Sequential Logic (con't)
(Class 3.1 – 2/2/2021)

CSE 4357/CSE 5357 – Advanced Digital Logic Design

Spring 2021

Instructor – Bill Carroll, PhD, PE, Professor

UNIVERSITY OF TEXAS ARLINGTON
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Today's Topics

- Course calendar
- Counters
- Clock dividers
- Basic timers
- Registers
- Register files
- Assignment #2 posted – due 2/6/21, 11:59 PM

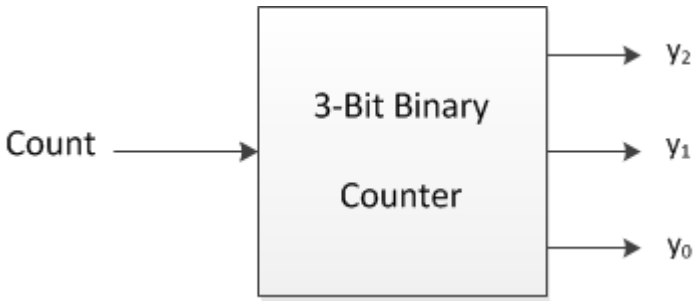
Course Calendar

Week	Suggested Reading	Class date	Tuesday (1) Lecture Topics	Class date	Thursday (2) Lecture Topics	Assignment	Due Date (11:59 PM)
1	Ciletti, Chs 1, 2 and 4	1/19	Course overview. Programmable logic. Review of combinational logic and Verilog.	1/21	Review of combinational logic and Verilog (continued).	Check out DE1-SoC development kit.	1/22
2	Ciletti, Chs 3 and 5	1/26	Using Quartus Prime 18.1. High-speed adders.	1/28	Review of sequential logic and Verilog -- recognizers.	Assignment 1 -- Ripple-Carry Adder/Subtractor	1/30
3	Ciletti, Chs 3 and 5	2/2	Review of sequential logic and Verilog -- registers and counters.	2/4	Review of sequential logic and Verilog -- controllers. Logic hazards.	Assignment 2 -- Knight Rider.	2/6
4	Ciletti, Chs 4 and 6	2/9	Test benches.	2/11	ModelSim.	Assignment 3 -- Test Bench for RCAddSub.	2/13
5	Ciletti, Chapter 8	2/16	FPGA organization and architecture.	2/18	Quartus Prime processes.	None	2/20
6		2/23	Examination 1 (SH 103)	2/25	Quartus Prime analysis tools.	Assignment 4 -- Two-function High-speed ALU.	2/27
7	Ciletti, Chapter 10	3/2	Basic multipliers.	3/4	High-speed multipliers.	TBA	
8	Ciletti, Chapter 10	3/9	High-speed multipliers (continued)	3/11	High-speed multipliers (continued)	TBA	
SB		3/16	Spring Break	3/18	Spring Break		
9	Lee, Chapter 8	3/23	Pipelined functional units.	3/25	Pipelined functional units (continued).	TBA	
10	Ciletti, Chapter 10	3/30	Dividers.	4/1	Dividers (continued).	TBA	
11	Ciletti, Chapter 11	4/6	Examination 2 (SEIR 194)	4/8	Post-synthesis design tasks.	TBA	
12	Ciletti, Chapter 11	4/13	Timing analysis.	4/15	Timing analysis (continued).	TBA	
13	Ciletti, Chapter 7	4/20	ALU design.	4/22	ALU design (continued).		
14		4/27	Project work	4/29	Project work		
15		5/4	Project work				

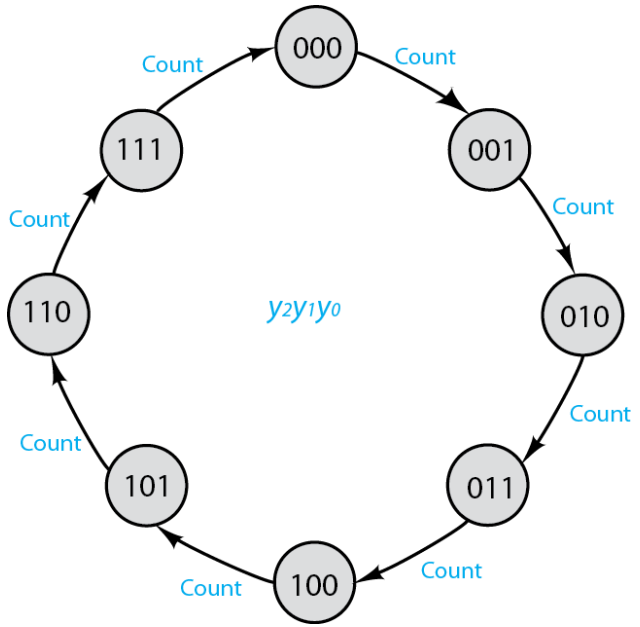
Counters

Binary Counter

- Counters – sequential circuits that tally, or count, in a predetermined code sequence, the number of input pulses received, e.g., 3-bit binary counter.



States = 2^n



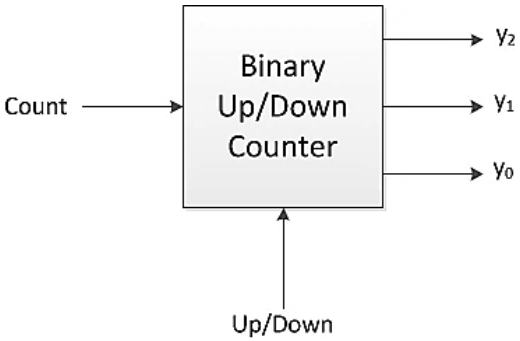
Binary Counter Verilog Model

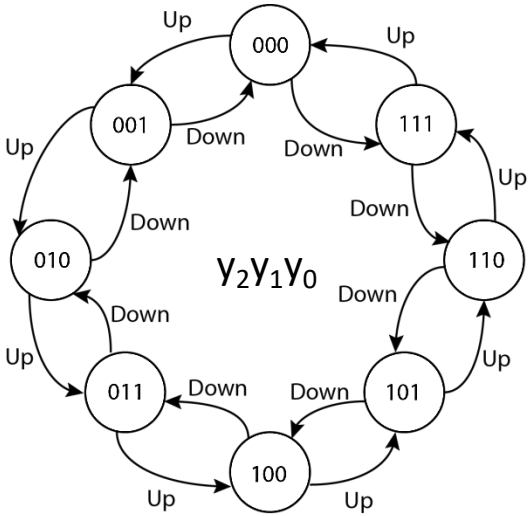
```
// Behavioral model of a three-bit binary counter
module binarycounter (COUNT, CLEAR, y);
input COUNT, CLEAR;
output reg [2:0] y;                    // y is defined as a 3-bit output register
        always @ (posedge COUNT, negedge CLEAR)
            if (CLEAR)  y <= 3'b0;  // y is loaded with binary 000
                    else
                        begin
                            if (y == 3'b111)
                                y <= 3'b0;        // Once y = 111 it is cleared
                            else
                                y <= y + 1'b1;   // y is incremented
                        end
endmodule
```

n-Bit Binary Counters

```
// n-bit binary counter
module NbitBinCount (COUNT,CLEAR,y);
    parameter N=4;
    input COUNT, CLEAR;
    output reg [N-1:0] y;              // y is defined as a register
    always @ (posedge COUNT, negedge CLEAR)
        if (CLEAR==1'b0) y <= 0;       // y is loaded with all 0's
            else
                begin
                    if (y == 2**N-1'b1) y <= 0;       // Once y = 2**N-1 it is cleared
                        else y <= y + 1'b1;           // y is incremented
                end
endmodule
```

Binary Up/Down Counter



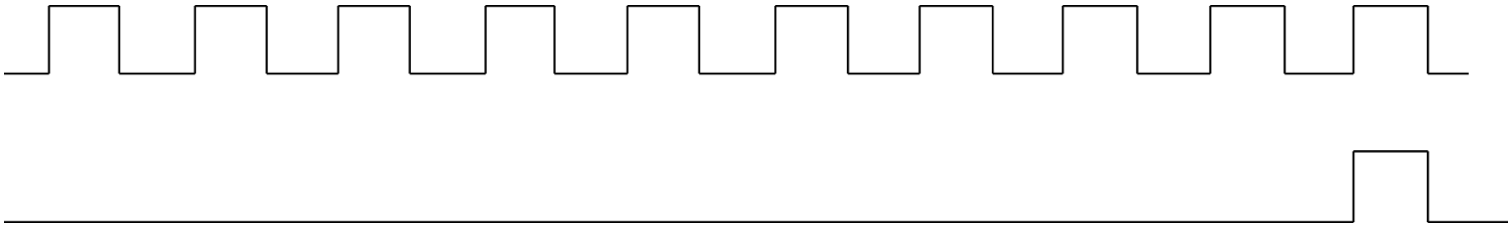


Binary Up/Down Counter

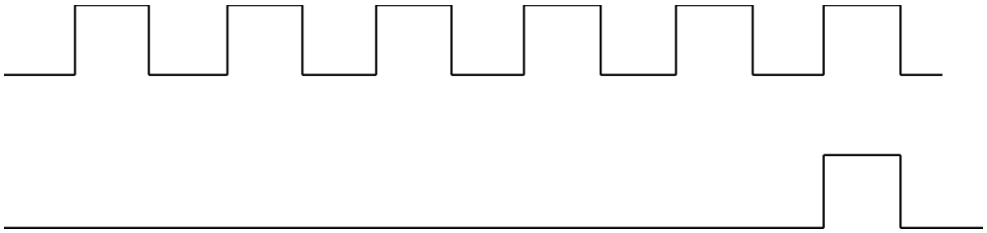
```
// Behavioral n-bit binary up/down counter.  Positive-edge triggered.  Active-low clear.
module BinaryUpDown (CLK, CLEAR, UP, COUNT);
    parameter N = 3'd3;
    input CLK, CLEAR, UP;
    output reg [N-1:0] COUNT;                    //COUNT is defined as an N-bit register
        always @ (posedge CLK, negedge CLEAR)    //trigger on clock or clear
            if (CLEAR==0) COUNT <= 0;            //COUNT is loaded with binary all 0's
            else
                if (UP==0)                       //count up or down?
                    COUNT <= COUNT + 1'b1;       //count up
                    else
                        COUNT <= COUNT - 1'b1;   //count down
endmodule
```

Divide-By Counters

Divide-by-10



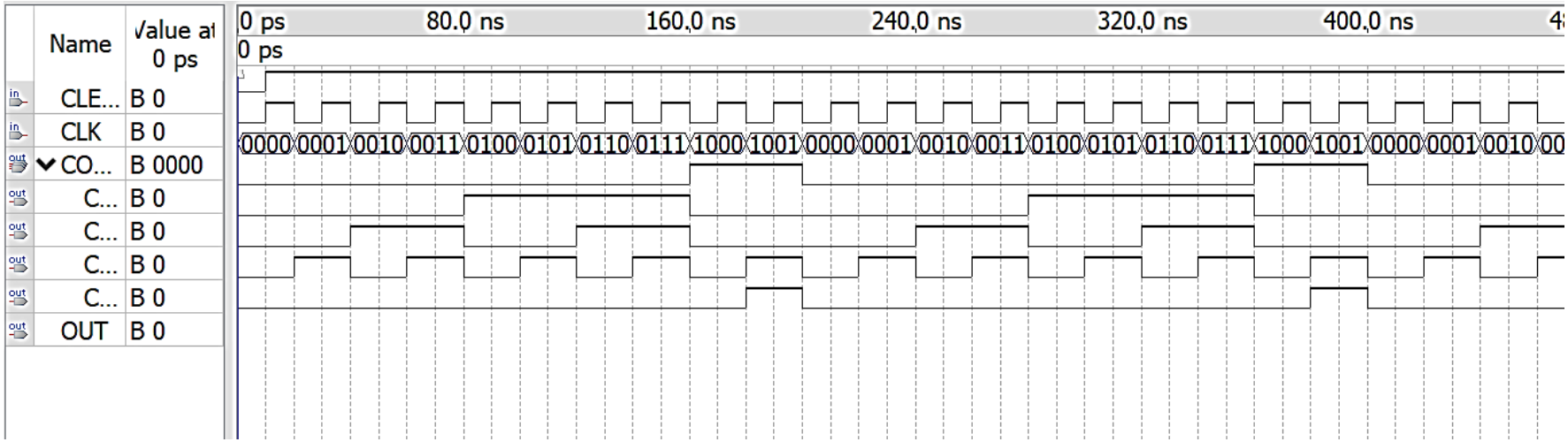
Divide-by-6



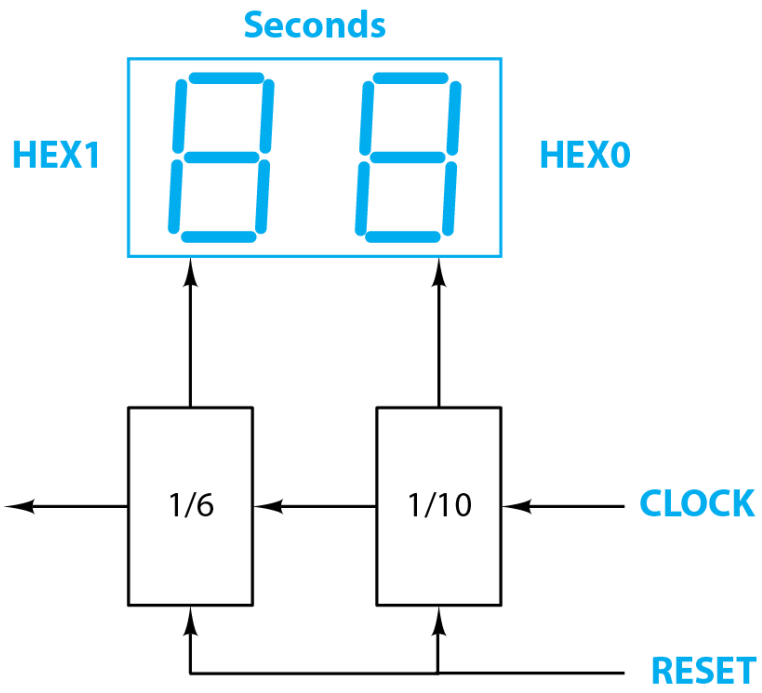
Divide-By-*N* Counter

```
// Divide by 10 counter
      module divideX10 (CLK,CLEAR,OUT,COUNT);
            parameter N=4'd10;
            input CLK, CLEAR;
            output reg [3:0] COUNT;                          // COUNT is defined as a 4-bit register
            output reg OUT;
            always @ (negedge CLK, negedge CLEAR)
                  if (CLEAR==1'b0)     COUNT <= 4'b0;        // COUNT is loaded with all 0's
                        else
                        begin
                              if (COUNT == N-2'd2)  begin OUT <= 1'b1; COUNT <= N-1'd1; end // Once COUNT = N-2 OUT = 1
                                    else
                                     if (COUNT == N-1'd1) begin OUT <=1'b0; COUNT <= 4'b0; end //Once COUNT = N-1 OUT=0
                                        else begin OUT <= 1'b0; COUNT <= COUNT + 1'b1; end      // COUNT is incremented
                        end
       endmodule
```

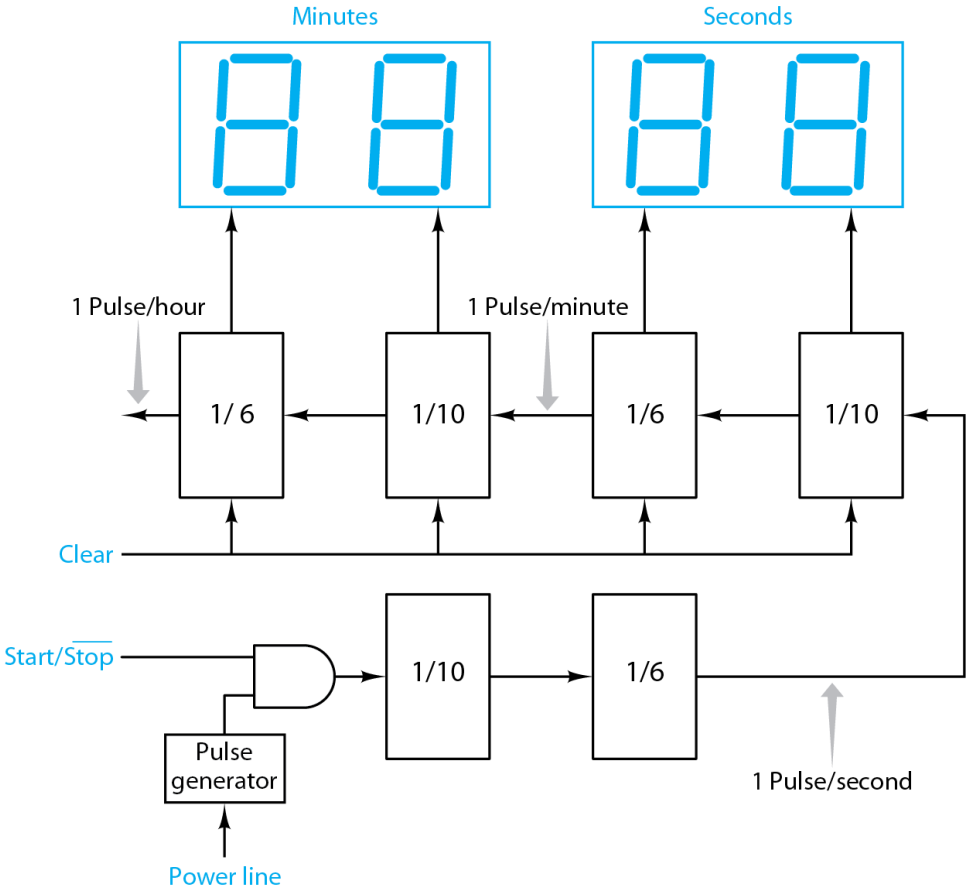
Divide-By-*N* Counter, *N*=10



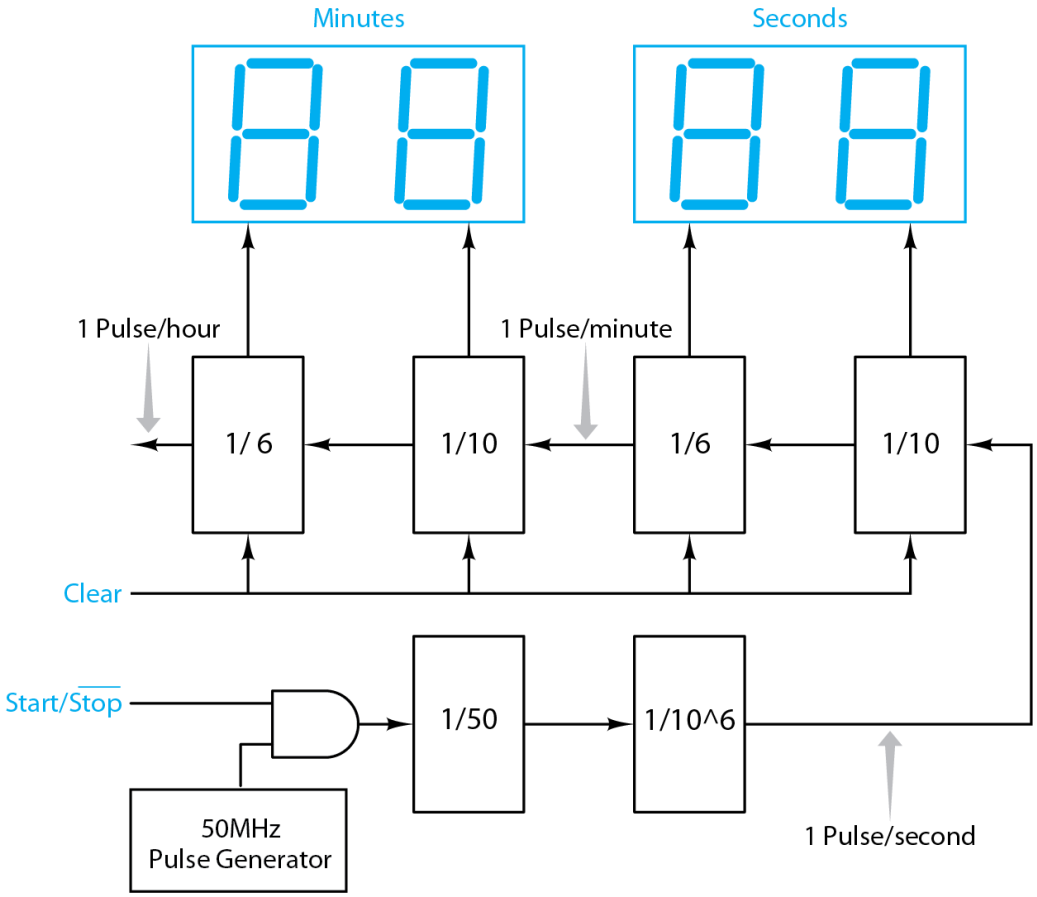
0 to 59 Second Timer



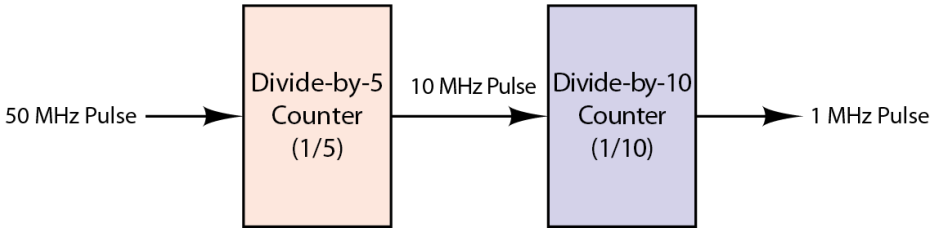
Digital Timer Design

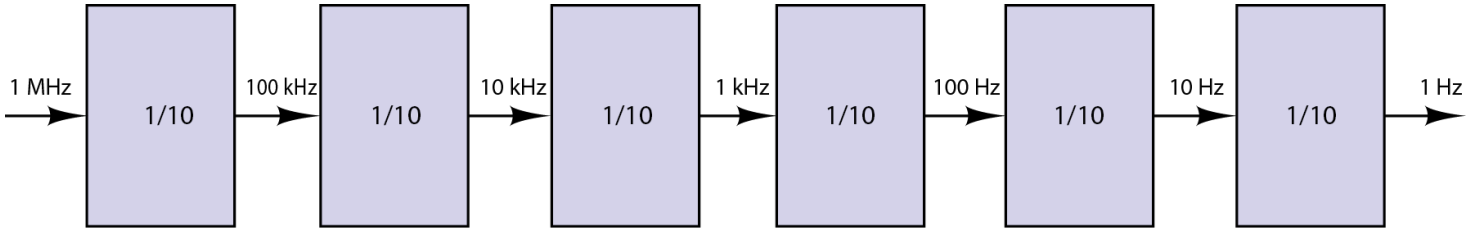


Digital Timer Design (revisited)



Divide-by-50x10^6 Counter





Frequency References

Power line – 60 Hz or 50 Hz

Crystal oscillators – various frequencies available

Phased-locked-loops – widely used (DE1-SoC has six)

CLOCK_50 – PIN_AF14
CLOCK2_50 – PIN_AA16
CLOCK3_50 – PIN_Y26
CLOCK4_50 – PIN_K14

All 50-MHz

Divide-By-Six Counter

```
// Divide by Six counter
      module divideX6 (CLK,CLEAR,OUT,COUNT);
            parameter N=3'd6;
            input CLK, CLEAR;
            output reg [2:0] COUNT;                        // COUNT is defined as a 3-bit register
            output reg OUT;
            always @ (negedge CLK, negedge CLEAR)
                  if (CLEAR==1'b0)     COUNT <= 3'b0;      // COUNT is loaded with all 0's
                        else
                        begin
                            if (COUNT == N-2'd2)  begin OUT <= 1'b1; COUNT <= N-1'd1; end // Once COUNT = N-2 OUT = 1
                                  else
                                   if (COUNT == N-1'd1) begin OUT <=1'b0; COUNT <= 3'b0; end //Once COUNT = N-1 OUT=0
                                      else begin OUT <= 1'b0; COUNT <= COUNT + 1'b1; end      // COUNT is incremented
                        end
       endmodule
```

Divide-By-10 Counter

```
// Divide by 10 counter
      module divideX10 (CLK,CLEAR,OUT,COUNT);
            parameter N=4'd10;
            input CLK, CLEAR;
            output reg [3:0] COUNT;                        // COUNT is defined as a 4-bit register
            output reg OUT;
            always @ (negedge CLK, negedge CLEAR)
                  if (CLEAR==1'b0)     COUNT <= 4'b0;      // COUNT is loaded with all 0's
                        else
                        begin
                              if (COUNT == N-2'd2)  begin OUT <= 1'b1; COUNT <= N-1'd1; end // Once COUNT = N-2 OUT = 1
                                    else
                                     if (COUNT == N-1'd1) begin OUT <=1'b0; COUNT <= 4'b0; end //Once COUNT = N-1 OUT=0
                                        else begin OUT <= 1'b0; COUNT <= COUNT + 1'b1; end      // COUNT is incremented
                        end
       endmodule
```

Divide-By-Five Counter

```
// Divide by Five counter
    module divideX5 (CLK,CLEAR,OUT,COUNT);
        parameter N=3'd5;
        input CLK, CLEAR;
        output reg [2:0] COUNT;                    // COUNT is defined as a 3-bit register
        output reg OUT;
        always @ (negedge CLK, negedge CLEAR)
            if (CLEAR==1'b0)     COUNT <= 3'b0;    // COUNT is loaded with all 0's
                else
                begin
                    if (COUNT == N-2'd2)  begin OUT <= 1'b1; COUNT <= N-1'd1; end // Once COUNT = N-2 OUT = 1
                        else
                            if (COUNT == N-1'd1) begin OUT <=1'b0; COUNT <= 3'b0; end //Once COUNT = N-1 OUT=0
                                else begin OUT <= 1'b0; COUNT <= COUNT + 1'b1; end      // COUNT is incremented
                end
     endmodule
```

Divide-By-1000 Counter

```
// Divide by 1000 counter
      module divideX1000 (CLK,CLEAR,OUT,COUNT);
            parameter N=10'd1000;
            input CLK, CLEAR;
            output reg [9:0] COUNT;                          // COUNT is defined as a 10-bit register
            output reg OUT;
            always @ (negedge CLK, negedge CLEAR)
                  if (CLEAR==1'b0)     COUNT <= 10'b0;       // COUNT is loaded with all 0's
                        else
                        begin
                            if (COUNT == N-2'd2)  begin OUT <= 1'b1; COUNT <= N-1'd1; end // Once COUNT = N-2 OUT = 1
                                  else
                                   if (COUNT == N-1'd1) begin OUT <=1'b0; COUNT <= 10'b0; end //Once COUNT = N-1 OUT=0
                                      else begin OUT <= 1'b0; COUNT <= COUNT + 1'b1; end       // COUNT is incremented
                        end
       endmodule
```

Digital Timer Verilog Code

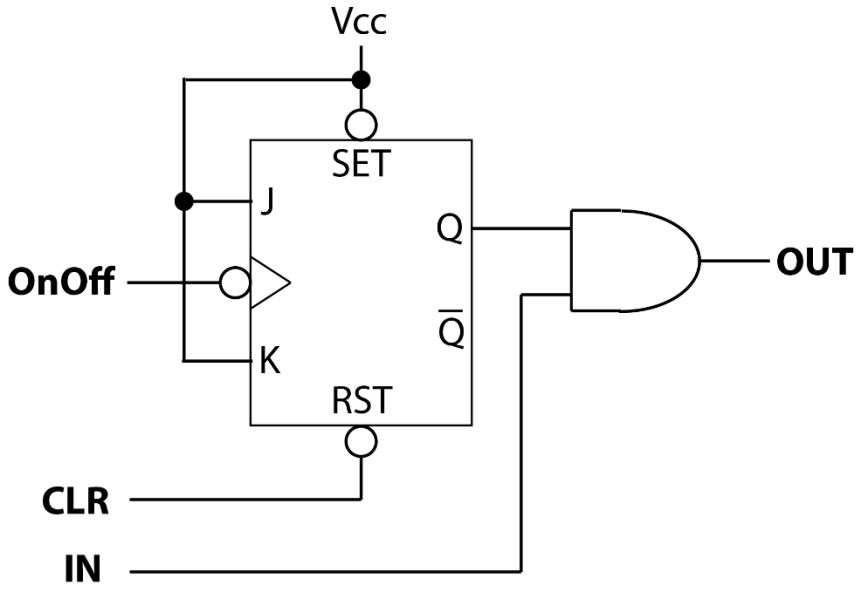
```
//Zero to 59 second timer
module timerdemo
        (
         input clock, reset,
         output [0:6] Aout, Bout
          );
          wire [3:0] A, B;
          wire five, ten, k1, k0, sec0, sec1;
          wire [3:0] count5, count10;
          wire [9:0] count1000L, count1000M;
//instantiations
 divideX5 div5
(
     .CLK(clock) ,                    // input  50MHz clock
     .CLEAR(reset) ,                  // input  reset
     .OUT(five) ,                     // output  every 5 input pulses
     .COUNT(count5)                   // output [3:0] count bits
);
divideX10 div10
(
     .CLK(five) ,                     // input  10MHz clock
      .CLEAR(reset) ,                 // input  reset
      .OUT(ten) ,                     // output  every 10 input pulses
       .COUNT(count10)                // output [3:0] count bits
);
divideX1000 div1000M
(
        .CLK(ten) ,   // input  1MHz clock
         .CLEAR(reset) ,                 // input  reset
         .OUT(k1) ,   // output  every 1000 input pulses
         .COUNT(count1000M)          // output [9:0] count bits
);
```

Digital Timer Verilog Code

```
divideX1000 div1000L
(
        .CLK(k1) ,    // input  1KHz clock
         .CLEAR(reset) ,                  // input  reset
         .OUT(k0) ,  // output  1000 input pulses
         .COUNT(count1000L)           // output [9:0] count bits
);
divideX10 seconds_low
(
         .CLK(k0) ,   // input  1Hz clock
         .CLEAR(reset) ,                  // input  reset
         .OUT(sec0) ,                     // output  every 10 input pulses
         .COUNT(B)                        // output [3:0] seconds low count
);
binary2seven seconds_low_display
(
         .BIN(B) ,     // input [3:0] seconds low count
.         SEV(Bout)                       // output [0:6] low seconds display
);
divideX6 seconds_high
(
.         CLK(sec0) ,                     // input  from low seconds
         .CLEAR(reset) ,                  // input  CLEAR_sig
         .OUT(sec1) ,                     // output  to low minutes
         .COUNT(A)                        // output [3:0] high seconds count
);
binary2seven seconds_high_display
(
         .BIN(A) ,     // input [3:0] high seconds count
         .SEV(Aout)                       // output [0:6] high seconds display
);
endmodule
```

OnOff-Toggle Latch

If CLR = 0, OUT = 0. Otherwise, OUT = IN when toggled *On* and OUT = 0 when toggled *Off.*

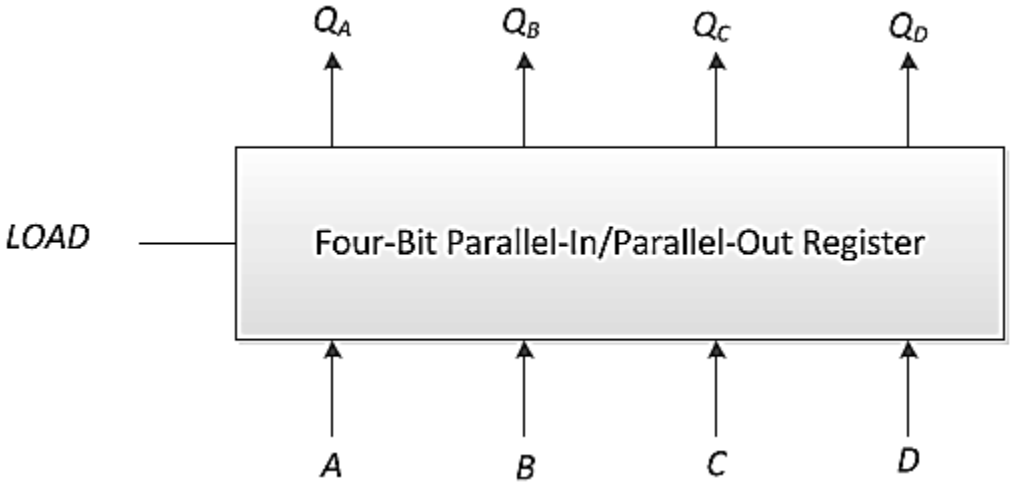


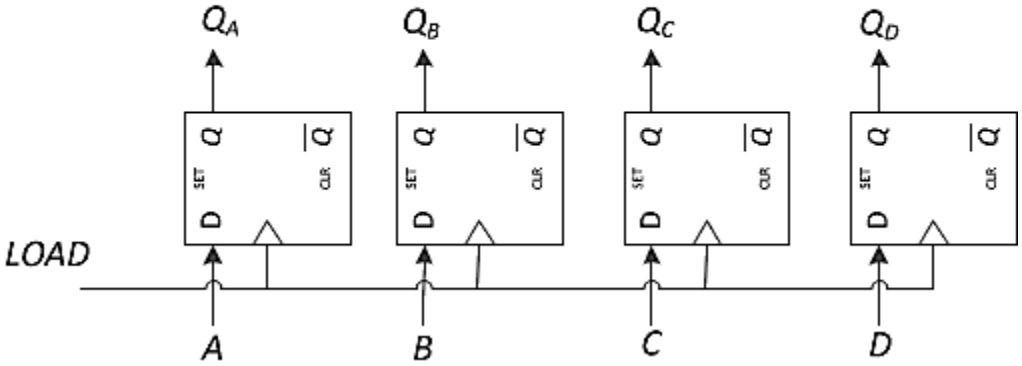
Verilog OnOff-Toggle Latch

```
//OnOff Toggle Latch.  Assumes an normally-on push button switch for OnOff.
module ToggleLatch (OnOff, CLR, IN, OUT);
input OnOff, IN, CLR;
output OUT;
reg state, nextstate;
parameter ON=1'b1, OFF=1'b0;
      always @ (negedge OnOff, negedge CLR)  //Active-low CLR.
            if (CLR==1'b0) state <= OFF;            //CLR turns the switch off.
                  else state <= nextstate;
      always @ (state)
                  case(state)
                        OFF: nextstate = ON;      //Pushing OnOff turns the switch on.
                        ON: nextstate = OFF;      //Pushing OnOff turns the switch off.
                  endcase
      assign OUT = state*IN;  //Out = In when switch in on.  Otherwise, Out = 0.
endmodule
```

Registers

Basic Register

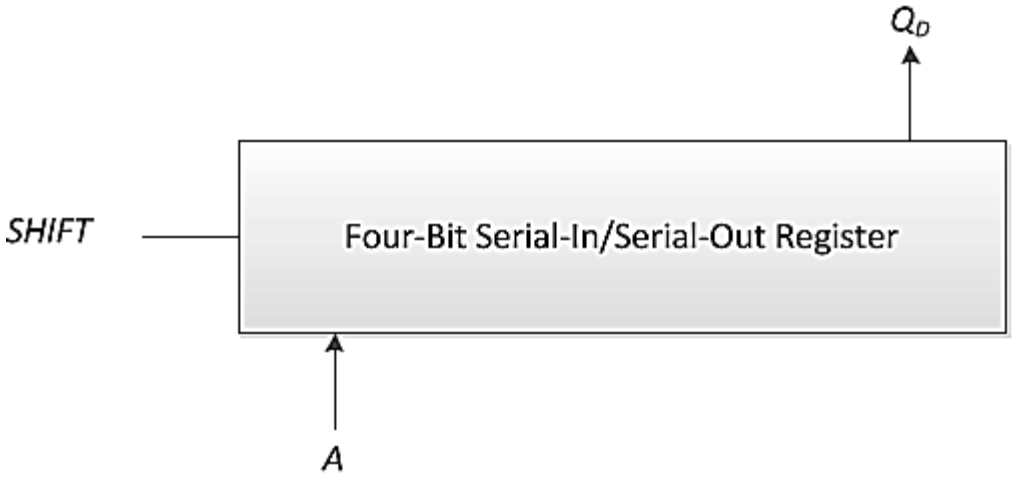


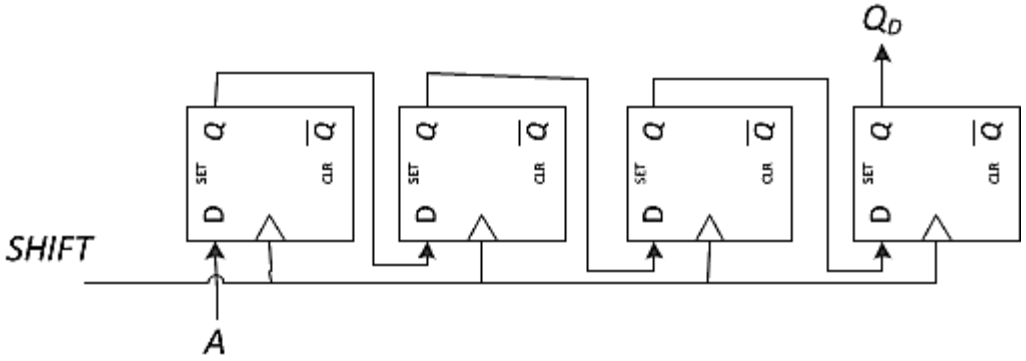


Basic Register – Verilog Model

```
//Verilog Model of an N-bit register with active-low asynchronous clear
module NbitRegister (D,Q,CLK,CLR);
    parameter N = 4;                //declare default value for N
    input [N-1:0] D;                //declare N-bit data input
    input CLK, CLR;                 //declare clock and clear inputs
    output reg [N-1:0] Q;           //declare N-bit data output
        always @ (posedge CLK, negedge CLR) //detect change of clock or clear
            begin
                if (CLR==1'b0) Q <= 0;              //register loaded with all 0's
                    else if (CLK==1'b1) Q <= D;     //data input values loaded in register
            end
endmodule
```

Serial-In/Serial-Out Register

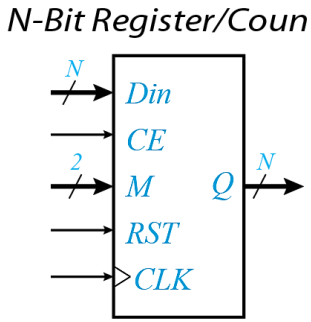




Serial-In/Serial-Out Register – Verilog

```
//Verilog Model of an N-bit Serial-In Serial-Out Shift Register with active-low clear
module NbitSerialInSerialOut (A,Q,SHIFT,CLR);
    parameter N = 4;                //declare default value for N
    input A;                        //declare data input
    input SHIFT, CLR;               //declare shift and clear inputs
    output reg [N-1:0] Q;           //declare N-bit data output
    integer i;                                          //declare index variable
        always @ (posedge SHIFT, negedge CLR)           //detect change of clock or clear
        begin
            if (CLR==1'b0) Q <= 0;                      //register loaded with all 0's
                else if (SHIFT==1'b1)
                begin                                   //shift and load input A
                    for (i = 1; i <= N-1; i = i + 1)
                    begin Q[i] <= Q[i-1]; Q[0] <= A; end
                end
        end
endmodule
```

Multifunction Registers



(a)

RST	CE	M1	M0	Q+	Function
1	x	x	x	$Q_i+ <= 0$	Reset
0	0	x	x	$Q_i+ <= Q$	Hold
0	1	0	0	$Q_i+ <= Q$	Hold
0	1	0	1	$Q_i+ <= Q_{i+1}$	Shift
0	1	1	0	$Q+ <= Q+1$	Count
0	1	1	1	$Q+ <= Din$	Load

(b)

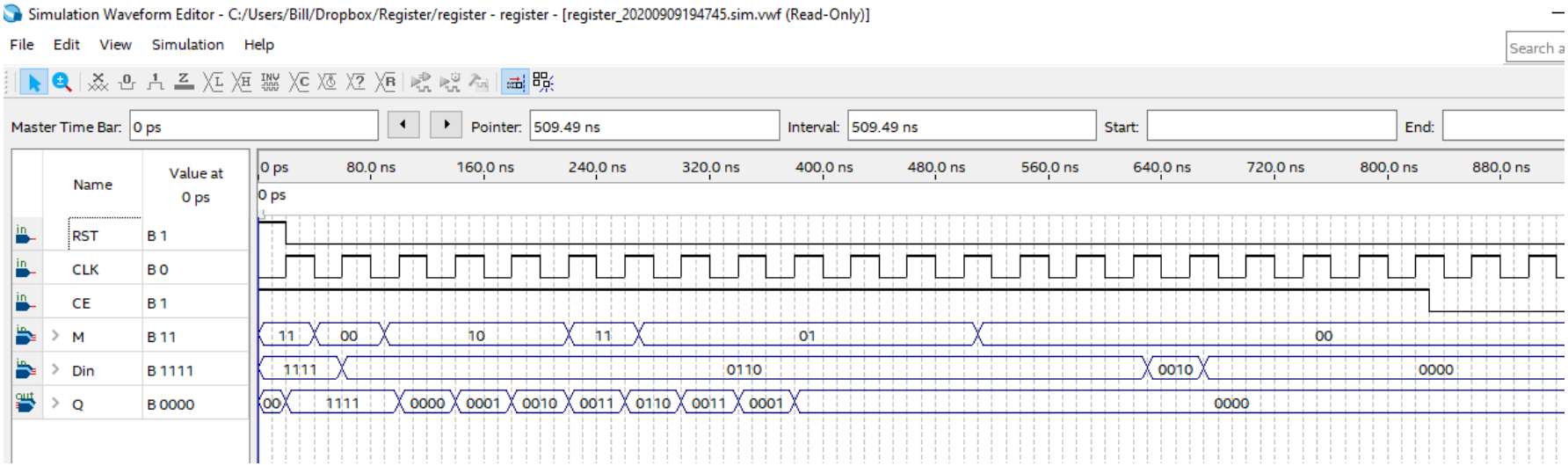
Input/Output Signals *Function Table*

Din – *N*-bit data input.
Q – *N*-bit data output.
CE – Active-high clock enable.
M – 2-bit function code (*M1*, *M0*).
RST – Active-high asynchronous reset.
CLK – Active-high clock. Triggers operations on the rising edge.

Multifunction Registers

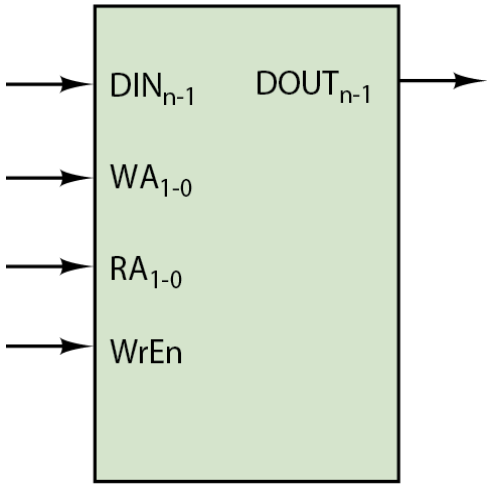
```
//Verilog model of an N-bit multifunction register
module register #(parameter N = 4)
    (
    input CLK, RST, CE,             //declare clock, reset, and clock enable inputs
    input [1:0] M,                  //declare function inputs
    input [N-1:0] Din,              //declare data inputs
    output reg [N-1:0] Q            //declare data outputs
    );
        always @ (posedge CLK or posedge RST) begin
                if (RST) Q <= 0;                //reset detected
                    else if (CE) begin          //check for clock enable
                        if (M == 2'b01) Q <= Q >> 1'b1;             //right shift
                            else if (M == 2'b10) Q <= Q + 1'b1;     //increment
                            else if (M == 2'b11) Q <= Din;          //load
                    end
        end
endmodule
```

Multifunction Register Simulation



Register File

n-bit x *N* Register File



Input/Output Diagram (*N* = 4)

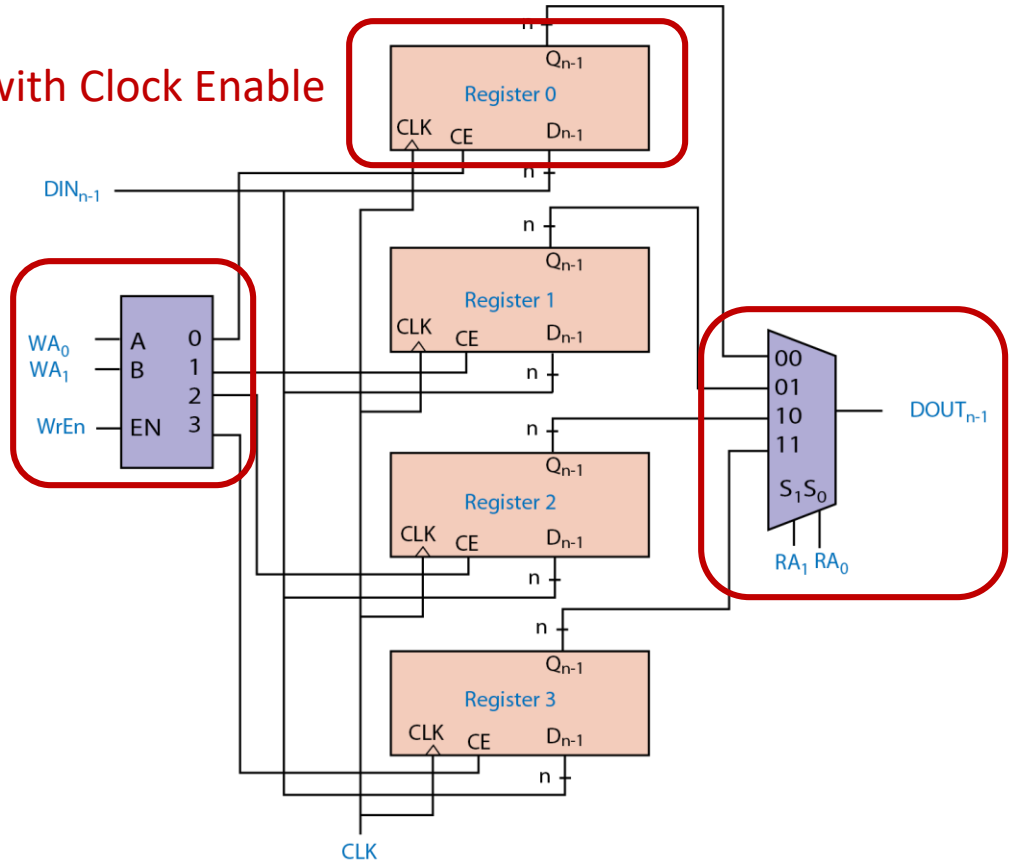
DIN – data input (*n*-bits)
DOUT – data output (*n*-bits)
WA – write address (2-bits)
RA – read address (2-bits)
WrEn – write enable (1-bit)

Register File Organization

Register with Clock Enable

Two-to-four decoder with Enable

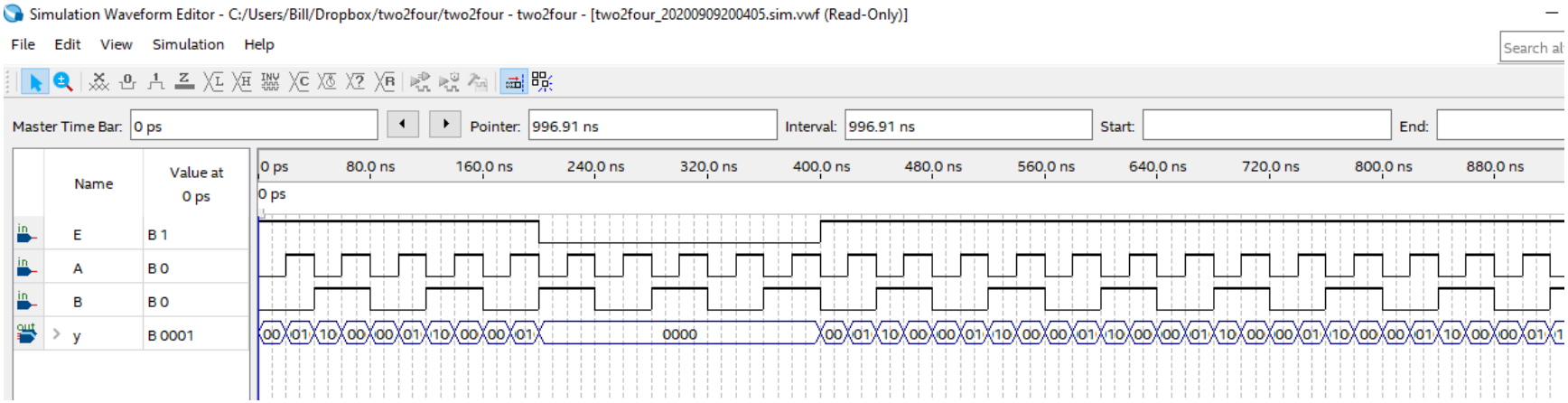
Four-to-one *n*-bit multiplexer



Two-to-Four Decoder with Enable

```
//Two to Four Active High Decoder with Enable
module Two2FourActiveHigh (
      input A,B,E,
      output reg [3:0] y);
      always @ (A, B, E)
            if (~E) y = 4'b0;  //output all 0's if not enabled
                  else
                  case ({B,A})  //decode input if enabled
                        2'b00: y = 4'b0001;
                        2'b01: y = 4'b0010;
                        2'b10: y = 4'b0100;
                        2'b11: y = 4'b1000;
                  endcase
endmodule
```

Two-to-Four Decoder Simulation



n-bit Register with Clock Enable

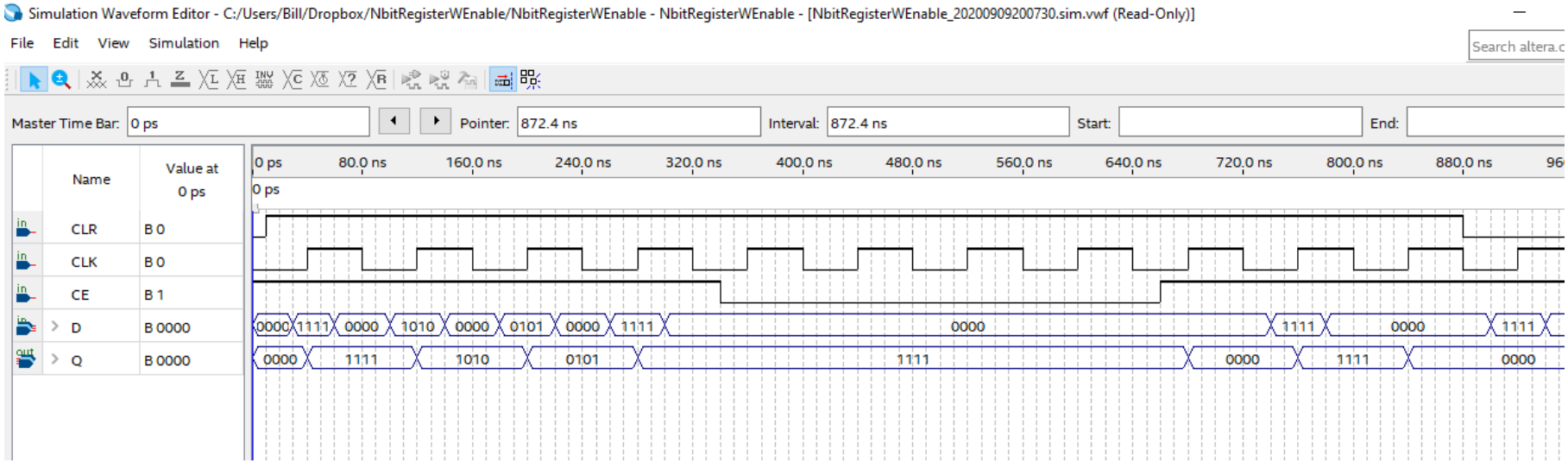
Recall

```
//Verilog Model of an N-bit register with active-low asynchronous clear
module NbitRegister (D,Q,CLK,CLR);
    parameter N = 4;                //declare default value for N
    input [N-1:0] D;                //declare N-bit data input
    input CLK, CLR;                 //declare clock and clear inputs
    output reg [N-1:0] Q;           //declare N-bit data output
        always @ (posedge CLK, negedge CLR) //detect change of clock or clear
            begin
                if (CLR==1'b0) Q <= 0;              //register loaded with all 0's
                    else if (CLK==1'b1) Q <= D;     //data input values loaded in register
            end
endmodule
```

n-bit Register with Clock Enable

```
//Verilog Model of an N-bit register with active-low clear and clock enable
module NbitRegister (
    input [N-1:0] D;                    //declare N-bit data input
    input CLK, CLR, CE,                 //declare clock, clear, and clock enable inputs
    output reg [N-1:0] Q;               //declare N-bit data output
    parameter N = 4;                    //declare default value for N
        always @ (posedge CLK, negedge CLR)      //detect change of clock or clear
            begin
                if (CLR==1'b0) Q <= 0;           //register loaded with all 0's
                    else if (CLK&CE==1'b1) Q <= D;   //data input values loaded in register
                        else Q <= Q;             // hold current values if not enabled
            end
endmodule
```

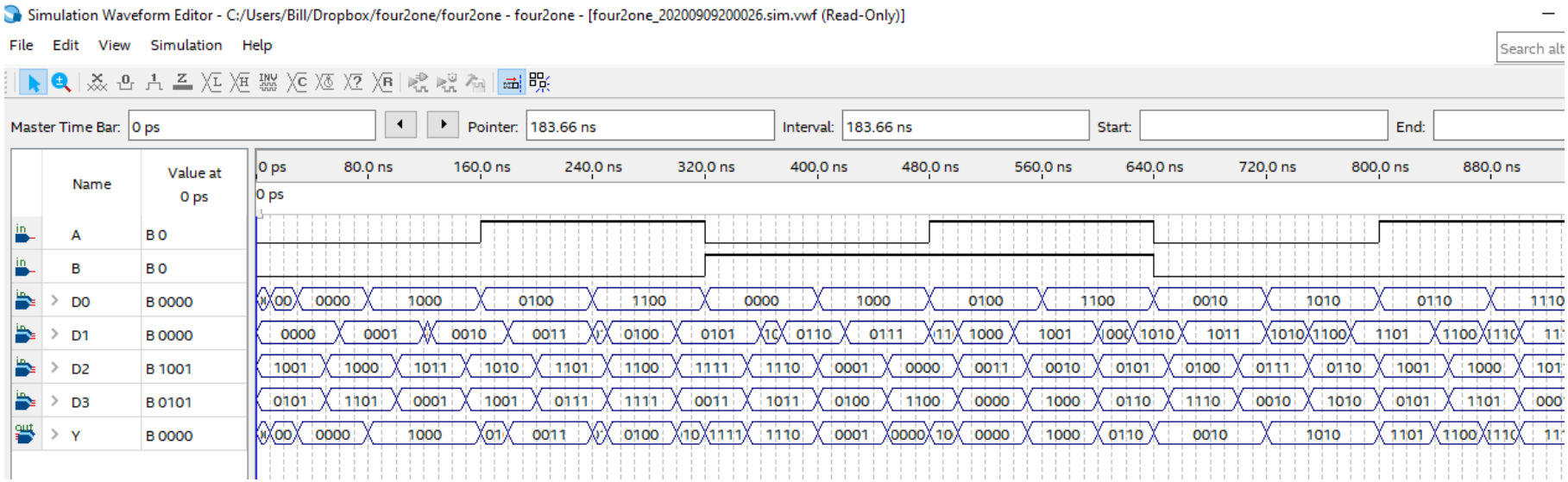
Four-bit Register Simulation



Four-to-One *n*-Bit Multiplexer

```
//Four-to-one, N-bit multiplexer
module four2one #(parameter N = 4)
    (
    input A,B,                          //declare select inputs
    input [N-1:0] D0,D1,D2,D3,          //declare data inputs
    output reg [N-1:0] Y                //declare data outputs
    );
    always @ (A,B,D0,D1,D2,D3)          //detect input change
        case ({B,A})                    //derive the output
            2'b00: Y = D0;
            2'b01: Y = D1;
            2'b10: Y = D2;
            2'b11: Y = D3;
        endcase
endmodule
```

Four-to-One Multiplexer Simulation



Instantiation Revisited

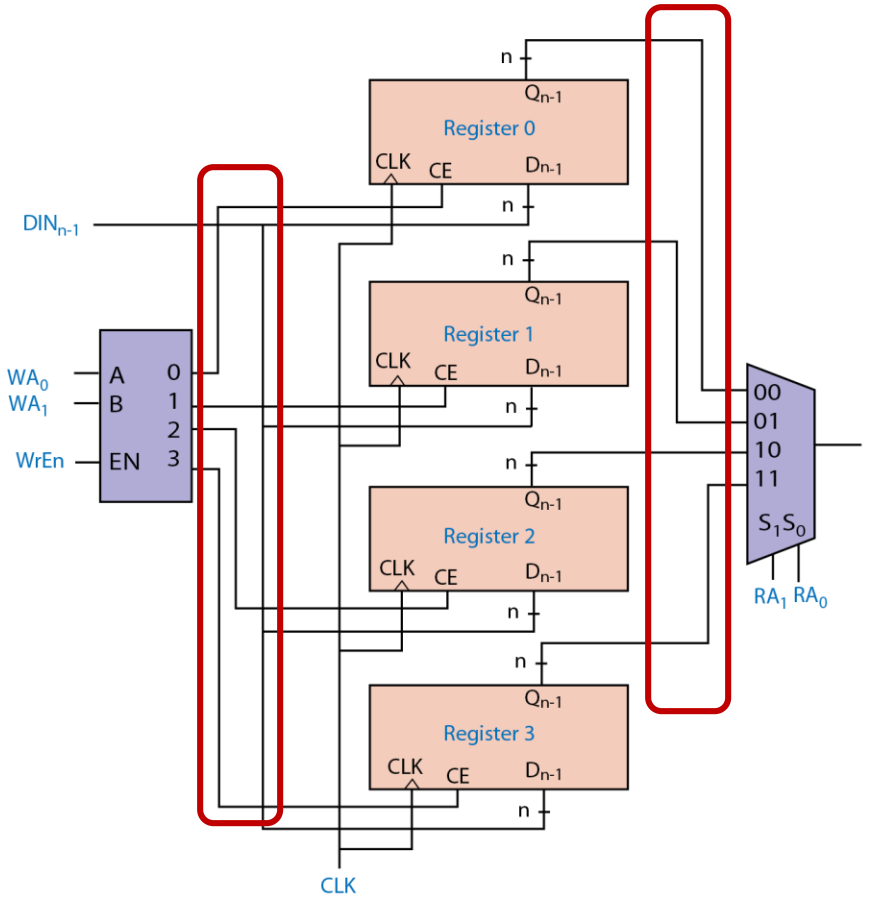
```
//Use of instances of binary2seven decoder module
module instances
(
        input [3:0] A, B,               //A:SW7,SW6,SW5,SW4; B:SW3,SW2,SW1,SW0
        output [0:6] outA, outB         //outA:HEX1; outB:HEX0
);
binary2seven hex0       //By position
(
        B,              // input [3:0] B
        outB            // output [0:6] outB
);
binary2seven hex1       //By name
(
        .BIN(A) ,       // input [3:0] A
        .SEV(outA)      // output [0:6] outA
);
endmodule
```

Instantiation Revisited

```
//Register 0
NbitRegisterWEnable R0          //By position
(
        .D(Din) ,       // input [3:0] Din
        .CLK(CLK) ,     // input  CLK
        .CLR(RST) ,     // input  RST
        .CE(y[0]) ,     // input  y[0]
        .Q(Q0)          // output [3:0] Q0
);
//Register 1
NbitRegisterWEnable R1          //By name
(
        .D(Din) ,       // input [3:0] Din
        .CLK(CLK) ,     // input  CLK
        .CLR(RST) ,     // input  RST
        .CE(y[1]) ,     // input  y[1]
        .Q(Q1)          // output [3:0] Q1
);
```

Register File Organization

Internal Nodes, y



Internal nodes, Q

Interconnecting Modules

Define internal nodes

```
wire [3:0] Q3, Q2, Q1, Q0, y;
```

Four-to-One *n*-Bit Multiplexer

```
//Four-to-one, N-bit multiplexer
module four2one #(parameter N = 4)
    (
    input A,B,                          //declare select inputs
    input [N-1:0] D0,D1,D2,D3,          //declare data inputs
    output reg [N-1:0] Y                //declare data outputs
    );
    always @ (A,B,D0,D1,D2,D3)          //detect input change
        case ({B,A})                    //derive the output
            2'b00: Y = D0;
            2'b01: Y = D1;
            2'b10: Y = D2;
            2'b11: Y = D3;
        endcase
endmodule
```

Pin Assignments Revisited

Pin Assignments Revisited

- Assignments Editor – manual
- Pin Planner
- TCL (Tool Command Language) console
- Import .QSF (Quartus Settings File) files

Pin Assignments Revisited

```
//Slide switches to LEDs
module switch2led
        (
        input [9:0] SW,
        output [9:0] LEDR
        );
        assign LEDR = SW;
endmodule
```

SW0 – PIN_AB12	**LEDR0 – PIN_V16**
SW1 – PIN_AC12	**LEDR1 – PIN_W16**
SW2 – PIN_AF9	**LEDR2 – PIN_V17**
SW3 – PIN_AF10	**LEDR3 – PIN_V18**
SW4 – PIN_AD11	**LEDR4 – PIN_W17**
SW5 – PIN_AD12	**LEDR5 – PIN_W19**
SW6 – PIN_AE11	**LEDR6 – PIN_Y19**
SW7 – PIN_AC9	**LEDR7 – PIN_W20**
SW8 – PIN_AD10	**LEDR8 – PIN_W21**
SW9 – PIN_AE12	**LEDR9 – PIN_Y21**

set_location_assignment <pin#> -to <name>

Pin Assignments Revisited

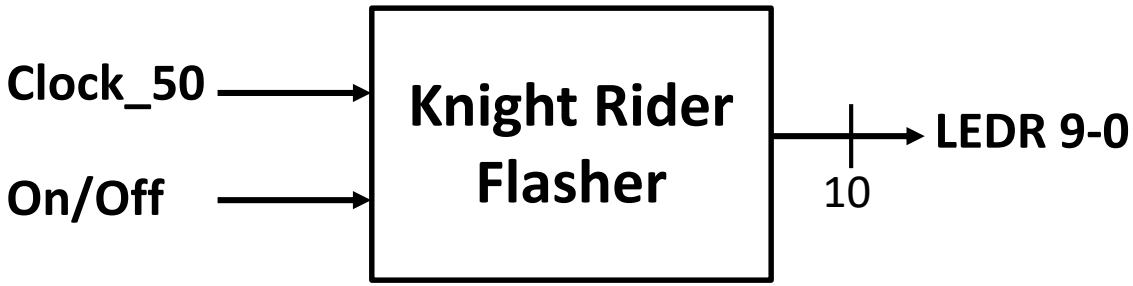
```
//Behavioral Verilog model of a digital lock.  Code 110010.
module toplock (
  input [9:0] SW,                          //Declare slide switch inputs
  input [3:0] KEY,                         //Declare pushbutton inputs
  output [9:0] LEDR,                       //Declare LED outputs
  output [0:6] HEX0);                      //Declare HEX0 output
  reg Open;                                //Declare lock Open output
  reg [2:0] state, nextstate;              //Declare lock state variables
  wire [0:6] stateout;                     //Declare lock state output
  wire x, Enter, ResetLock;                //Declare lock inputs
  parameter A=3'b000,B=3'b001,C=3'b010,D=3'b011,E=3'b100,F=3'b101,G=3'b110,H=3'b111;
    assign ResetLock = KEY[0];             //Map lock inputs & outputs to switches and LEDS
    assign Enter = KEY[1];
    assign x = SW[1];
    assign LEDR[0] = Open;
    assign LEDR[1] = SW[1];
    assign HEX0 = stateout;
   always @ (negedge Enter, negedge ResetLock) //Begin lock behavioral model
          if (ResetLock == 1'b0) state <= A;
          else state <= nextstate;
```

Pin Assignments Revisited

```
always @ (state, x)
        case (state)
                A: if (~x) nextstate = H; else nextstate = B;
                B: if (~x) nextstate = H; else nextstate = C;
                C: if (~x) nextstate = D; else nextstate = H;
                D: if (~x) nextstate = E; else nextstate = H;
                E: if (~x) nextstate = H; else nextstate = F;
                F: if (~x) nextstate = G; else nextstate = H;
                G: nextstate = G;
                H: nextstate = H;
        endcase
 always @ (state, x)
        case (state)
                A,B,C,D,E,F,H: Open = 1'b0;
                G: Open = 1'b1;
        endcase
//
binary2seven binary2seven_inst
(
        .BIN(state) ,    // input [3:0] state
        .SEV(stateout)                  // output [0:6] stateout
);
endmodule
```

Assignment #2 Preview

Design, implement, and demonstrate a Knight Rider Flasher.



- ✓ Clock_50 50-MHz clock
- ✓ On/Off toggle
- ✓ Clock divider
- ✓ Up/Down counter
- ✓ Module instantiation
- ✓ Pin assignment

Due – February 6, 2021, 11:59 PM

Q & A