# Digital Logic Design - Vaibbhav

Group 8

# Chap15 - Non-synthesizable
# Chap 22 - Multiple Clock Domain

# Chapter 15 Non-synthesizable Verilog Constructs and Testbenches

The chapter discusses about the inter-delay, intra-delay assignments and other non-synthesizable constructs useful during the testbenches.

1. For Simulation
2. Inter-delay & Intra-delay
   a. Blocking Assignments with Inter-assignment Delays
   b. Blocking Assignments with Intra-assignment Delays
   c. Non-blocking Assignments with Inter-assignment Delays
   d. Non-blocking Assignments with Intra-assignment Delays
3. Role of Testbenches
4. Multiple Assignments Within the begin–end
5. Multiple Assignments Within the fork–join
6. Display Tasks

# Simulation - The always and initial Procedural Block

**Table 1** Difference between initial and always block

| Initial | Always |
|---|---|
| In this block assignment executes in the 0-simulation time and continues for the next specified sequence | In this block assignments continues to execute in simulation time 0 and repeats forever depending on the sensitivity list event |
| This block is executed only once, and the simulation stops at the end of this block | The simulation in this block continues forever. If wait construct is there, then it will be held during simulation session |
| It is non-synthesizable construct | It is synthesizable construct |

# Blocking Assignments & Non-blocking Assignments

```
////////////////////////////////////////////////////////////////////////////////

module test_design;
reg clk;
reg [7:0] a,b,c,d;
always #10 clk = ~clk;
always@(posedge clk)
begin
    a <=b+c;
    b <=a+d;
    c <=a+b;
end

initial
begin
    clk=0;
    a=8'h2;
    b=8'h3;
    c=8'h4;
    d=8'h5;
end
endmodule
////////////////////////////////////////////////////////////////////////////////
```

# Inter-delay & Intra-delay

Useful during the testbenches

# Blocking Assignments with Inter-assignment Delays

```
always@(posedge clk)
begin
  b=a+a;
#3 c=b+a;
#1 d=c+a;
end
```

```
initial
begin
    clk =0;
    a=4;
    b=3;
    c=2;
   d=5;
end
```


**Waveform 3** Simulation result for the Verilog blocking assignment with inter-assignment delay

# Blocking Assignments with Intra-assignment Delays

```
always@(posedge clk)
begin
  b=a+a;
 c= #3  b+a;
d= #1 c+a;
end
```

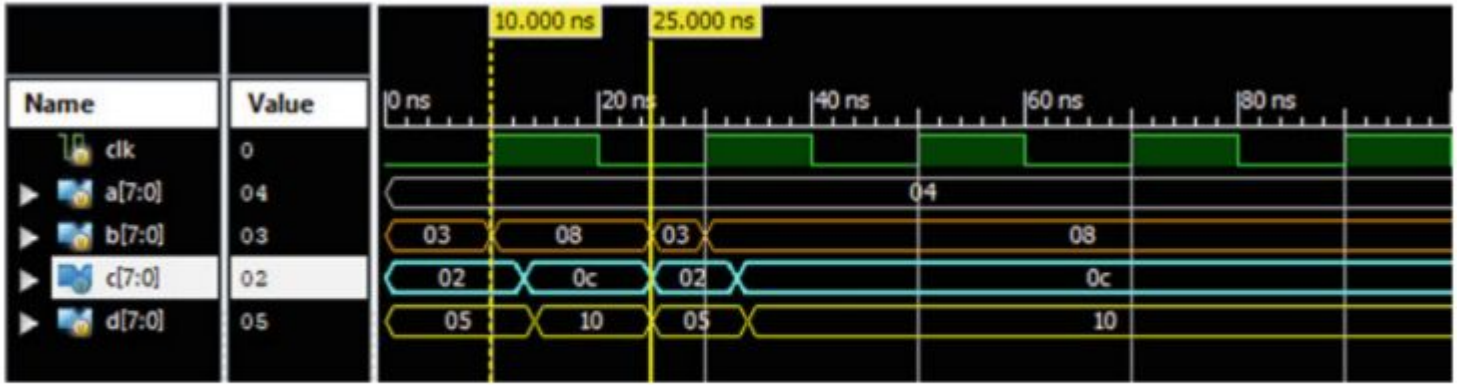
```
initial
begin
    clk =0;
    a=4;
    b=3;
    c=2;
   d=5;
end
endmodule
```


**Waveform 4** Simulation result for the Verilog blocking assignment with intra-assignment delay

# Non-blocking Assignments with Inter-assignment Delays

```
initial
begin
    clk =0;
    a=4;
    b=3;
    c=2;
    d=5;
#25
    a=4;
    b=3;
    c=2;
    d=5;
end
endmodule
```

```
always@(posedge clk)
begin
  b <=a+a;
#3 c <= b+a;
#1 d <= c+a;
end
```




**Waveform 5** Simulation result for the Verilog non-blocking assignment with inter-assignment delay

# Non-blocking Assignments with Intra-assignment Delays

```
initial
begin
    clk =0;
    a=4;
    b=3;
    c=2;
    d=5;
#25
    a=4;
    b=3;
    c=2;
    d=5;
end
```

```
always@(posedge clk)
begin
  b <=a+a;
  c  <= #3  b+a;
  d  <= #1 c+a;
end
```


**Waveform 6** Simulation result for the Verilog non-blocking assignment with intra-assignment delay

# Role of Testbenches

To test the functional correctness

Ex: Compare behavior model with RTL code

# Multiple Assignments Within the begin–end

All the blocking assignments within begin..end are executed sequentially.

```
always #10 clk=~clk;

initial
begin
  clk=0;
  a = 0;
  b = 0;
end
```

```
always @(posedge clk)
begin
  #30 a = 1;
  #20 b = 1;
end
```


**Fig. 15.1** Simulation result for Example 9

# Multiple Assignments Within the fork–join

```
module fork_join;
reg clk;
reg a;
reg b ;
always #10 clk=~clk;
 initial
 begin
    clk=0;
    a = 0;
    b = 0;
 end
```

```
 always @(posedge clk)
fork
    #30 a = 1;
    #20 b = 1;
 join
endmodule
```


**Fig. 15.2** Simulation result of Example 10

# What if……

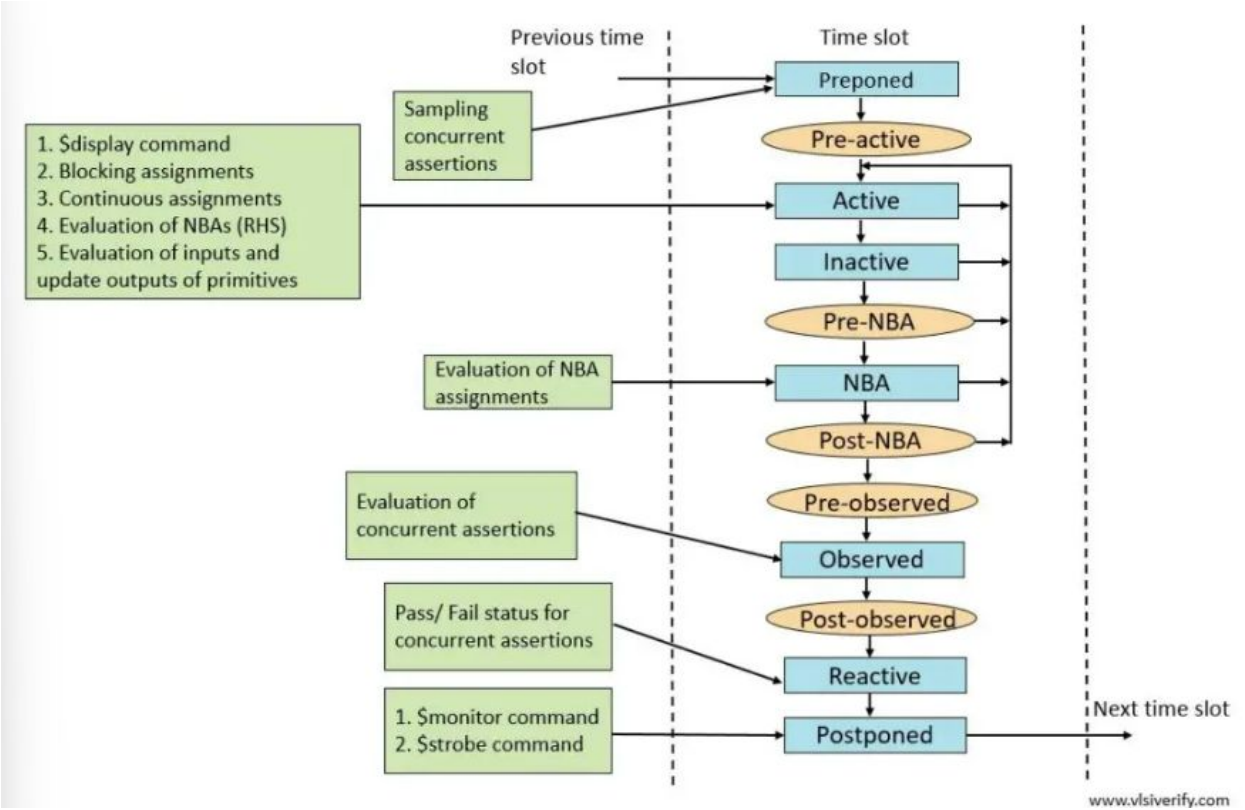
```
module test;

  reg clk;
  reg reset;

  reg a;
  reg b;


  initial begin
        a= 0;b=1;
    clk = 0;
    reset = 1;
    forever #5 clk = ~clk;


  end
```

```
  initial $monitor("a=%b, b=%b",a,b);

  always @(posedge clk) begin
    fork
    a = b;
    b = a;
      join
  end


endmodule
```

# Display Tasks

```
a<=0; b<=0;

c<=0; d<=0;

# 10 a<=1;

       b<=1;

$display (" time=%0t a=%0b b=%0b c=%0b y=%0b", $time, a, b, c, y);

$strobe (" tme=%0t a=%0b b=%0b c=%0b y=%0b", $time, a, b, c, y);

#10 y <= a & b ^ c;

$monitor (" Θme=%0t a=%0b b=%0b c=%0b y=%0b", $time, a, b, c, y);
```

# Display Tasks



# Display Tasks

Using the waveforms is difficult and time consuming to check for the functional correctness


**Fig. 15.3** Simulation result of Example 11

The ***$display*** is used to print the value immediately.
The ***$strobe*** is used to print the value at the current time stamp.
The ***$monitor*** to display the changes in the values at the end of every time stamp.

```
time=10000 a=0 b=0 c=0 y=0
time=10000 a=1 b=0 c=1 y=0
time=20000 a=1 b=0 c=1 y=1
```

# Chapter 22 Multiple Clock Domain Design

1. What Is Multiple Clock Domain?
2. What Is Clock Domain Crossing (CDC)
3. Metastability
4. Solutions
    a. Level Synchronizers
    b. Pulse Stretcher
    c. Handshaking Mechanism + Pulse Synchronizers

# What Is Multiple Clock Domain?

The clock sources CLK1 and CLK2 are derived from different clock source or may have the same or different frequency.

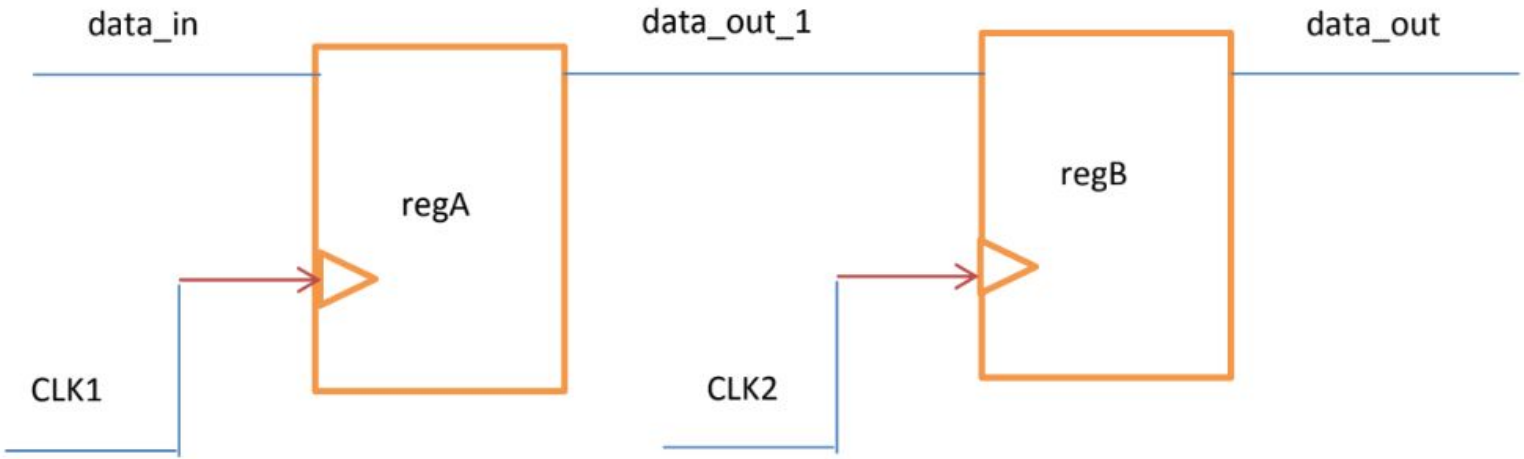



**Fig. 22.1** Multiple clock domain logic

# What Is Clock Domain Crossing (CDC)

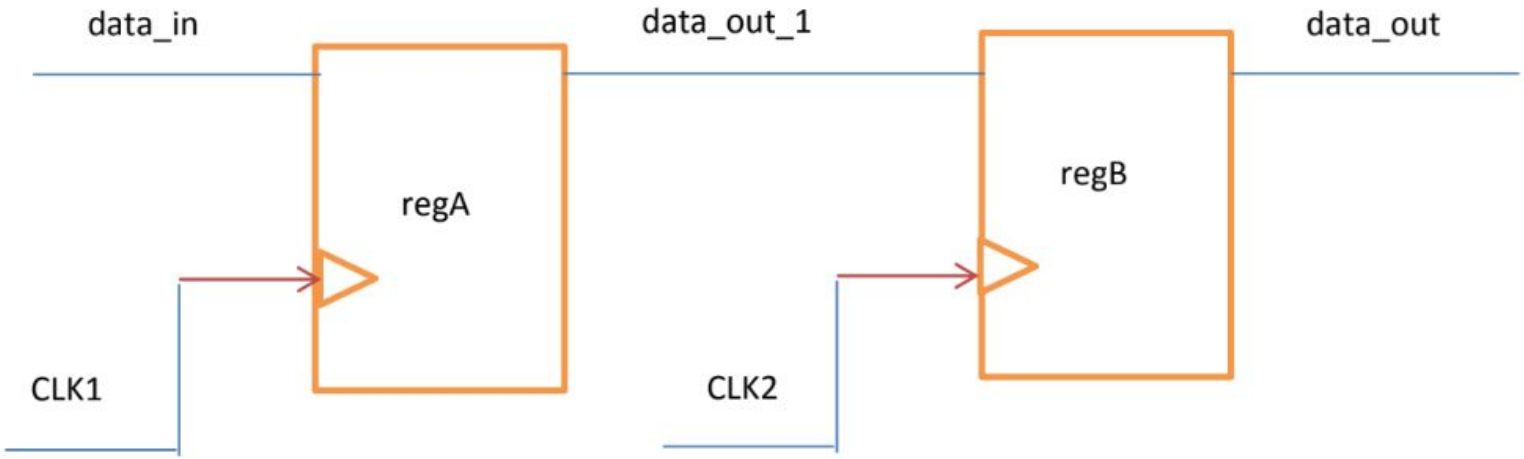
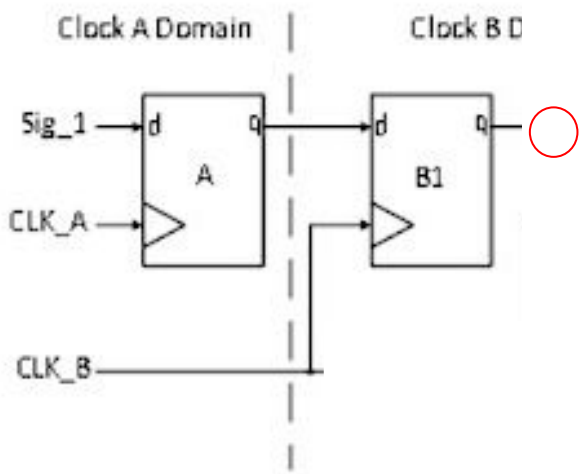
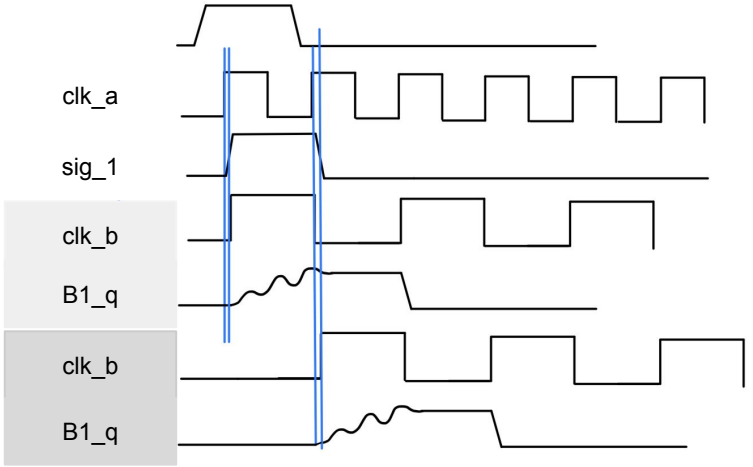



**Fig. 22.1** Multiple clock domain logic

# Issue - Metastability

Metastability is the scenario in the design due to occurrences of multiple events close to each other, and the design has setup and hold time violations.




https://www.edn.com/synchronizer-techniques-for-multi-clock-domain-socs-fpgas/

# The Level Synchronizers

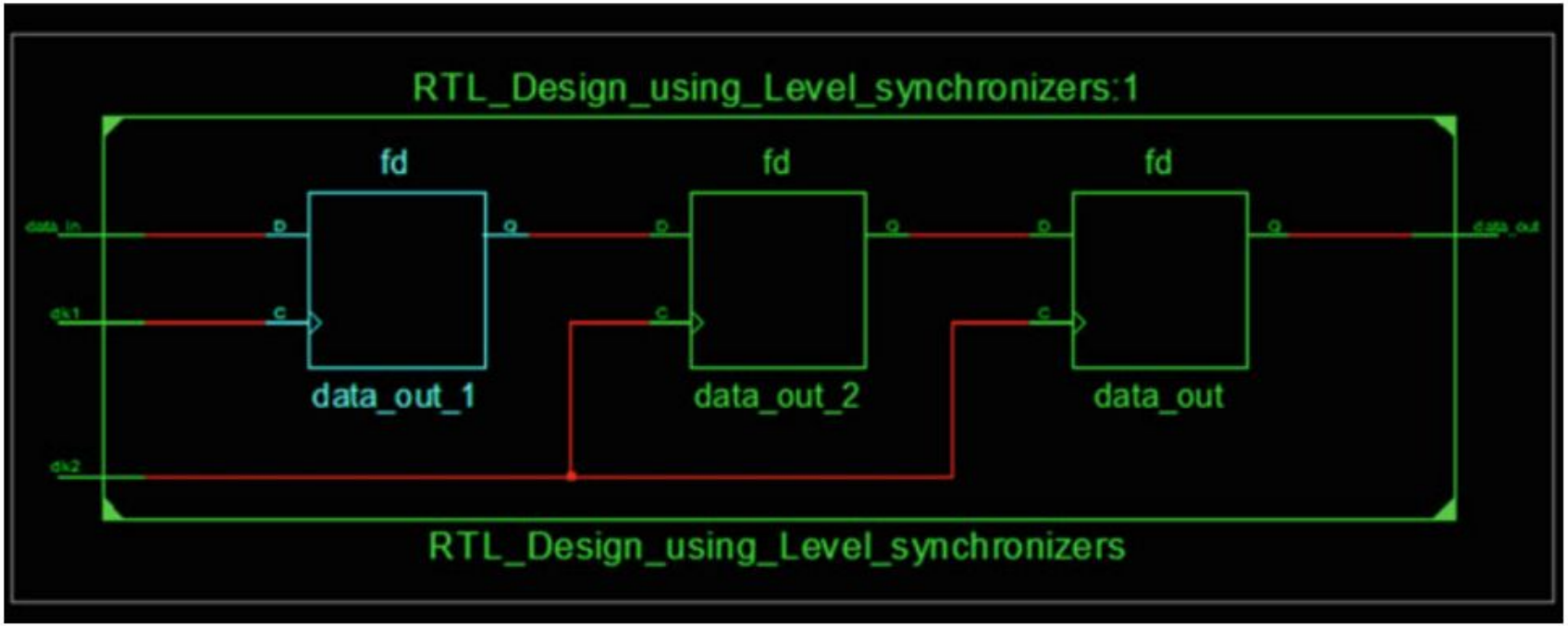



**Fig. 22.6** Two-stage level synchronizer in the control path

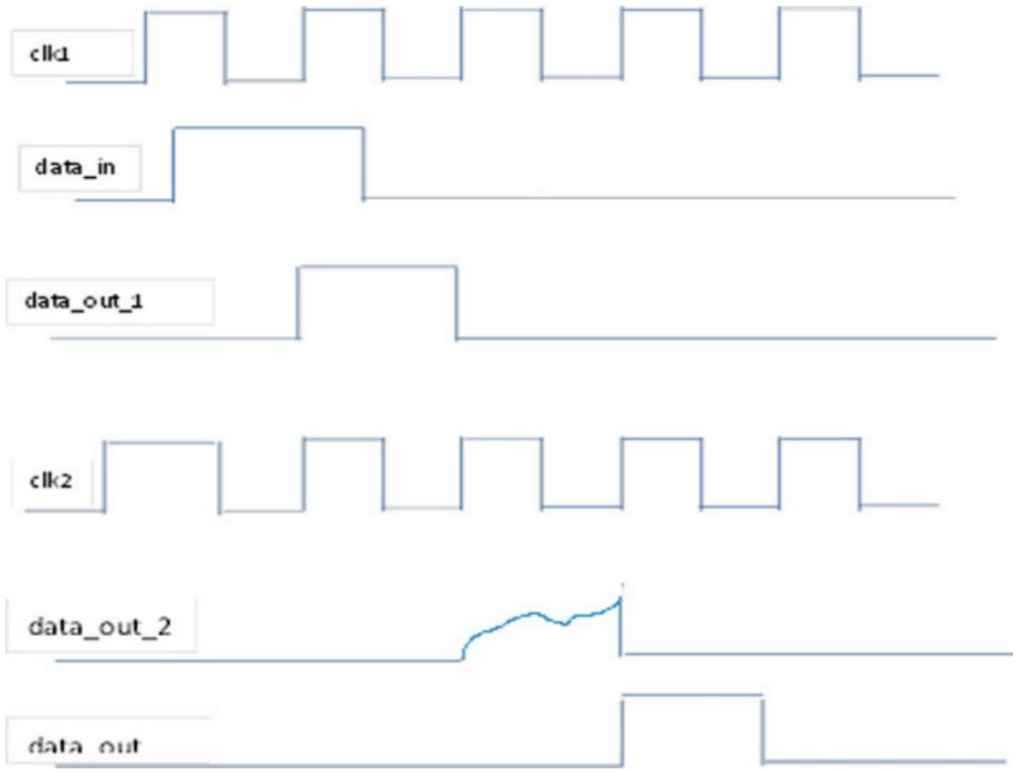
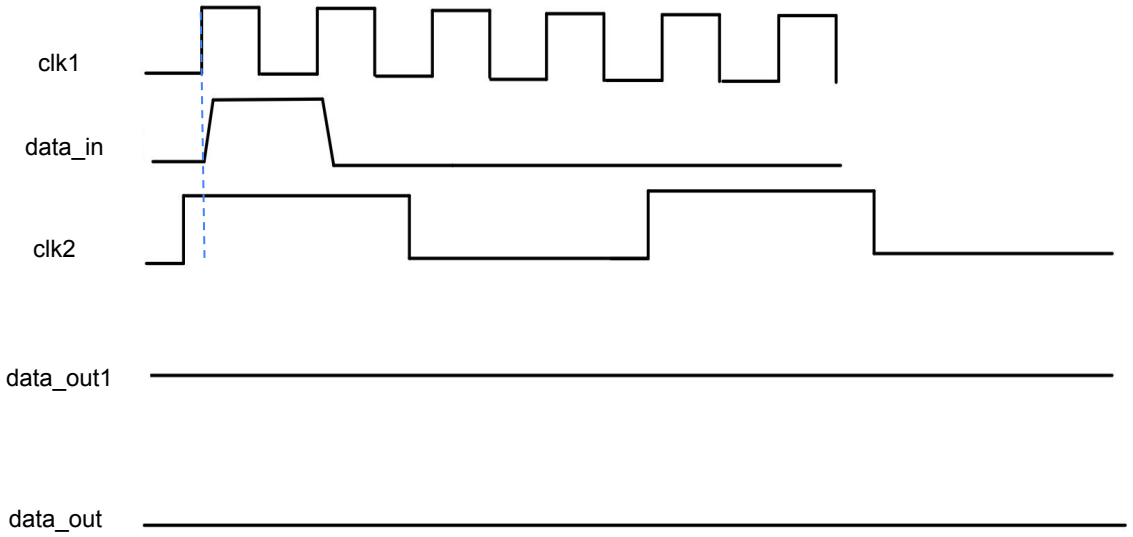



**Fig. 22.7** Timing sequence with use of the two-stage synchronizer

# Constraint - Only for slow clk to fast clk



# Pulse Stretcher

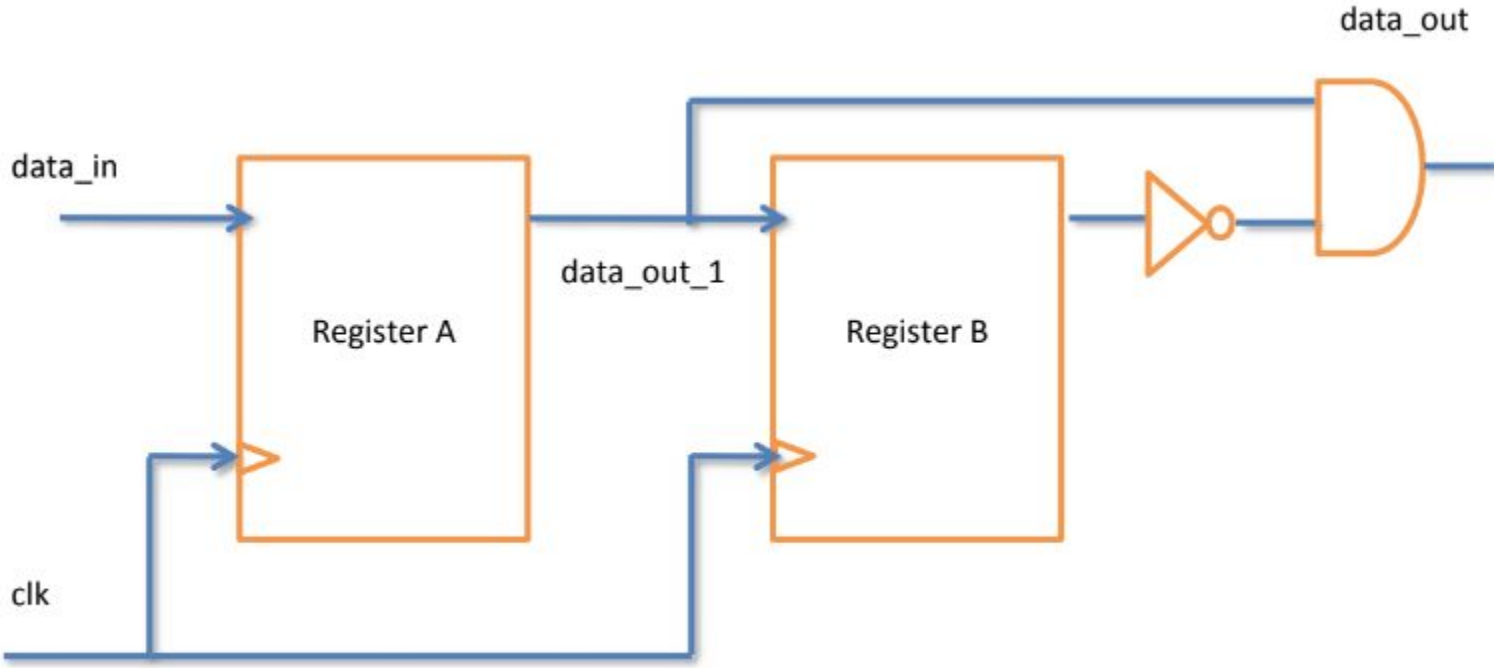
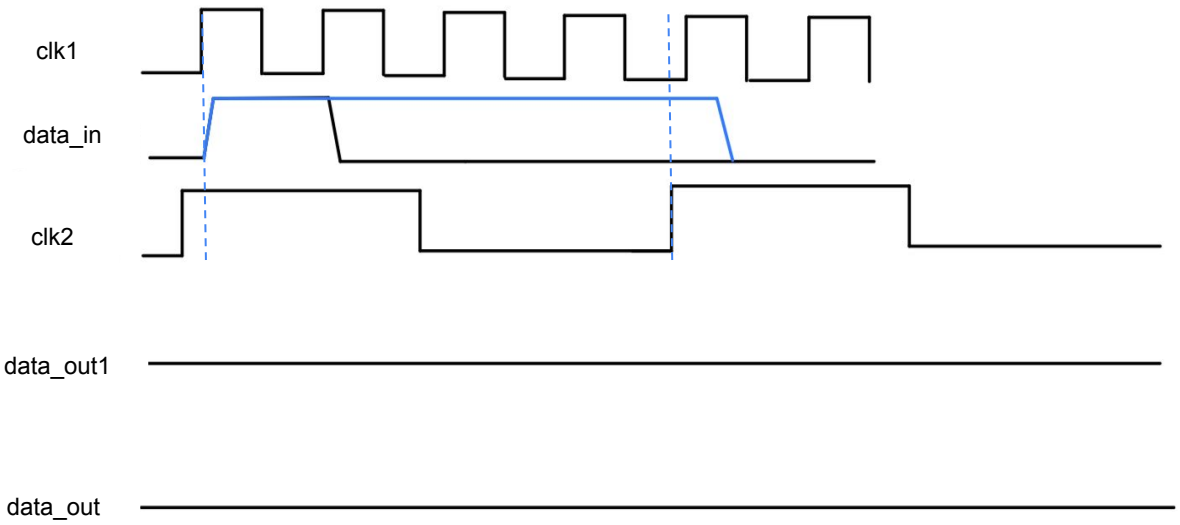
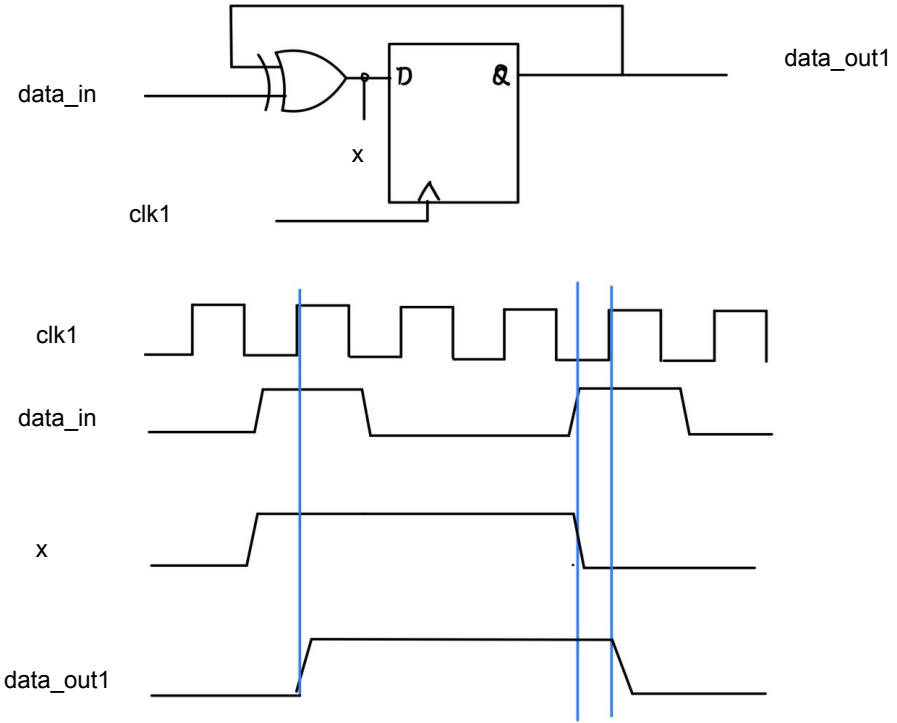



**Fig. 22.11** Level-to-pulse conversion logic

# Maybe…



# Pulse Stretcher



# Handshaking Mechanism

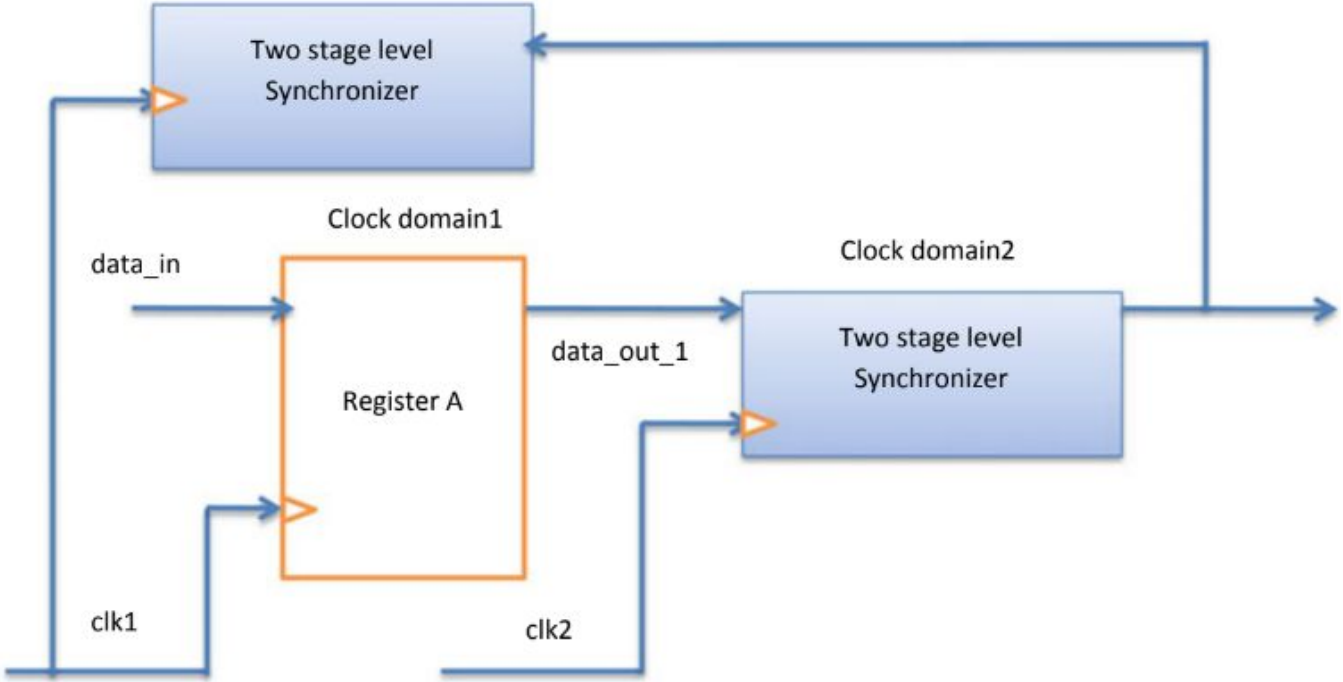



**Fig. 22.12** Handshake control signal mechanism

# Pulse Synchronizers / Toggle Synchronizer

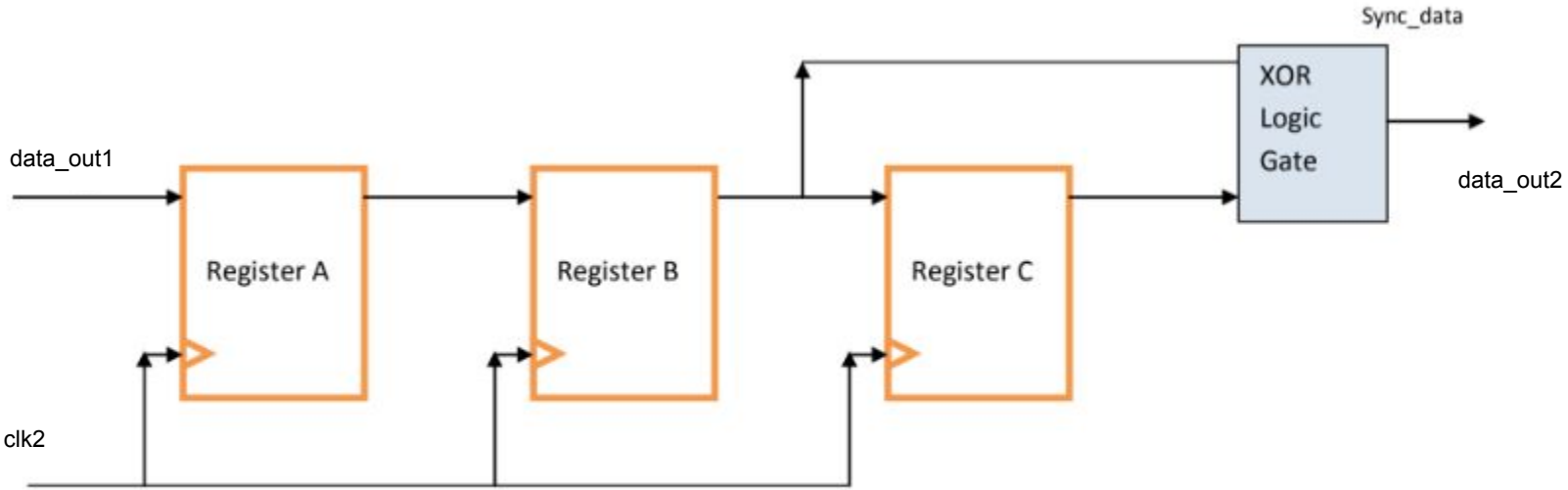
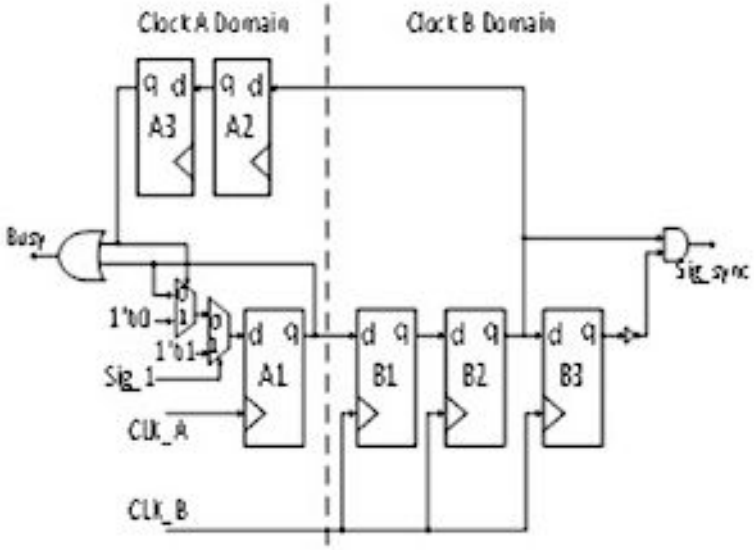
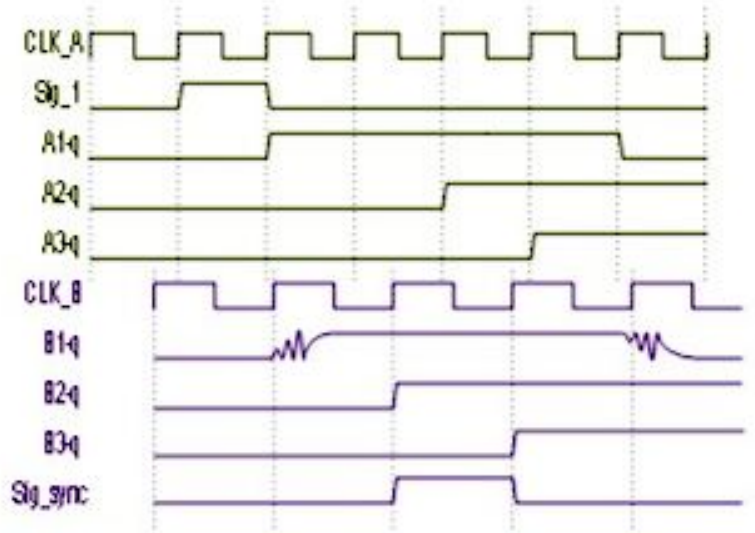



**Fig. 22.13** Pulse synchronizer

# Handshaking Mechanism + Pulse Synchronizers




https://www.edn.com/synchronizer-techniques-for-multi-clock-domain-socs-fpgas/

# Why we need 3 registers, what if there are only 2

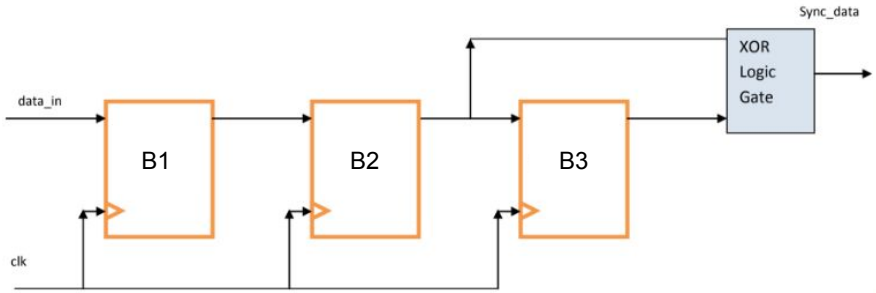
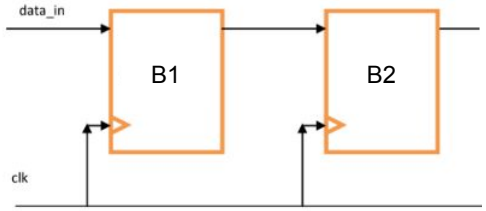
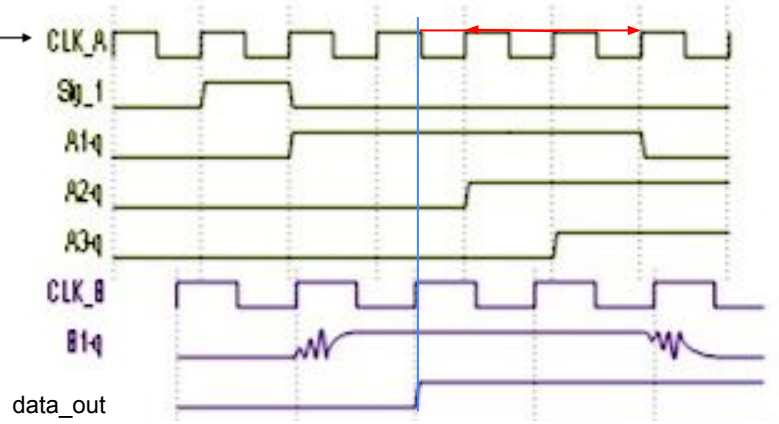



**Fig. 22.13** Pulse synchronizer



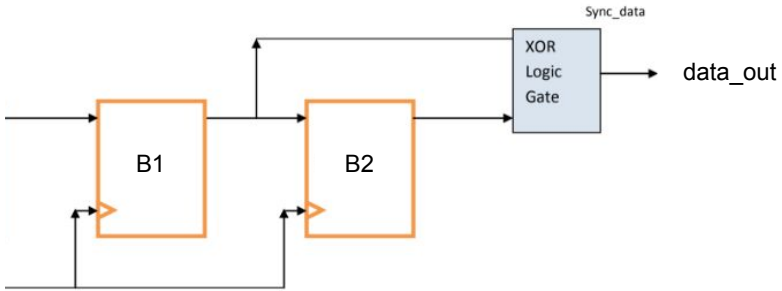

data_out

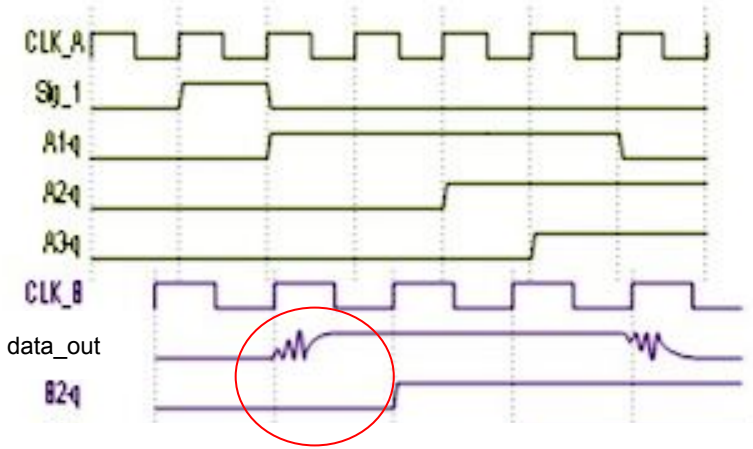
**Fig. 22.13** Pulse synchronizer




Make sure output is a pulse in a cycle time

# What if there are only 2 registers & a XOR




We cannot process with metastable signal.