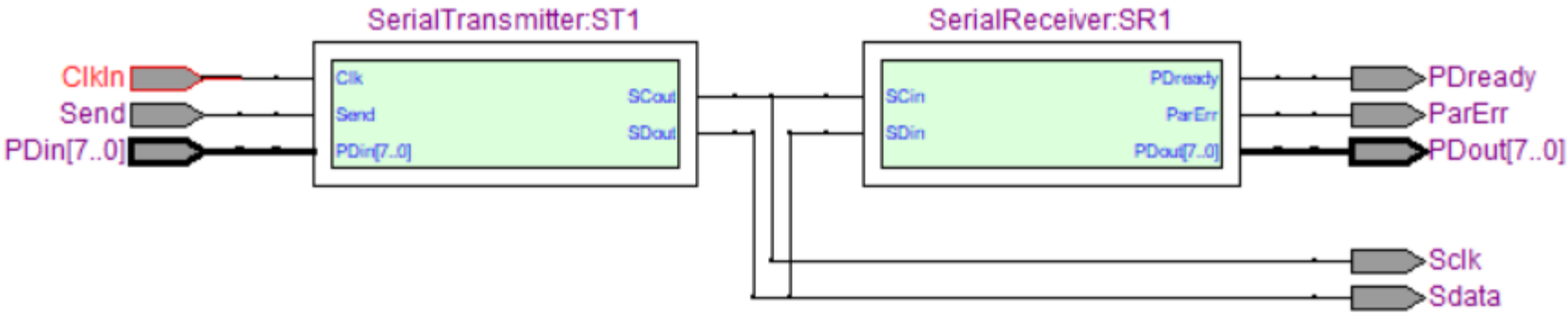
EE342 Lab-5
Instructions and Example Designs

Mehmet Çalı

2-Line Serial Transmitter/Receiver - Part 2

- You have designed a 2-line serial transmitter in the last experiment
- In this experiment you are supposed to apply multiple modification to transmitter and receiver modules
- In the top module inputs does not change and ParErr pin is added to outputs.
- The interconnections between modules stays the same

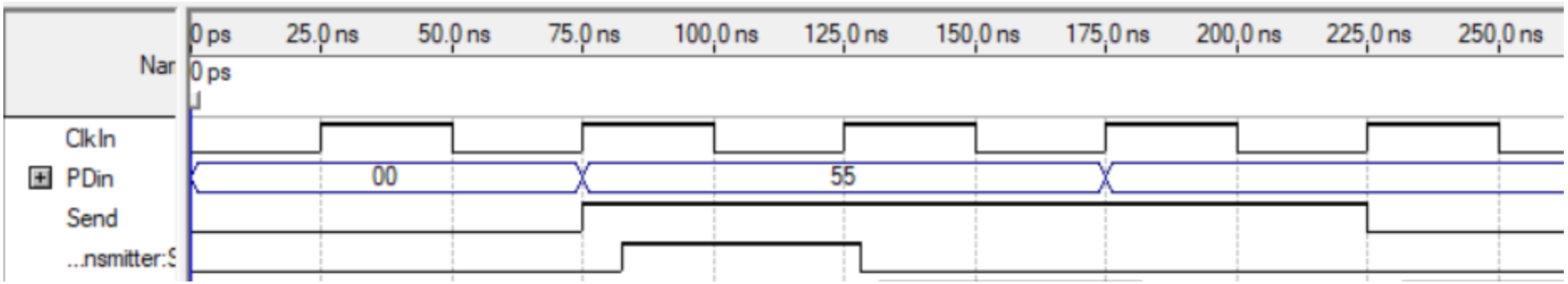


Transmitter

- **Inputs:**
  - **Clk:** 20 MHz clock
  - **Send:** Can be different than 1 clock cycle
  - **Pdata[7:0]** Parallel data
- **Outputs:**
  - **SClk**: 20 MHz clock
  - **Sdata**: Serial output signal that consist of a start bit and serialized Pdata from MSB to LSB

Part-1: Long Send input pulse:

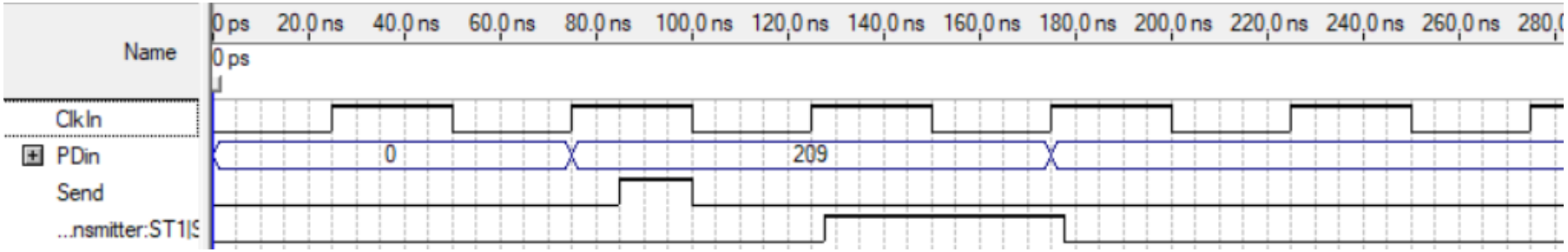
- In this part send signal can be more than one clock cycle
- In the previous experiment your code should work for single clock cycle
- The simplest way to handle this modification is to create another send signal that has a length of one clock cycle and use the produced new signal instead of the original one



Transmitter

Part-2: Asynchronous Send and parallel data inputs

- In this part the original send signal should be detected asynchronously (as soon as it is set) and then a synchronous one clock cycle modified send signal should be created to trigger transmission (**asynchronous detection**, **synchronous transmission**)
- In the experiment sheet it is stated that «Send and PDin[7:0] remain valid for at least two clock cycles» however please note that previous transmitter module can also work perfectly for send signals that has a length of over 2 clock cycles
- In order to check whether yor codes work or not you should also try testing for send signals that last less than one clock cycle as shown below. Hence the asynchronous detection of send signal has a meaningful outcome difference that is observed in the waveforms.
- Note that the duration that the send signal is set is far away from the rising edge of the next clock cycle



Transmitter/Receiver

- Due to timing constraints you should create two separate project and synthesize your code before the evaluation period if the synthesis takes too much time on your computer.
- Otherwise, you can separate previous two parts using macros and «`ifdef èndif» block or manually commenting and uncommenting.
- You do not need to create separate parts for the following modifications
- You can simply implement third and forth modification (parity bit and parity bit check) to your first and second part of your code.

Modification-3:

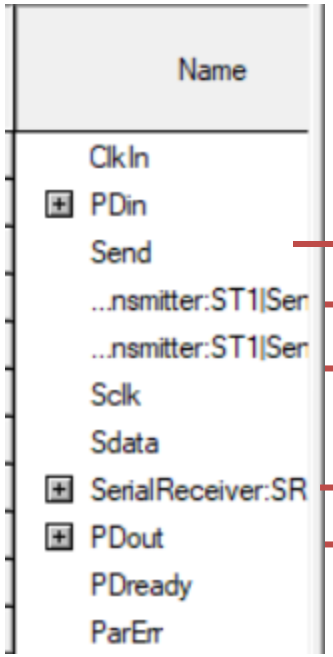
- Add a parity bit at the end of the transmitted data
- Use even parity (if the number of ones are odd, parity bit becomes one otherwise it becomes zero)
- To test **Modification-4** invert the parity bit so it gives an intentional wrong parity bit which will set **ParErr** at the receiver end.
- Note that this parity change can require adjustment in both transmitter and receiver shift registers

Modification-4:

- The receiver should compare parity bit and data bits. If there is an error, an output pin called **ParErr** should be set. **ParErr** should be set simultaneously with PDReady and it should remain high until the next start bit.

Waveform tips

- Your waveform file should include the following nodes:



Original send signal

Intermediate node/s that are used while creating final send signal (if exist)

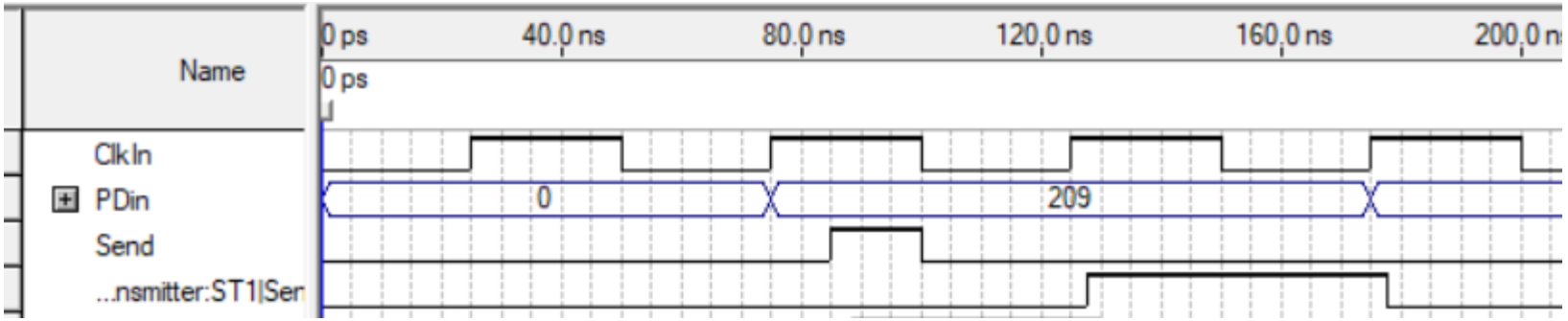
Final single clock cycle send signal

Counter used at the receiver side (if exist)

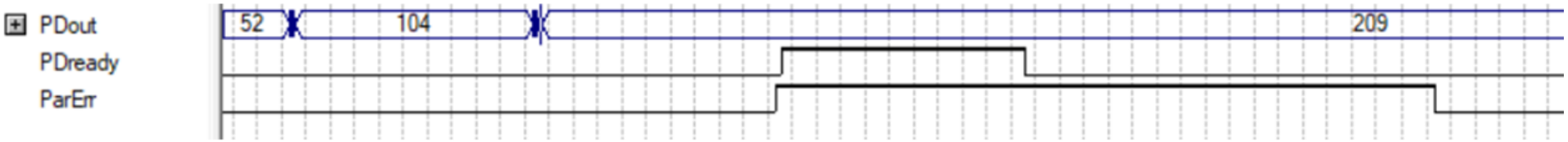
Parallel data output

Waveform tips

- Example outputs for second part:
- Transmitter nodes:

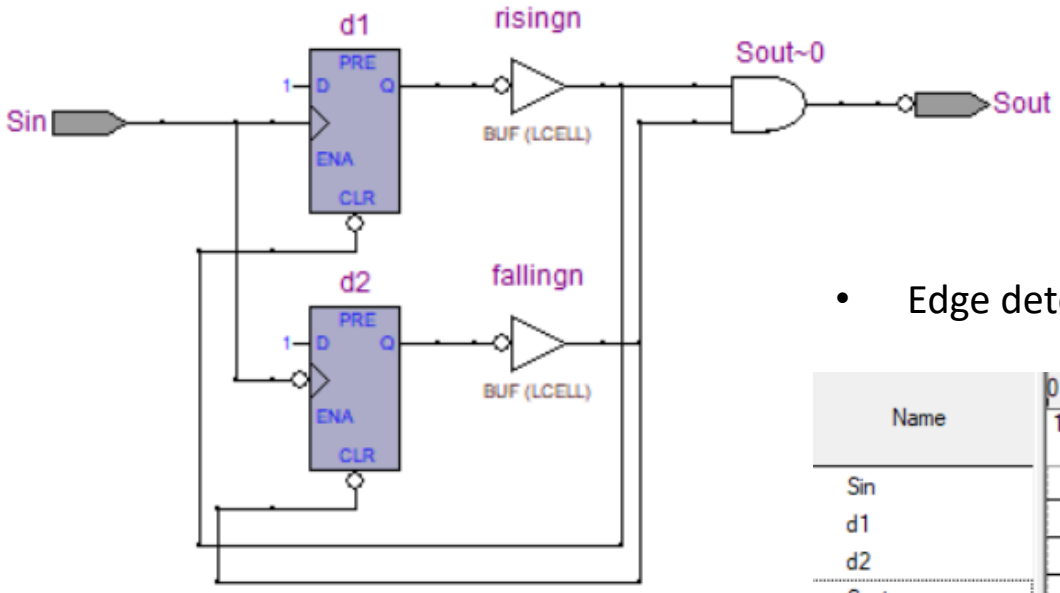


- Receiver nodes:

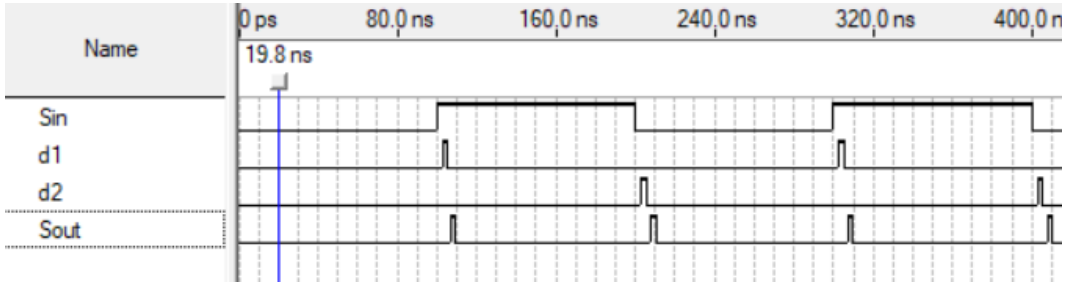


Related Design Example: Single Line Transmission

- We have seen differential encoding and an edge detector that constitutes first sub-module of a single line serial transmitter/receiver.
- Now we will move the second part of the single line transmission but before that lets add some modification to out edge detector.
- Reference design of last week's edge detector:



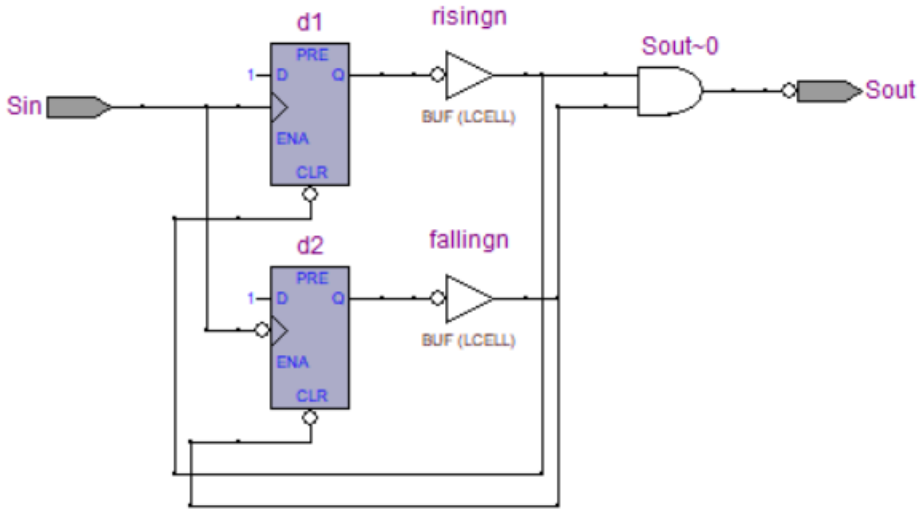
- Edge detector reference output:



Edge Detection

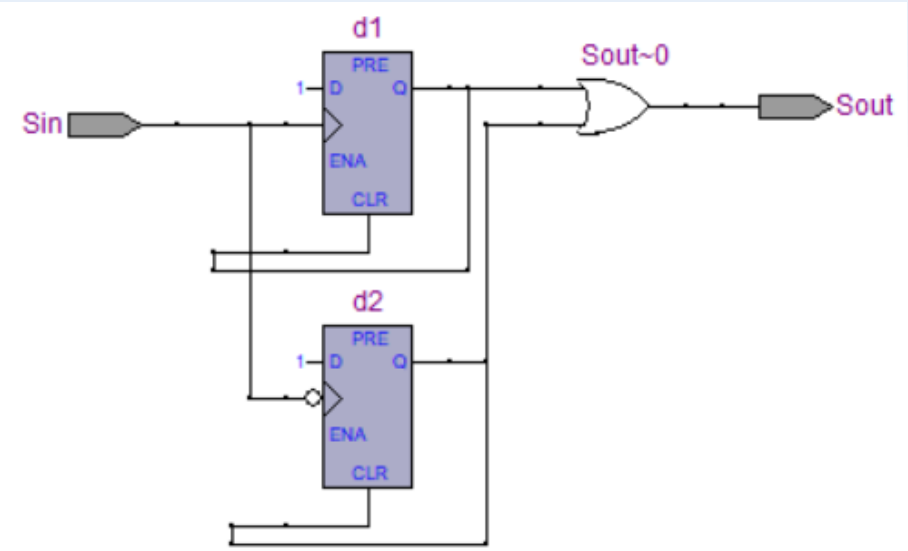
```
`ifdef part3
    module example1(Sin, Sout);
    input Sin;
    output Sout;
    wire rising, falling;

    supply1 vcc;
    supply0 gnd;

    DFF d1          (
                    .d(vcc),
                    .clk(Sin),
                    .clrn(~rising),
                    .prn(vcc),
                    .q(rising)
                    );

    DFF d2          (
                    .d(vcc),
                    .clk(~Sin),
                    .clrn(~falling),
                    .prn(vcc),
                    .q(falling)
                    );

    assign Sout = rising || falling;
    endmodule
`endif
```

Reference Schematic:



Edge Detection

- Part-3 schematic:



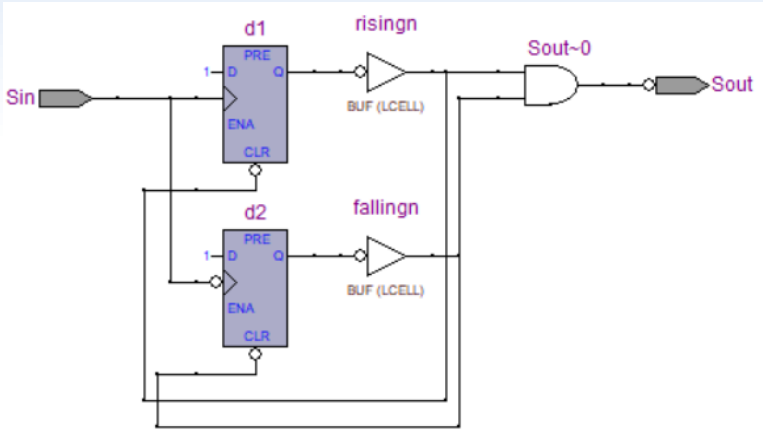
- Part-3 waveform:

Edge Detection

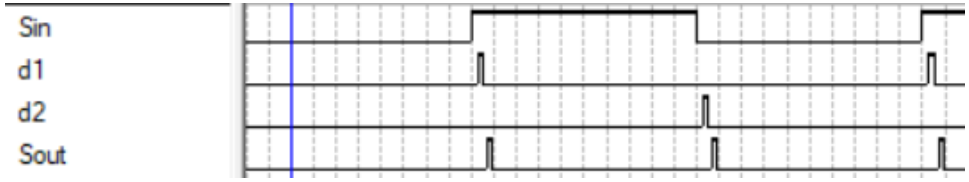
- Lets say we want to increase the pulse widths of edge detection output

```
`ifdef part4
    module example1(Sin, Sout);
    input Sin;
    output Sout;
    wire rising, falling;
    wire risingn;
    wire fallingn;
    supply1 vcc;
    supply0 gnd;
    DFF d1          (
                    .d(vcc),
                    .clk(Sin),
                    .clrn(risingn),
                    .prn(vcc),
                    .q(rising)
                    );

    DFF d2          (
                    .d(vcc),
                    .clk(~Sin),
                    .clrn(fallingn),
                    .prn(vcc),
                    .q(falling)
                    );
    assign risingn = ~rising;
    assign fallingn = ~falling;
    assign Sout = ~(risingn && fallingn);
    endmodule
`endif
```

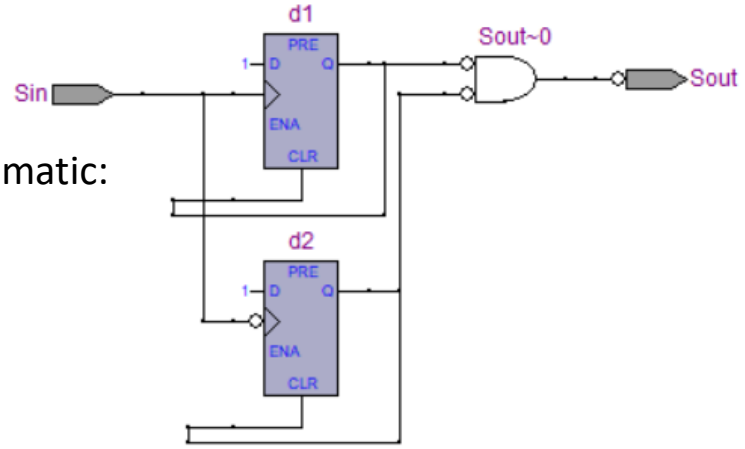
Reference Schematic:



Actual Output:



Actual Schematic:



Edge Detection

- Synthesizer has changed the output gate however it does not add the buffers (or inverters).
- Therefore the pulse with stayed the same.
- If we want to keep a wire (prevent it from being syntesized away by Quartus) we should add a comment /* synthesis keep */ between the wire name and semicolon.
- Example usage:

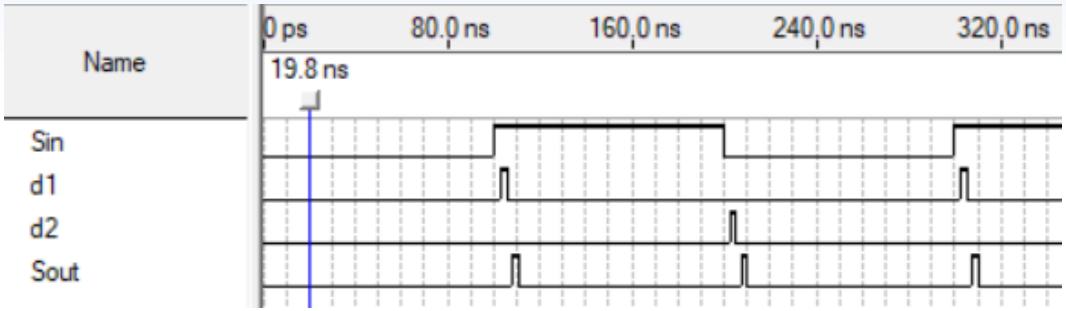
```
wire risingn /* synthesis keep */;
```

Edge Detection

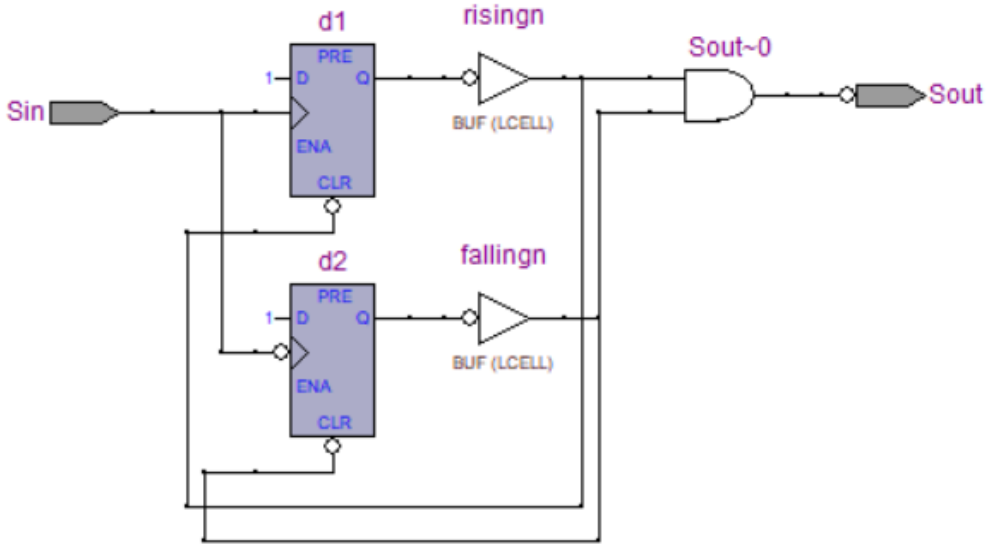
- The correct code to increase pulse delay:

```
`ifdef part5
    module example1(Sin, Sout);
    input Sin;
    output Sout;
    wire rising, falling;
    wire risingn /* synthesis keep */;
    wire fallingn /* synthesis keep */;
    supply1 vcc;
    supply0 gnd;

    DFF d1          (
                    .d(vcc),
                    .clk(Sin),
                    .clrn(risingn),
                    .prn(vcc),
                    .q(rising)
                    );

    DFF d2          (
                    .d(vcc),
                    .clk(~Sin),
                    .clrn(fallingn),
                    .prn(vcc),
                    .q(falling)
                    );
    assign risingn = ~rising;
    assign fallingn = ~falling;
    assign Sout = ~(risingn && fallingn);
    endmodule
`endif
```

Actual Output:



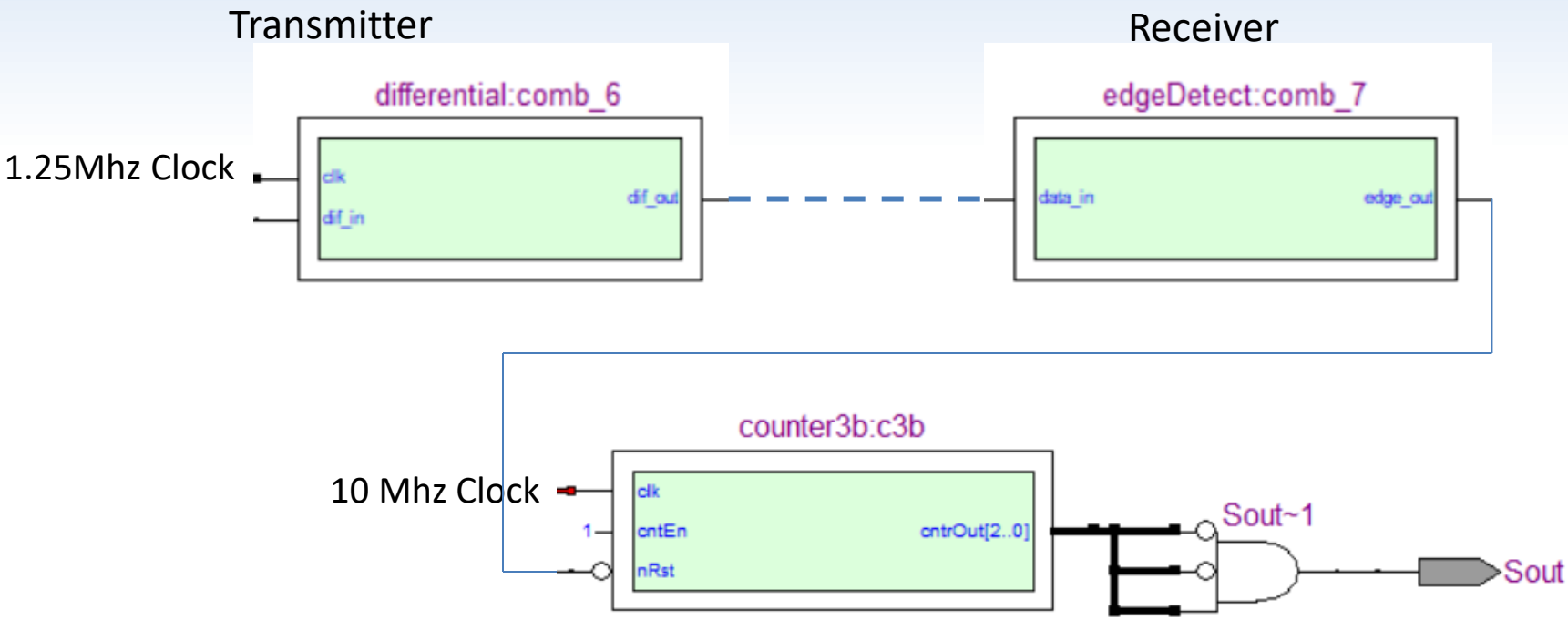
Actual Schematic:

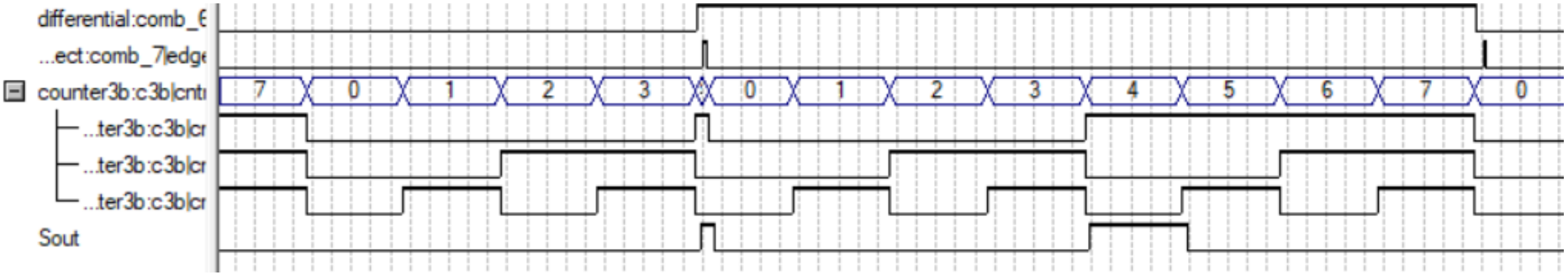


Clock Generator: Second Part

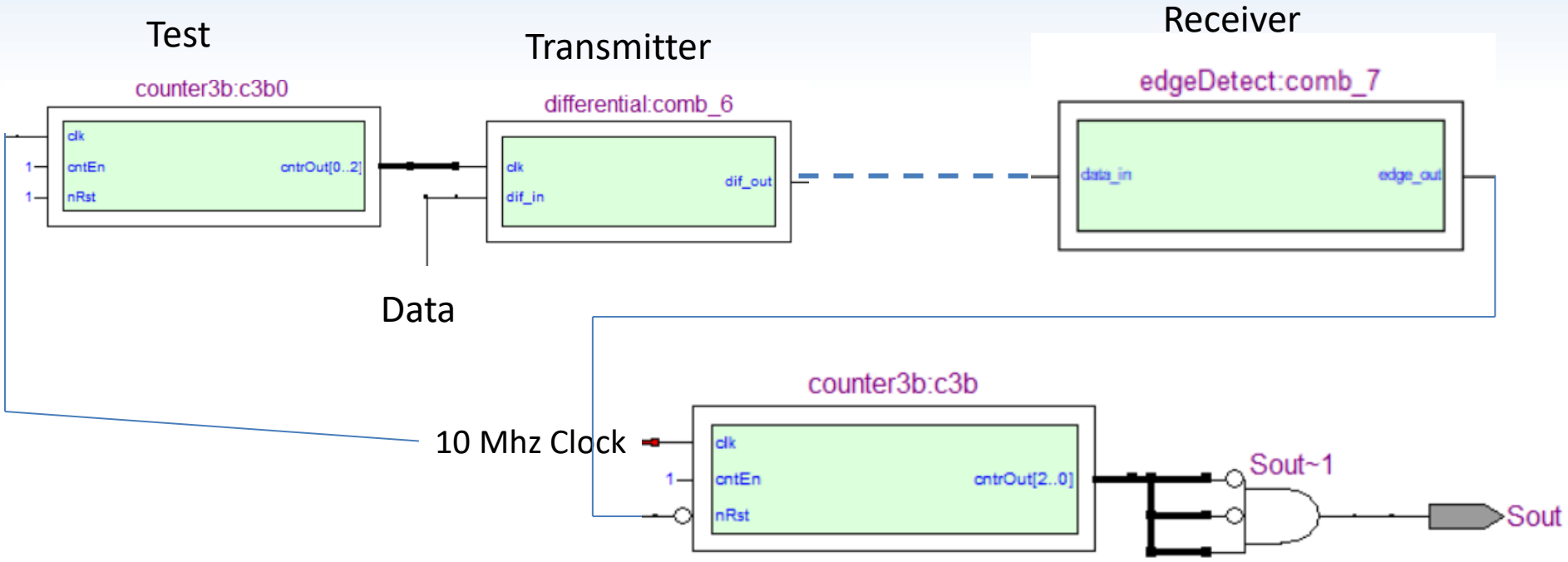
- We can add any number of dummy buffer to increase the pulse duration of our edge detector output using /* synthesis keep */ comment.
- The only differnce between single and two line serial transmission was the lack of clock input in the receiver side
- In order to generate our clock with a proper phase (the rising edge of the clock should correspond to middle part of the each bit)

Reference Design





Test Setup

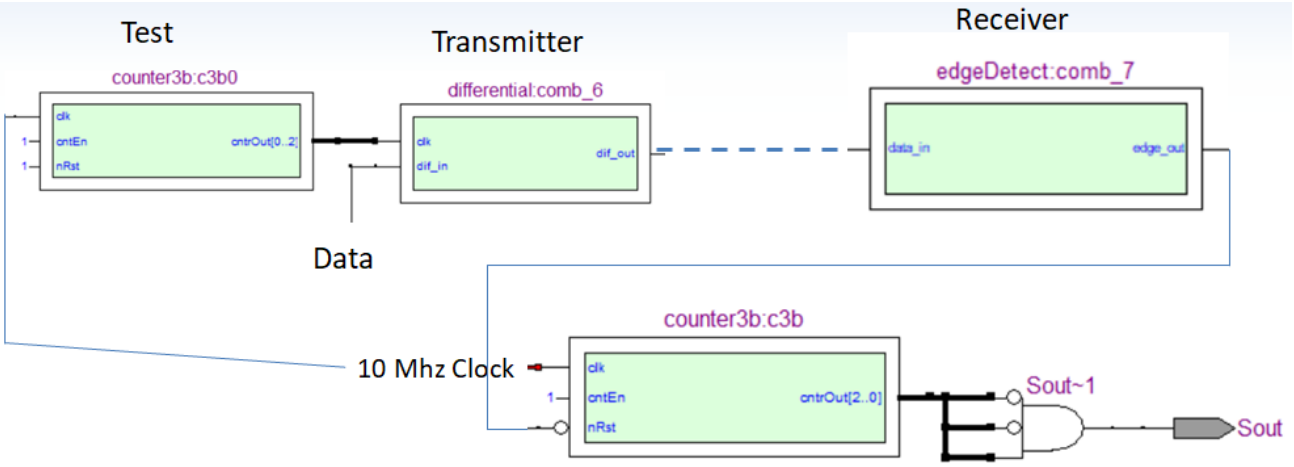

Test
Transmitter
Receiver
counter3b:c3b0
clk
cntEn
nRst
cntrOut[0..2]
1
1
differential:comb_6
clk
dif_in
dif_out
edgeDetect:comb_7
data_in
edge_out
Data
counter3b:c3b
10 Mhz Clock
clk
cntEn
nRst
cntrOut[2..0]
1
Sout~1
Sout

Counter Module

```
//Counter
module counter3b(clk, nRst, cntEn, cntrOut);

input  clk;
input  nRst;
input  cntEn;
output reg [2:0] cntrOut;

always @(posedge clk or negedge nRst)
  if (nRst == 1'b0)
    cntrOut[2:0] <= 3'd0;
  else
    if (cntEn == 1'b0)
      cntrOut[2:0] <= cntrOut[2:0];
    else
      if (cntrOut[2:0] == 3'd7)
        cntrOut[2:0] <= 3'd0;
      else
        cntrOut[2:0] <= cntrOut[2:0] + 4'd1;
```

Top Module

```
//Top Module
module example1(Sin, Clk, Sout);
input Sin, Clk;
output Sout;
wire receiver_in, edge_out;
wire [2:0] count_out;
wire [2:0] Clkd8;

supply1 vcc;
supply0 gnd;

counter3b c3b0(.clk(Clk), .nRst(vcc), .cntEn(vcc), .cntrOut(Clkd8));
differential(.dif_in(Sin),.clk(Clkd8[2]),.dif_out(receiver_in));
edgeDetect(.data_in(receiver_in), .edge_out(edge_out));
counter3b c3b(.clk(Clk), .nRst(~edge_out), .cntEn(vcc), .cntrOut(count_out));
assign Sout = count_out[2] && ~count_out[1] && ~count_out[0];

endmodule
```



Overall Waveform