CME 341: Synchronous Logic, Part 3

Brian Berscheid

Department of Electrical and Computer Engineering
University of Saskatchewan

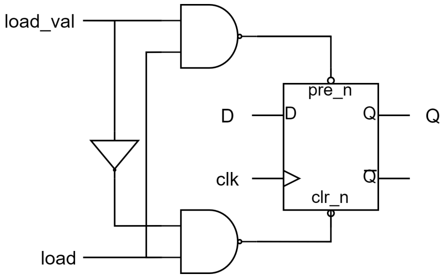
Today's agenda

1. Asynchronous vs synchronous loads in D-FFs
2. Example synchronous circuits
3. Synchronous logic summary

Asynchronous vs synchronous loads in D-FFs

Recall: asynchronous load in D-FFs

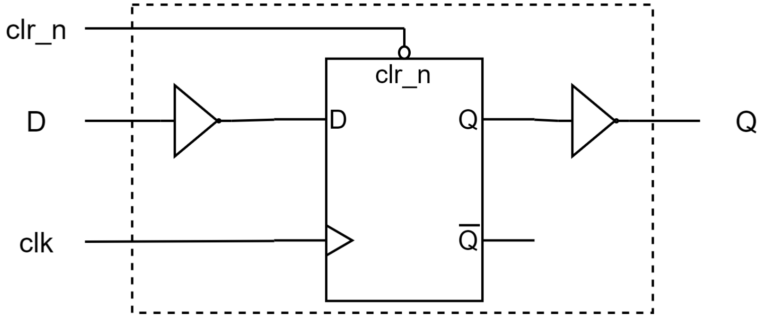
```
reg q;

always @ (posedge clk or posedge load)
if (load == 1'b1)
  q = load_val;  // load_val is a variable
else
  q = d;
```



Problem: most FPGAs use FFs that only have an asynchronous clear port!
(no asynchronous reset)

Asynchronous set with only asynchronous clear

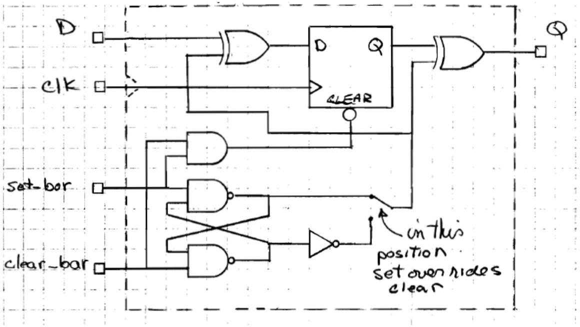
Quartus will actually build this if you ask for async set functionality



- Inverters on input and output cancel with respect to the D → Q path
- Asserting clear "clears" the internal FF, but the observable Q output is "set"

Asynchronous load with only asynchronous clear

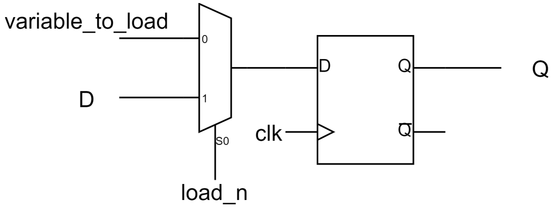
Quartus will actually build this if you ask for async load functionality



- Either set or clear can trigger the clear port on the FF
- Latch circuit (bottom of diagram) used to control whether to use inverters or not
  - ▷ Latches generally undesirable... typically better to avoid asynchronous loads
  - ▷ Generally recommend using synchronous loads instead

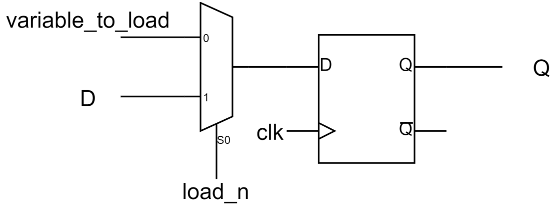
D-FFs with synchronous load

- Loads a constant or variable into a FF *on a clock edge*
- Only clock needs to go in the procedure sensitivity list
- Compiler will build logic in front of the FF to control the FF's D input to perform the load



Verilog code for D-FFs with synchronous load

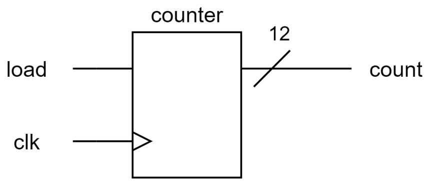
```
always @ (posedge clk)
  if (load_n == 1'b0)
    Q = variable_to_load;
  else
    Q = D;
```



Example synchronous circuits

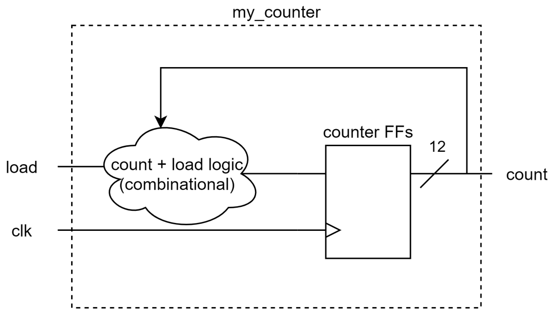
Example 1: Synchronous counter with synchronous load

Design a 12-bit counter which can be synchronously loaded with the value 43.

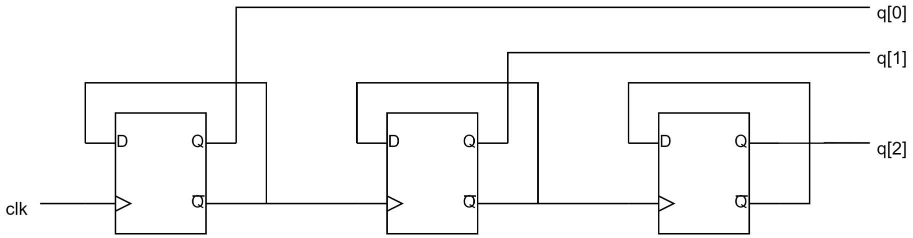


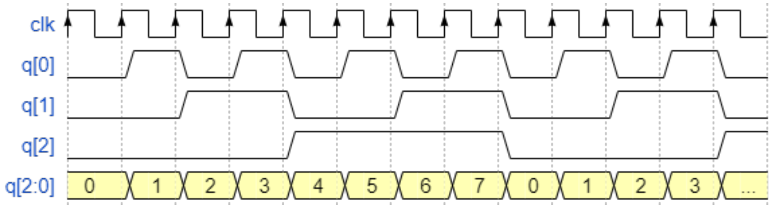
Synchronous counter with synchronous load: a solution

```
module my_counter (
  input wire        clk,
  input wire        load,
  output reg [11:0] count
);

always @ (posedge clk)
  if (load == 1'b1)
    count = 12'd43;
  else
    count = count + 12'd1;

endmodule
```



Example 2: 3-bit ripple counter





3-bit ripple counter: a solution

```
module ripple_counter (
  input wire clk,
  output reg [2:0] q
);

always @ (posedge clk)
  q[0] = ~q[0];
always @ (negedge q[0])
  q[1] = ~q[1];
always @ (negedge q[1])
  q[2] = ~q[2];

endmodule
```

Example 3: Decimal counter

Create a counter that counts from 2 to 9 and then rolls over back to 2 and repeats.

Decimal counter: a solution

```
module decimal_counter (
  input  wire       clk,
  output reg [3:0] count
);

always @ (posedge clk)
  if (count == 4'd9)
    count = 4'd2;
  else
    count = count + 4'd1;

endmodule
```

What would happen if count somehow became equal to 4'd11? How could we fix this problem?

Example 4: Ring counter

Create a 3-bit "ring counter" that generates the following sequence of outputs:
$001 \rightarrow 010 \rightarrow 100 \rightarrow 001 \rightarrow 010 \rightarrow$ etc...

Sketch the hardware corresponding to the circuit you designed.

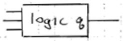
Synchronous logic summary

Synchronous logic

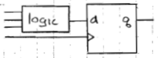
- The basic element of synchronous logic is the FF
- Synchronous logic is built in procedures with edge-sensitive signals in the sensitivity list: ie, `always @ (posedge clk)`
- If multiple edge-sensitive signals are in the sensitivity list, all but one must be checked in the procedure body
  - ▷ The checked signal(s) will be used for asynchronous set/clear functionality
  - ▷ The unchecked signal (one only!) will be used as the clock
- Synchronous set/clear/load functionality is built using combinational logic in front of the FFs
  - ▷ The set/clear/load trigger signal is not put in the sensitivity list in this case

Review exercise 1: identify the hardware (A, B, or C) for each code block

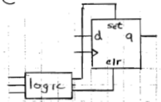
A



B



C



1.

```
always @(posedge clk)
if (clear==1'b1)  q = 1'b1;
else  q = d;
```

2.

```
always @ ( posedge clk or
            posedge clear)
if (clear == 1'b1)  q = 1'b0;
else  q = d;
```

3

```
always @ (posedge clk or
            posedge load)
if (load ==1'b1) q = d1;
else  q = d;
```

4

```
always @ *
if (clear == 1'b1)  q = 1'b1;
else  q = d;
```

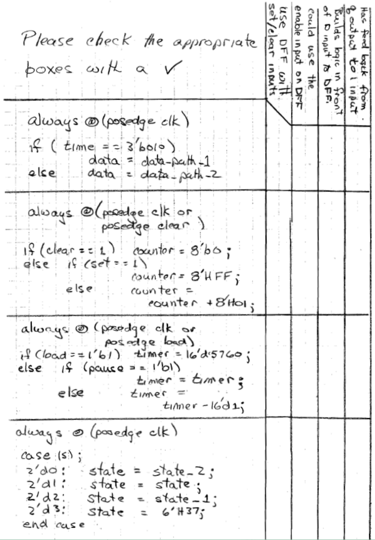
5

```
always @ (posedge clk)
if (load ==1'b1) q = d1;
else  q = d
```

6

```
always @ *
if (load == 1'b1)  q = d1;
else  q = d;
```

Review exercise 2: identify the properties of each code block



Please check the appropriate boxes with a ✓

	use DFF with set/clear inputs	Could use the enable input on DFF	Builds logic in front of D input to DFF.	Has feed back from Q output to 1 input.

always @(posedge clk)
if (time == 3'b010)
data = data-path-1
else data = data-path-2				

always @(posedge clk or posedge clear)
if (clear == 1) counter = 8'b0 ;
else if (set == 1)
counter = 8'HFF;
else counter =
counter + 8'H01;				

always @ (posedge clk or posedge load)
if (load == 1'b1) timer = 16'd5760 ;
else if (pause == 1'b1)
timer = timer;
else timer =
timer - 16'd1;				

always @ (posedge clk)
case (s) ;
2'd0 : state = state-2;
2'd1 : state = state ;
2'd2 : State = state-1;
2'd3 : State = 6'H37;
end case				

Thank you!
Have a great day!