Design of Sequential Logic: Flip flops, counter, state machine, stacks

1 Today's goal

- Learn how to use always and if statements to design flip flops.
- Learn how to design sequential logic such as counters, state machines and stacks.
- Learn how to write the postfix notation of a formula.

2 Today's contents

Step 1 Write a flip flop(List 1) and its test bench(List 2), and perform the simulation to confirm that it works properly.

Step 2 *Check 1* Write a counter(List 3) and its test bench (List 4) and perform the simulation to confirm that it works properly.

Step 3 *Check 2* Write a top module (List 5) of the counter and its ucf (List 6). Implement the bit file in the FPGA and confirm that it works correctly.

Step 3 *Check 3* Write a state machine (List 7) and its test bench (List 8). Perform the simulation.

Step 4 *Check 4* Write a top module (List 9) and implement the state machine using ucf (List 10) in the FPGA.

Step 5 *Check 5* Write a stack (List 11), an operation stack (List 12) and its test bench (List 13). Perform the simulation.

3 Flip flops

In sequential logic, the output depends on the past and the present input. To implement sequential logic, flip flops (D-flip flops) are used to store the information of the past input. Table 1 shows the behavior of the flip flop. It has 3 input bits, d (data input), clk (clock), and reset, and one output bit, q (data output).

Table 1: Behavior of a Flip flop

Input		Output
clk	reset	q
-	0	0 (Asynchronous reset)
↑	1	d (Synchronous latch)
not ↑	1	previous q(keeps the same value)

In the event list in List 1, "posedge clk" and "negedge reset" mean rising edge of clk and falling edge of reset, respectively. Thus, if event negedge reset occurs, reset is always 0, and q <= 0 is executed. In the event of posedge clk, we have two cases:

- if reset is 0, then q <= 0 is executed, although q is 0 before the event.
- if reset is 1, then q <= d is executed.

Thus, the flip flop in List 1 satisfies the specification in List 1.

List 1: Flip flop ff.v

```
1  module ff(clk, reset, d, q);
2
3    input clk, reset, d;
4    output q;
5    reg q;
6
7    always @(posedge clk or negedge reset)
8      if(!reset) q <= 0;
9      else q <= d;
10
11 endmodule
```

List 2 shows an example of the test bench for flip flop. The frequency of clk is 10MHz (i.e. 100ns).

4 Counter

Let us design N-bit counter whose specification is defined in List 2 The counter has four 1-bit input clk, reset, load, and

List 2: Test bench for Flip flop ff.v

```
1  `timescale 1ns/1ps
2  module ff_tb;
3
4   reg clk,reset,d;
5   wire q;
6
7   ff ff0(.clk(clk),.reset(reset),.d(d),.q(q));
8
9   initial begin
10    clk = 0;
11    forever
12        #50 clk = ~clk;
13  end
14
15   initial begin
16      reset = 0; d = 0;
17    #100 reset = 1; d = 1;
18    #200 d = 0;
19    #200 d = 1;
20    #100 reset= 0;
21    #100 reset = 1;
22    #200 d = 1;
23    #100 d = 0;
24    end
25
26 endmodule
```

inc. It also has N-bit input d and N-bit output q.

Table 2: Specification of a counter

input				output
clk	reset	load	inc	q
-	0	-	-	0 (asynchronous reset)
↑	1	1	0	d (synchronous latch)
↑	1	0	1	q+1(increment)
↑	1	0	0	q(keeps the same value)
not ↑	1	-	-	q(keeps the same value)

List 3 is a Verilog HDL description of an N-bit counter. The default value of N is 16.

List 5 is a UCF file for a counter. If you have an error, then add a new constraint CLOCK_DEDICATED_ROUTE = FALSE; to the BTN_EAST (i.e. clk).

5 State machine

Figure 1 illustrates the state machine for CPU that we will design later of this course. Two states FETCHA and FETCHB

List 3: N-bit Counter counter.v

```
1  module counter(clk,reset,load,inc,d,q);
2     parameter N = 16;
3
4     input clk,reset,load,inc;
5     input [N−1:0] d;
6     output [N−1:0] q;
7     reg [N−1:0]  q;
8
9     always @(posedge clk or negedge reset)
10      if(!reset) q <= 0;
11      else if(load) q <= d;
12      else if(inc) q <= q + 1;
13
14 endmodule
```

List 4: Test bench for a counter counter_tb.v

```
1  `timescale 1ns / 1ps
2  module counter_tb;
3
4    reg clk,reset,load,inc;
5    reg [15:0] d;
6    wire [15:0] q;
7    counter counter0(.clk(clk), .reset(reset), .load(load), .inc(inc)
         , .d(d), .q(q));
8
9    initial begin
10     clk = 0;
11     forever
12       #50 clk = ~clk;
13   end
14
15   initial begin
16     reset = 0; load = 0; inc = 0; d=16'h0000;
17     #100 reset = 1;
18     #100 inc = 1;
19     #300 inc = 0; load = 1; d = 16'h1234;
20     #100 inc = 1; load = 0; d = 16'h0000;
21     #500 reset = 0;
22   end
23
24 endmodule
```

List 5: top module counter_top.v

```
1  module counter_top(BTN_NORTH, BTN_EAST, BTN_WEST,
       BTN_SOUTH, LED);
2    input BTN_NORTH, BTN_EAST, BTN_WEST,
         BTN_SOUTH;
3    output [7:0] LED;
4
5    counter #(8) counter0(.clk(BTN_EAST),.reset(~
         BTN_SOUTH),.load(BTN_NORTH),.inc(BTN_WEST),.
         d(8'h55),.q(LED));
6  endmodule
```

List 6: UCF file for counter_top.ucf(Spartan-3A/3AN Starter kit)

```
1  # PUSH SWITCH
2  NET "BTN_NORTH" LOC = "T14" | IOSTANDARD =
       LVTTL | PULLDOWN ;
3  NET "BTN_EAST" LOC = "T16" | IOSTANDARD = LVTTL
       | PULLDOWN ;
4  NET "BTN_WEST" LOC = "U15" | IOSTANDARD = LVTTL
       | PULLDOWN ;
5  NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD =
       LVTTL | PULLDOWN ;
6
7  # LED
8  NET "LED<7>" LOC = "W21" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
9  NET "LED<6>" LOC = "Y22" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
10 NET "LED<5>" LOC = "V20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
11 NET "LED<4>" LOC = "V19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
12 NET "LED<3>" LOC = "U19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
13 NET "LED<2>" LOC = "U20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
14 NET "LED<1>" LOC = "T19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
15 NET "LED<0>" LOC = "R20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
```

are used to fetch instruction codes from a memory, and EXECA and EXECB are used to execute an instruction according to the instruciton code. Since the state machine has 5 states, three bits are used to store the current state.

List 7 is a state machine for CPU and List 8 is its test bench.

List 7: State machine state.v

```
1  `define IDLE 3'b000
2  `define FETCHA 3'b001
3  `define FETCHB 3'b010
4  `define EXECA 3'b011
5  `define EXECB 3'b100
6
7  module state(clk,reset,run,cont,halt,cs);
8
9      input clk, reset, run, cont, halt;
10     output [2:0] cs;
11     reg [2:0]     cs;
12
13     always @(posedge clk or negedge reset)
14       if(!reset) cs <= `IDLE;
15       else
16         case(cs)
17           `IDLE: if(run) cs <= `FETCHA;
18           `FETCHA: cs <= `FETCHB;
19           `FETCHB: cs <= `EXECA;
20           `EXECA: if(halt) cs <= `IDLE;
21                   else if(cont) cs <= `EXECB;
22                   else cs <= `FETCHA;
23           `EXECB: cs <= `FETCHA;
24           default: cs <= 3'bxxx;
25         endcase
26
27 endmodule
```

6 Stack

Stack is a Last In First Out (LIFO) memory, which is used to store intermediate value for evaluating formula. Stack has 5 1-bit input clk, reset, load, push, pop, 16-bit input d, and 2 16-bit outputs qtop, qnext. Also, it has an array of 16-bit registers. 3 inputs load, push, pop are used to control the array of registers.

7 Operation Stack

An operation stack consists of stack and ALU. The operation stack is used to evaluate the postfix notation of formulas. For

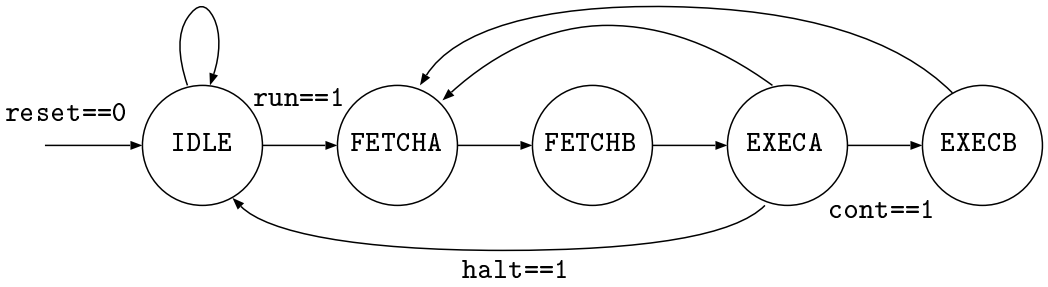

Figure 1: State machine

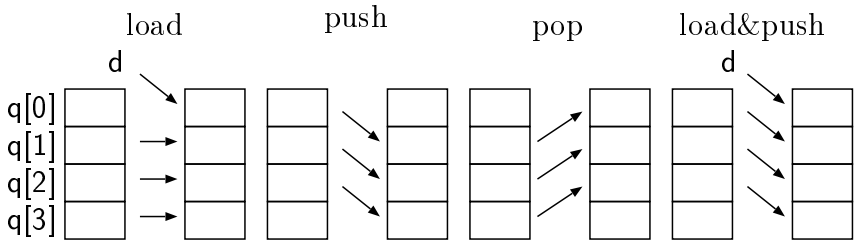

Figure 2: Behavior of Stack

List 8: Test bench state_tb.v

```
1  `timescale 1ns / 1ps
2  module state_tb;
3
4    reg clk,reset,run,halt,cont;
5    wire [2:0] cs;
6
7    state state0(.clk(clk),.reset(reset),.run(run),.cont(cont),.halt(
         halt),.cs(cs));
8
9    initial begin
10   clk = 0;
11   forever
12       #50 clk = ~clk;
13   end
14
15   initial begin
16     reset = 0; run = 0; halt = 0; cont = 0;
17     #100 reset = 1; run = 1;
18     #100 run = 0;
19     #200 cont = 1;
20     #100 cont = 0;
21     #600 halt = 1;
22     #100 halt = 0;
23   end
24
25 endmodule
```

List 9: Top module state_top.v of the state machine

```
1  `define IDLE 3'b000
2  `define FETCHA 3'b001
3  `define FETCHB 3'b010
4  `define EXECA 3'b011
5  `define EXECB 3'b100
6
7  module state_top(BTN_EAST, BTN_SOUTH, SW, LED);
8    input BTN_EAST, BTN_SOUTH;
9    input [2:0] SW;
10   output [4:0] LED;
11   wire [2:0] cs;
12
13   state state0(.clk(BTN_EAST),.reset(~BTN_SOUTH),.run(
         SW[2]),.cont(SW[1]),.halt(SW[0]),.cs(cs));
14
15   assign LED[4] = (cs == `IDLE);
16   assign LED[3] = (cs == `FETCHA);
17   assign LED[2] = (cs == `FETCHB);
18   assign LED[1] = (cs == `EXECA);
19   assign LED[0] = (cs == `EXECB);
20
21 endmodule
```

List 10: UCF state_top.ucf for state machine(Spartan-3A/3AN)

```
1  # PUSH SWITCH
2  NET "BTN_EAST" LOC = "T16" | IOSTANDARD = LVTTL
       | PULLDOWN ;
3  NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD =
       LVTTL | PULLDOWN ;
4
5  # LED
6  NET "LED<4>" LOC = "V19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
7  NET "LED<3>" LOC = "U19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
8  NET "LED<2>" LOC = "U20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
9  NET "LED<1>" LOC = "T19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
10 NET "LED<0>" LOC = "R20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
11
12 # SLIDE SWITCH
13 NET "SW<2>" LOC = "U8" | IOSTANDARD = LVTTL |
       PULLUP ;
14 NET "SW<1>" LOC = "U10" | IOSTANDARD = LVTTL |
       PULLUP ;
15 NET "SW<0>" LOC = "V8" | IOSTANDARD = LVTTL |
       PULLUP ;
```

example, usual formua uses infix notation as follows:

$$(1 + 2) * (3 + 4)$$

To evaluate this formula on the operation stack, the postfix notation below are used:

$$1 \quad 2 \quad + \quad 3 \quad 4 \quad + \quad *$$

Table 3 shows the specification of the operation stack.

List 12 shows an Verilog HDL description of operaiton stack. It isntantiate ALU and stack.

List 13 is an example of a test bench for operation stack. In this example, the infix notation of the following formula is evaluated.

$$(-(2 + 3 * 4) < 5) \;||\; (6 > 7)$$

This infix notation must be converted to the postfix notation as follows:

$$2\ 3\ 4\ *\ +\ -\ 5\ <\ 6\ 7\ >\ ||$$

8 Homework

Homework 1 Design 4-bit counter by instantiating 4 flip flops and 4 full adders. The 4 flip flops are used to store

Table 3: Operation Stack

input				operation	behavior			
clk	reset	num	op		load	push	pop	d
-	0	-	-	asyncronous reset	-	-	-	-
↑	1	1	-	push x	1	1	0	x
↑	1	-	1	operetion specified by x	1	0	1 (binary operation) 0 (unary operation)	s of alu

List 11: Stack stack.v

```
1  module stack(clk, reset, load, push, pop, d, qtop, qnext);
2
3    input clk, reset, load, push, pop;
4    input [15:0] d;
5    output [15:0] qtop, qnext;
6    reg [15:0] q [3:0];
7
8    assign qtop = q[0];
9    assign qnext = q[1];
10
11   always @(posedge clk or negedge reset)
12     if(!reset) q[0] <= 0;
13     else if(load) q[0] <= d;
14     else if(pop) q[0] <= q[1];
15
16   always @(posedge clk or negedge reset)
17     if(!reset) q[1] <= 0;
18     else if(push) q[1] <= q[0];
19     else if(pop) q[1] <= q[2];
20
21   always @(posedge clk or negedge reset)
22     if(!reset) q[2] <= 0;
23     else if(push) q[2] <= q[1];
24     else if(pop) q[2] <= q[3];
25
26   always @(posedge clk or negedge reset)
27     if(!reset) q[3] <= 0;
28     else if(push) q[3] <= q[2];
29
30 endmodule
```

List 12: Operation stack opstack.v

```
1  module opstack(clk,reset,num,op,x);
2
3    input clk, reset, num, op;
4    input [15:0] x;
5    wire [15:0] qtop, qnext, aluout;
6    wire load, push, pop;
7    reg [15:0] stackin;
8
9    alu alu0(.a(qtop), .b(qnext), .f(x[4:0]), .s(aluout));
10   stack stack0(.clk(clk), .reset(reset), .load(load), .push(push),
         .pop(pop), .d(stackin), .qtop(qtop), .qnext(qnext));
11
12   assign load = num | op;
13   assign push = num;
14   assign pop = op & ~x[4];
15
16   always @(num or op or x or aluout)
17     if(num) stackin = x;
18     else if(op) stackin = aluout;
19     else stackin = 16'hxxxx;
20
21 endmodule
```

List 13: Test bench opstack_tb.v for operation stack

```
1  `timescale 1ns / 1ps
2
3  `define ADD 5'b00000
4  `define SUB 5'b00001
5  `define MUL 5'b00010
6  `define SHL 5'b00011
7  `define SHR 5'b00100
8  `define BAND 5'b00101
9  `define BOR 5'b00110
10 `define BXOR 5'b00111
11 `define AND 5'b01000
12 `define OR 5'b01001
13 `define EQ 5'b01010
14 `define NE 5'b01011
15 `define GE 5'b01100
16 `define LE 5'b01101
17 `define GT 5'b01110
18 `define LT 5'b01111
19 `define NEG 5'b10000
20 `define BNOT 5'b10001
21 `define NOT 5'b10010
22
23 module opstack_tb;
24
25   reg clk, reset, num, op;
26   reg [15:0] x;
27
28   opstack opstack0(.clk(clk), .reset(reset), .num(num), .op(op)
         , .x(x));
29
30   initial begin
31   clk = 0;
32   forever
33     #50 clk = ~clk;
34   end
35
36   initial begin
37   reset = 0; num = 0; op = 0; x = 0;
38   #100 reset = 1; num = 1; op = 0; x = 2;
39   #100 num = 1; op = 0; x = 3;
40   #100 num = 1; op = 0; x = 4;
41   #100 num = 0; op = 1; x = `MUL;
42   #100 num = 0; op = 1; x = `ADD;
43   #100 num = 0; op = 1; x = `NEG;
44   #100 num = 1; op = 0; x = 5;
45   #100 num = 0; op = 1; x = `LT;
46   #100 num = 1; op = 0; x = 6;
47   #100 num = 1; op = 0; x = 7;
48   #100 num = 0; op = 1; x = `GT;
49   #100 num = 0; op = 1; x = `OR;
50   #100 num = 0; op = 0; x = 0;
51   end
52
53 endmodule
```

4-bit integer, and the 4 full adders ure used to compute plus one (+1). Write the test bench for it and perform the simulation to confirm that it works properly. Also you need to write the illustration of the diagram of this 4-bit counter.

Homework 2 Design a state machine with 6 states as follows:

- 6 states are assigned using 3 bits as follows: State0: 3'b000, State1: 3'b001, State2: 3'b010, State3: 3'b011, State4: 3'b100, State5: 3'b101.
- 2 control inputs nextstate and jumpstate are used as follows: If the current state is State i and nextstate is 1, then next state is State (i+1 mod 6). If the current state is State i and jumpstate is 1, then next state is State (i+2 mod 6). If both control bits are 0, then the state is not changed.

Homework 3 Extend the stack (List 11) such that it has six 16-bit registars. After that, write the infix notation of a formula that has at least 10 operations. Convert it to the postfix notatin, and write the test bench to evaluate the formula on the operation stack. Perform the simulation.