ECE 337: ASIC Design Lab 5

UART: A Generic Asynchronous Serial Interface

Week of 9/16/2013

Deadlines & Reminders

- Lab 5 Deadlines
  - Required Preparation Phase (Design planning)
    - Due by the end of the 3rd TA Office hours after your lab
  - Automated Design Grading
    - Due by the start of Lab 6
  - Postlab (Teamwork report)
    - Due by the start of Lab 6
- Project Idea submissions will be in 2 weeks
  - Start thinking of who you would like to work with
  - Start thinking of possible ideas

What is a UART?

- A general purpose serial communication interface.
  - Uses: computer serial ports, USB interfaces, most modems, infrared, and wireless interfaces.
  - Why? Lets you use one wire or channel for each direction.

Why Study UART Design?

- Serial interfaces are a very common pieces of ASIC and processor designs
  - Minimize pins on packaging
    - Minimizes packaging costs
    - Minimizes packaged chip size
  - Enable standard pin interfaces for a variety of ASICs

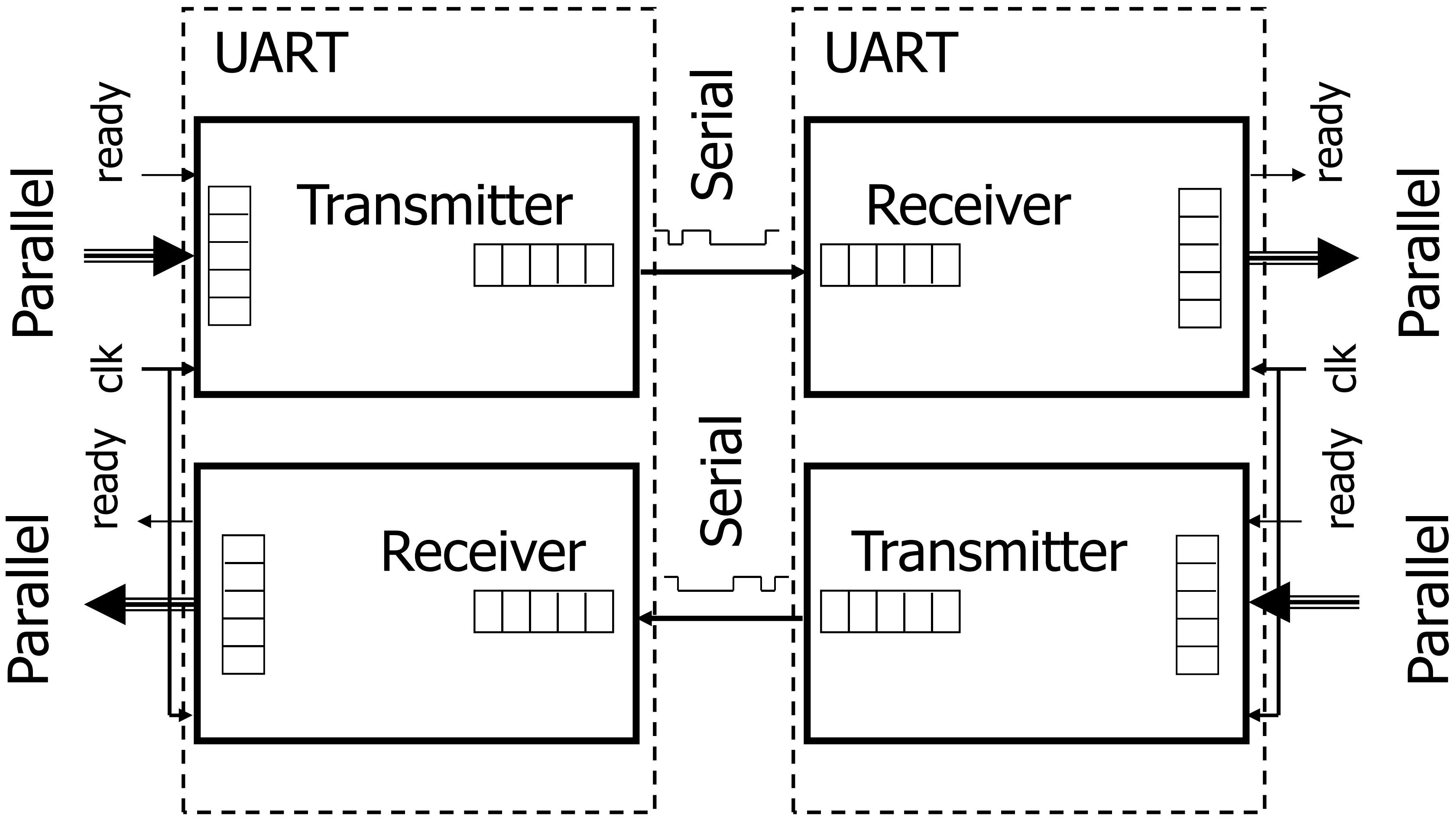
In lab

- Completing the design of a receiver portion of a UART
  - Universal Asynchronous Receiver Transmitter
  - Given most pieces of the design
    - Only have to make the controller, timer, and top-level blocks
    - Hint – the timer is just a counter or chain of counters
  - Given a starter test bench
  - No Gold model this time
- This design is a warm-up for lab 6 which is a “simplified” USB or I^{2}C serial bus interface

Major functions of a UART

- Receiver Block
  - Receives serial data, stores in parallel to register
  - Register is read by an external device
- Transmitter Block
  - External device loads data in parallel into register
  - Serially transmit the data
- Data is transmitted/received one byte at a time
  - Each byte handled separately

Top Level View

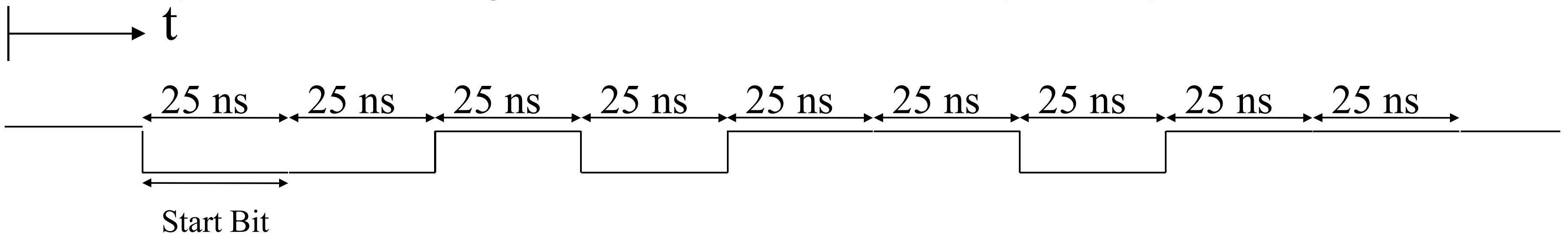


What makes it Asynchronous?

- Separate clock for sender & receiver
  - thus, incoming data not synchronized to local clock
- Serial input data can start at any time
- Only one guarantee regarding timing:
  - Duration of each bit will be uniform, within some tolerance

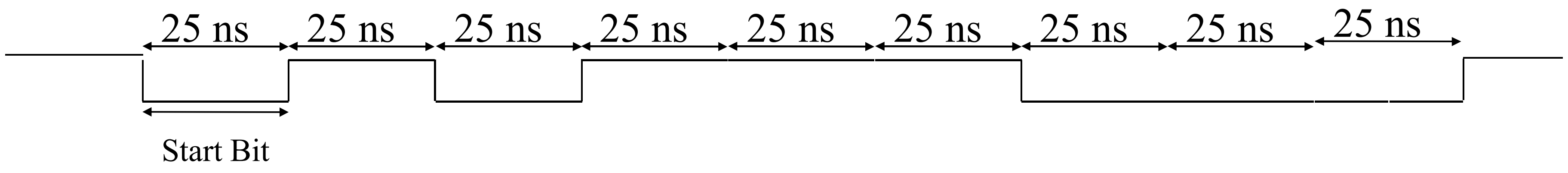
Serial Data Transmission

(note: a wide range of transmission rates are possible)



What is the data value here? Assume LSB first, high=1/low=0 (stop bit omitted)

a. 11011010 b. 10110100 c. 00011101 d. 01011011



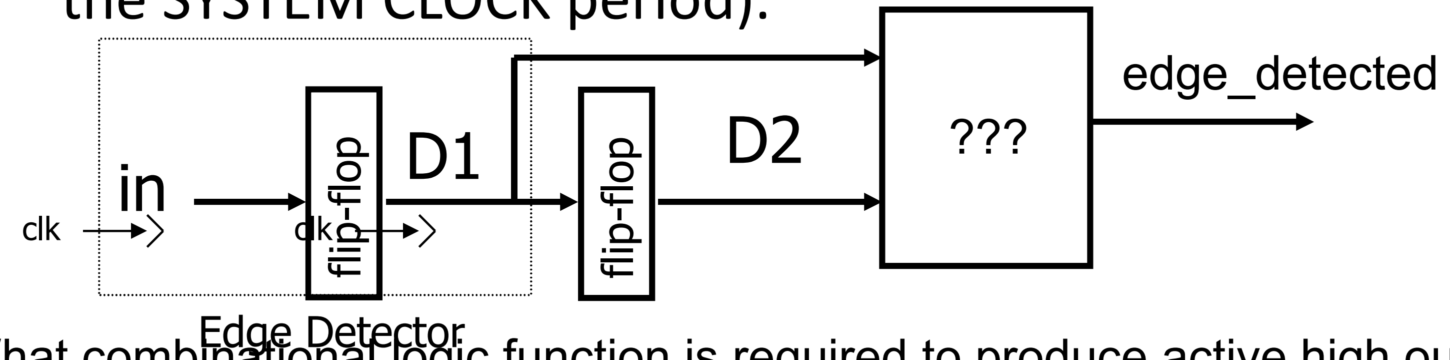
What is the data value here? Assume LSB first, high=1/low=0 (stop bit omitted)

Serial Data Transmission

- Try to choose the most accurate answer
- When not transmitting, what is a good value to output & why?
  a. A Logic ‘1’ – kind of like a dial tone on the phone – you can tell whether the line is live or not.
  b. A Logic ‘0’ – save power when not transmitting
  c. Doesn’t matter
- How do you know when there is incoming data?
  a. you see data transitions on the input
  b. you see a start bit

Detecting a start bit

- Start Bit for your UART is logic ‘0’
- Receiver Block must detect ‘1’ to ‘0’ transition, <u>WITHOUT</u> using *negedge property*.
  - Start Bit can, and will arrive asynchronously (at any point in the SYSTEM CLOCK period).

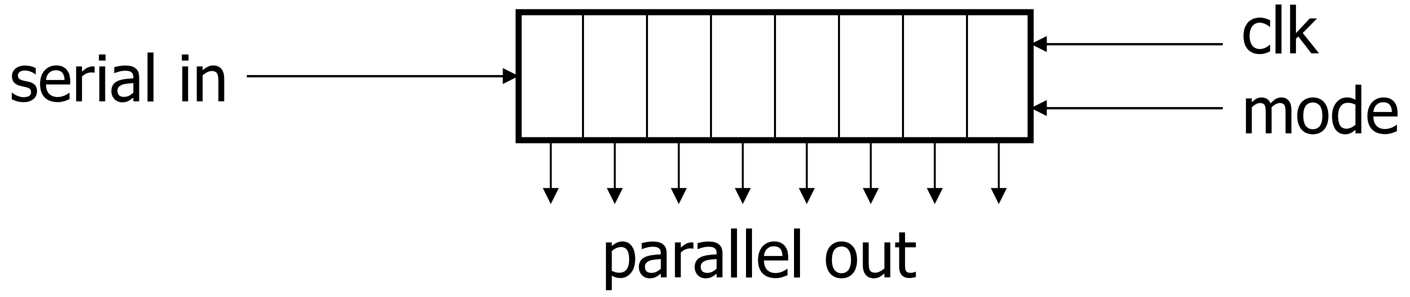


What combinational logic function is required to produce active high output when falling edge is detected?

Can this circuit also serve as a synchronizer?

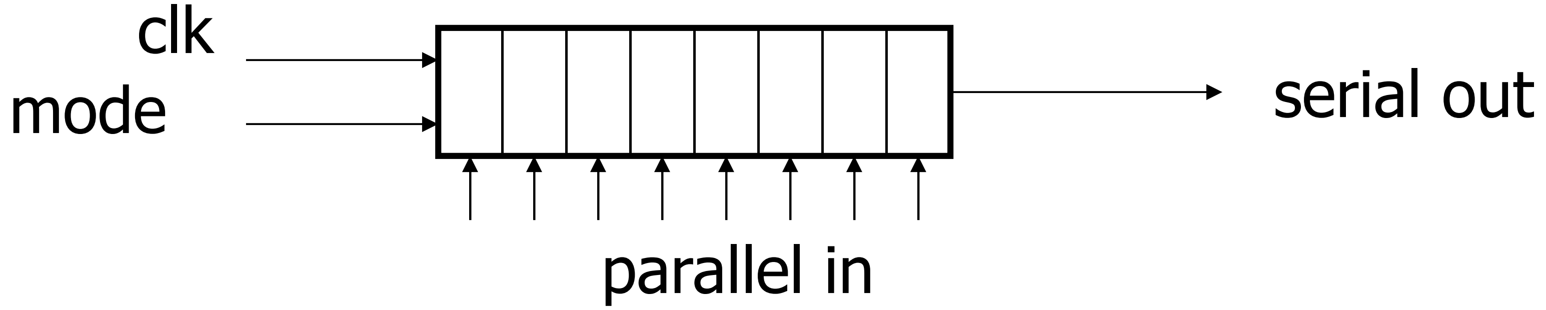
How do you read in serial data?

A shift register

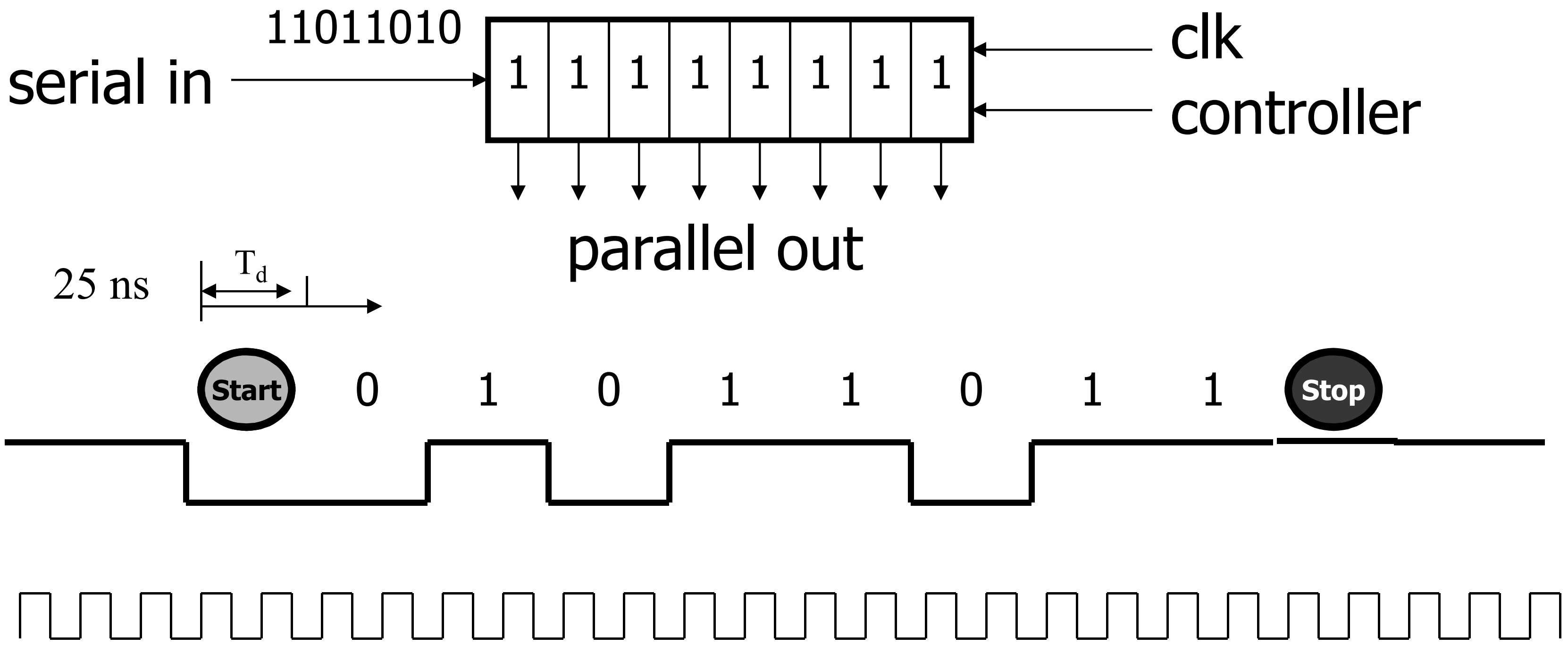


How do you send serial data?

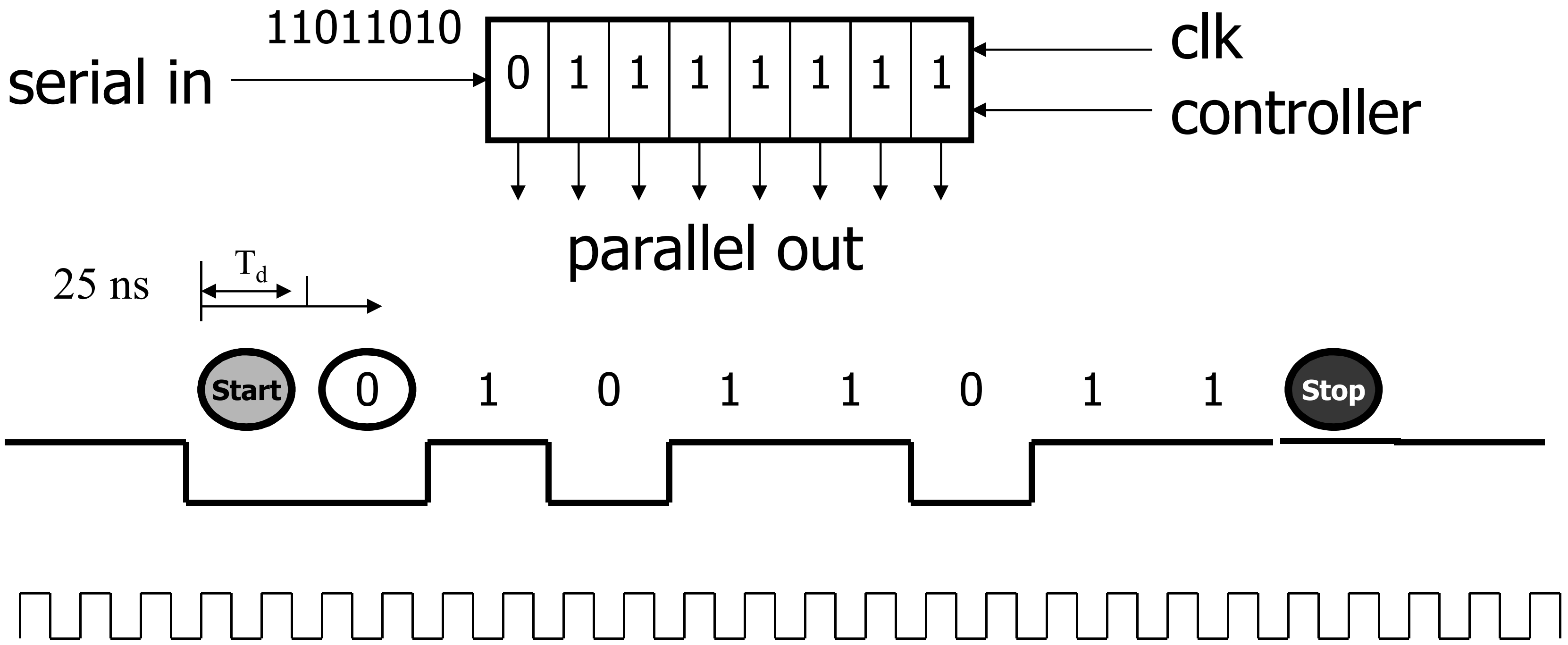
A shift register



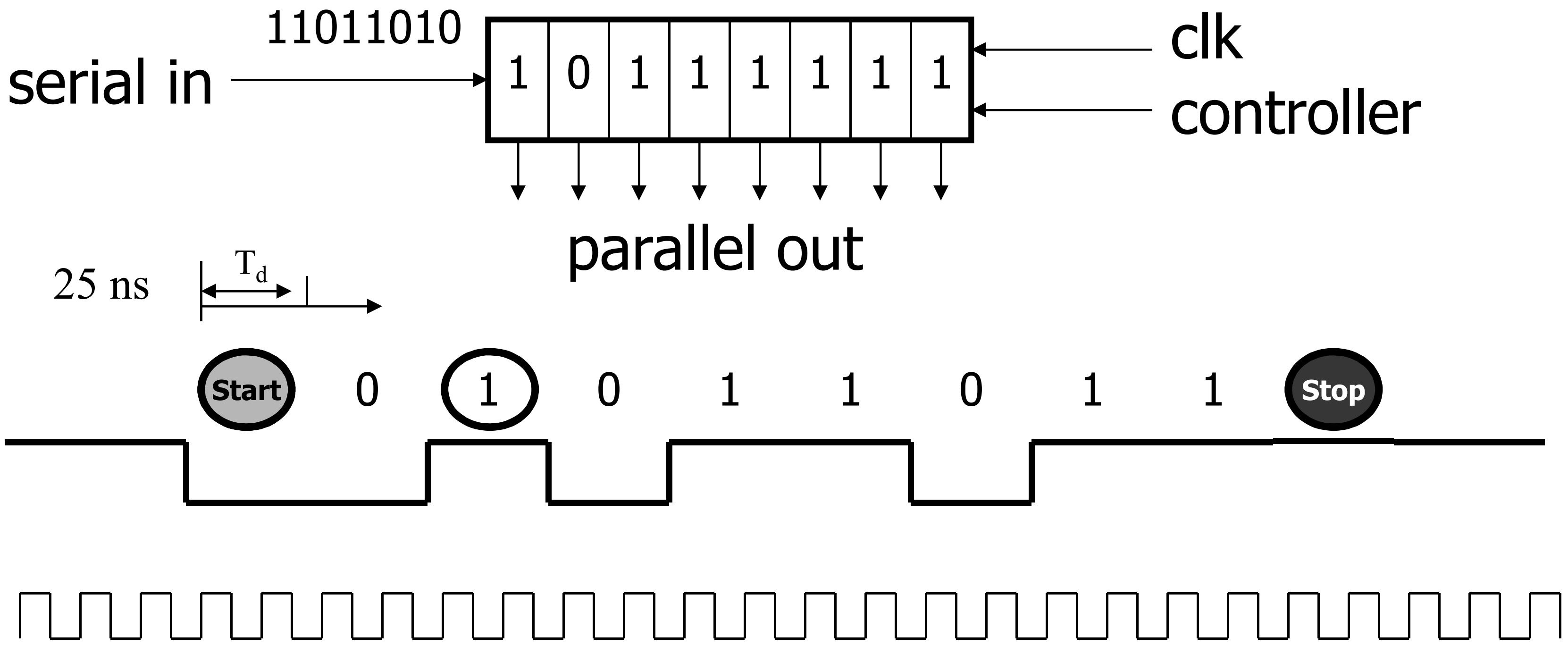
Shift Operation



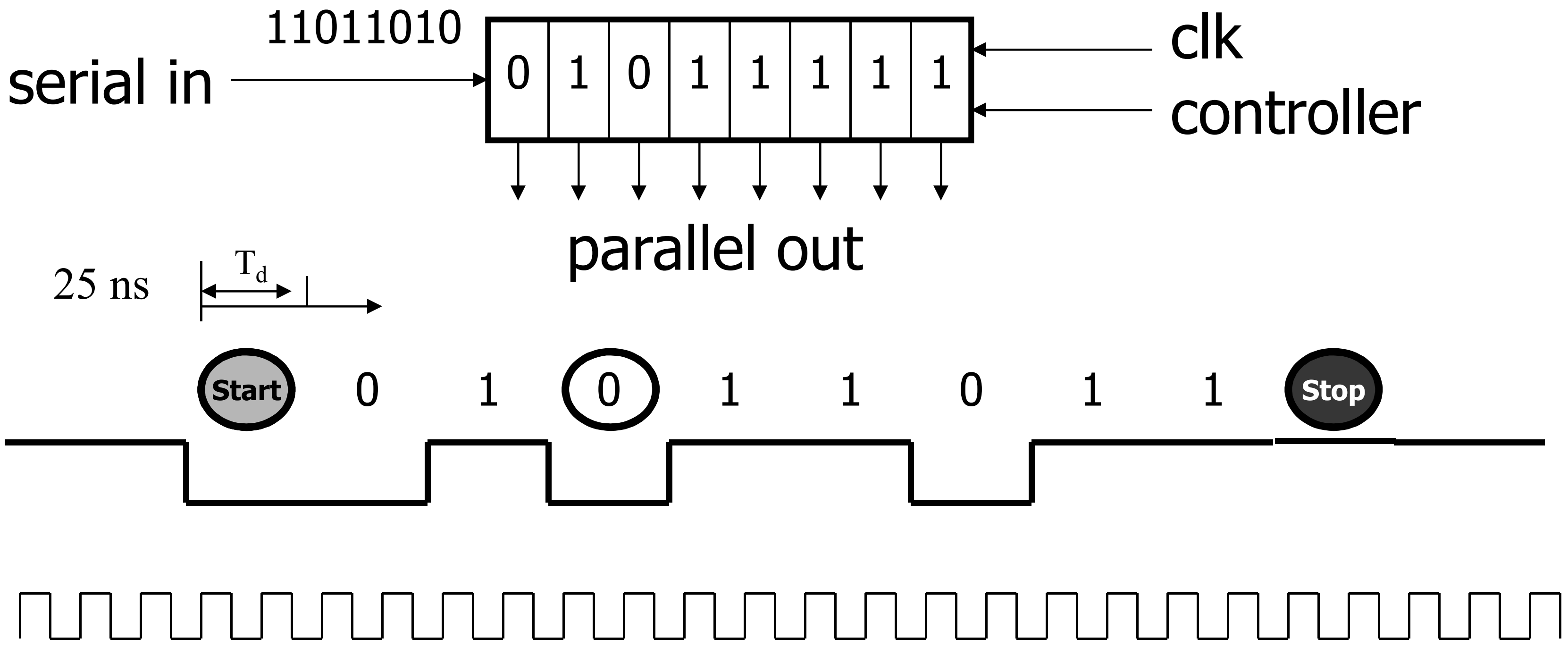
Shift Operation



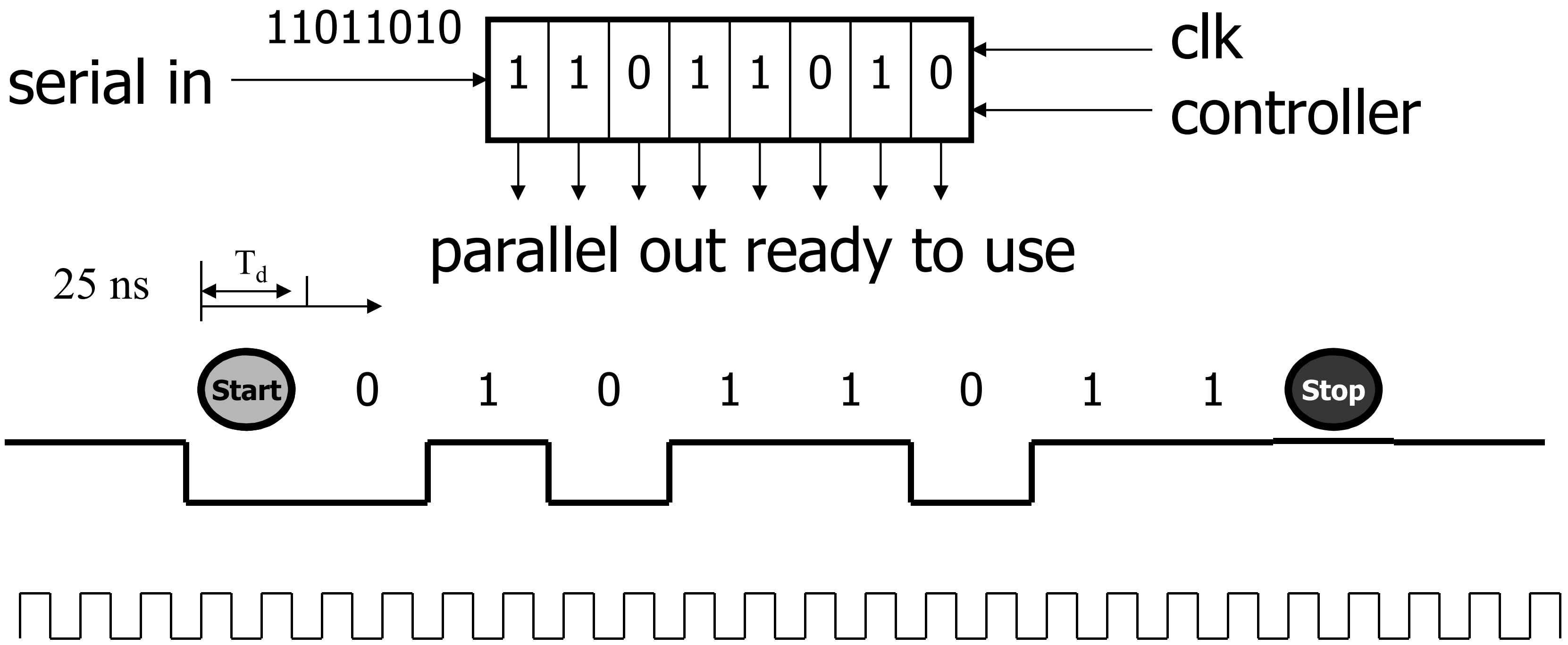
Shift Operation



Shift Operation



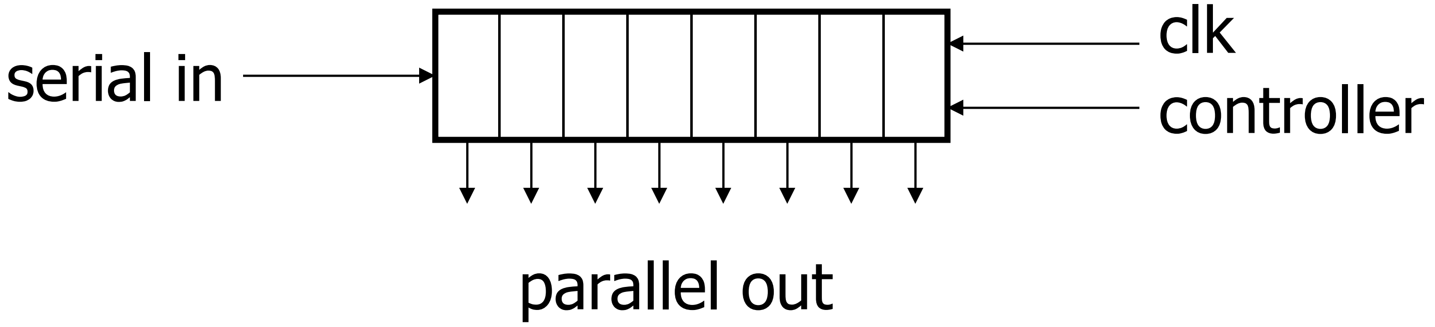
Shift Operation

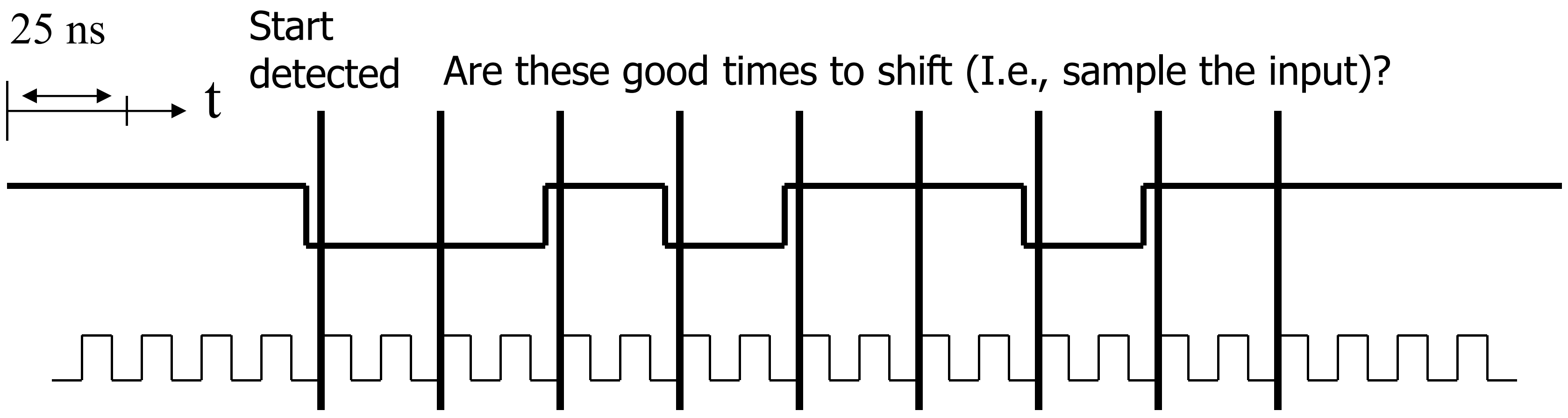


How should bits be shifted?

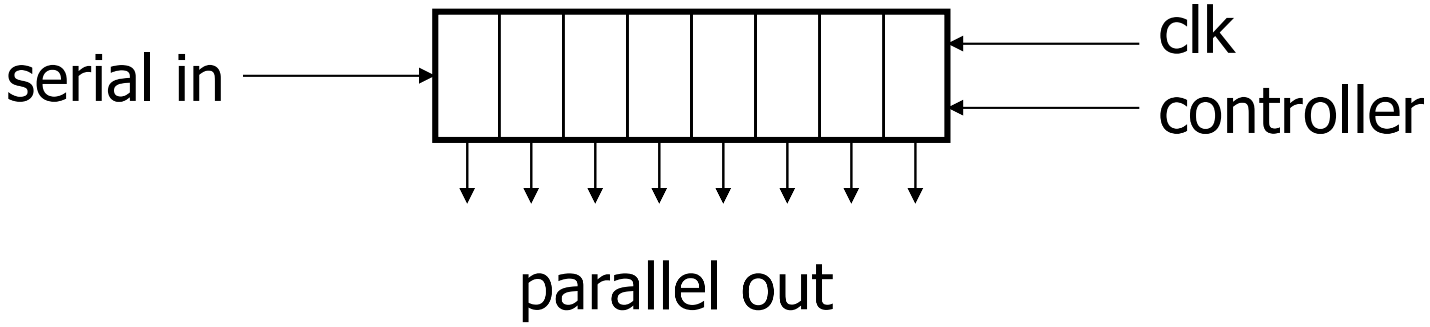
- One bit per system clock cycle?
  - Not unless system clock is synchronized to data and matches data rate
- We will use a control input to trigger shifting
  - You will create a timer block to generate pulses at the right time

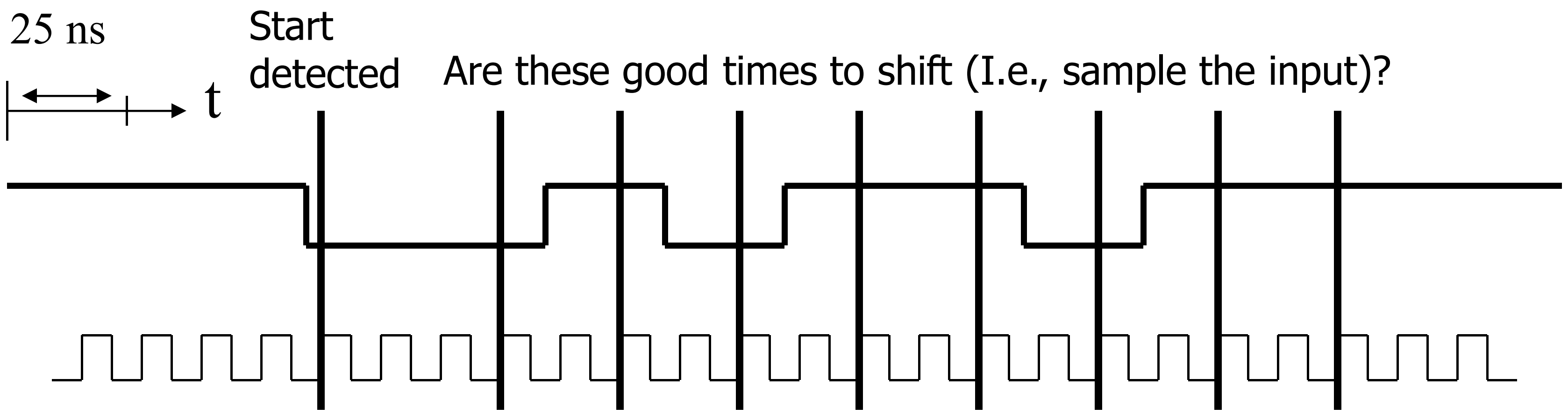
You decide when to shift



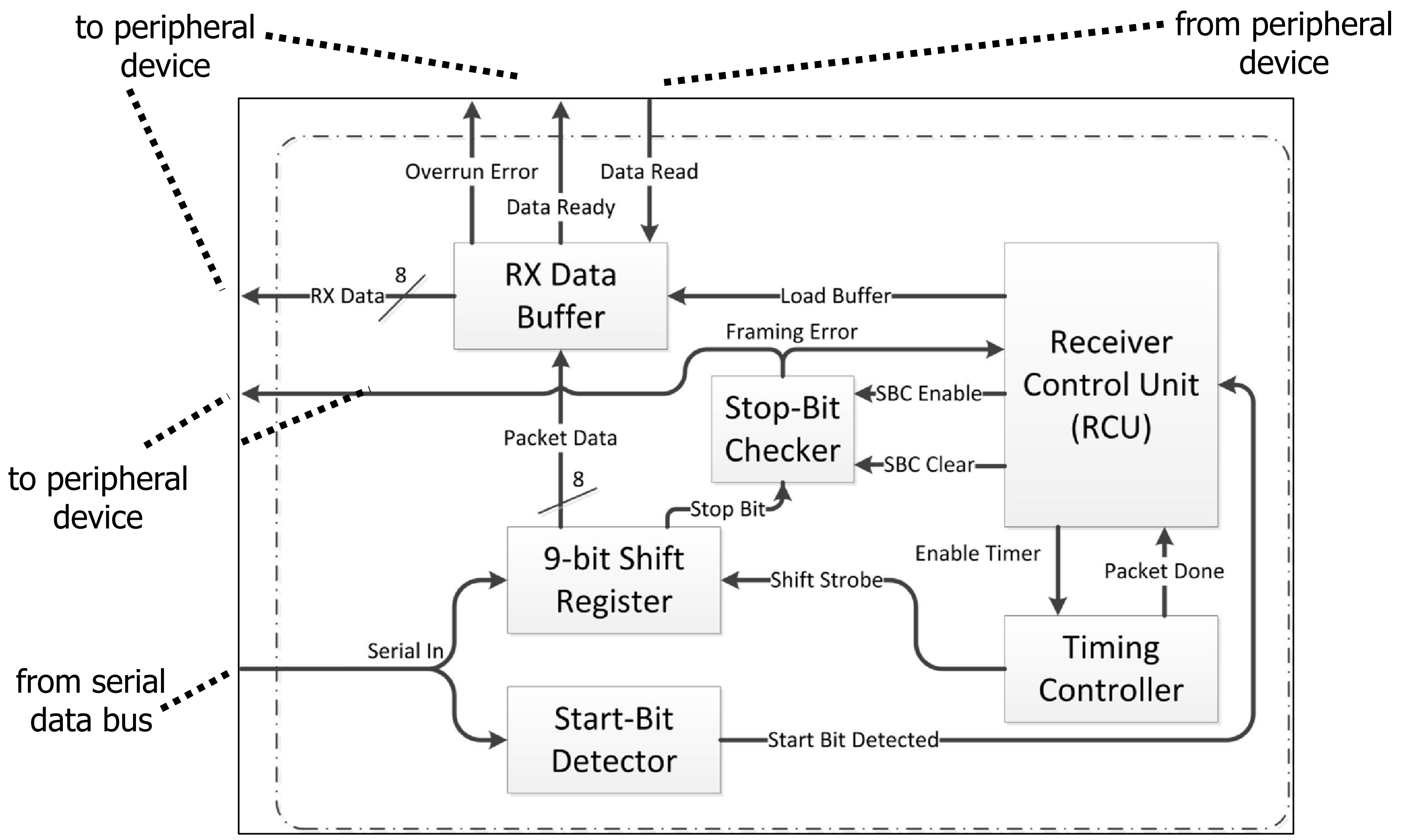


You decide when to shift





UART Receive Architecture



Sequence of Operation

1. Input line is idle (‘1’), receiver is waiting for start bit
2. Start bit (1->0 transition is detected)
3. Control unit enables timer
4. Timer enables shift register at appropriate clock cycles to sample each bit and stop bit
5. Timer signals control unit that all bits have been sampled
6. Control unit enables stop-bit checker.
    1. If there is an error, framing_error is asserted and receiver goes back to wait state

Sequence of Operation Continued

7. Control unit enables receiver buffer register to load data from shift register.
8. Peripheral device (or your test bench) should read data from the device and assert the data_read signal to inform the buffer that the data has been read

Some group discussion questions:

- How many times should the timer enable a right-shift after being triggered by the RCU?
- Approximately how many clock cycles should elapse between the time a start bit is detected and the time that the timer asserts the stop receiving signal?

Things you should check:

- Have you run LEDA?
- Have you checked for a varying data packet speed?
- Does your reset TRULY reset all flip-flops?
- Does your design meet the minimum input specs?
- Are your makefile variables filled in

More food for thought:

- Exhaustive testing is unnecessary
- Test for corner cases
- Ensure that all signals (input, output, and internal) change during your test bench
- Make sure your design can escape error conditions (without using the reset)

Making Debugging Easier

- Set the path length on signal names in wave view
- Saving/making .do files to setup waveforms
  - Make separate .do files for each subblock’s signals
    - Save/set the signals you need for each subblock
  - Save/set the radix and formatting for signals

Starter Test Bench Overview

- Has a task to send a packet to the DUT
  - Takes in the test data to send, the stop bit value, and the data bit period
  - Directly applies the serial data to the tb_serial_in variable
  - Implements the timing needed to send the specified values
- Tasks are good options to use for test bench designs
  - Use runs the same code for every use
    - No copy paste errors for encapsulated code
    - Helps organize/keep large test benches readable
- You will need to add more test cases only 2 are given

Recommended Expansions

- Create a new task to encapsulate the code need to run a test case
  - Use conditionals to allow the task to handle test cases with bad stop bits, and other error cases
- Add Test Cases for checking that all of your resets are working
  - including resetting the design while it’s receiving a packet
- Ask a TA if you have questions about the testbench