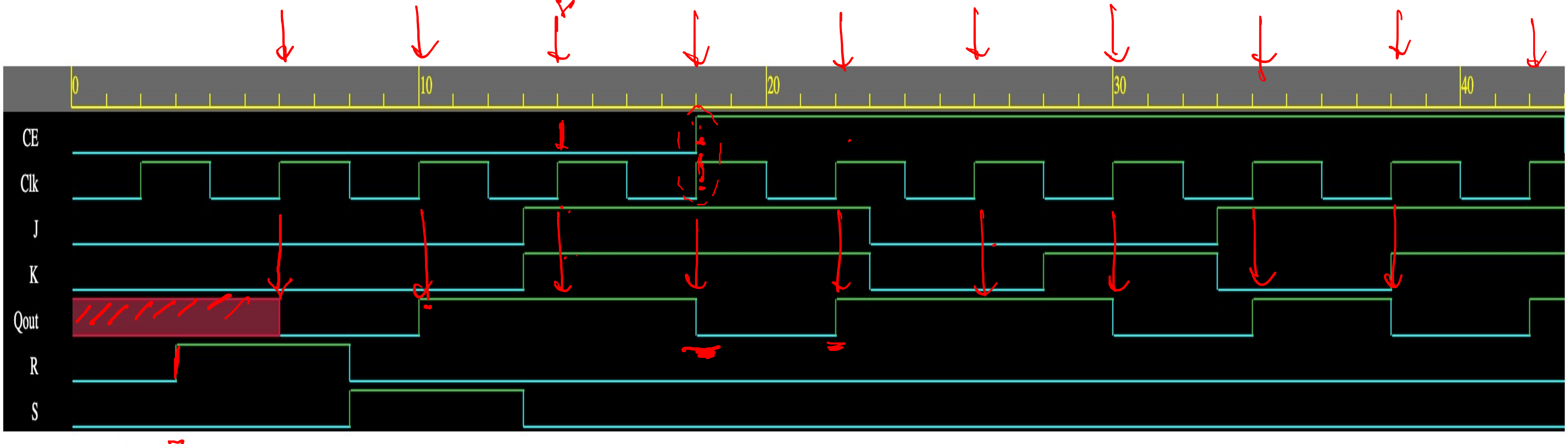
J	K	Qn+1
1	1	~Qn
1	0	1
0	1	0
0	0	Qn

```
module tb_jkff;

    // Inputs
    reg J;
    reg K;
    reg Clk;
    reg R;
    reg S;
    reg CE;

    // Outputs
    wire Qout;

    // Instantiate the Unit Under Test (UUT)
    FJKRSE uut (
        .J(J),
        .K(K),
        .Clk(Clk),
        .R(R),
        .S(S),
        .CE(CE),
        .Qout(Qout)
    );
```

```
initial begin

    $dumpfile("jkff.vcd");
                $dumpvars(1);

    // Initialize Inputs
    Clk = 0;
    J = 0;
    K = 0;
    R = 0;
    S = 0;
    CE = 0;
    #3;
    //Apply inputs
    R = 1;  #5;
    R = 0;
    S = 1;  #5;
    S = 0;
    J = 1;  K = 1;  #5;
    CE = 1; #5;
    J = 0;  K = 0;  #5;
    J = 0;  K = 1;  #5;
    J = 1;  K = 0;  #5;
    J = 1;  K = 1;  #5;
    CE = 0;
    $finish;
  end

  always begin
    #2 Clk = ~Clk;
  end

endmodule
```

CLOCK PERIOD 4 UNITS

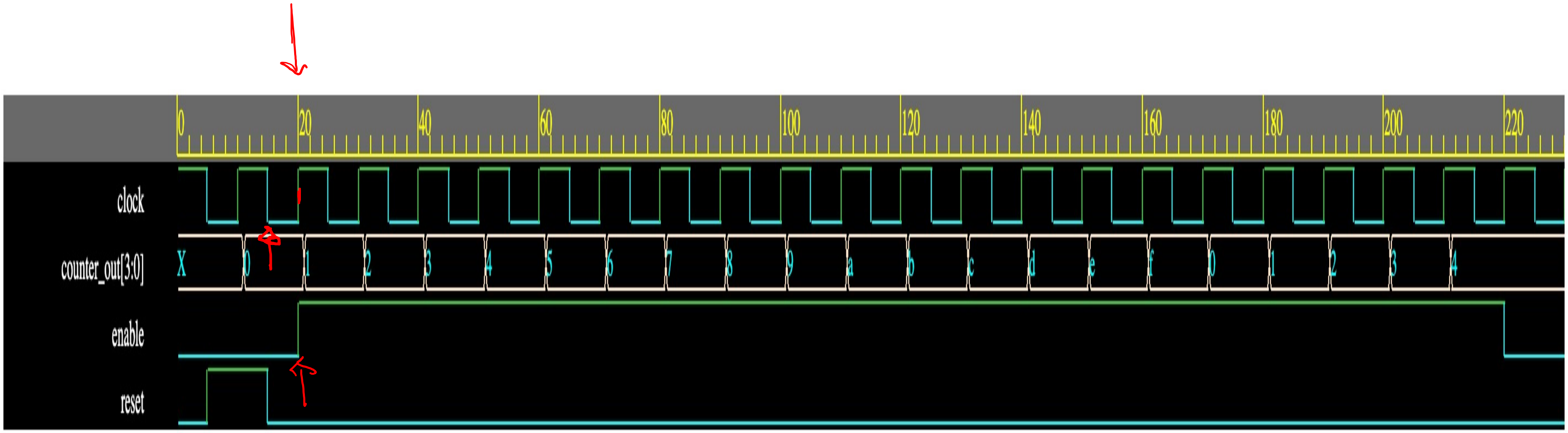
→ edaplayground.com .



RESET

```
module first_counter (
  clock,
  reset,
  enable,
  counter_out
);

  input clock;
  input reset;
  input enable;

  output [3:0] counter_out;

  wire clock;
  wire reset;
  wire enable;
  reg [3:0] counter_out;

  always @ (posedge clock)

    begin: COUNTER

      if (reset == 1'b1) begin
        counter_out <= #1 4'b0000;
      end

      else if (enable == 1'b1) begin
        counter_out <= #1 counter_out+1;
      end

    end

endmodule
```

```
module first_counter_tb();

  reg clock, reset, enable;
  wire [3:0] counter_out;

  initial begin

    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(1);

    $display("time\t clk reset enable counter");

    $monitor("%g %b %b %b %b",
             $time, clock, reset, enable, counter_out);

    clock = 1;
    reset = 0;
    enable = 0;
    #5 reset = 1;
    #10 reset = 0;
    #5 enable = 1;
    #200 enable = 0;
    #10 $finish;

  end

  always begin
    #5 clock = ~clock;
  end

  first_counter U_counter(
    clock,
    reset,
    enable,
    counter_out
  );

endmodule
```


0
20
40
60
80
100
120
140
160
180
200
220
clock
counter_out[3:0]
enable
reset
X
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
0
1
2
3
4

FINITE STATE MACHINE

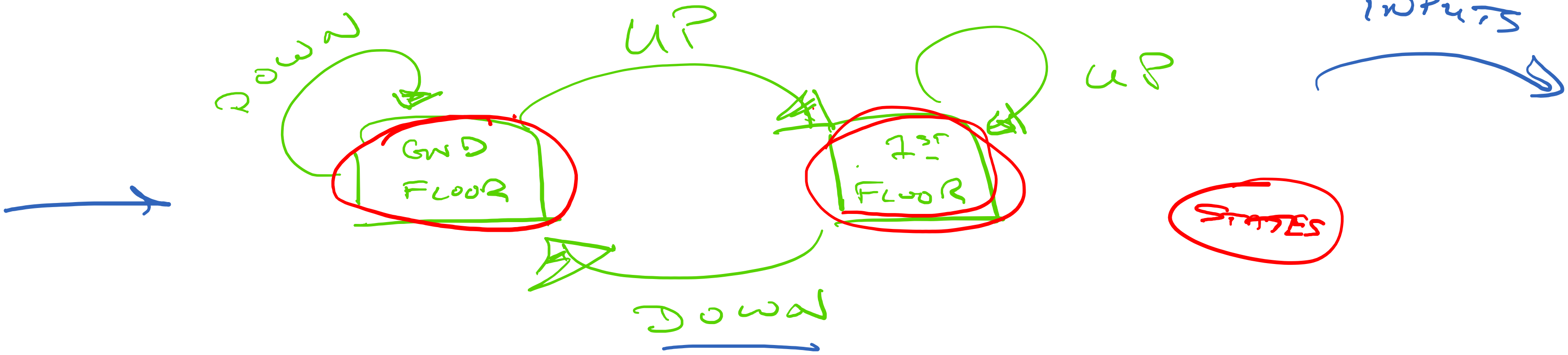
KARNAUGH MAP

FINITE SET OF STATES

INPUTS INDUCE TRANSITIONS BASED ON CURRENT STATE & INPUT VALUES

STORE UP TO 2^N STATES IN OUTPUTS OF N FLIP FLOPS

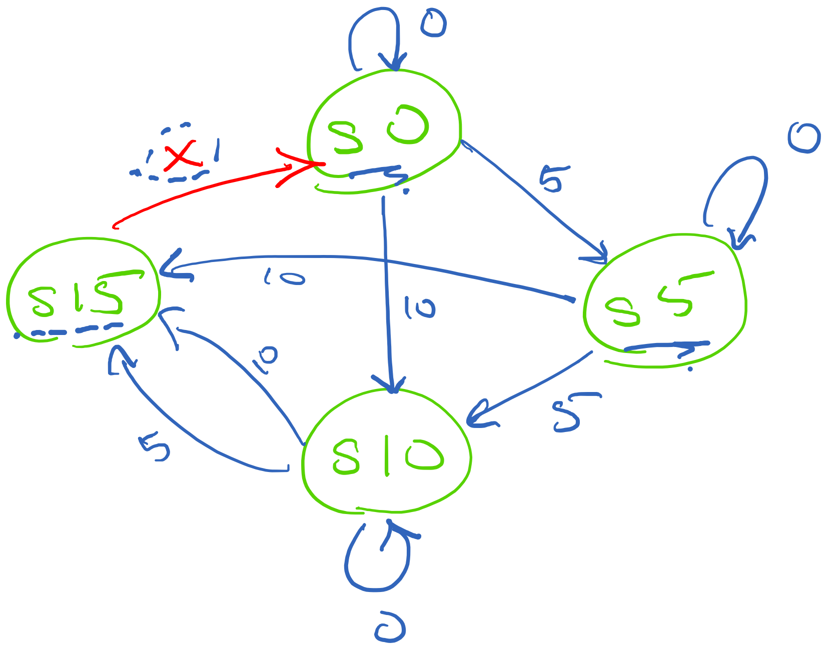
STATE DIAGRAM



VENDING MACHINE

TYPE II P.D.
PLZ CHIP

ACCEPT 5¢, 10¢
@ 15¢, DISPENSE NEWSPAPER



OUTPUT

0 → "BLUE"

1 → "RED"
GIVE N. PAPER.

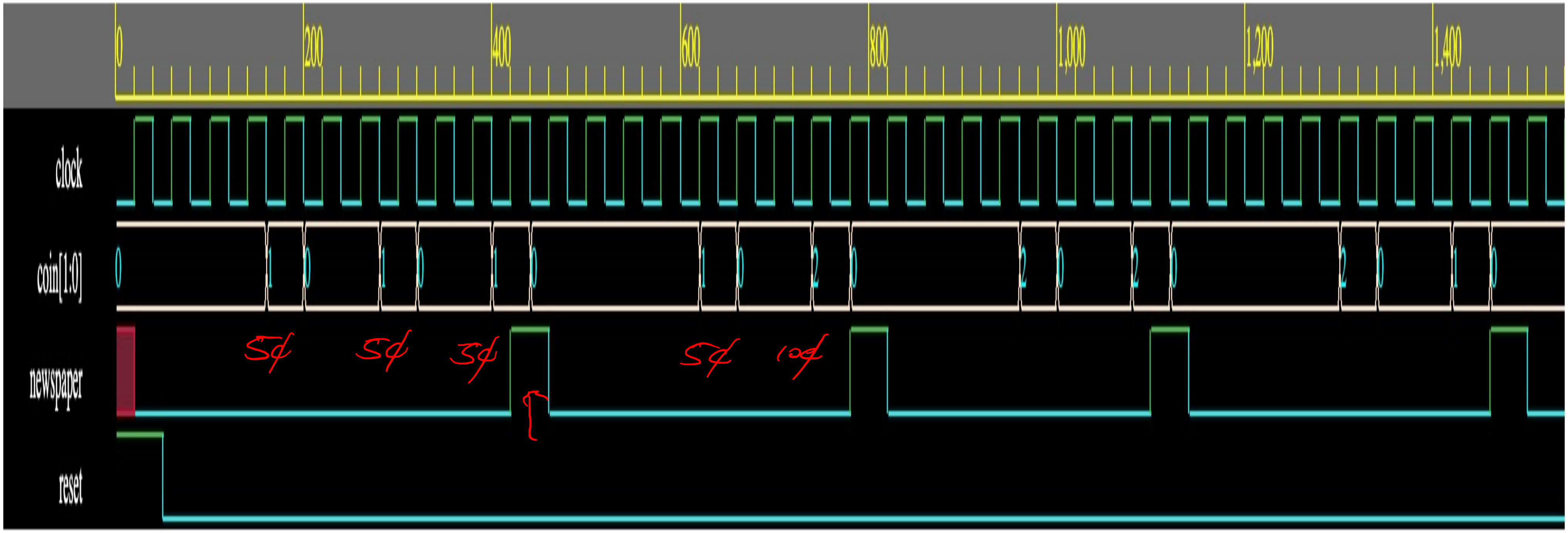
```verilog
module vend(coin, clock, reset, newspaper);

 // port declarations
 input [1:0] coin;
 input clock;
 input reset;
 output newspaper;

 wire newspaper;

 // internal FSM declarartions
 wire [1:0] NEXT_STATE;
 reg [1:0] PRES_STATE;

 // state encodings
 parameter s0 = 2'b00;
 parameter s5 = 2'b01;
 parameter s10 = 2'b10;
 parameter s15 = 2'b11;

 function [2:0] fsm;
  input [1:0] fsm_coin;
  input [1:0] fsm_PRES_STATE;

  reg fsm_newspaper;
  reg [1:0] fsm_NEXT_STATE;

  begin
   case (fsm_PRES_STATE)
    s0:
     begin
      if (fsm_coin == 2'b10)
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s10;
       end
      else if (fsm_coin == 2'b01)
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s5;
       end
      else
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s0;
       end
     end
```

```verilog
s5:
     begin
      if (fsm_coin == 2'b10)
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s15;
       end
      else if (fsm_coin == 2'b01)
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s10;
       end
      else
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s5;
       end
     end

    s10:
     begin
      if (fsm_coin == 2'b10)
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s15;
       end
      else if (fsm_coin == 2'b01)
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s15;
       end
      else
       begin
        fsm_newspaper = 1'b0;
        fsm_NEXT_STATE = s10;
       end
     end

    s15:
     begin
      fsm_newspaper = 1'b1;
      fsm_NEXT_STATE = s0;
     end

   endcase
```

```verilog
fsm = {fsm_newspaper, fsm_NEXT_STATE};

  end

 endfunction

 assign {newspaper, NEXT_STATE} = fsm(coin, PRES_STATE);

 always @(posedge clock)
  begin
   if (reset == 1'b1)
    PRES_STATE <= s0;
   else
    PRES_STATE <= NEXT_STATE;
  end

endmodule
```

{OUTPUT, NEXT STATE}

```
module tb_vend;

 reg clock;
 reg [1:0] coin;
 reg reset;
 wire newspaper;

 vend U(coin, clock, reset, newspaper);

 initial
  begin
   $dumpfile("dump.vcd");
   $dumpvars(1);
   $display("\t \tTime Reset Newspaper\n");
   $monitor("%d %d %d", $time, reset, newspaper);
  end

 initial
  begin
   clock = 0;
   coin = 0;
   reset = 1;
   #50 reset = 0;

   @(negedge clock);

   //put in 3 nickels to get newspaper
   #80 coin = 1; #40 coin = 0;
   #80 coin = 1; #40 coin = 0;
   #80 coin = 1; #40 coin = 0;

   //put in 1 nickel then 1 dime
   #180 coin = 1; #40 coin = 0;
   #80 coin = 2; #40 coin = 0;

//put in 2 dimes
   #180 coin = 2; #40 coin = 0;
   #80 coin = 2; #40 coin = 0;

//put in 1 dime then 1 nickel
   #180 coin = 2; #40 coin = 0;
   #80 coin = 1; #40 coin = 0;

   #80 $finish;
  end

 always
  begin
   #20 clock = ~clock;
  end
endmodule
```


0
200
400
600
800
1,000
1,200
1,400
clock
coin[1:0]
0
1
0
1
0
1
0
1
0
2
0
2
0
2
0
2
0
1
0
newspaper
reset
5¢
5¢
3¢
5¢
10¢

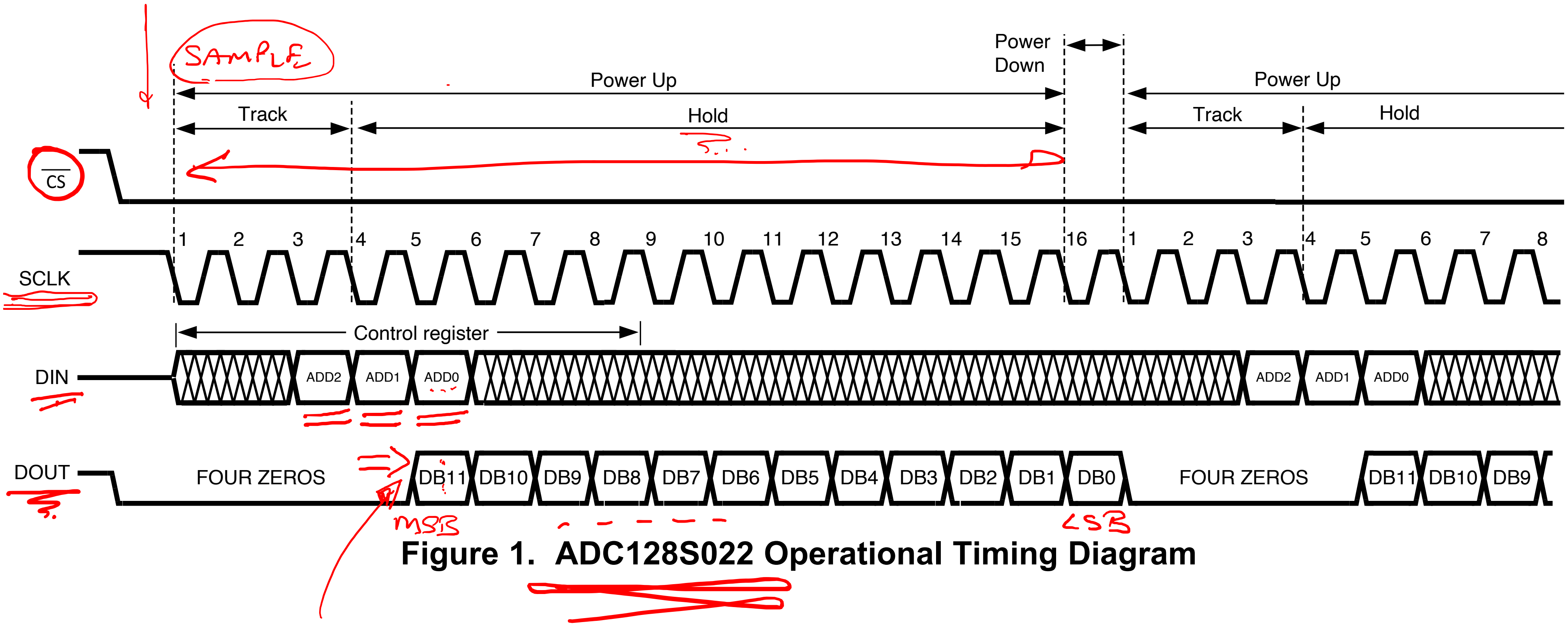


Figure 1. ADC128S022 Operational Timing Diagram

```
module ADC_CTRL(iRST,iCLK,iCLK_n,iGO,iCH,oLED,oDIN,oCS_n,oSCLK,iDOUT);

 input iRST;
 input iCLK;
 input iCLK_n;
 input iGO;
 input [2:0] iCH;
 input iDOUT;

 output [7:0] oLED;
 output oDIN;
 output oCS_n;
 output oSCLK;

 reg data;
 reg go_en;
 wire [2:0] ch_sel;
 reg sclk;
 reg [3:0] cont;
 reg [3:0] m_cont;
 reg [11:0] adc_data;
 reg [7:0] led;

 assign oCS_n = ~go_en;
 assign oSCLK = (go_en) ? iCLK:1;
 assign oDIN = data;
 assign ch_sel = iCH;
 assign oLED = adc_data[11:4];
 //assign oLED = led;

 always @(posedge iGO or negedge iRST)
  begin
   if(!iRST)
    go_en <= 0;
   else
    begin
     if(iGO)
      go_en <= 1;
    end
  end
```

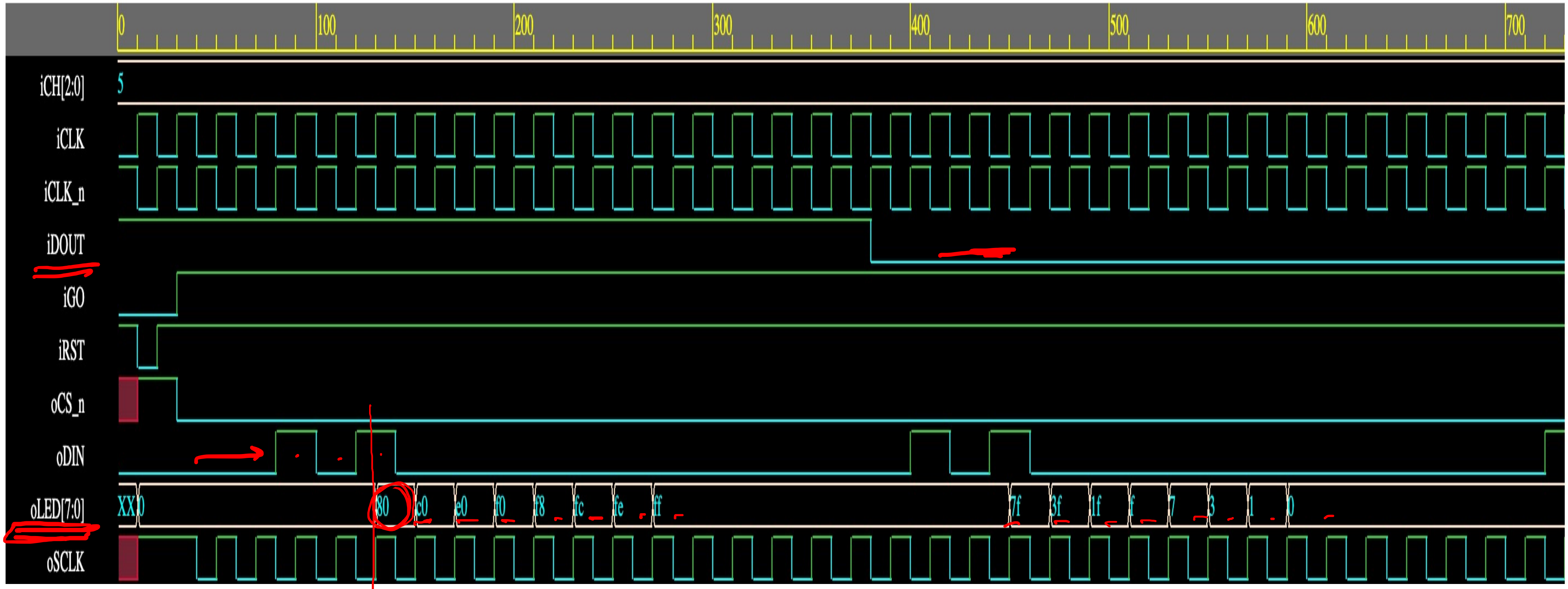
```
always @(posedge iCLK or negedge go_en)
  begin
   if(!go_en)
    cont <= 0;
   else
    begin
     if(iCLK)
      cont <= cont+1;
    end
  end

 always @(posedge iCLK_n)
  begin
   if(iCLK_n)
    m_cont <= cont;
  end

 always @(posedge iCLK_n or negedge go_en)
  begin
   if(!go_en)
    begin
     data <= 0;
    end
   else
    begin
     if(iCLK_n)
      begin
       if(cont == 2)
        data <= ch_sel[2];
       else if(cont ==3)
        data <= ch_sel[1];
       else if(cont == 4)
        data <= ch_sel[0];
       else
        data <= 0;
      end
    end
  end
```

iCLK_n IS USED TO SEND "data" [ADC ADDRESS] WHICH OF 8 ADC UNITS TO TALK TO

iCLK → USED TO CAPTURE Dout

```
always @(posedge iCLK or negedge go_en)
  begin
   if(!go_en)
    begin
     adc_data <= 0;
    end
   else
    begin
     if(iCLK)
      begin
       if (m_cont == 4)
        adc_data[11] <= iDOUT;
       else if (m_cont == 5)
        adc_data[10] <= iDOUT;
       else if (m_cont == 6)
        adc_data[9] <= iDOUT;
       else if (m_cont == 7)
        adc_data[8] <= iDOUT;
       else if (m_cont == 8)
        adc_data[7] <= iDOUT;
       else if (m_cont == 9)
        adc_data[6] <= iDOUT;
       else if (m_cont == 10)
        adc_data[5] <= iDOUT;
       else if (m_cont == 11)
        adc_data[4] <= iDOUT;
       else if (m_cont == 12)
        adc_data[3] <= iDOUT;
       else if (m_cont == 13)
        adc_data[2] <= iDOUT;
       else if (m_cont == 14)
        adc_data[1] <= iDOUT;
       else if (m_cont == 15)
        adc_data[0] <= iDOUT;
        //led <= adc_data[11:4];
        //else if (m_cont == 1);
        //led <= adc_data[11:4];
      end
    end
  end
endmodule
```

```
module ADC_tb();

 reg iRST;
 reg iCLK;
 reg iCLK_n;
 reg iGO;
 reg [2:0] iCH;
 reg iDOUT;

 wire [7:0] oLED;
 wire oDIN;
 wire oCS_n;
 wire oSCLK;

 ADC_CTRL U(iRST,iCLK,iCLK_n,iGO,iCH,oLED,oDIN,oCS_n,oSCLK,iDOUT);

 initial
  begin
   $dumpfile("dump.vcd");
   $dumpvars(1);
  end

 initial
  begin
   iCLK = 0;
   iCLK_n = 1;
   iRST = 1;
   iGO = 0;
   iCH = 3'b101;
   iDOUT = 1;
   #10 iRST = 0;
   #10 iRST = 1;
   #10 iGO = 1;
   #350
   iDOUT = 0;
   #350 $finish;
  end

 always
  begin
   #10 iCLK = ~iCLK;
   iCLK_n = ~iCLK_n;
  end
endmodule
```

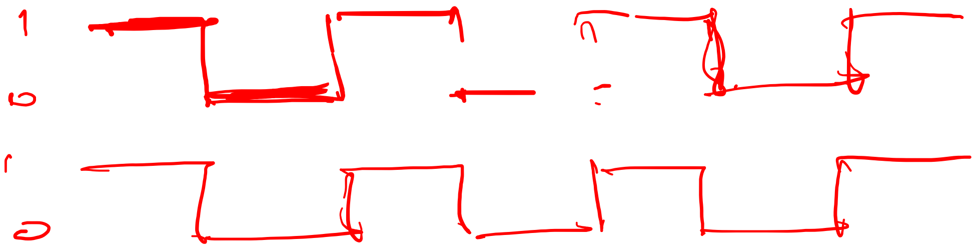


0
100
200
300
400
500
600
700
iCH[2:0]
5
iCLK
iCLK_n
iDOUT
iGO
iRST
oCS_n
oDIN
oLED[7:0]
XX
0
80
c0
e0
f0
f8
fc
fe
ff
7f
3f
1f
f
7
3
1
0
oSCLK

8'h xx

"VERILOG MULTIPLY ACCUMULATE"

```
module unsig_altmult_accum(

        input [7:0] dataa,
        input [7:0] datab,
        input clk, aclr, clken, sload,
        output reg [15:0] adder_out
);



        // Declare registers and wires
        reg  [15:0] dataa_reg, datab_reg;
        reg  sload_reg;
        reg [15:0] old_result;
        wire [15:0] multa;


        // Store the results of the operations on the current data
        assign multa = dataa_reg * datab_reg;


        // Store the value of the accumulation (or clear it)
        always @ (adder_out, sload_reg)
        begin
                if (sload_reg)
                        old_result <= 0;
                else
                        old_result <= adder_out;
        end

// Clear or update data, as appropriate
        always @ (posedge clk or posedge aclr)
        begin
                if (aclr)
                begin
                        dataa_reg <= 0;
                        datab_reg <= 0;
                        sload_reg <= 0;
                        adder_out <= 0;
                end
                else if (clken)
                begin
                        dataa_reg <= dataa;
                        datab_reg <= datab;
                        sload_reg <= sload;
                        adder_out <= old_result + multa;
                end
        end
endmodule
```



<MULT> → 1/2 IN PHASE

→ 1/4 INCOHERENT

→ 0 OUT OF PHASE