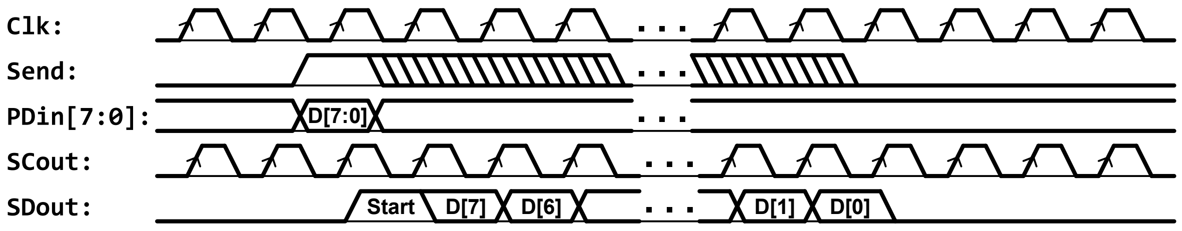
EE342 - Digital System Design

Laboratory Experiment - 5
2-Line Serial Transmitter/Receiver - Part 2

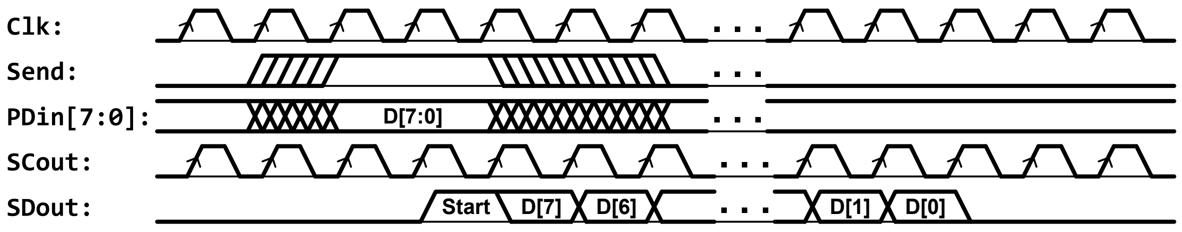
Preliminary Work

Make the following modifications in the two-line serial transmitter and receiver modules written in the previous experiment.

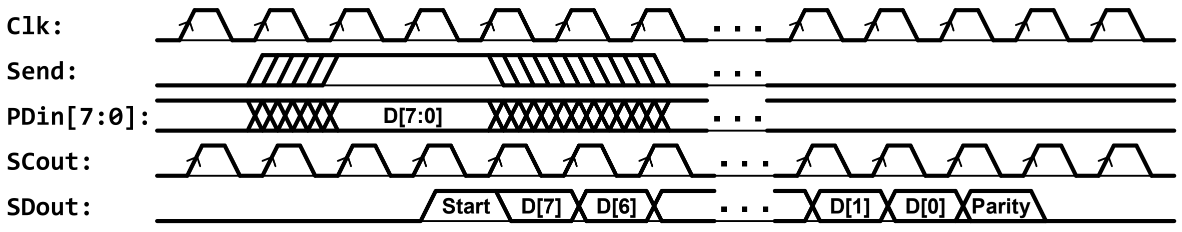
1. Long `Send` input pulse: Modify the serial transmitter so that it works when the **`Send`** input pulse is longer than one clock cycle as shown below. **`Send`** input is valid at the rising **`Clk`** edge after it is set to **1**, but it may remain at **1** for several clock cycles.



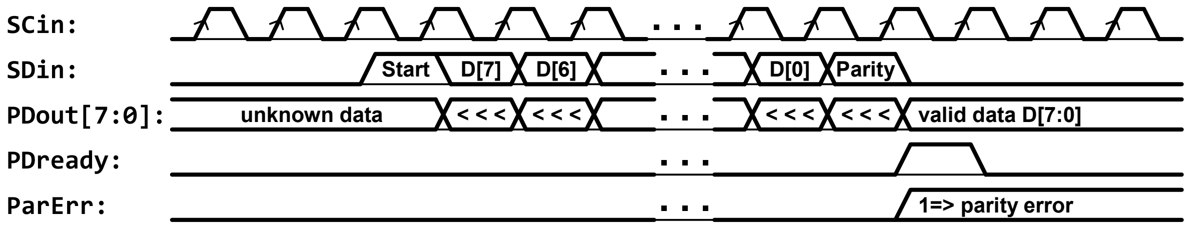
2. Asynchronous `Send` and parallel data inputs: Modify the serial transmitter so that it works when the **`Send`** and **`PDin[7:0]`** inputs can change at any time during a clock cycle independent of the **`Clk`** timing. Assume that **`Send`** and **`PDin[7:0]`** remain valid for at least two clock cycles.



3. Parity bit added to serial data output: Modify the serial transmitter so that an odd-parity bit is added at the end of the serial data sequence. The parity bit will be **1** when there are an odd number of 1's in the transmitted data and it will be **0** otherwise.



4. Serial receiver with parity check: Modify the serial receiver to check the parity bit received after the last data bit. The receiver module will have one more output, **`ParErr`**, that indicates the error condition. **`ParErr`** will remain at **0** when the parity result obtained from the received data matches the parity bit. In case of error, **`ParErr`** will be set to **1** simultaneously with the **`PDready`** output pulse and it will remain at 1 until the next data transmission.



Procedure

1. Use the Quartus II project and the top-level Verilog module written in the previous laboratory experiment to verify the new transmitter code modified in step-1 of the preliminary work. Check if the correct serial data output is obtained when the **`Send`** input is set to **1** for one clock cycle and for more than ten clock cycles.

2. Verify the transmitter code modified in step-2 of the preliminary work. Run simulation several times, changing the **`Send`** input timing so that a 0-to-1 transition occurs at different points relative to the rising edge of the clock input.

3. Verify the transmitter and receiver codes modified in step-3 and step-4 of the preliminary work. Add an output to the top-level module to connect the **`ParErr`** output of the receiver module. Check for parity error condition when there are odd and even number of 1's in the transmitted data.

Modify the transmitter code to produce a constant "1" parity result regardless of the data bits to force an error condition in the receiver module. Check the parity error detection in the receiver again when there are odd and even number of 1's in the transmitted data.