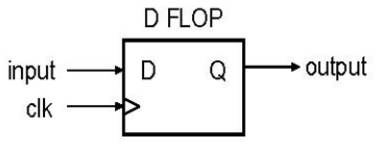
Metastability

And why you should care about it

What is a metastable state?

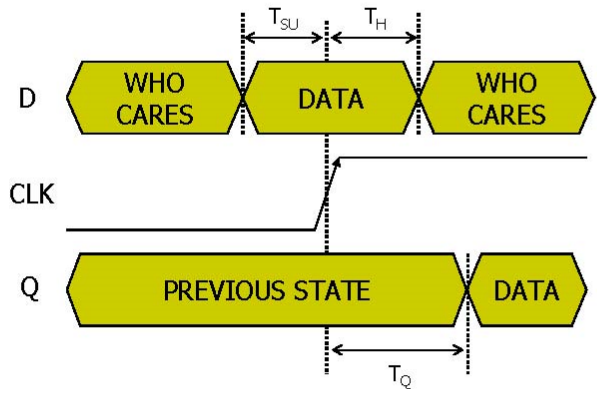
- Consider a D Flip-flop
  - Its output Q is either a 1 or a 0
  - Q gets the value of the input D on the active edge of the clock



  - However, input data must be stable a short time prior to and a short time after the active clock edge
    - Times are known as setup time t_{su} and hold time t_h

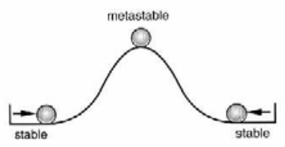
What is a metastable state?

- Predictable behavior of a flip flop is guaranteed if input data is stable for setup and hold time

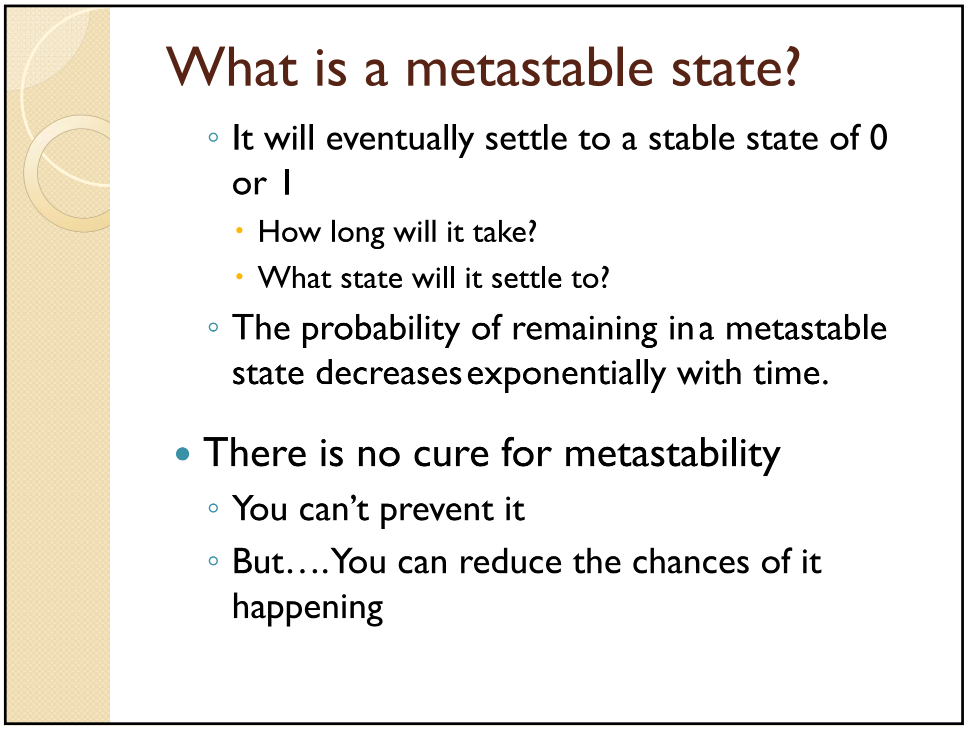


What is a metastable state?

- If data changes within setup and hold time window, the output of the flip flop cannot be predicted
  - It may go to 1 or 0
  - Or it may go metastable
- The metastable state is halfway between 1 and 0
  - Considered unstable equilibrium
  - Like a ball on a hill



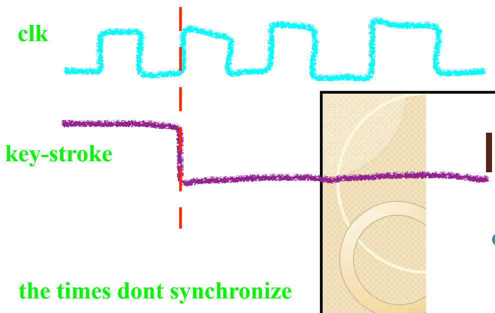
its like a temporary state that the system has no idea how to deal with

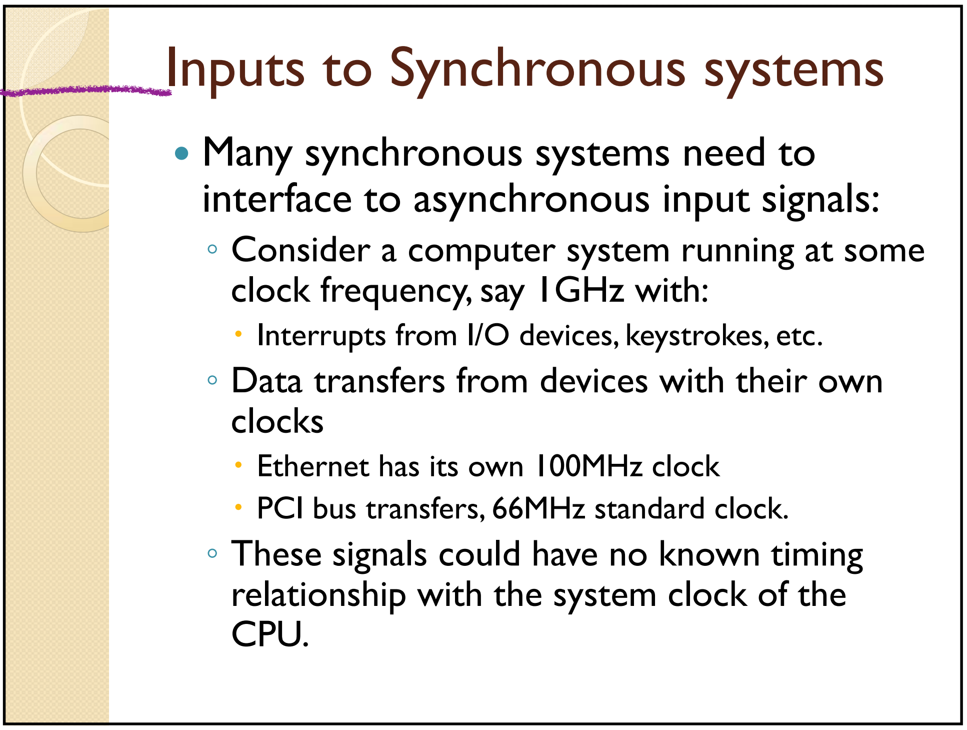


What is a metastable state?

- It will eventually settle to a stable state of 0 or 1
  - How long will it take?
  - What state will it settle to?
- The probability of remaining in a metastable state decreases exponentially with time.

- There is no cure for metastability
  - You can't prevent it
  - But….You can reduce the chances of it happening





Inputs to Synchronous systems

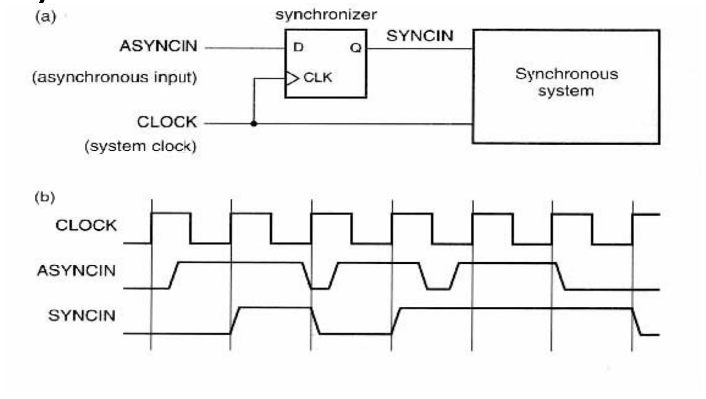
- Many synchronous systems need to interface to asynchronous input signals:
  - Consider a computer system running at some clock frequency, say 1GHz with:
    - Interrupts from I/O devices, keystrokes, etc.
  - Data transfers from devices with their own clocks
    - Ethernet has its own 100MHz clock
    - PCI bus transfers, 66MHz standard clock.
  - These signals could have no known timing relationship with the system clock of the CPU.

Inputs to synchronous systems

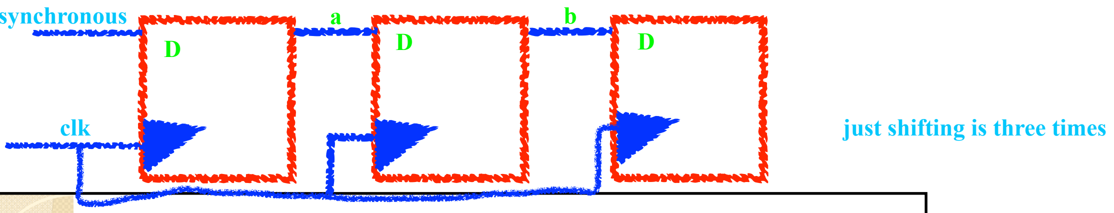
- If the input is not synchronous with the system's clock the potential exists to violate setup or hold times
  - May result in a metastable state
  - System may not work as expected
- If we can't prevent metastability how do we reduce the chance of it occuring?

Synchronizing Circuit

- For a single asynchronous input, use a simple flip-flop to bring the external input signal into the timing domain of the system clock:

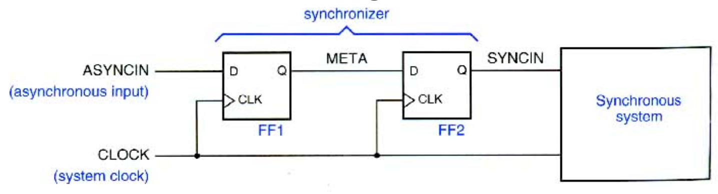


reccomended , syncronise for DSD students



Reliable Synchronizer Design

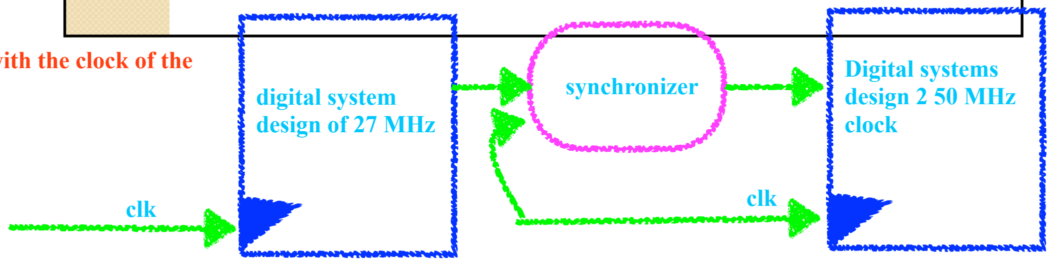
- The probability that a flip-flop stays in the metastable state decreases exponentially with time
  - Therefore, any scheme that delays using the signal can be used to decrease the probability of failure
  - Recommended design



Clock Domains

- Why would you have multiple clock domains?
  - Independent (sub)systems with different reference clocks, needing to share/exchange information.
  - Impractical to distribute or use a reference clock.

Always synchronize with the clock of the recieving system


digital system design of 27 MHz
synchronizer
Digital systems design 2 50 MHz clock
clk
clk

```
Synch: Process (clk, reset_n) IS
    BEGIN
  IF( reset_n = '0') THEN
          a <= '0';
          b <= '0';
          synch <= '0';
ELSIF (clk'EVENT AND clk = '1') THEN
          q <= Asynch;
          b <= a;
          synch <= b;
END IF;
```

Crossing Clock Domains

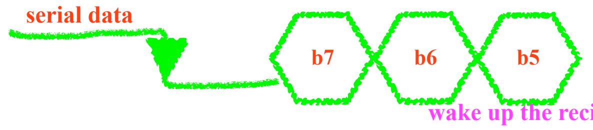
- For asynchronous clock domain relationships:
  - For a single signal, use the same two flip-flop synchronizer used for asynchronous inputs
- Think about…
  - What would the VHDL look like for a circuit to synchronize a signal coming from one clock domain into another clock domain?

Synchronizer in Lab 11

- The receive_data signal will be coming from the transmitting DE0 board.
  - Even though they are both on a 50Mhz clock, there is no guarantee that they are in sync
  - The incoming data must be synchronized to the receivers clock.
  - This is done with synchronizing flip-flops

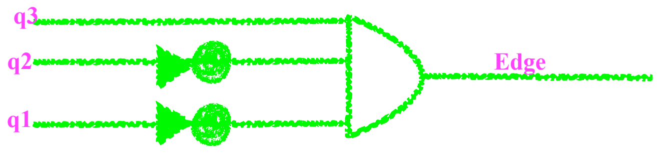
we need a falling edge detector

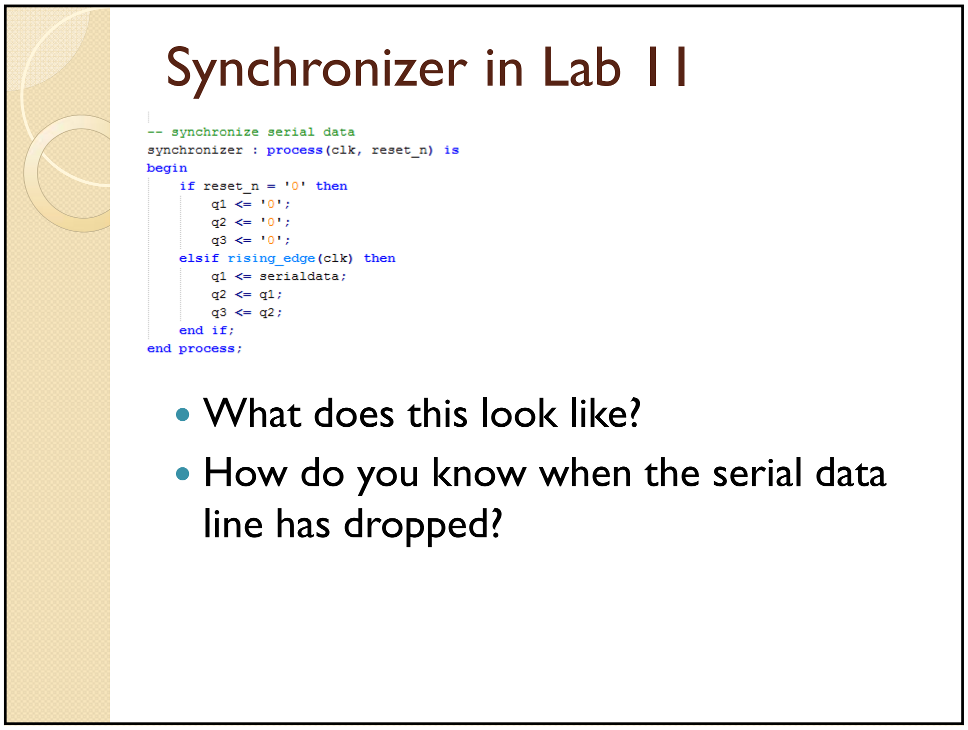
Lab 11 Data



wake up the reciver, BUT needs to be synchonized

IF (q3 = '1' AND q2='0' AND q1='0') THEN





Synchronizer in Lab 11

```
-- synchronize serial data
synchronizer : process(clk, reset_n) is
begin
    if reset_n = '0' then
        q1 <= '0';
        q2 <= '0';
        q3 <= '0';
    elsif rising_edge(clk) then
        q1 <= serialdata;
        q2 <= q1;
        q3 <= q2;
    end if;
end process;
```

- What does this look like?
- How do you know when the serial data line has dropped?