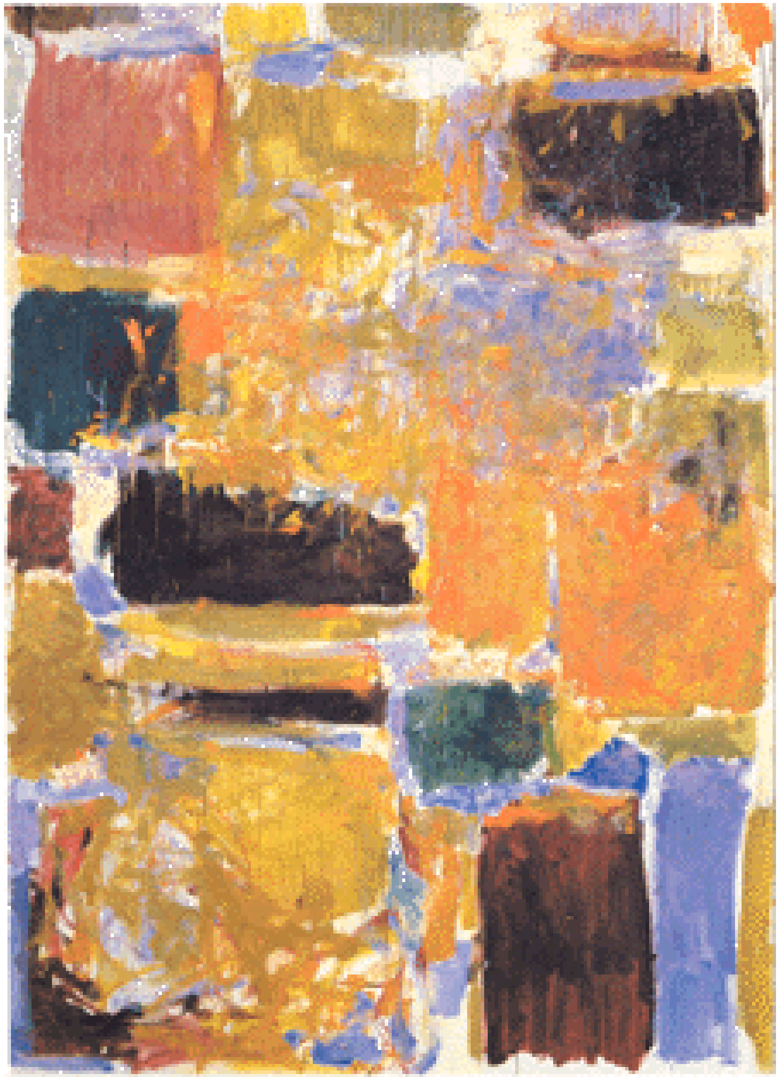


CS150 Discussion 2

Simon Scott

Registers

- What is a register?

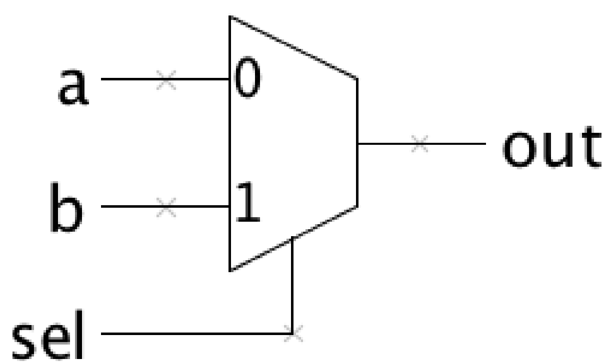
- Given the following timing for counter, what is maximum possible clock rate?
  - Setup time = 2.5ns
  - Hold time = 1.5ns
  - Clk-to-q time = 1ns
  - Delay through adder = 3ns

- Minimum delay through adder to avoid race condition?

The Sequential always Block

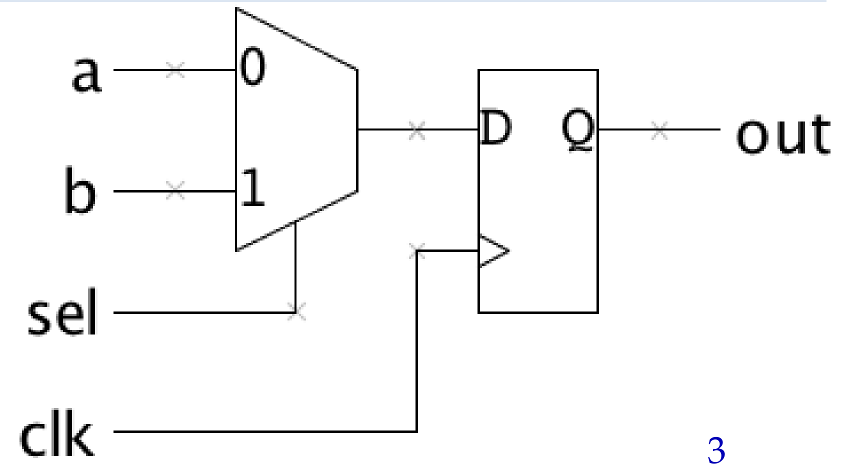
Combinational

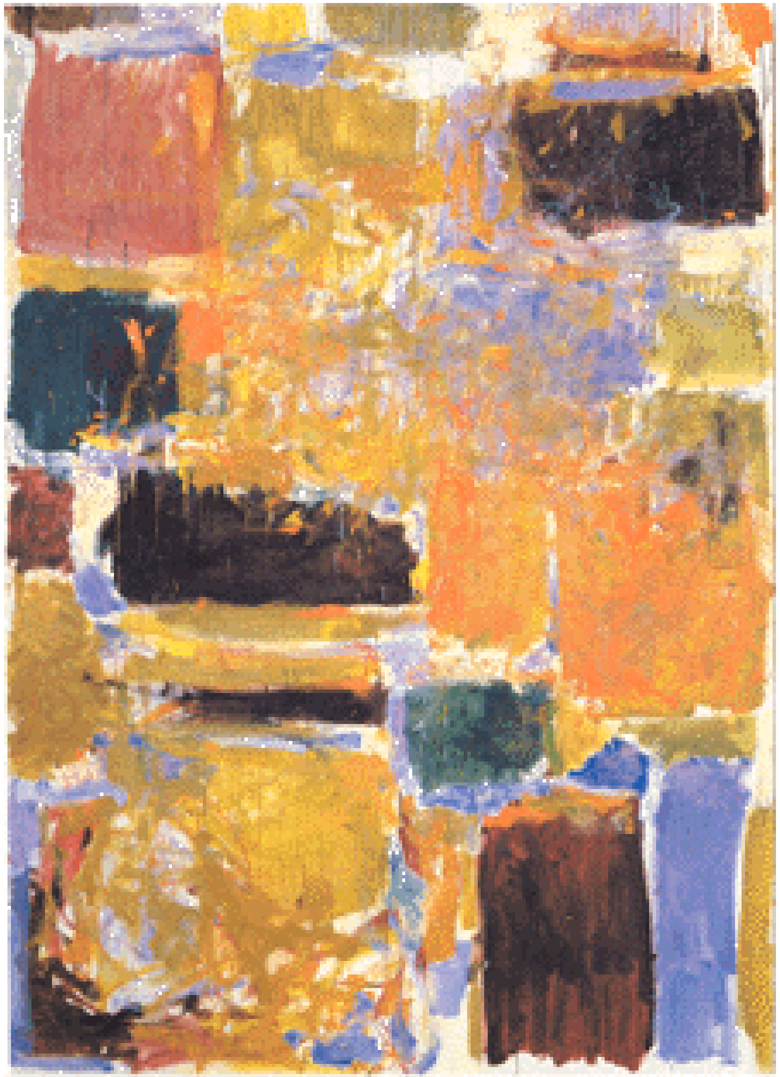
```
module comb(input a, b, sel,
            output reg out);

  always @(*) begin
    if (sel) out = b;
    else out = a;
  end

endmodule
```



Sequential

```
module seq(input a, b, sel, clk,
           output reg out);

  always @(posedge clk) begin
    if (sel) out <= b;
    else out <= a;
  end

endmodule
```



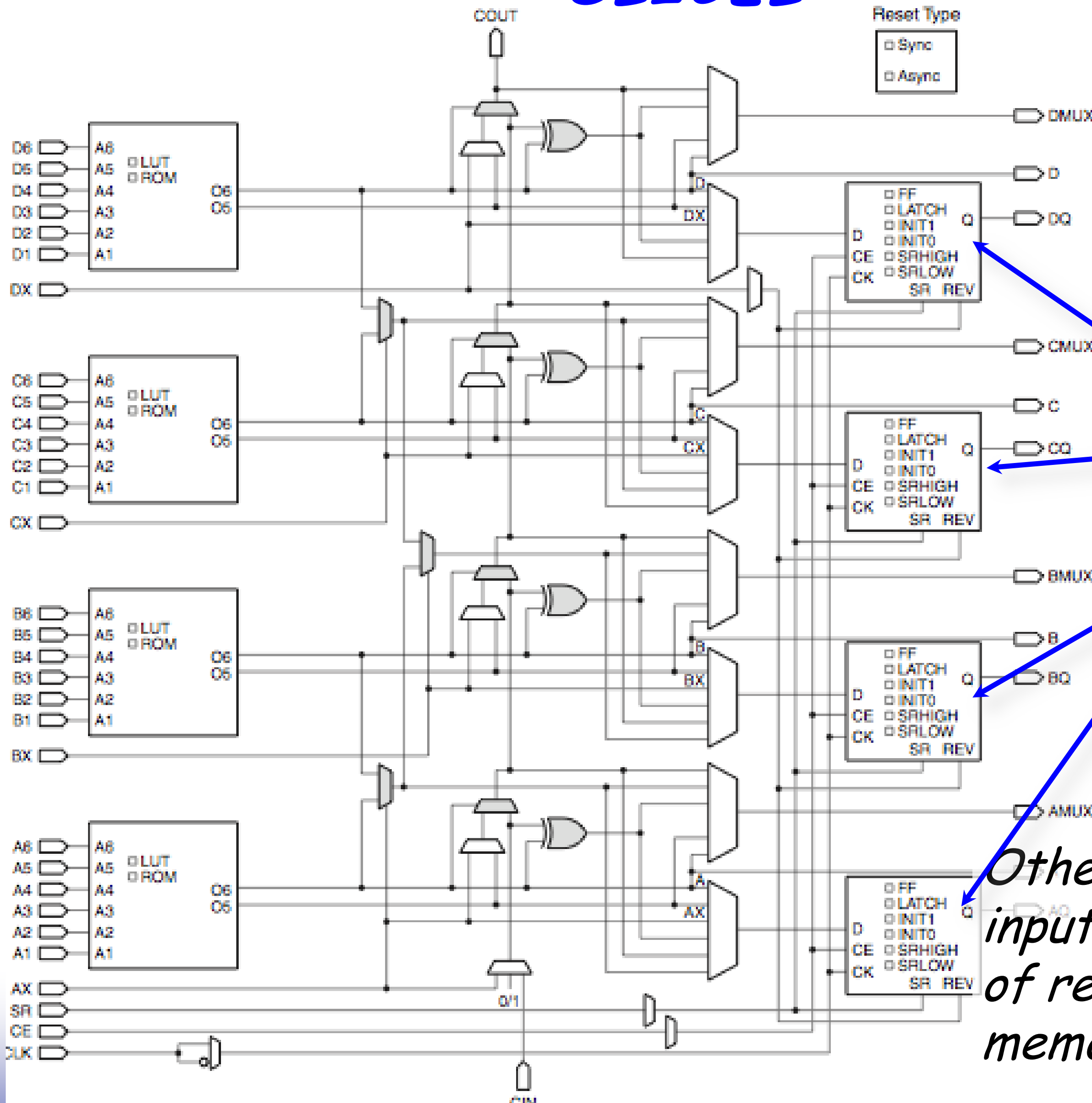


FPGA Flip-flops

(adapted from slides by John Wawzrynek)

Flip-flops on Virtex5 FPGA

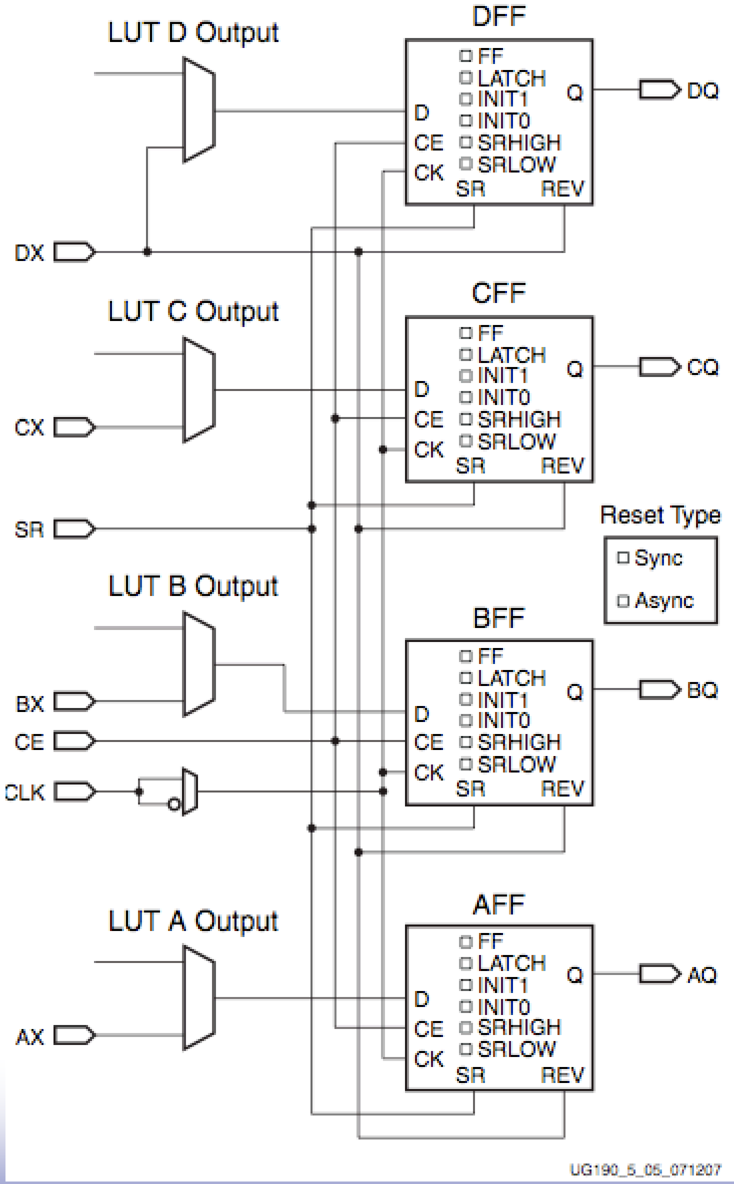
SLICEL



Four flip-flops per slice.
17,280 slices in an LX110T.

Other flip-flops in the chip input/output cells, and in the form of registers in the DSP slices and memory block interfaces.

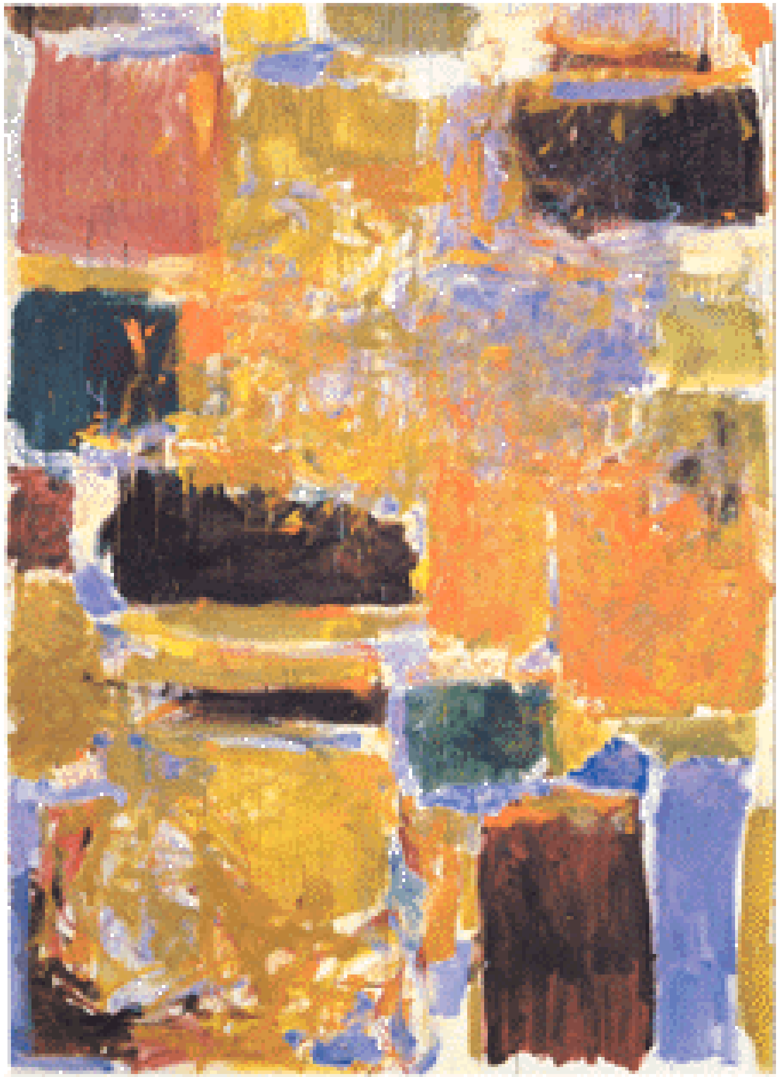
Virtex5 Slice Flip-flops



4 flip-flops / slice (corresponding to the 4 6-LUTs)

Each takes input from LUT output or primary slice input.

Edge-triggered FF vs. level-sensitive latch.
Clock-enable input (can be set to 1 to disable) (shared).
Positive versus negative clock-edge.
Synchronous vs. asynchronous reset.
SRHIGH/SRLOW select reset (SR) set.
REV forces opposite state.
INIT0/INIT1 used for global reset (not shown - usually just after power-on and configuration).



Verilog Simulator

Verilog Events

IEEE 1364-2001 Verilog Standard: ***Section 5.3 The stratified event queue***

The Verilog event queue is logically segmented into five different regions. Events are added to any of the five regions but are only removed from the active region.

1. Events that occur at the current simulation time and can be processed in any order. These are the active events.

2. Events that occur at the current simulation time, but that shall be processed after all the active events are processed. These are the inactive events.

3. Events that have been evaluated during some previous simulation time, but that shall be assigned at this simulation time after all the active and inactive events are processed. These are the nonblocking assign update events.

4. Events that shall be processed after all the active, inactive, and nonblocking assign update events are processed. These are the monitor events.

5. Events that occur at some future simulation time. These are the future events. Future events are divided into future inactive events, and future nonblocking assignment update events.

Coding Guidelines

The following are helpful guidelines that ensure your simulation results will match what they synthesized hardware will do:

1. *When modeling sequential logic, use nonblocking assignments.*
2. *When modeling latches, use nonblocking assignments.*
3. *When modeling combinational logic with an always block, use blocking assignments.*
4. *When modeling both sequential and combinational logic within the same always block, use nonblocking assignments.*
5. *Do not mix blocking and nonblocking assignments in the same always block.*
6. *Do not make assignments to the same variable from more than one always block.*
7. *Use $strobe to display values that have been assigned using nonblocking assignments.*
8. *Do not make assignments using #0 delays.*

#1 thing we will be checking in your Verilog submissions!

= vs. <= inside always

```
module main;
  reg a,b,clk;

  initial begin
    clk = 0; a = 0; b = 1;
    #10 clk = 1;
    #10 $display("a=%d b=%d\n",a,b);
    $finish;
  end
endmodule
```

A

```
always @(posedge clk) begin
  a = b;    // blocking assignment
  b = a;    // execute sequentially
end
```

B

```
always @(posedge clk) begin
  a <= b;  // non-blocking assignment
  b <= a;  // eval all RHSs first
end
```

C

```
always @(posedge clk) a = b;
always @(posedge clk) b = a;
```

D

```
always @(posedge clk) a <= b;
always @(posedge clk) b <= a;
```

E

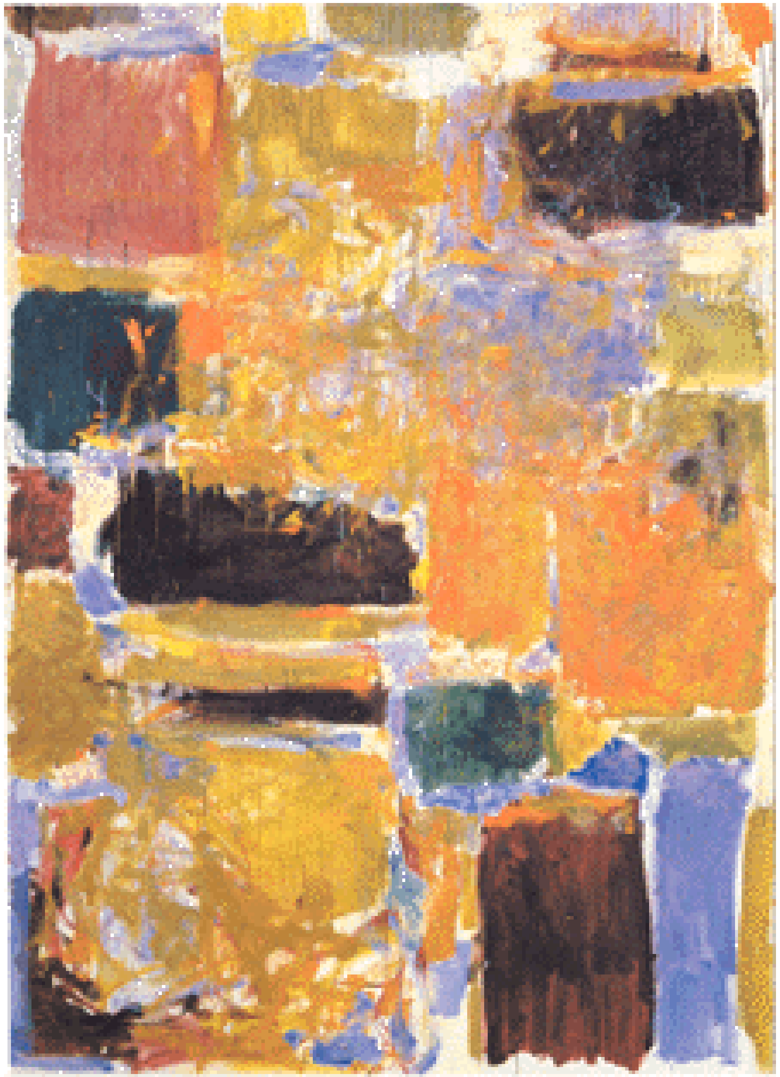
```
always @(posedge clk) begin
  a <= b;
  b = a;    // urk! Be consistent!
end
```

Rule: always change state using <=
(e.g., inside `always @(posedge clk)`...)

Exercises

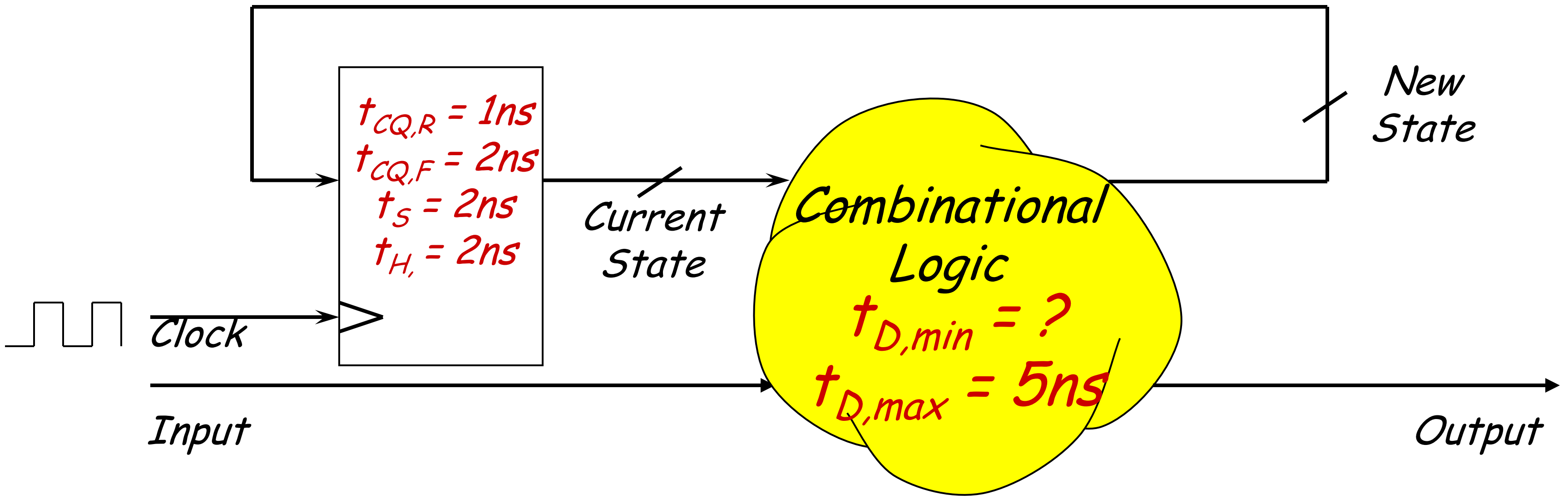
```
module flipflop <or>
latch
(
    input clk,
    input d,
    output reg q
);

        <your code here>
endmodule
```

- Write Verilog module for:
  - D-type flip flop
  - Latch
  - JK flip flop
    - Output changes on rising clock edge according to following rules:
    - J = 0, K = 0: out unchanged
    - J = 1, K = 0: out = 0
    - J = 0, K = 1: out = 1
    - J = K = 1: output toggles



Timing

Sequential Circuit Timing

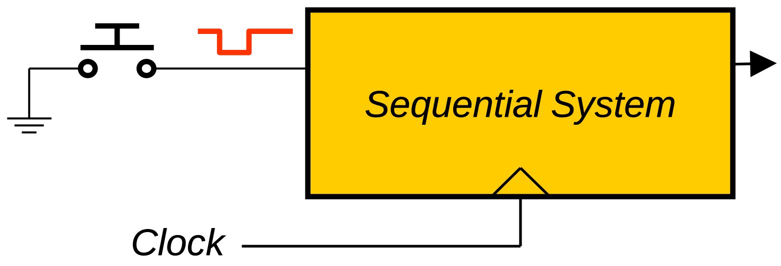


Questions:

- Constraints on $t_{D,min}$ for the logic?
- Minimum clock period?
- Setup, Hold times for Inputs?

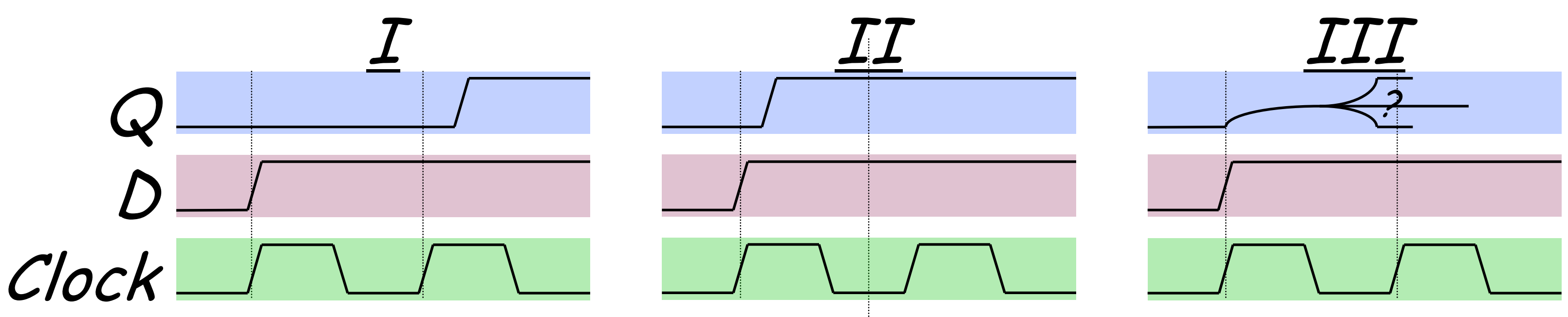
Asynchronous Inputs in Sequential Systems

What about external signals?



Can't guarantee setup and hold times will be met!

When an asynchronous signal causes a setup/hold violation...



Transition is missed on first clock cycle, but caught on next clock cycle.

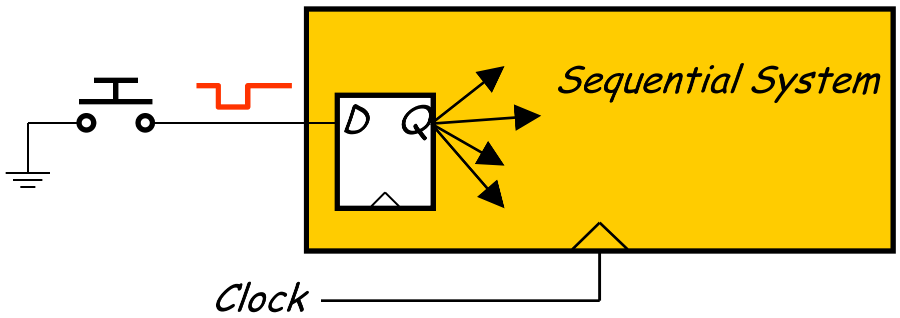
Transition is caught on first clock cycle.

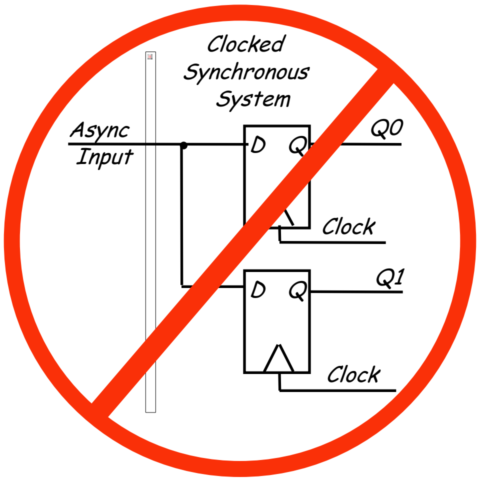
Output is metastable for an indeterminate amount of time.

Asynchronous Inputs in Sequential Systems

All of them can be, *if more than one happens simultaneously within the same circuit.*

Guideline: ensure that external signals directly feed *exactly one* *flip-flop*

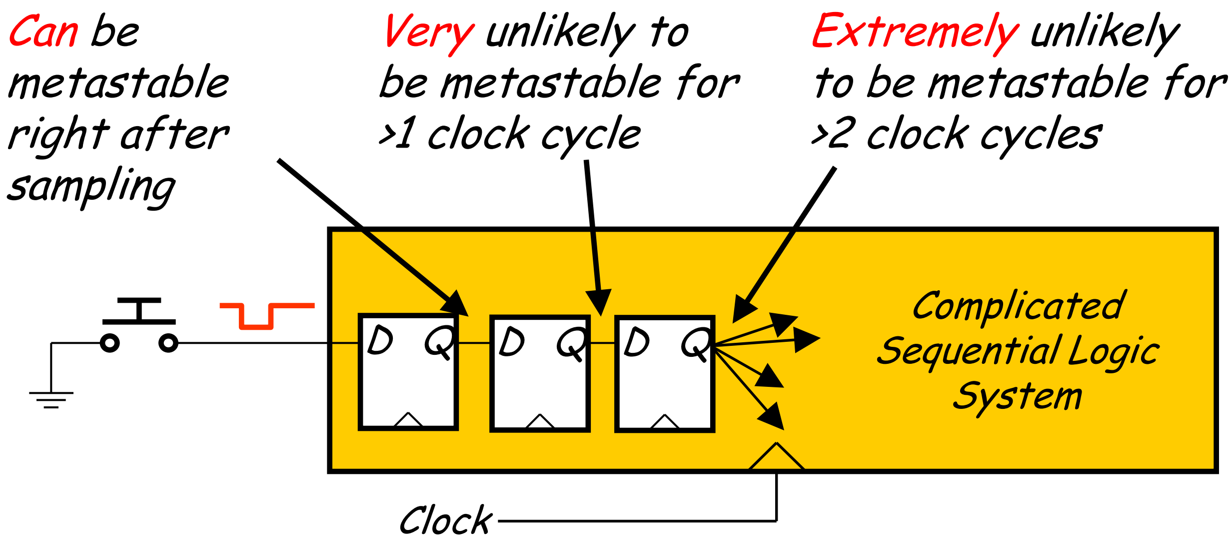




This prevents the possibility of I and II occurring in different places in the circuit, but what about metastability?

Handling Metastability

- Preventing metastability turns out to be an impossible problem
- High gain of digital devices makes it likely that metastable conditions will resolve themselves quickly
- Solution to metastability: allow time for signals to stabilize



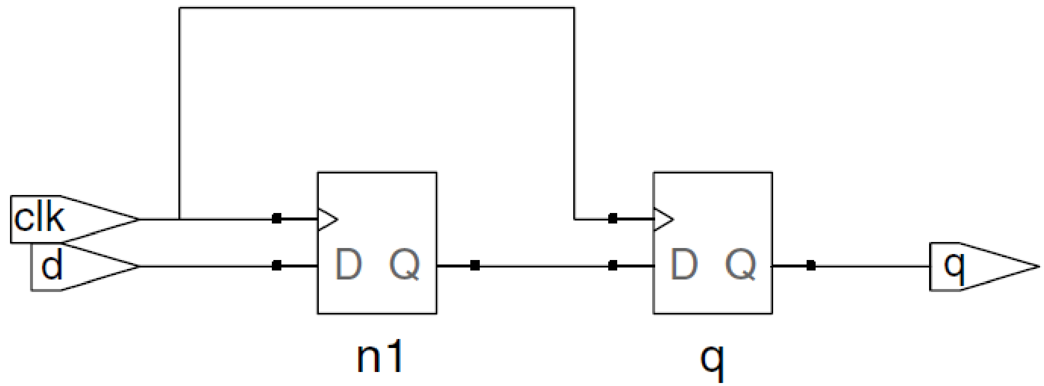
How many registers are necessary?

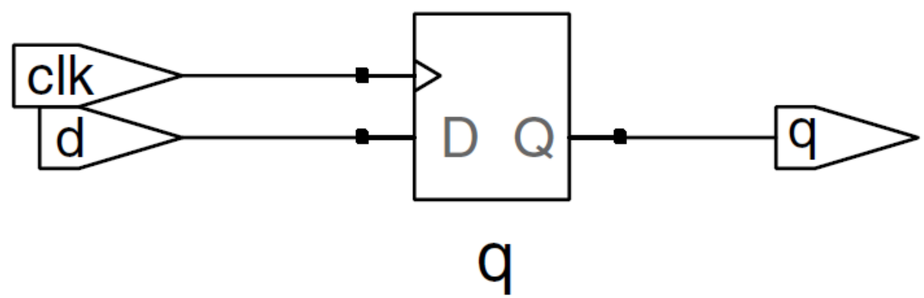
- *Depends on many design parameters (clock speed, device speeds, …)*
- *In the labs, a pair of synchronization registers is sufficient*

Blocking vs Non-blocking example

- Use a synchronizer to handle asynchronous inputs.

```
module sync(input      clk,
            input      d,
            output reg q);

  reg n1;

  always @ (posedge clk)
    begin
      n1 = d;
      q = n1;
    end
endmodule
```





Design Question

- Design an edge detector
- Inputs: clk, sig_in
- Output: edge_out
- When sig_in transitions high, edge_out is high for just 1 clock cycle