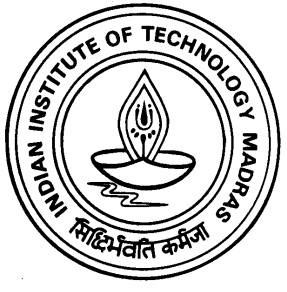
EE2001 - Digital systems lab
Synthesis and Implementation using an FPGA

Vinita Vasudevan



Sequential Circuits: Blocking statements

Shift Register

```
module sr( q, d );

output [3:0] q;
reg [3:0] q;
input d;
wire clock;

clock c1( .c(clock) );

always@(posedge clock )
   begin
      q[0] = d;
      q[1] = q[0];
      q[2] = q[1];
      q[3] = q[2];
   end
endmodule // sr
```

```
module clock( c );
 output c;
 reg c;
 initial
    begin
       c = 0;
    end
 always
    begin
       #5 c = ~c;
    end
 endmodule // clock
```

Wrong output!

0	clk=0	d=0	q = xxxx
4	clk=0	d=1	q = xxxx
5	clk=1	d=1	q = 1111
10	clk=0	d=1	q = 1111
14	clk=0	d=0	q = 1111
15	clk=1	d=0	q = 0000

At the positive edge, the input should be registered only in the first flip-flop. In this case, all flip-flops are set/reset.

Non-blocking Statements

Solution: Use non-blocking statements.

```
module sr( q, d );

output [3:0] q;
reg [3:0] q;
input d;
wire clock;

clock c1( .c(clock) );

always@(posedge clock )
   begin
      q[0] <= d;
      q[1] <= q[0];
      q[2] <= q[1];
      q[3] <= q[2];
   end
endmodule // sr
```

0	clk=0	d=0	q = xxxx
4	clk=0	d=1	q = xxxx
5	clk=1	d=1	q = xxx1
10	clk=0	d=1	q = xxx1
14	clk=0	d=0	q = xxx1
15	clk=1	d=0	q = xx10
20	clk=0	d=0	q = xx10
24	clk=0	d=1	q = xx10
25	clk=1	d=1	q = x101
30	clk=0	d=1	q = x101
34	clk=0	d=0	q = x101
35	clk=1	d=0	q = 1010
40	clk=0	d=0	q = 1010
44	clk=0	d=1	q = 1010
45	clk=1	d=1	q = 0101
50	clk=0	d=1	q = 0101
55	clk=1	d=1	q = 1011

Gather all the righthand side values before the edge and assign them to the left hand side variables after the clock edge. If blocking statements are used, the left hand side variables are assigned sequentially - $q[1]$ is assigned after $q[0]$ and so on.

Counters - Behavioural Model

Divide-by-two:

```
module clk_divide (Count, clk, reset);
 output Count;
 reg [3:0] Counter; //4 bit counter

 assign Count = Counter[1]; //pick 2nd bit
 always @ (negedge clk)
 begin
     if (!reset)
       begin
         Counter <= 0
       end
     else
       begin
         Counter <= Counter+1;
       end
 end
endmodule
```

0	clk=0	Count=0
5	clk=1	Count=0
10	clk=0	Count=0
15	clk=1	Count=0
20	clk=0	Count=1
25	clk=1	Count=1
30	clk=0	Count=1
35	clk=1	Count=1
40	clk=0	Count=0

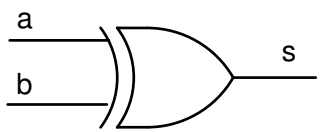
Digital design

- Specification
- Modelling using HDLs
- Functional simulation and testing
- Timing analysis
- Mapping to a hardware library/ FPGA (Synthesis)
- Place and Route
- Timing analysis.

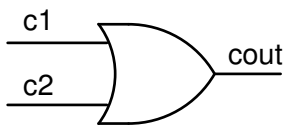
Synthesis

Map verilog models to hardware - Bit operators, gates, flipflops

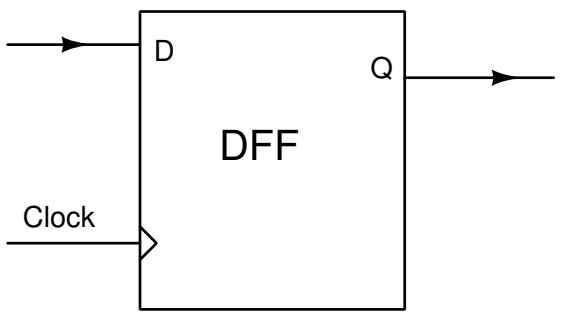
```
assign s = a^b;
```

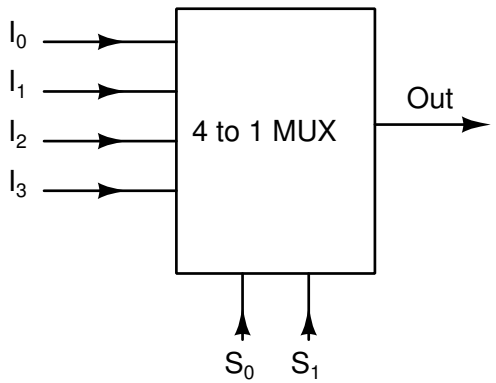



```
or or1( cout, c1, c2);
```




```
always @(posedge clock)
begin
  Q <= D;
end
```



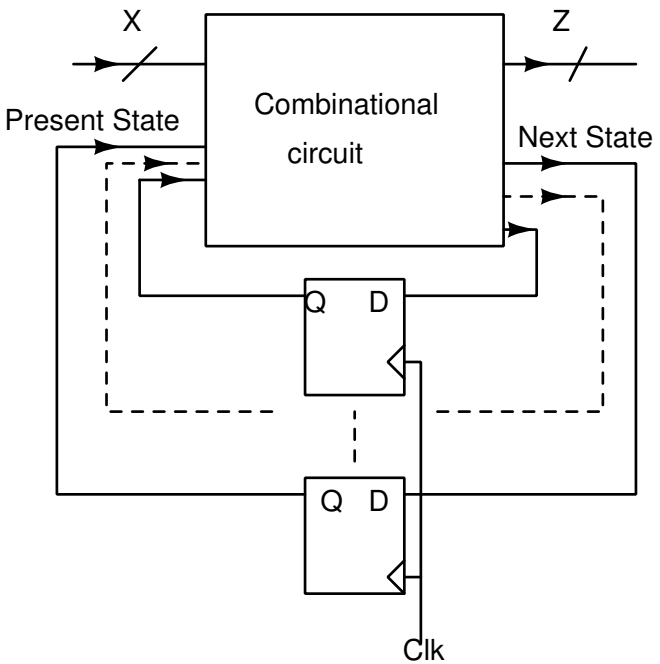



```
module mux4( I0, I1, I2, I3, s, out);

input I0, I1, I2, I3 ;
input[1:0] s;
output out;
reg out;

 assign t0 = (~s[1] & I0) | (s[1] & I2);
 assign t1 = (~s[1] & I1) | (s[1] & I3);
 assign out = (~s[0] & t0) | (s[0] & t1);
endmodule
```

```
assign out = (s == 0)? I0:
             (s == 1)? I1:
             (s == 2)? I2:
             (s == 3)? I3: 1'bx;
```

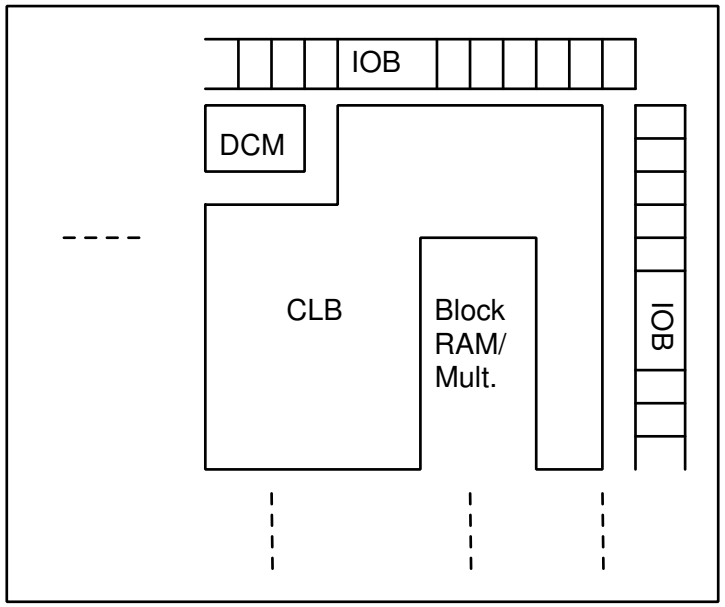
```
 always@(I0 or I1 or I2 or I3 or s)
begin
case( s )
  0: out = I0;
  1: out = I1;
  2: out = I3;
  3: out = I3;
end
```

State Machines: Modelling using Verilog




```
module StateMach( X, Z, clock );

input X ;
input clock;
output Z;
reg PresentState, NextState;

 always@(posedge clock)
begin
PresentState <= NextState
end

assign Z = X & PresentState;

assign NextState = X | PresentState;

endmodule
```

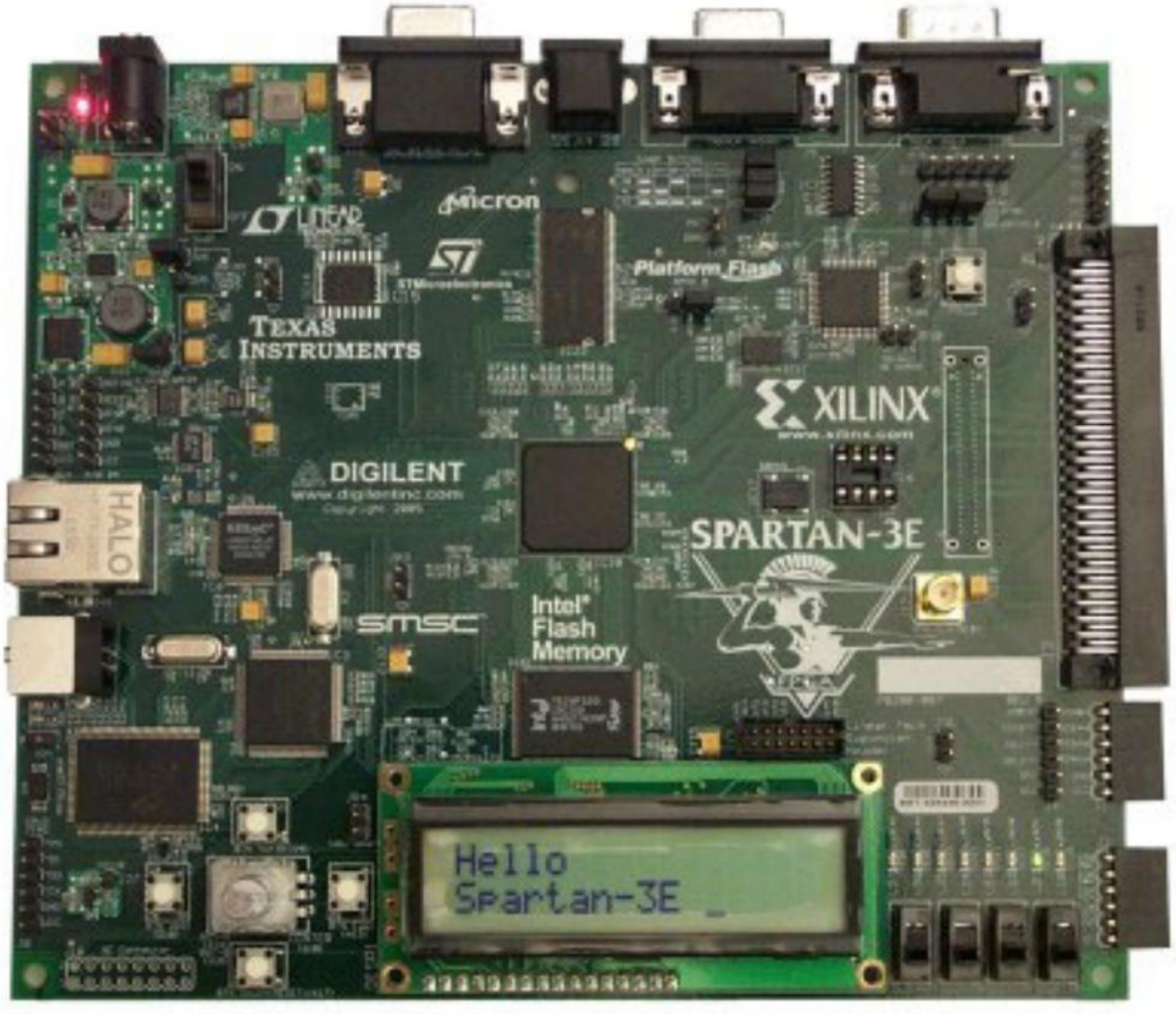
Field programmable gate arrays - FPGA



- CLB - Configurable logic blocks
- IOB - Programmable IO blocks
- Switch matrix - Programmable interconnects
- DCM - digital clock management

- CLBs contain Lookup tables (LUT), multiplexers, D flip flops and fast carry chain
- CLBs can be programmed to implement logic functions, can act as distributed RAM
- CLBs can be connected to obtain more complex logic functions using programmable switches and interconnects
- The IOBs can be programmed to act as input, output, tristate ....
- DCMs can be programmed to get various clock frequencies, phase shifts ...
- Block RAMs, multipliers

Spartan 3E development board



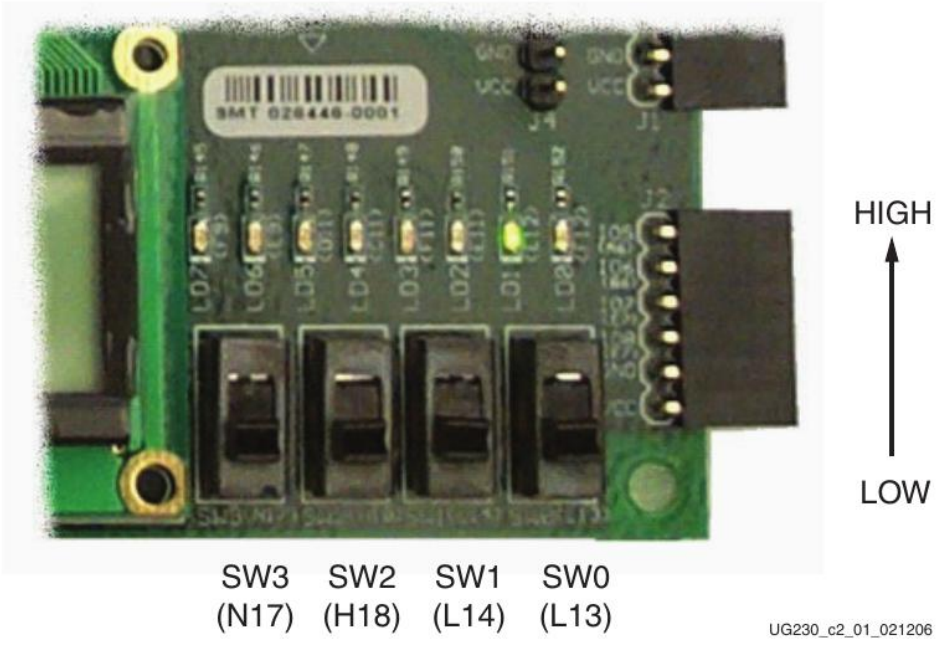


Figure 2-1: **Four Slide Switches**

```
NET "SW<0>" LOC = "L13" |
NET "SW<1>" LOC = "L14" |
NET "SW<2>" LOC = "H18" |
NET "SW<3>" LOC = "N17" |

NET "LED<7>" LOC = "F9" |
NET "LED<6>" LOC = "E9" |
NET "LED<5>" LOC = "D11"
NET "LED<4>" LOC = "C11"
NET "LED<3>" LOC = "F11"
NET "LED<2>" LOC = "E11"
NET "LED<1>" LOC = "E12"
NET "LED<0>" LOC = "F12"
```

The slide switches and LEDs are hardwired to certain pins in the FPGA. Must connect the appropriate signal to that pin. Specified in the UCF file.

Experiment 11

Decoder, Multiplier and counter the Spartan 3E FPGA

1. Write Verilog code for a 3-to-8 decoder using the case statement. Verify functionality and synthesize the circuit. Download and test on the board.
2. Write Verilog code for a 2-bit multiplier using gates.
3. Implement a clock divider that divides the 50MHz input clock to approximately 1Hz
4. Implement a 3-bit up/down counter using the 1Hz clock. Use a slide switch to indicate up/down. Display the number using an appropriate number of LEDs. Get the pin location of the clock from the user guide.

In both cases, the inputs are to be given using the switches present in the FPGA board and the result should be displayed using LEDs.