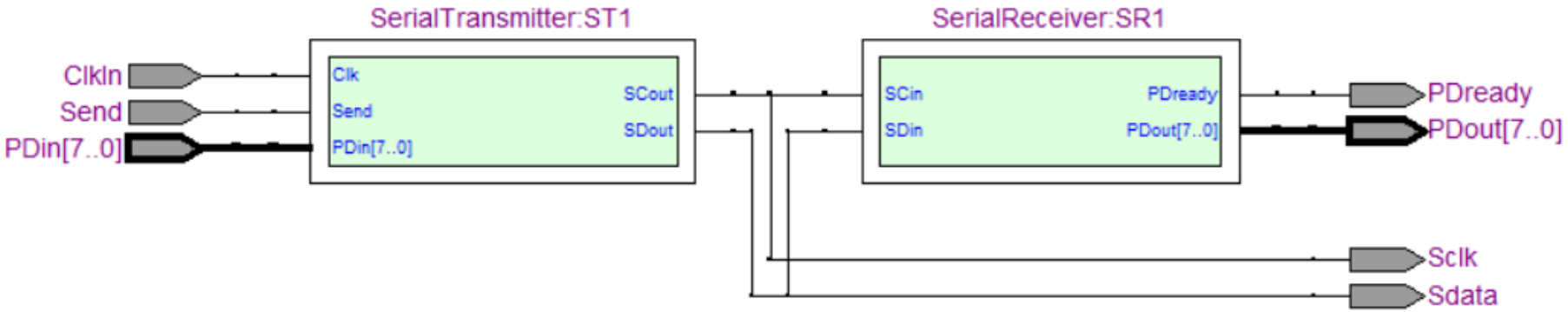
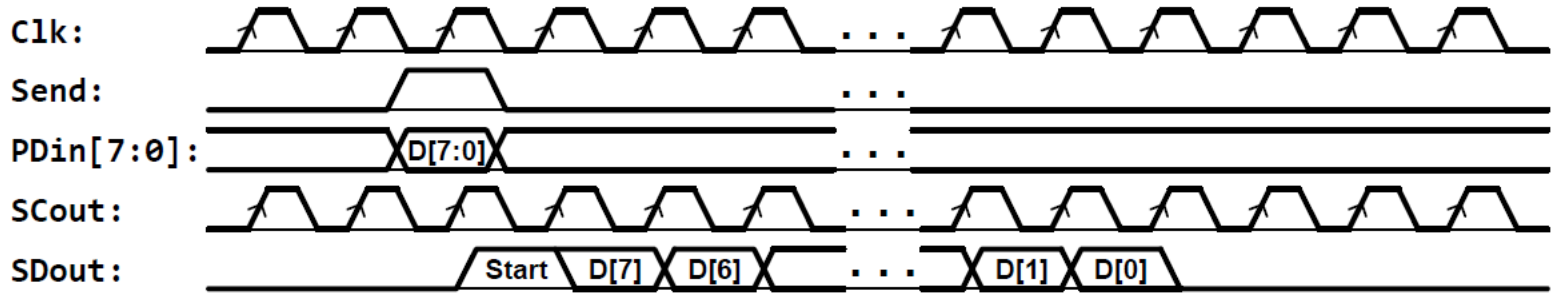
EE342 Lab-4
Instructions and Example Designs

Mehmet Çalı

2-Line Serial Transmitter/Receiver

- You have seen the 3 line serial transmission in Lecture Notes-5 Serial Interface.
- You are now required to design a 2 line transmitter and receiver and connect them in a top module





Transmitter

- **Inputs:**
  - **Clk:** 20 MHz clock
  - **Send:** 1 Clock cycle signal to inform that the paralel input is ready
  - **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

In **three line transmitter** we have seen that transmitter consist of a counter and a shift register:

```
// Describe the shift register:
reg  [7:0] SR;  // 8-bit shift register to store 8-bit parallel data

always @(posedge Clk)
begin
  if (WE == 1'b1)
    SR[7:0] <= Pdata[7:0];  // store parallel input data
  else
  begin  // start shifting when WE is 0
    SR[7:1] <= SR[6:0];  // MSB goes out first
    SR[0] <= 1'b0;  // clear LSB while shifting left
  end
end
assign Sdata = SR[7];  // MSB of shift register is the Sdata output
```

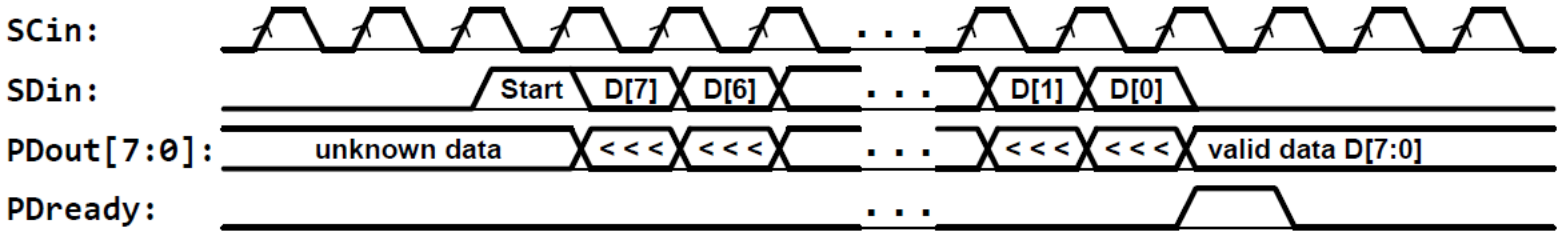
Transmitter

```
// Describe the counter (a better way):
reg  [2:0] Cntr;  // 3-bit counter for Sen timing
reg  SRshift;     // set while shift register is active
always @(posedge Clk)  // a simple counter enabled by SRshift
begin
  if (SRshift == 1'b0)
    Cntr[2:0] <= 3'd0;  // reset the counter
  else
    Cntr[2:0] <= Cntr[2:0] + 3'd1;  // increment the counter
end
always @(posedge Clk)
begin
  if (WE == 1'b1)
    SRshift <= 1'b1;  // set when parallel data is loaded
  else
    if (Cntr[2:0] == 3'd7)
      SRshift <= 1'b0;  // reset when counter reaches 7
    else
      SRshift <= SRshift;  // keep the last value
end
assign Sen = SRshift;  // Sen is free from glitches
```

We do not need this part in our 2 line transmitter however a similar structure will be needed in the receiver part

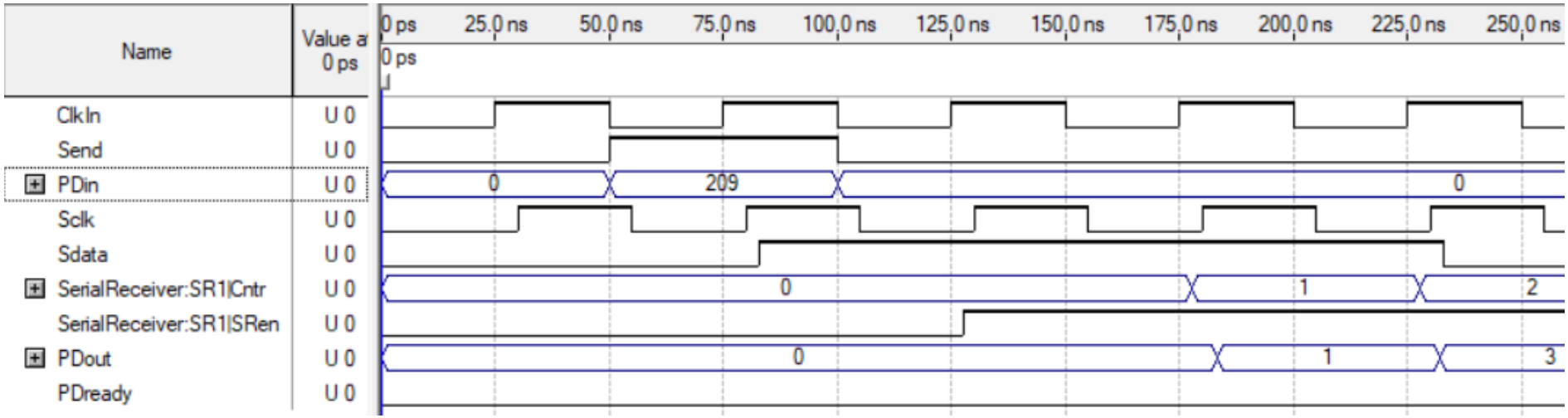
Receiver

- **Inputs:**
  - **SCin:** 20 MHz clock
  - **Sdin:** Serial data
- **Outputs:**
  - **PDout[7:0]**: Parallel data
  - **PDready**: One clock cycle signal that indicates the paralel data is ready



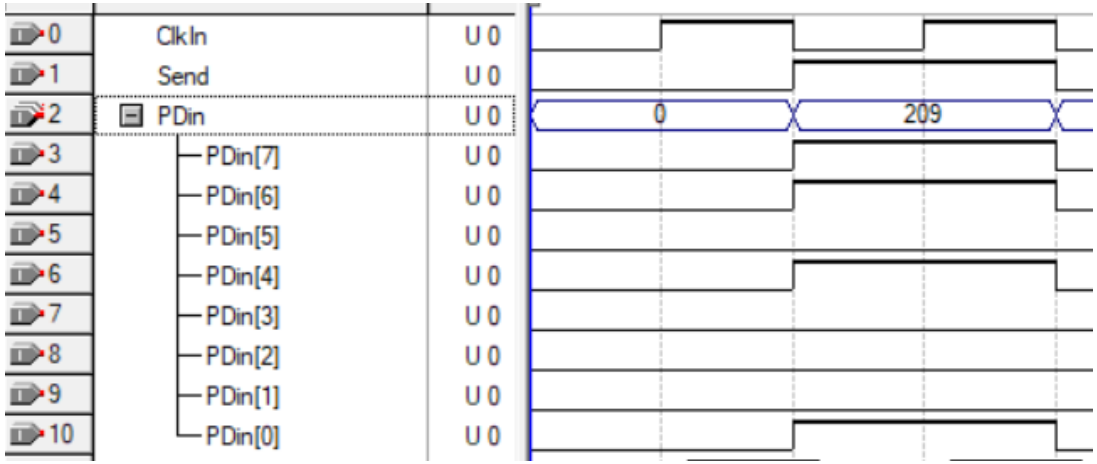
Waveform tips

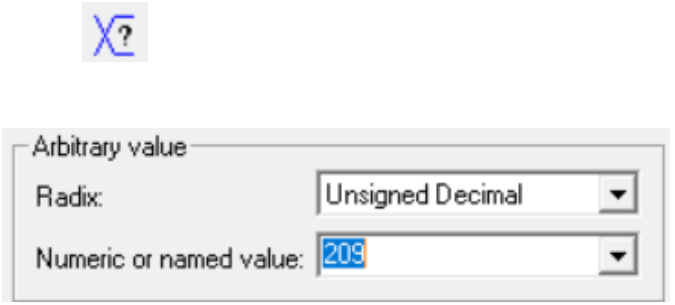
- You should connect outputs of transmitter and inputs of receiver
- You should create a single waveform file to test your top module with the following nodes



Waveform tips

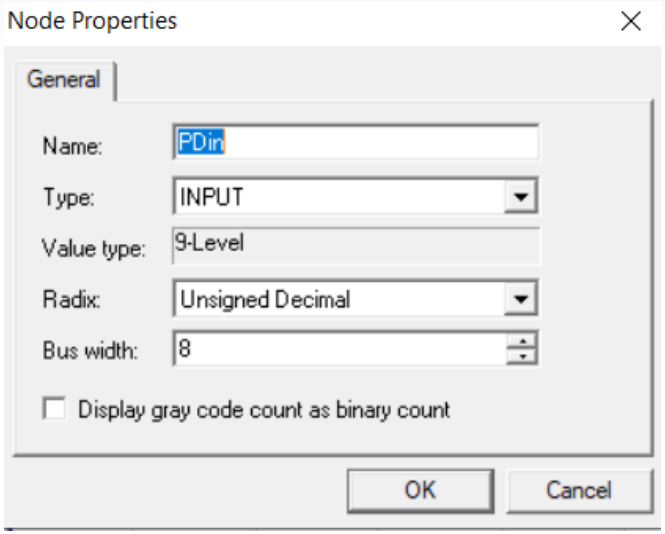
- Assign PDin to a asymetric 8 bit number in unsigned decimal formal
- The most and the least significant bits should be one

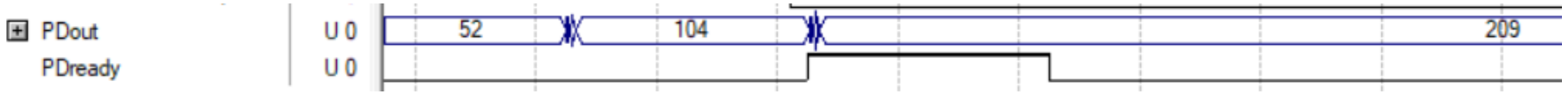




Waveform tips

- From right click to node name > Properties, assign Radix tab to unsigned decimal for PDin, serial receiver counter and PDout as well





- Be sure that parallel output is same as parallel input and the Pdready signal is set when Pdout is completely received as shown above

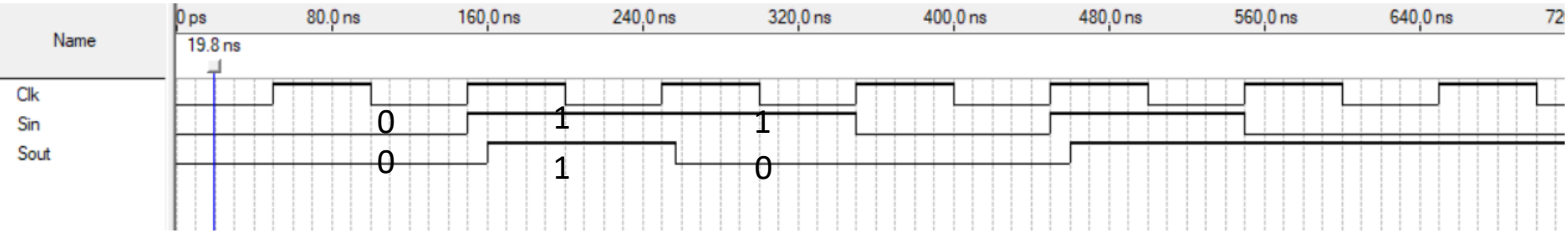
Some related design examples

- Before the transmission of signals, they are usualy encoded in various ways
- One of the most basic one is differential coding
- In differential coding the signal is encoded according to changes in the encoded output from the original signal

	t1	t2	t3	t4	t5	t6	t7	t8
Original signal x(i)	1	0	1	0	0	0	1	0
Encoded signal y(i)	1	1	0	0	0	0	1	1

0

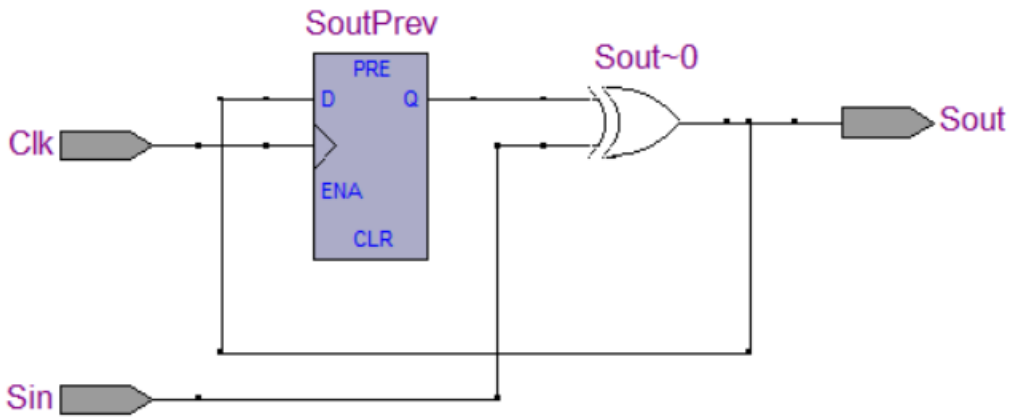
$$y(i) = y(i-1)\hat{}x(i)$$

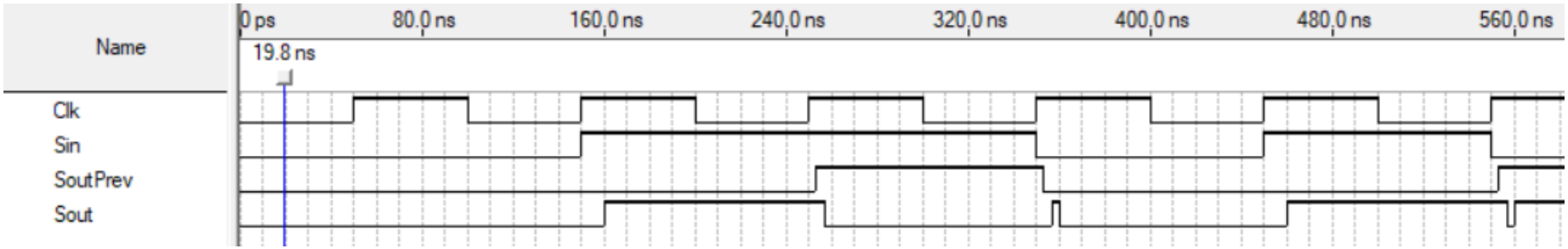


Differential Coding

- Wrong approach-1:

```
1   `ifdef part1
2       //Differential encoding
3       module example1(Sin, Clk, Sout)
4       input Sin, Clk;
5       output Sout;
6       reg SoutPrev;
7
8       always @(posedge Clk)
9       begin
10          SoutPrev = Sout;
11      end
12
13      assign Sout = SoutPrev ^ Sin;
14
15      endmodule
16
17
18  `endif
```

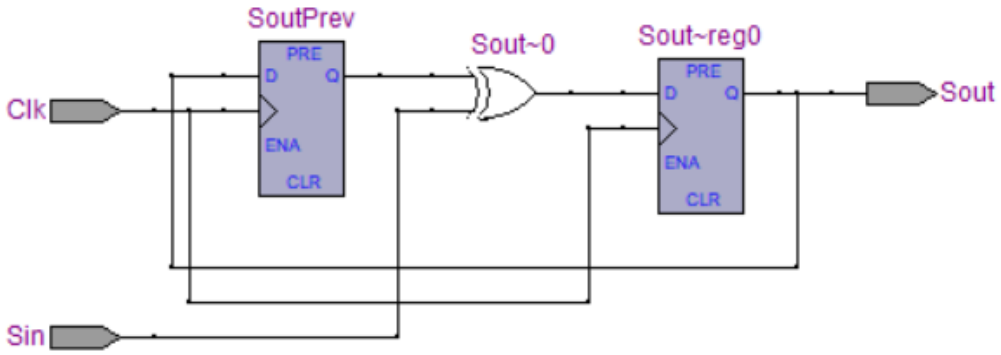




Differential Coding

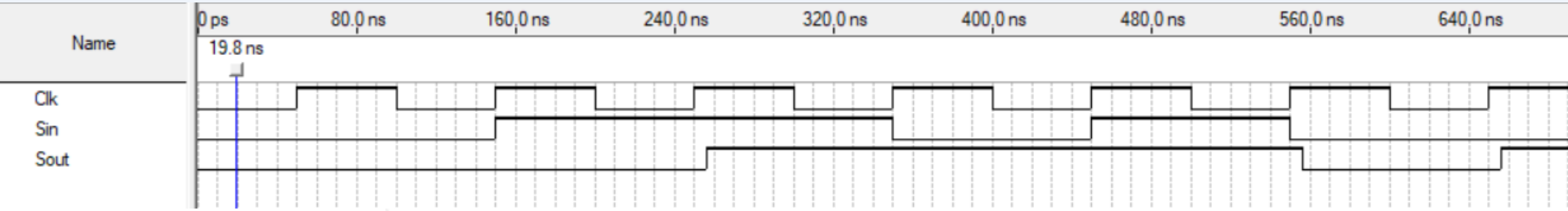
- Wrong approach-2:

```
`ifdef partla

    //Differential encoding
    module example1(Sin, Clk, Sout);
    input Sin, Clk;
    output reg Sout;
    reg SoutPrev;

    always @(posedge Clk)
    begin
        SoutPrev <= Sout;
    end

    always @(posedge Clk)
    begin
        Sout <= SoutPrev ^ Sin;
    end

    endmodule

`endif
```



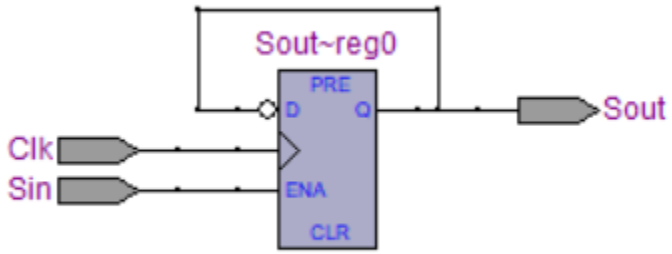
Differential Coding

Result of wrong approach-2:

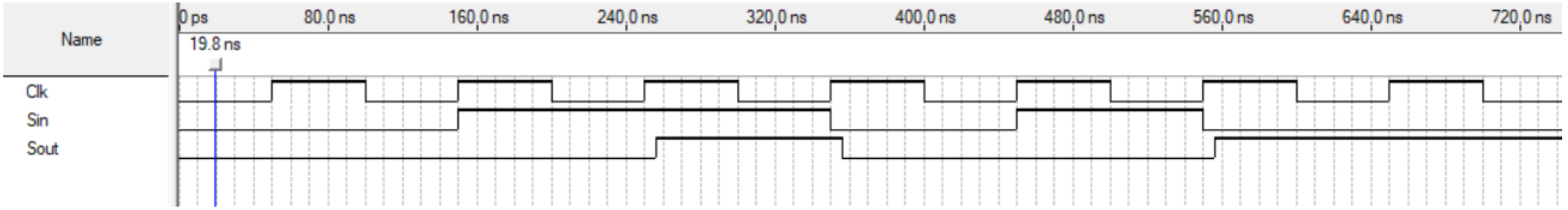


Third approach:

```
`ifdef partlb
    //Differential encoding
    module example1(Sin, Clk, Sout);
    input Sin, Clk;
    output reg Sout;

    always @(posedge Clk)
    begin
        Sout <= Sout ^ Sin;
    end
    endmodule
`endif
```

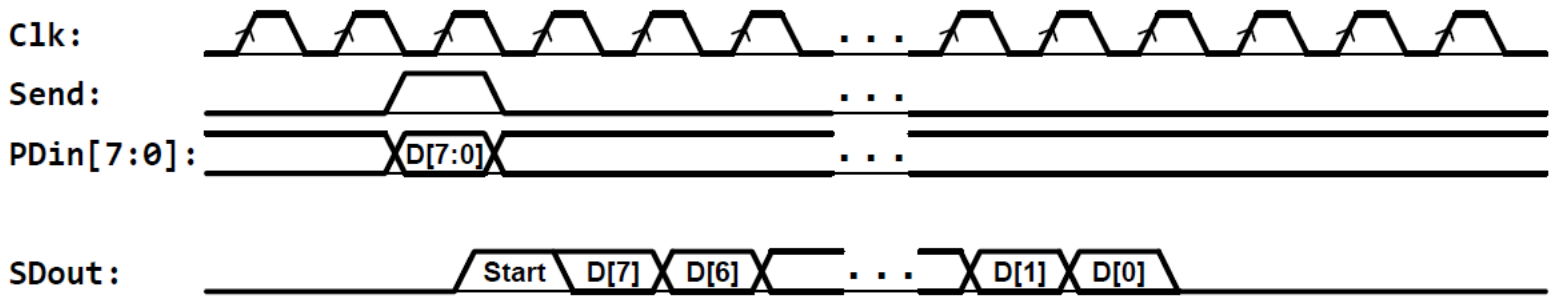


Result of third approach:



Differential Coding

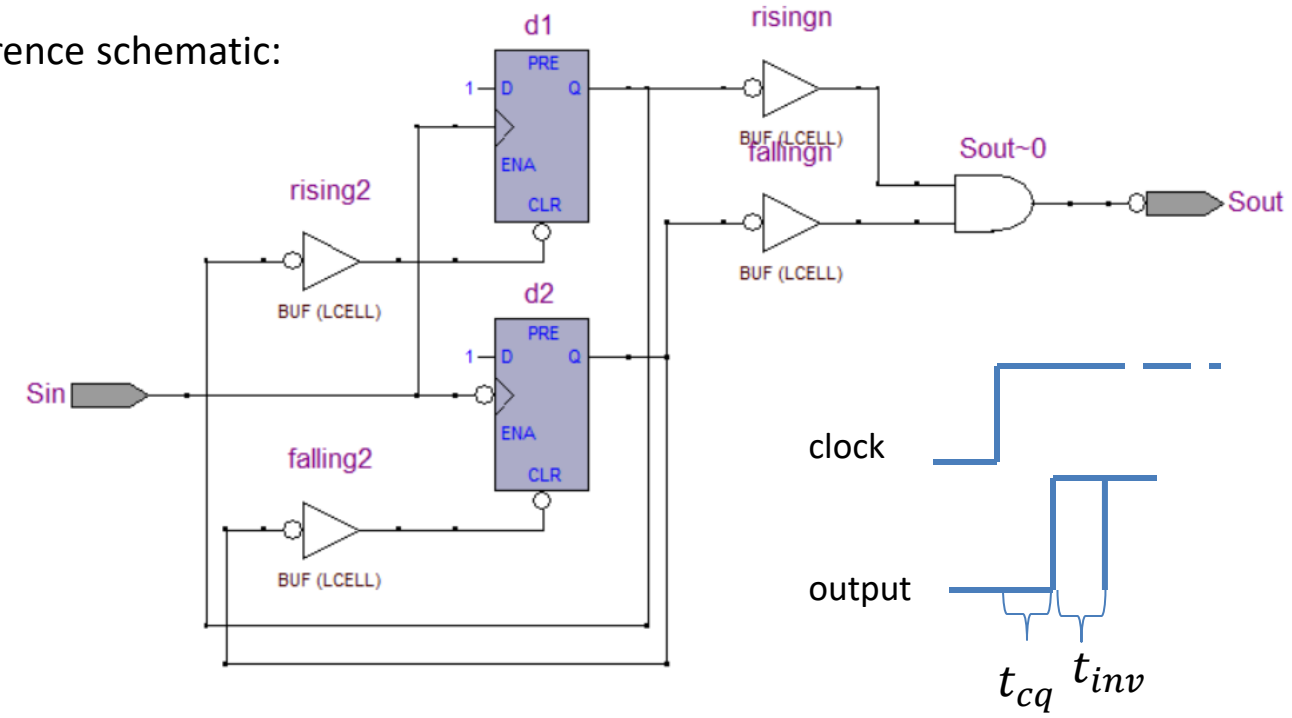
- After the optional encoding process serialized signal can be sent in various line number.
- You have seen the three line transmission and you are responsible for the two line transmission in the next week. What about single line transmission?
- Single line transmission obviously involves transmission of the data without a clock signal
- When we lower the line number from 3 to 2 by removing the data enable signal, we could extract it from the start bit in the receiver side.
- How could we extract the clock signal at the receiver side?
- There are various ways, but we will start with a first phase of a relatively simple clock extractor approach



Edge Detection

- In order to extract the clock signal with a proper phase from the data, we need a high frequency input clock with arbitrary phase and a edge detection circuit in the method that we will apply.
- Edge detection will be applied to the data so that we could generate a clock from higher frequency clock whose rising clock edge will correspond to the middle section of the data

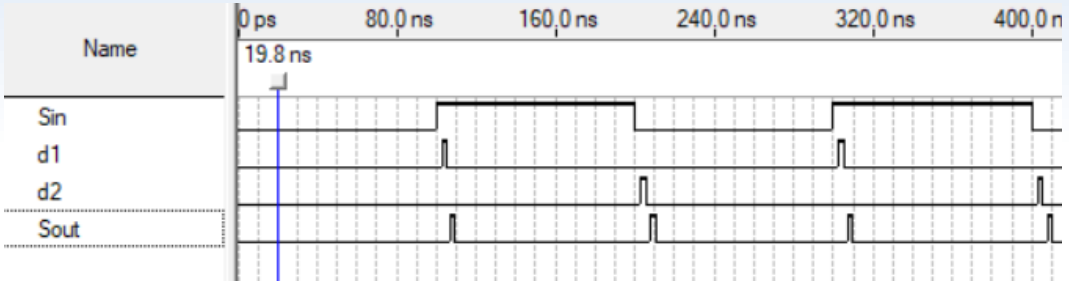
Edge detector reference schematic:



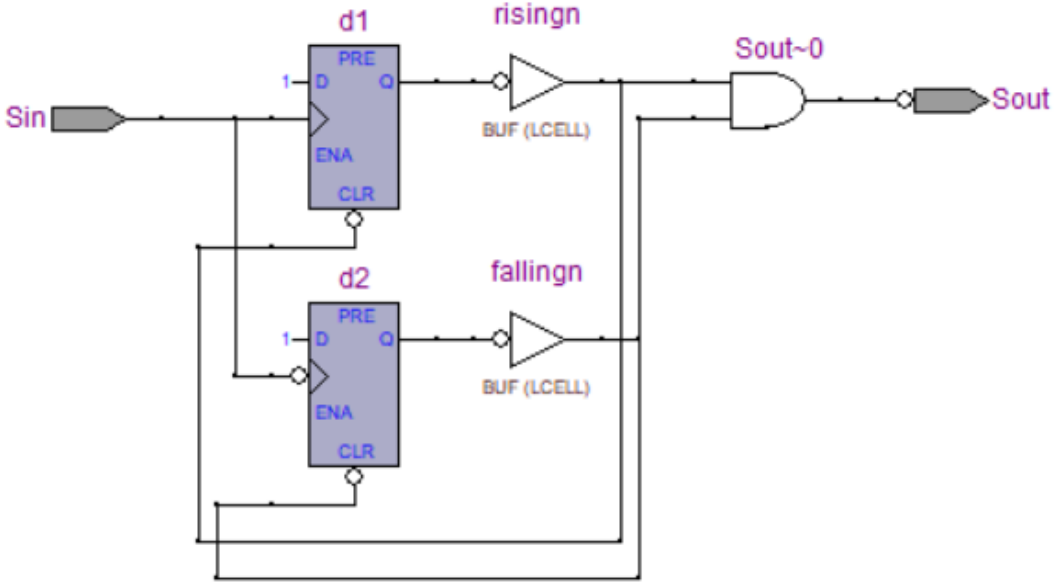
Edge Detection

```
`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
```

- Edge detector reference output:

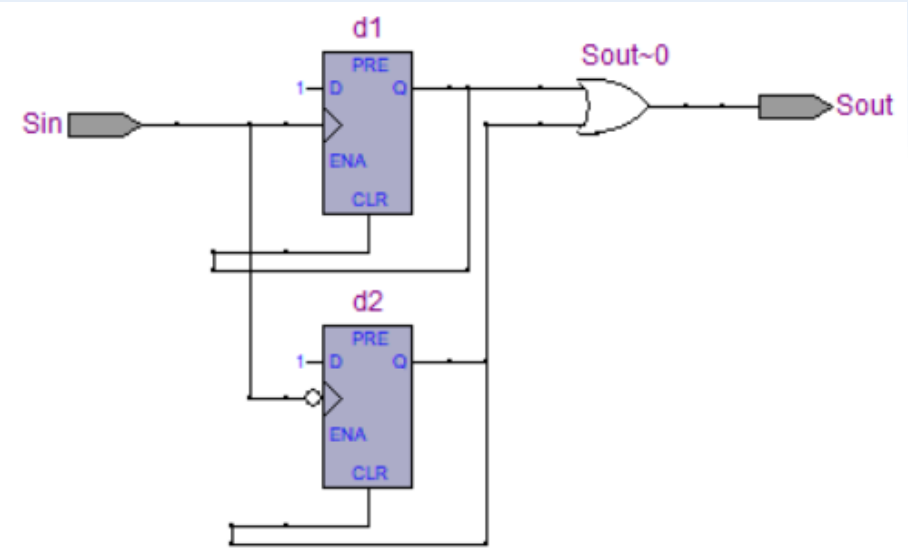


- Edge detector reference schematic:



Edge Detection

- Part-3 schematic:



- Part-3 waveform: