ENGR 210 / CSCI B441
"Digital Design"

SPI I

Test

Andrew Lukefahr

Announcements

- P8 – Elevator Controller is out
  - This one is hard. due Friday
- P9 – SPI is out → Last one ☺ ...
  - This one is new. Might be some changes.

until E315

- The end is in sight…

`for(;;)` Loops in Verilog

- Testbenches – just like C/C++
- Synthesizable modules - more like `#define`

Always specify defaults for **`always_comb`**!

BLOCKING (=) FOR

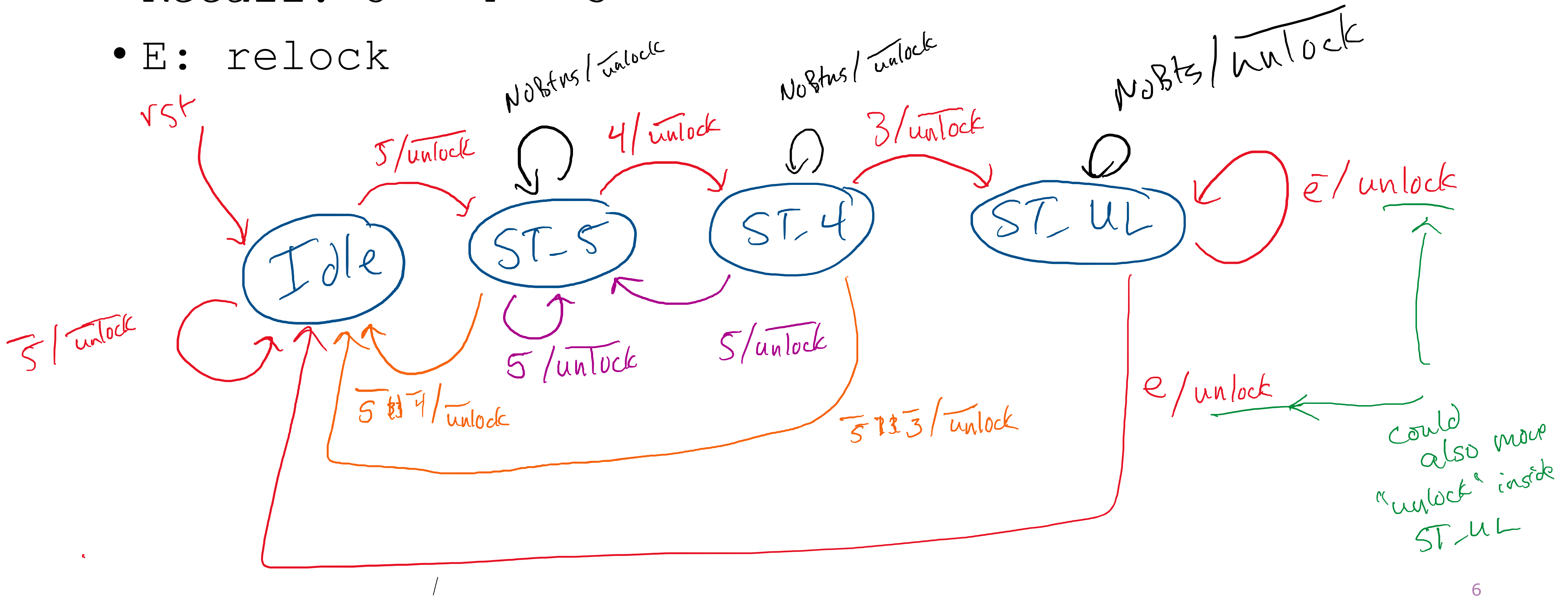
`always_comb`

NON-BLOCKING (<=) for

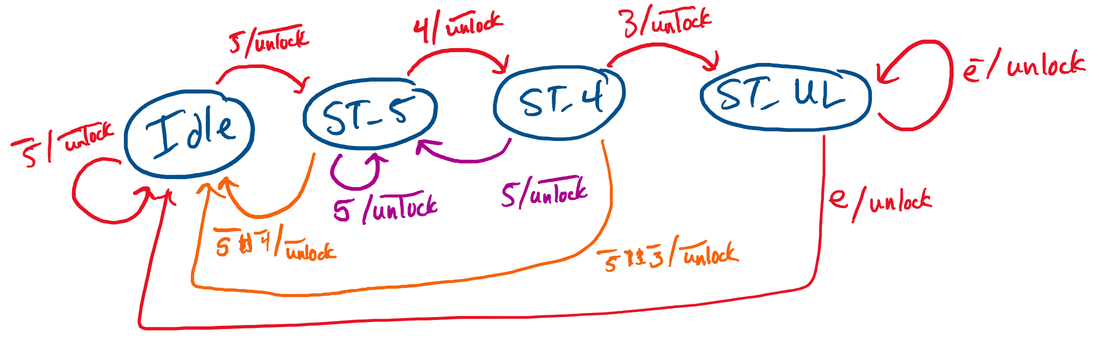
`always_ff`

Lock State Machine

- Recall: 5 - 4 - 3
- E: relock



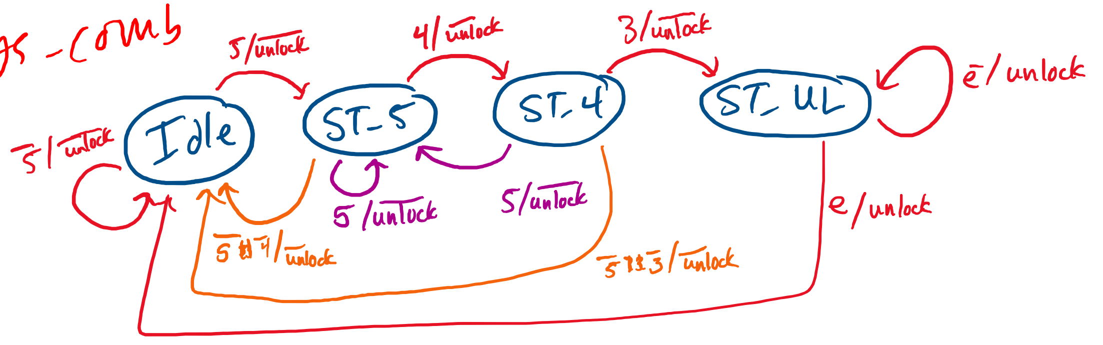
State Machine in Verilog



```
module Lock(
    input clk, rst,
    input [9:0] num,
    input e, //relock
    output unlock
);


enum {ST_IDLE, ST_5, ST_4, ST_UL } state, next_state;

//seq logic
always_ff @(posedge clk) begin
        if (rst) state <= ST_IDLE;
        else     state <= next_state;
end
```

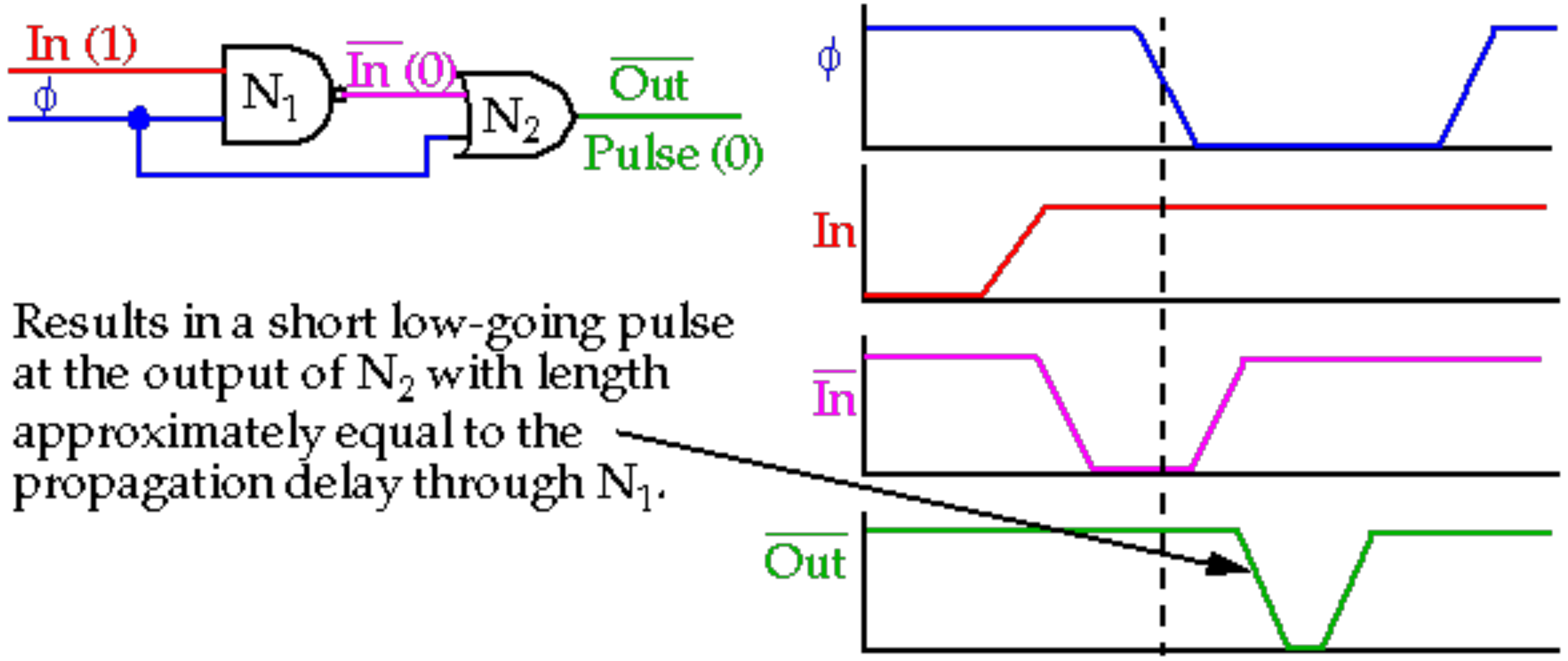
FixMe * → missing the default stuff of always-comb

State Machine in Verilog



```
case (state)
  ST_IDLE:
    if (num[5])
      next_state = ST_5;
  ST_5: begin
    if (num[4])
      next_state = ST_4;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e ) //other btns
      next_state = ST_IDLE;
  end
 ST_4: begin
    if (num[3])
      next_state = ST_UL;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e) // other btns
      next_state = ST_IDLE;
  end
  ST_UL: begin
    unlock = 1'h1;
    if (e)
      next_state = ST_IDLE;
  end
endcase
```

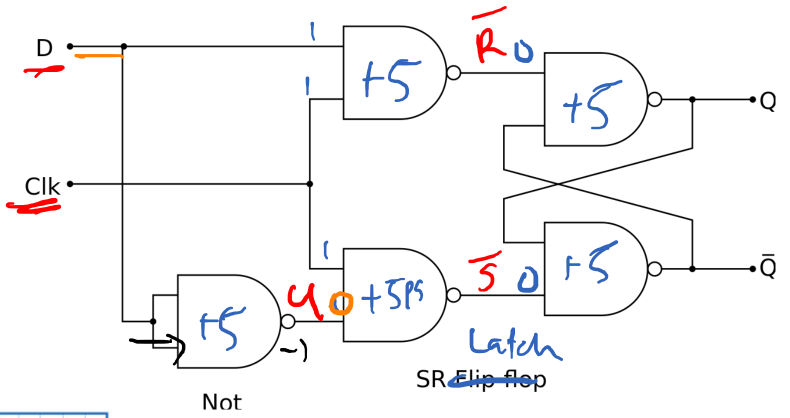
Glitch

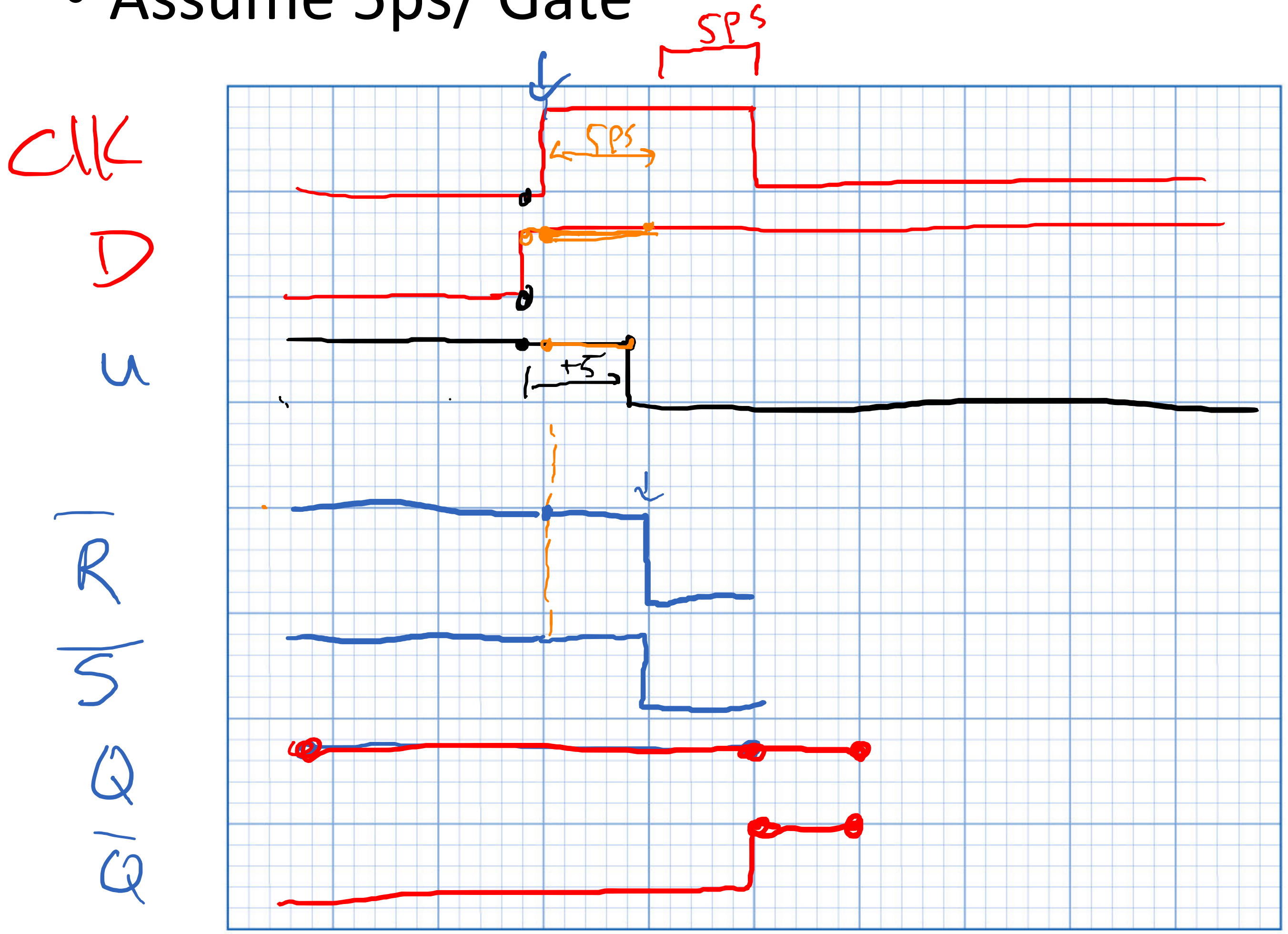


Results in a short low-going pulse at the output of N_2 with length approximately equal to the propagation delay through N_1.

Flip-Flop Timing

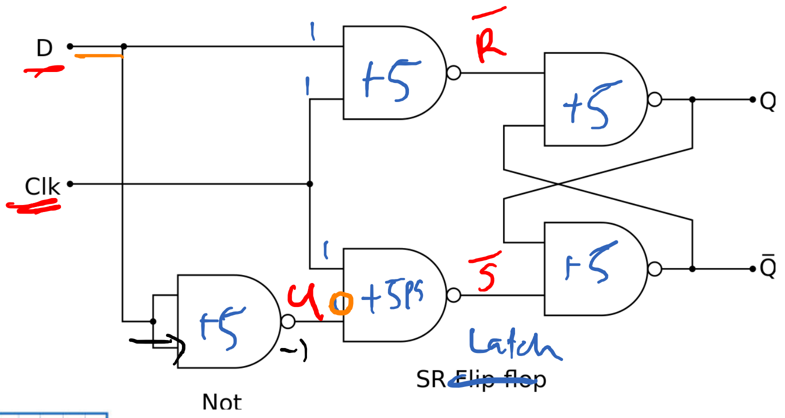
- Assume 5ps/ Gate

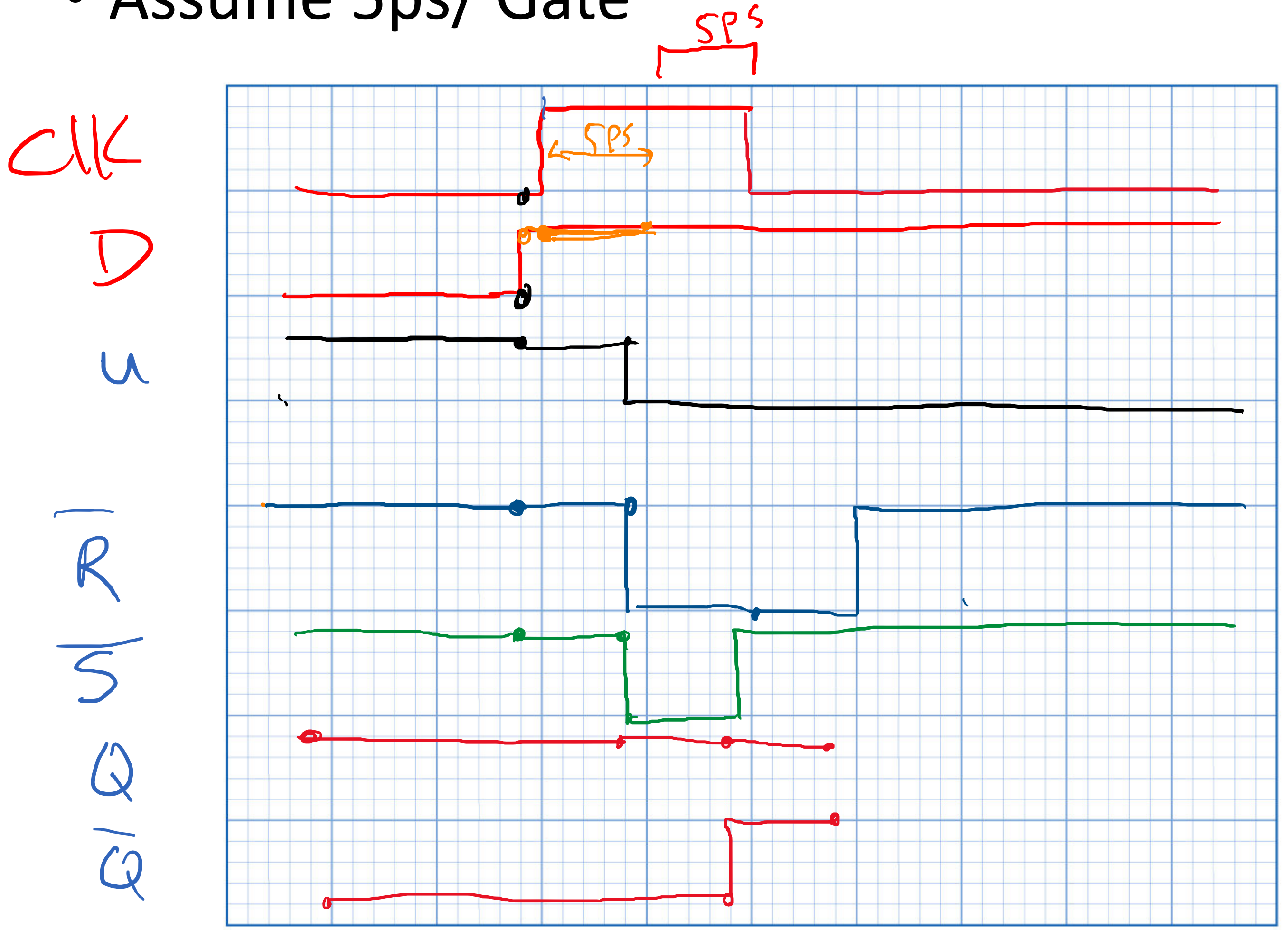




Flip-Flop Timing

- Assume 5ps/ Gate



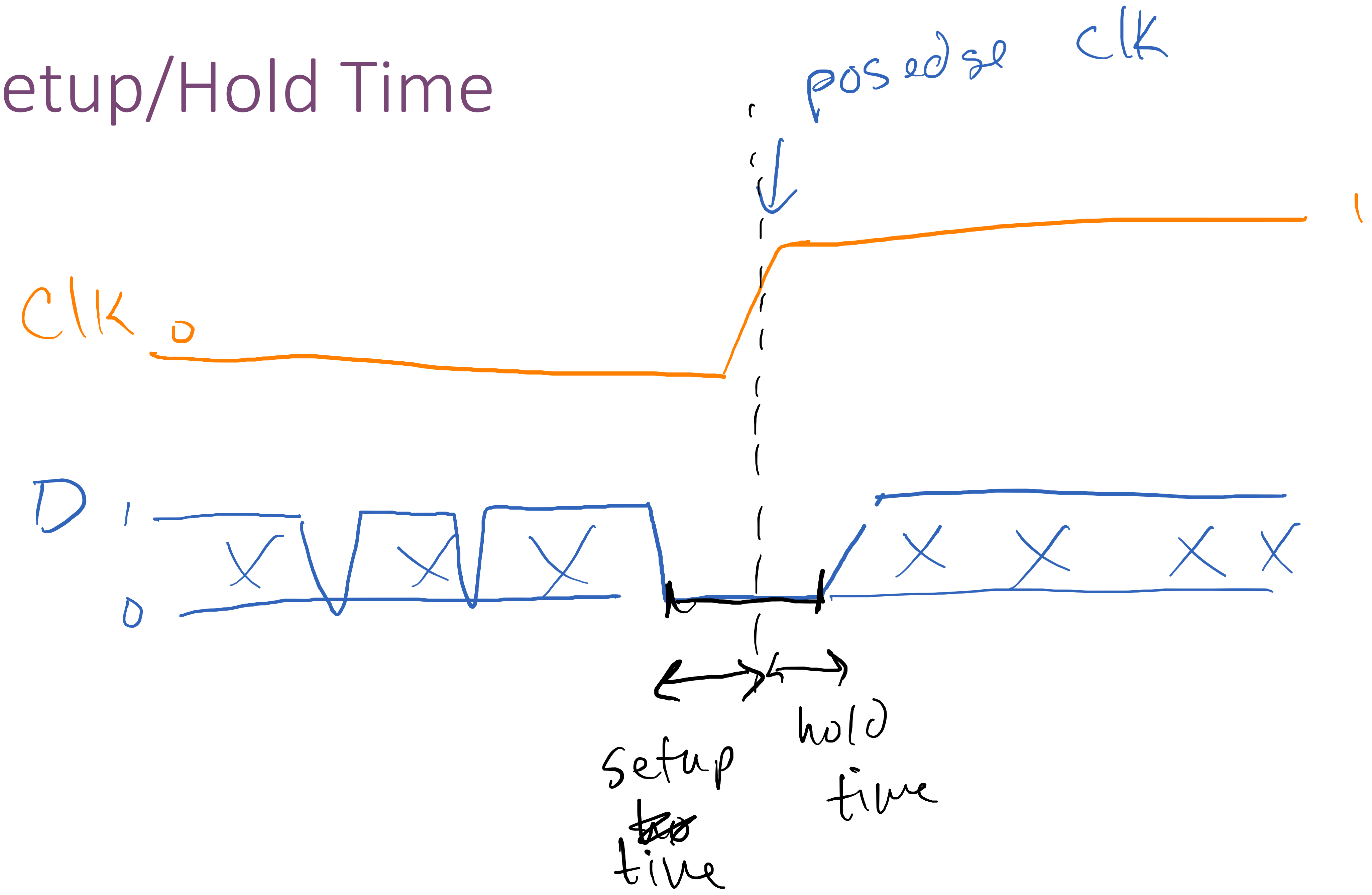


Setup and Hold Time

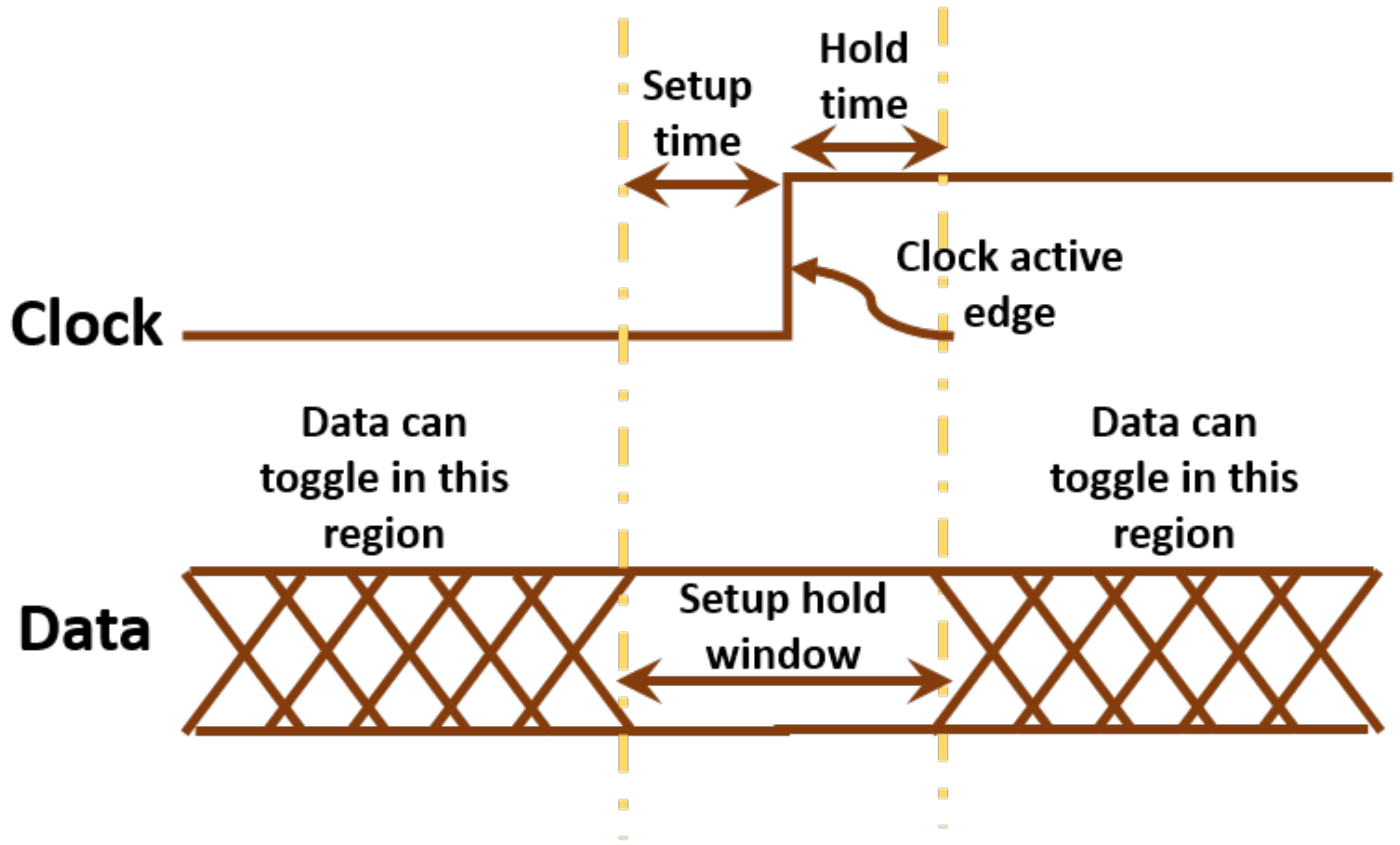
- **Setup Time**: minimum time the inputs to a flip-flop **must** be stable **before** the clock edge

- **Hold Time:** minimum time the inputs to a flip-flop **must** be stable **after** the clock edge

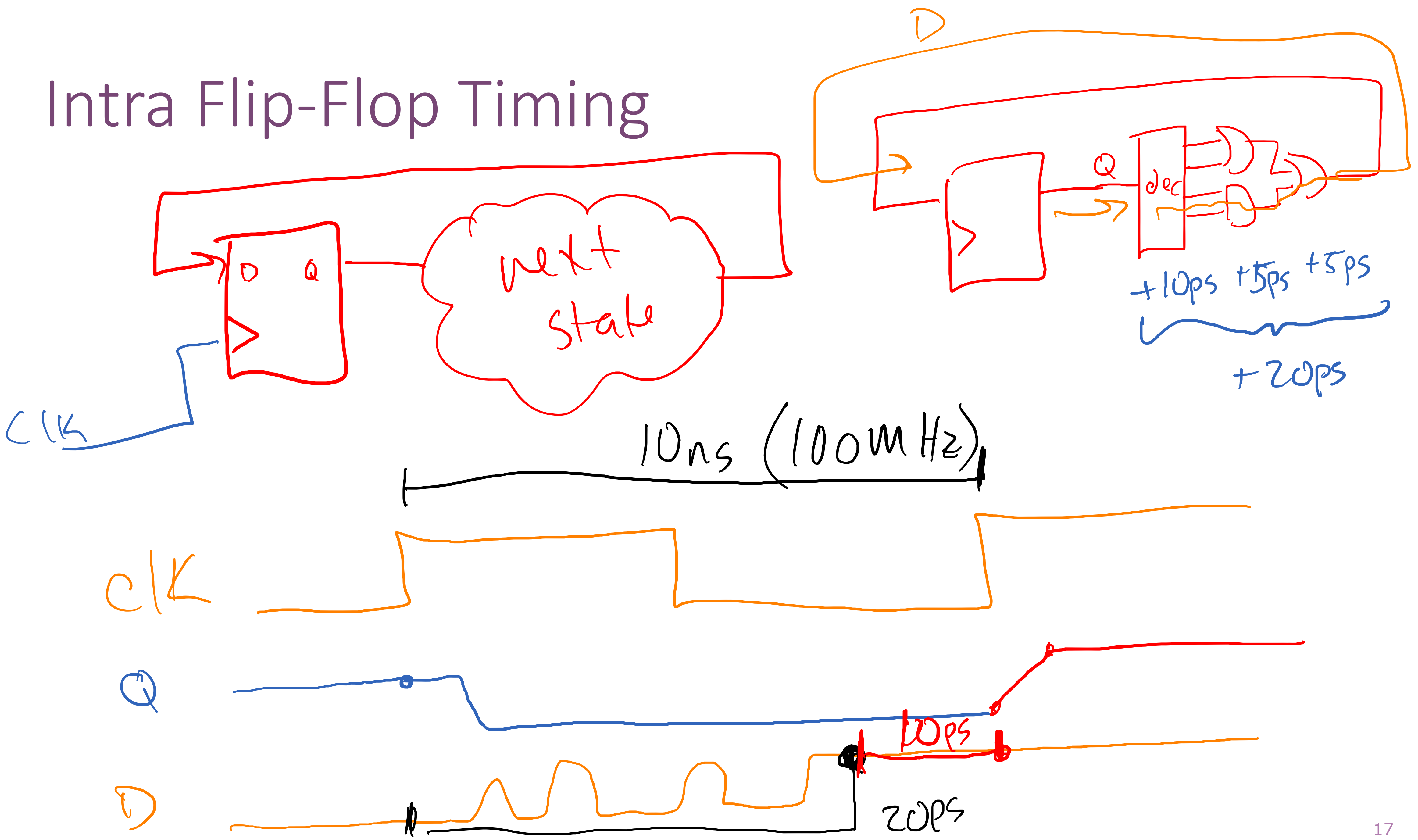
Setup/Hold Time



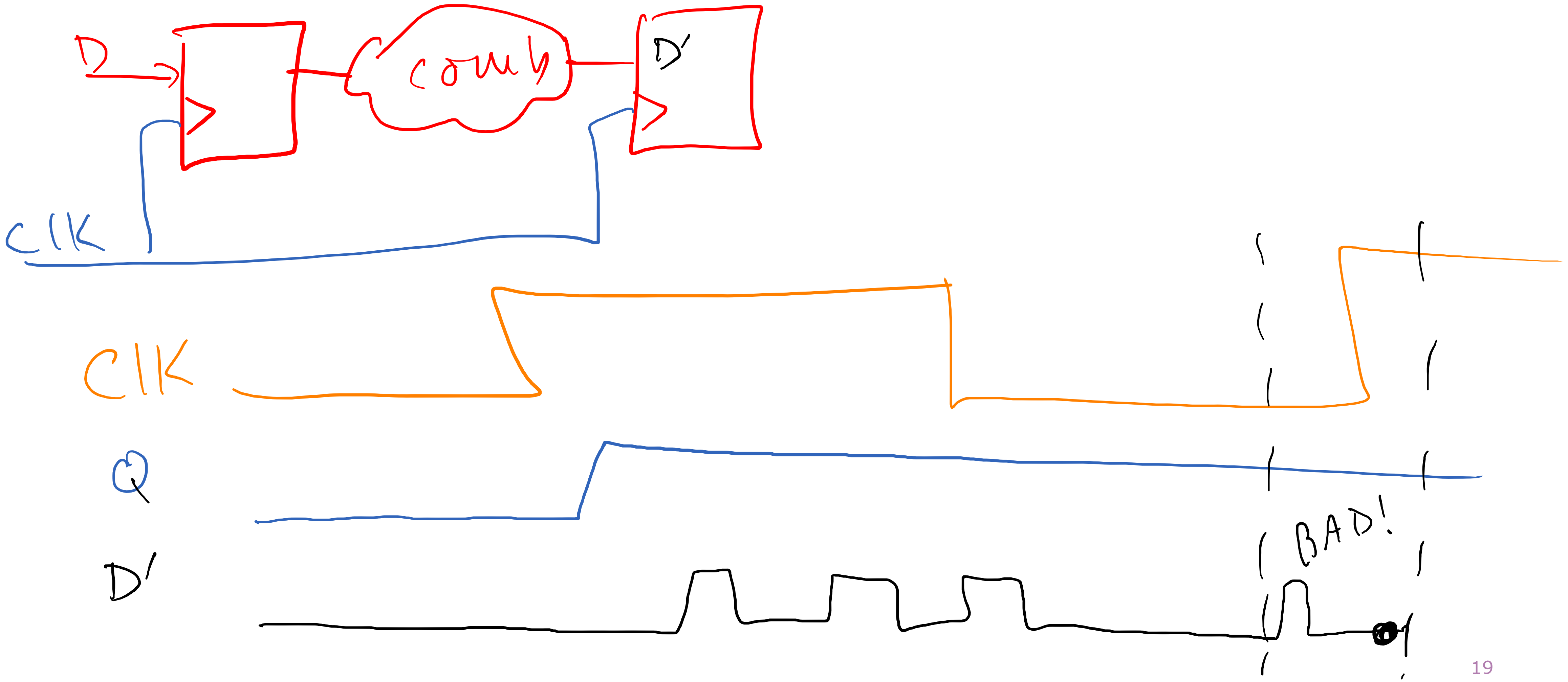
Setup/Hold Time



Intra Flip-Flop Timing



Inter Flip-Flop Timing



Intra Flip-Flop Timing

Register to register timing:

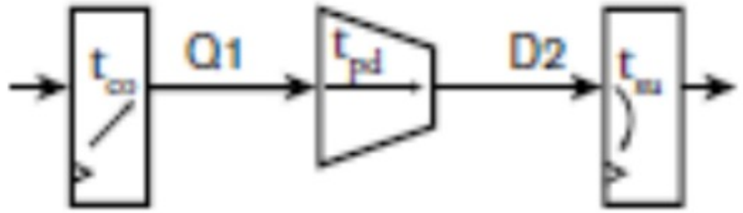
- output of a register *Q1*
- some combinational circuit
- input to the next register *D2*

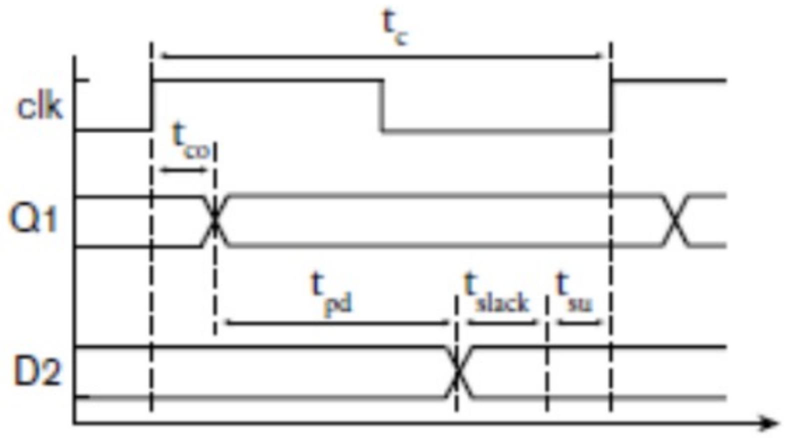
Delays:

- t_{co} , clock to output delay,
- t_{pd} , propagation delay in combinational circuit
- t_c , clock period

Timing requirement:

$$t_{co} + t_{pd} + t_{su} < t_c$$

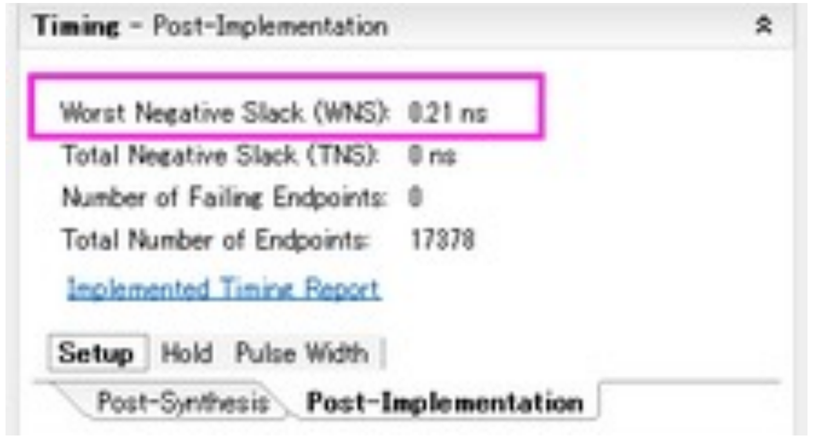




Slack

- Extra time between combinational propagation delay and setup time
- Time between stable input to Flip-Flop and next clock edge

- Vivado:
  - WNS: Worst-case Negative Slack


Timing - Post-Implementation

Worst Negative Slack (WNS): 0.21 ns
Total Negative Slack (TNS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 17378
Implemented Timing Report

- If this number is <0, your circuit will (probably) not work

`for(;;)` Loops in Verilog

- Testbenches – just like C/C++
- Synthesizable modules - more like `#define`

`for(;;)` Loops in Verilog

- Testbenches – just like C/C++
- Synthesizable modules - more like `#define`
- `for(;;)` loop is expanded at Compile (Synthesis) Time

Correct `for(;;)` Loops in Verilog

```
module MysteryFunction(
    input[3:0] a,
    input[1:0] right,
    output[3:0] y
);

//correct for loop
integer x; // OR genvar x
for (x = 0; x < 4; x++) begin
      assign y[x] = a[x+right];
end

endmodule
```

```
assign y[0] = a[right];
  "    y[1] = a[1+right];
       y[2] = a[2+right];
       y[3] = a[3+right];
```

Correct `for(;;)` Loops in Verilog

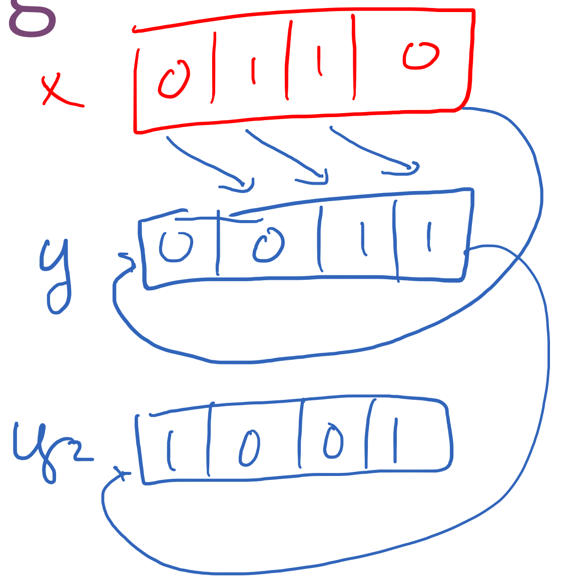
```
module RightBarrelShift(
    input[3:0] a,
    input[1:0] right,
    output[3:0] y
);

//correct for loop
integer x; // OR genvar x
for (x = 0; x < 4; x++) begin
        assign y[x] = a[x+right];
end

endmodule
```

x=1

assign y[0] = a[0+right]

assign y[1] = a[1+right]

...



Correct `for(;;)` Loops in Verilog

This Code:

```
integer x; //or genvar x
for (x = 0; x < 4; x++) begin
    assign y[x] = a[x+right];
end
```

Expands to:

```
assign y[0] = a[0 + right];
assign y[1] = a[1 + right];
assign y[2] = a[2 + right];
assign y[3] = a[3 + right];
```

Incorrect `for(;;)` Loops in Verilog

```
integer i;
wire carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry;
                assign carry = a[i] & b[i] | a[i] & carry | b[i] & carry;
        end
end
assign c_out = carry;
```

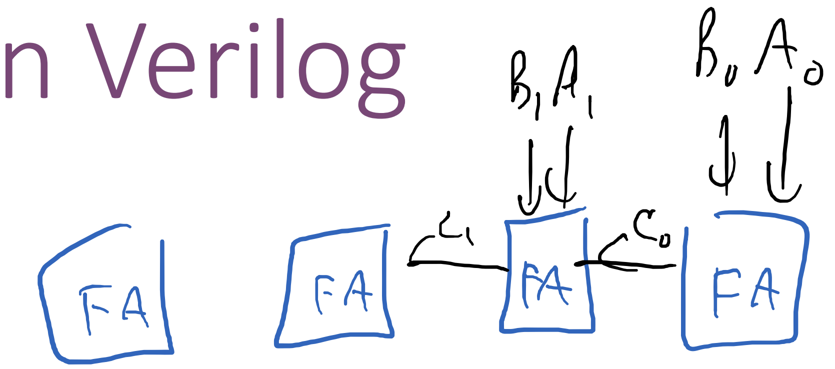
What is wrong here?

Incorrect `for(;;)` Loops in Verilog

```
integer i;
wire carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry;
                assign carry = a[i] & b[i] | a[i] & carry | b[i] & carry;
        end
end
assign c_out = carry;
```

What is wrong here?

Incorrect `for(;;)` Loops in Verilog



```
integer i;
wire carry[3:0]
assign sum[0] = a[0] ^ b[0] ^ c_in;
assign carry[0]= a[0] & b[0] | a[0] & c_in | b[0] & c_in;
assign sum[1] = a[1] ^ b[1] ^ carry[ ]
assign carry[1]= a[1] & b[1] | a[1] & carry | b[1] & carry;
assign sum[2] = a[2] ^ b[2] ^ carry[ ]
assign carry[2]= a[2] & b[2] | a[2] & carry | b[2] & carry;
assign sum[3] = a[3] ^ b[3] ^ carry[; ]
assign carry[3]= a[3] & b[3] | a[3] & carry | b[3] & carry;
assign c_out = carry;
```

Correcting `for(;;)` Loops in Verilog

```
integer i;
wire carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry;
                assign carry = a[i] & b[i] | a[i] & carry | b[i] & carry;
        end
end
assign c_out = carry;
```

How do we fix this?

Correcting `for(;;)` Loops in Verilog

```
integer i;
wire [3:0] carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry[i] = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry[i-1];
                assign carry[i] = a[i] & b[i] | a[i] & carry[i-1] | b[i] & carry[i-1];
        end
end
assign c_out = carry[3];
```

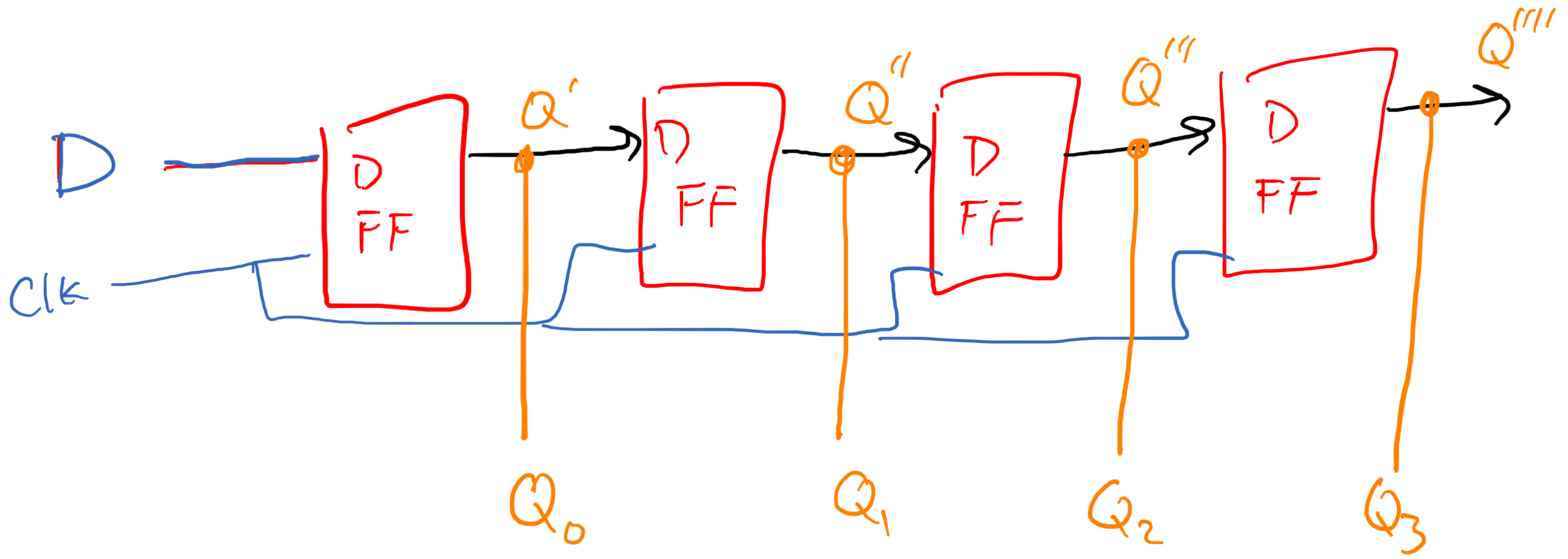
How do we fix this?

Correcting `for(;;)` Loops in Verilog

```
integer i;
wire carry;
assign sum[0] = a[0] ^ b[0] ^ c_in;
assign carry[0] = a[0] & b[0] | a[0] & c_in | b[0] & c_in;
assign sum[1] = a[1] ^ b[1] ^ carry[0];
assign carry[1] = a[1] & b[1] | a[1] & carry[0] | b[1] & carry[0];
assign sum[2] = a[2] ^ b[2] ^ carry[1];
assign carry[2] = a[2] & b[2] | a[2] & carry[1] | b[2] & carry[1];
assign sum[3] = a[3] ^ b[3] ^ carry[2];
assign carry[3] = a[3] & b[3] | a[3] & carry[2] | b[3] & carry[2];
assign c_out = carry[3];
```

Serial Communication

- Do you remember this circuit?

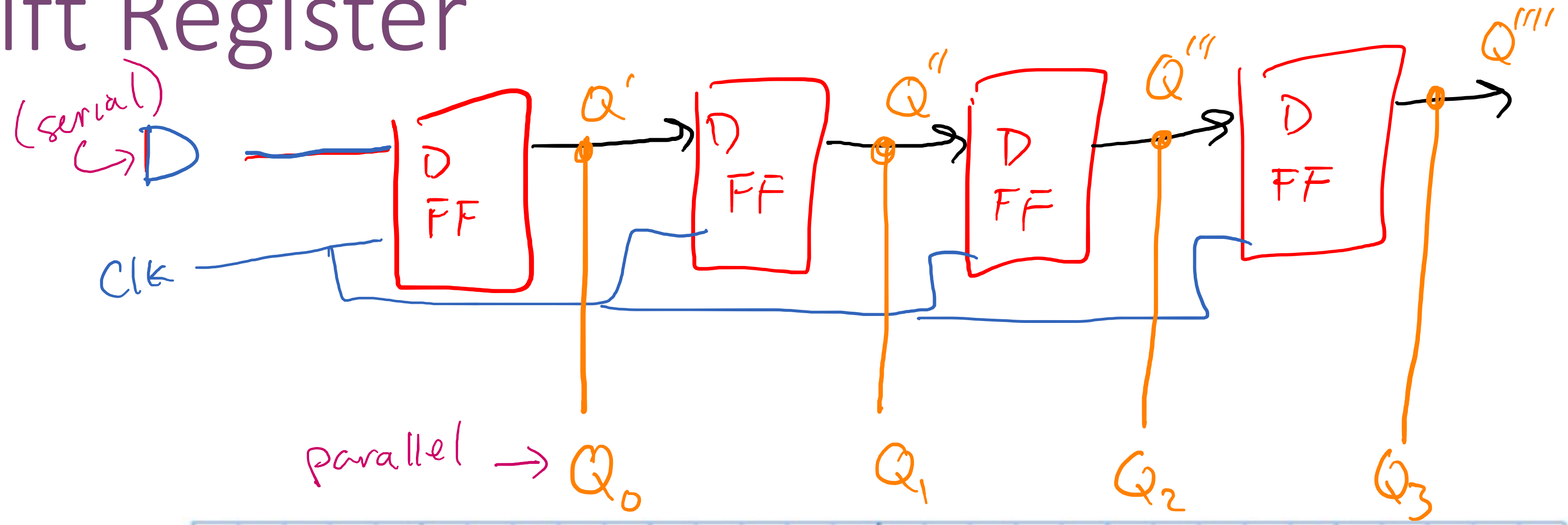


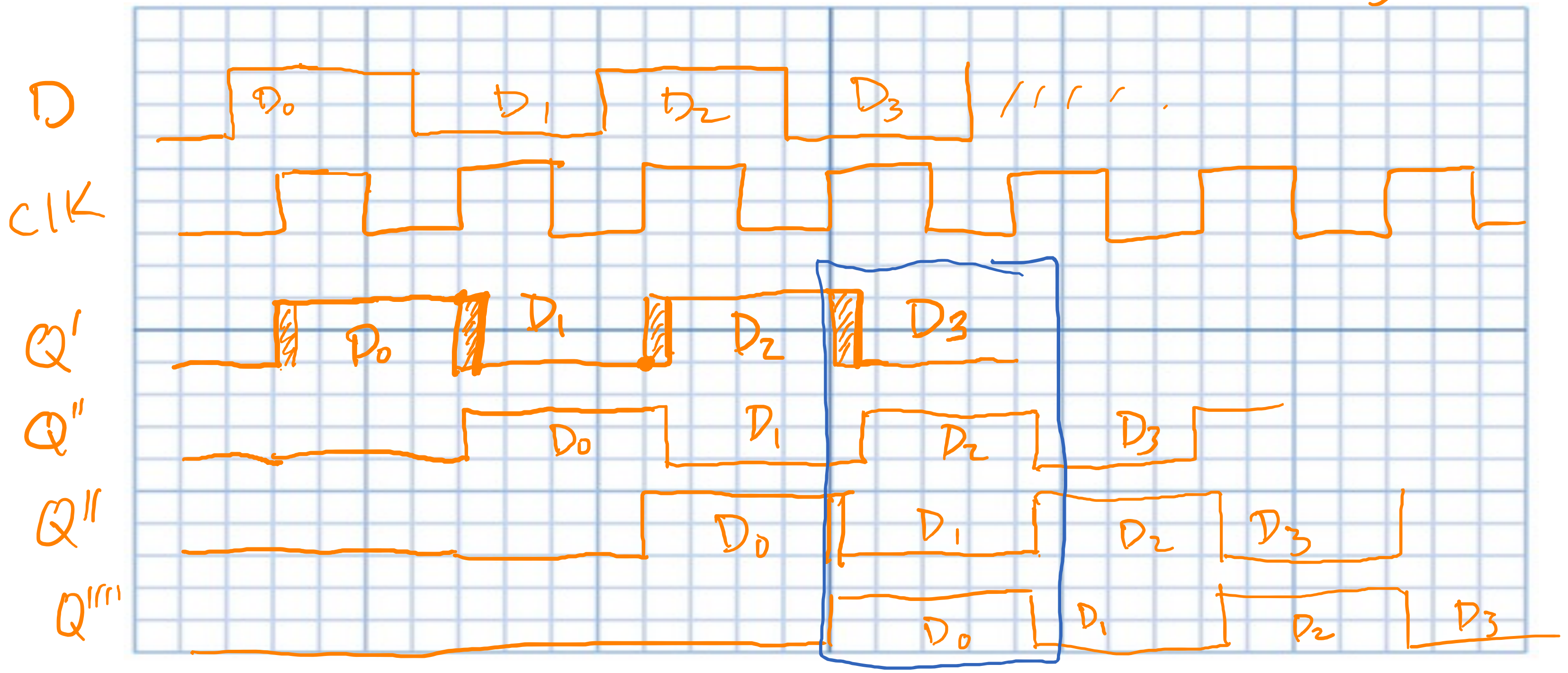
- What did it do?

shift register
shift in data serially & convert to parallel

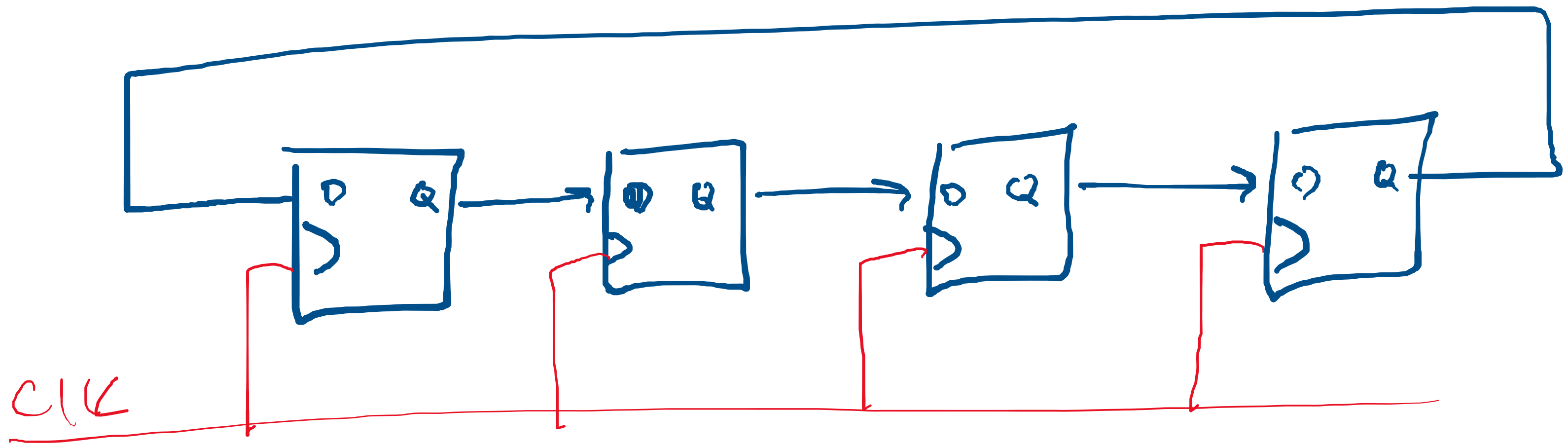
Converts serial input to parallel output

Shift Register



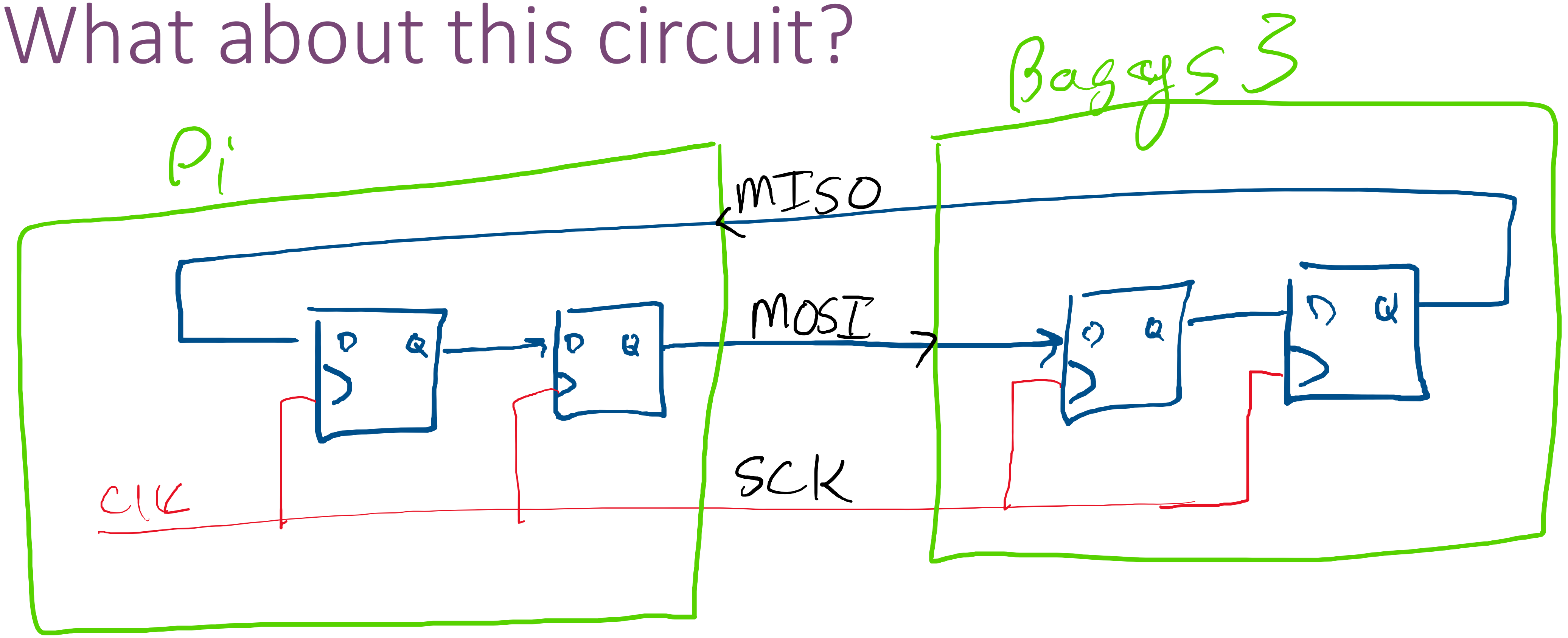


What does this circuit do?



circular shifter

What about this circuit?



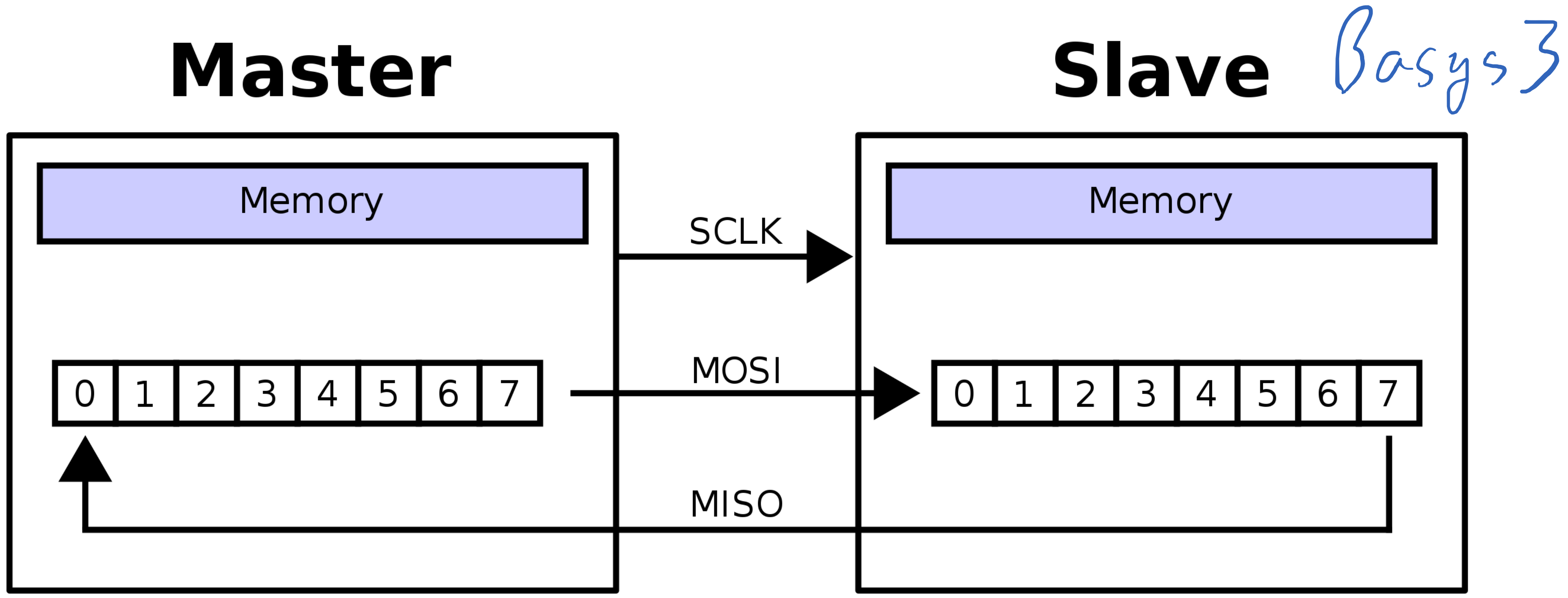
MOSI: Master Out Slave In
Serial Data Out (SDO)

MISO: Master In Slave Out
Serial Data In (SDI)

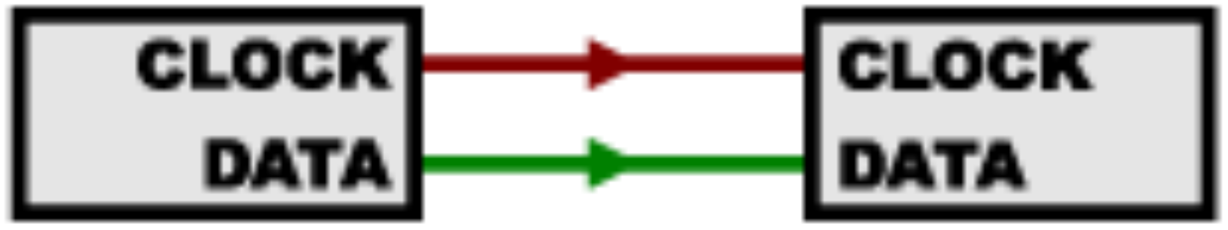
Serial Peripheral Interface (SPI)

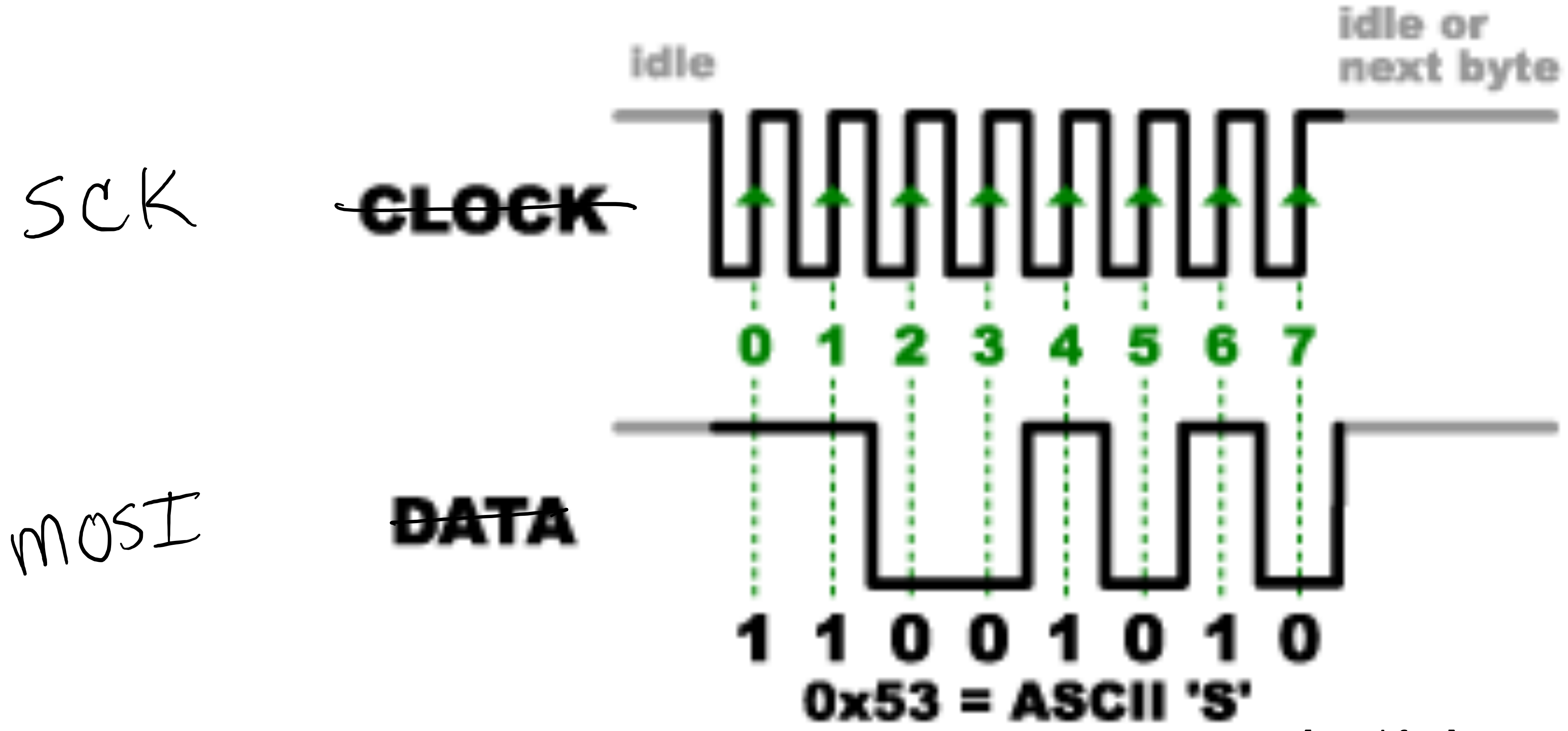
- The **Serial Peripheral Interface (SPI) is a** synchronous **serial communication interface** specification used for short-distance communication, primarily in embedded systems.
- SPI devices **communicate in full duplex** mode using a master-slave architecture. The master (controller) device **originates the frame for reading and writing**. Multiple slave-devices may be supported through selection with individual chip select (CS), sometimes called slave select (SS) lines.
- [Wiki]

SPI


Master
Slave
Basys3
Memory
Memory
SCLK
MOSI
MISO
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

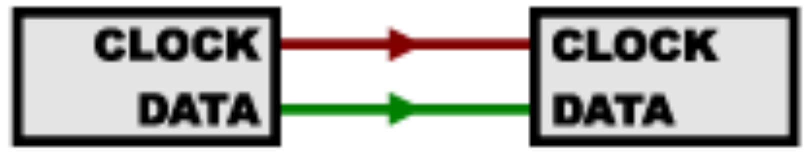
[wiki]


CLOCK
DATA
CLOCK
DATA

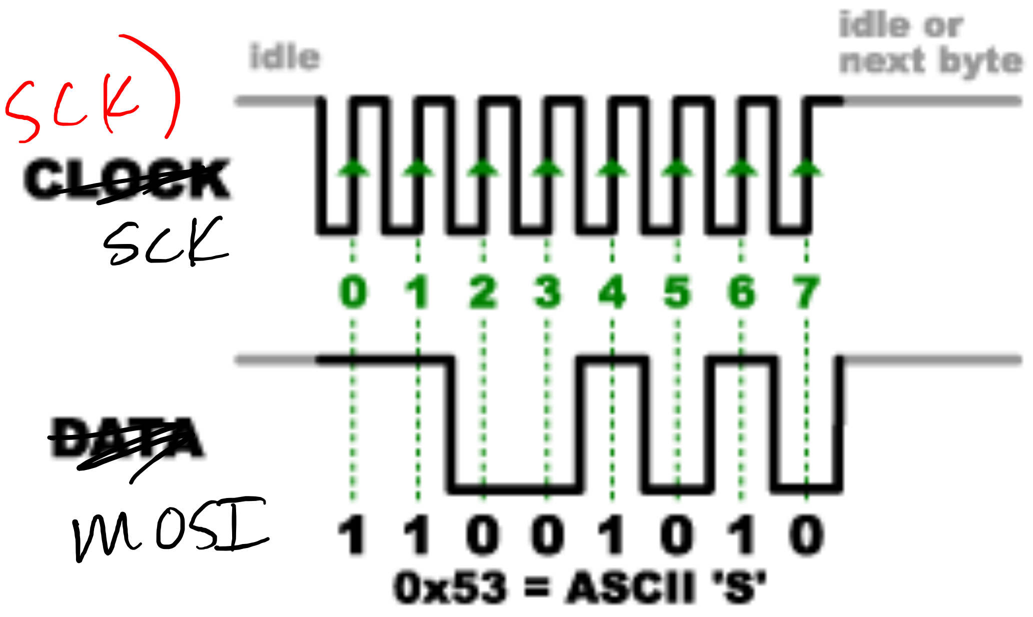

idle
idle or
next byte
SCK
~~CLOCK~~
0 1 2 3 4 5 6 7
MOSI
~~DATA~~
1 1 0 0 1 0 1 0
0x53 = ASCII 'S'

[sparkfun]

When to CAPTURE new data?

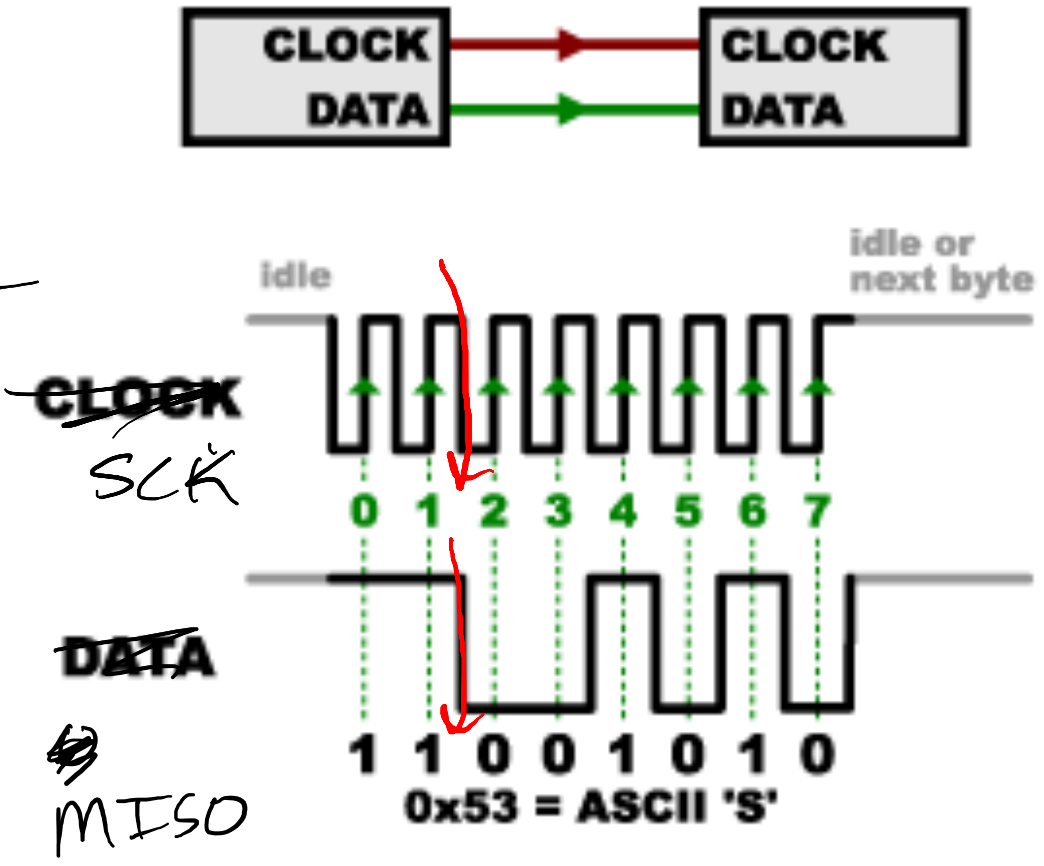


always_ff @(posedge SCK)
q <= MOSI

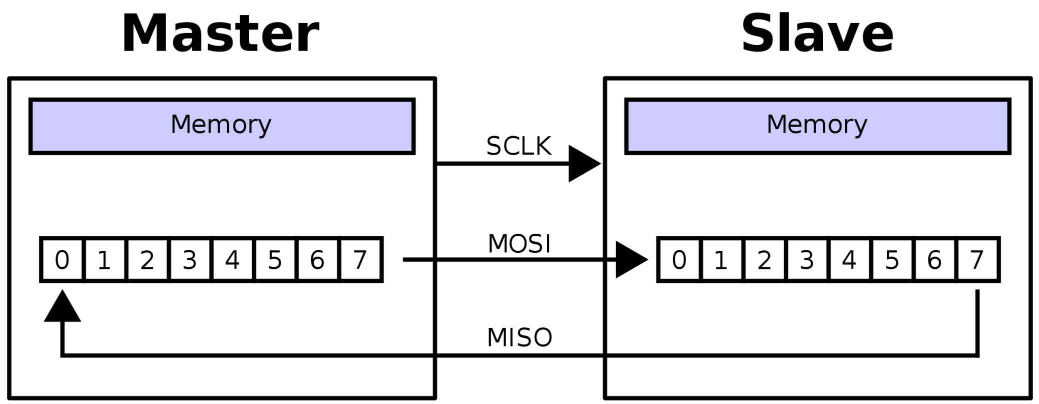


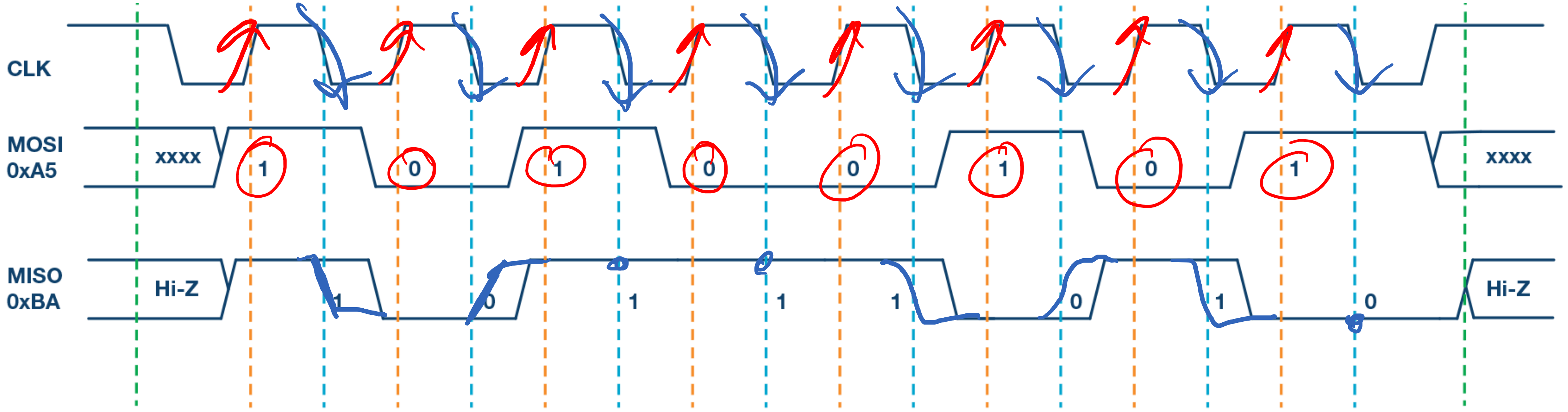
When to SEND new data?

@ falling edge of SCK



SPI Example





[analogDevices]

Next Time

- clk vs CLOCK 100MHz vs. SC~~K~~ CLK
- Synchronizers