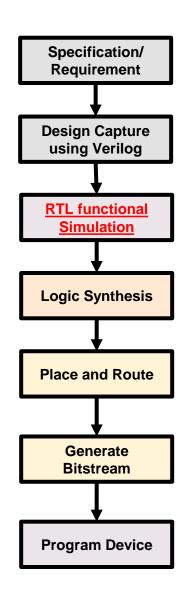
Overview UART

- Full UART Module with Parity
 - Implemented ASM chart.
 - 3 modules implemented and instantiated together.
 - Baud Rate Generator
 - Receiver
 - Transmitter
 - Implemented logic to combat metastability.
 - Implemented self-checking testbench

Method order and Goal

- 1. Specification/Requirements
- 2. Design Capture using Verilog
- 3. RTL functional Simulation
- 4. Logic Synthesis
- 5. Place and Route (Implementation)
- 6. Generate Bitstream
- 7. Program Device



Key Point-Overview Review: UART

•

Specification/Requirements

Baud Generator, Receiver, Transmitter

Baud Generator

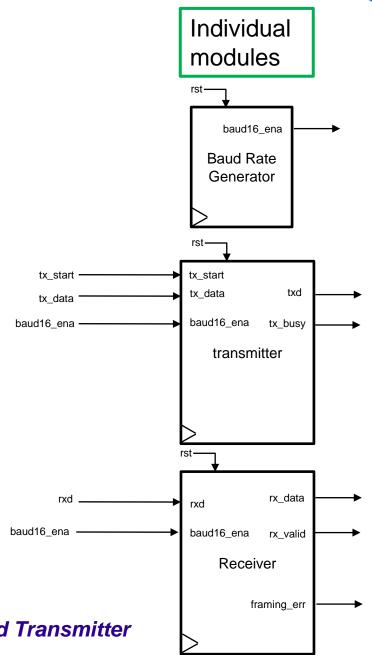
- Requirements/Specifications
 - To generate a sampling signal whose frequency is exactly 16 times the UART's designated baud rate from the system clock.
 - Account for any adjustments needed for an accurate count

Receiver

- Requirements/Specifications
 - Obtain data at a specific baud rate, oversampled x16
 - Implement false triggering detection, allowing to detect any Start Bit that is less than half-bit time as noise
 - Minimizing metastability
 - Implement a parity bit

Transmitter

- Requirements/Specifications
 - Shift register that loads data in parallel then shifts it out bit by bit at a specific rate(baud rate).
 - A bit is shifted out ,LSB first, every 16 enable ticks.
 - The receiver and transmitter must have the same baud rate as the baud rate generator
 - Implement a parity bit



Key Point – Specification/Requirements of Baud Generator, Receiver, and Transmitter

UART Receiver DESCRIPTION

What is UART Receiver?

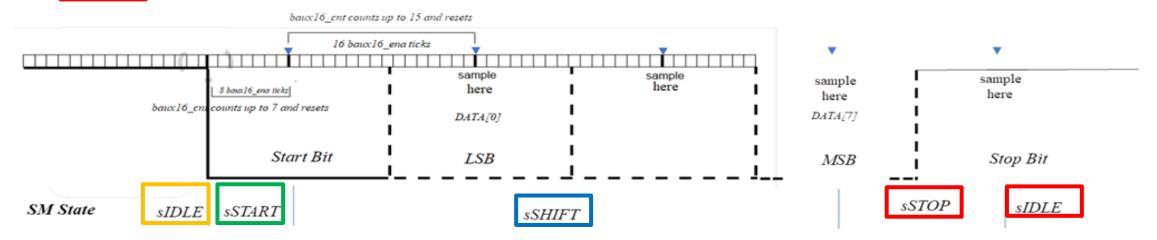
- UART (Universal asynchronous receiver and transmitter)
 - Includes a transmitter and receiver

Receiver

- Shift register that receives the data at a specific baud rate bit by bit via oversampling then reassembles the data.
- The receiver and transmitter must have the same baud rate → baud rate generator

States

- sIDLE: Active high(1), held in active high to show that the line and transmitter are not damaged.
- sSTART: Active low(0), to signals the receiver that new data is coming.
- sSHIFT: Receives the incoming data, LSB first, gets held in a shifting register.
- sSTOP: Signals to the receiver that the incoming data is finished.



UART Transmitter DESCRIPTION

What is UART Transmitter?

- UART (Universal asynchronous receiver and transmitter)
 - Includes a transmitter and receiver
 - **Transmitter** (The focus of the presentation)
 - Shift register that loads data in *parallel* then shifts it out bit by bit at a specific rate(baud rate).
 - A bit is shifted out ,LSB first, every 16 enable ticks.
 - The receiver and transmitter must have the same baud rate → baud rate generator
 - States
 - sIDLE: Active high(1) txd, awaits tx_start signal to load data word.
 - sWAIT: Waits one clock cycle before transitioning to Active Low(0) txd.
 - sSTART: Active low(0) txd, to signal the receiver that new data is coming, transmit first bit(LSB) in the data register.
 - sSHIFT: Transfers the data bit by bit every 16 enable ticks.
 - sSTOP: Transmits and active low(0) txd signal for 1 clock cycle.

Clock Improvement with Fractional Adjustment for Baud Rate Generator

- Without Adjustment (50MHz) 115200 baud rate
 - Rounded Value
 - Period = 27 x (20ns)= **540ns**
 - Oversampled $16x = (540ns) \times 16 = 8640ns$
 - Real Value
 - 27.126736 x (20ns) x 16=~**8680ns**
 - Absolute error
 - 8680ns 8640ns = **40ns** (A difference of **40ns**)
- With Adjustment (baudx16_ena period is extended by one 50MHz (20ns) clock every 8th baudx16_ena)
 - Oversampled adjusted value
 - 14x(27x20ns) + 2(28x20ns) = 8680ns
 - Absolute error
 - 8680ns-8680ns = ~ 0
 - The Error bit-time is near zero

Every 8th time the
baud generator
sends a pulse, it
has one extra to
adjust the clock
for the baud rate

Fractional Adjustment	Common Baud rates bits/second
→ 1/8 = 0.125	115200 : 27.126 ←
2/8 = 0.250	57600 : 55.253
3/8 = 0.375	38400 : 81.380
4/8 = 0.500	28800 : 108.507
5/8 = 0.625	230400 : 13.563
6/8 = 0.750	19200 : 162.760

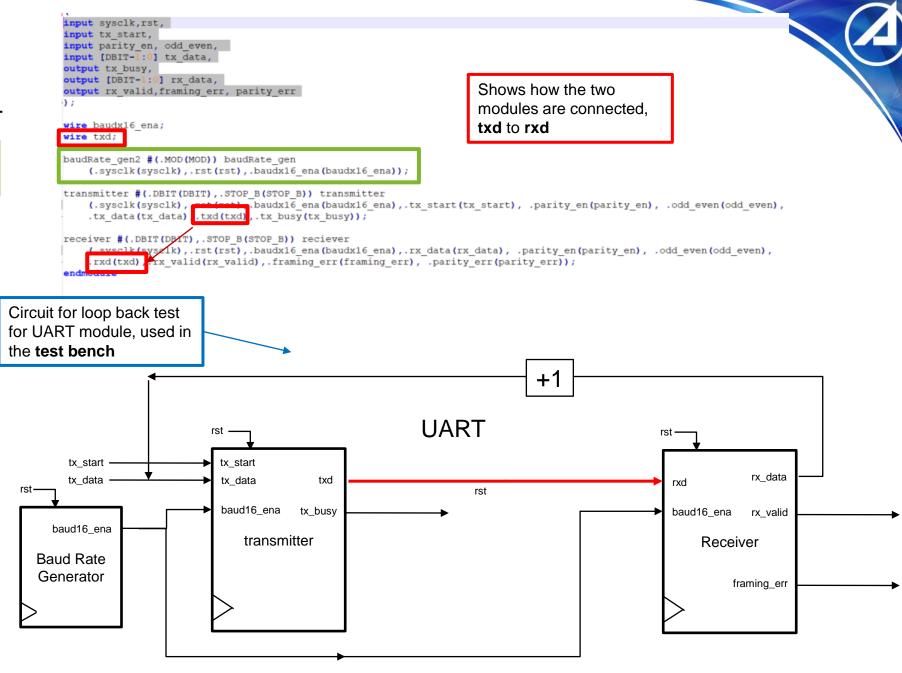
(System clock) divided by (desired baud rate)

Key Point – Clock Improvement for Baud Rate Generator

UART Instantiation

Structural Description

- Module instantiation of UART
 - Sharing the same baud rate generator
 - txd → rxd
 - Purpose:
 - Combining individual modules into one.
- Variables:
 - rst: resets the circuit
 - baud16_ena: sends a pulse every 16 count, responsible for the baud rate
 - tx_start receives a 1-clock-wide pulse to initiate transmission
 - tx_data: receives the data being transmitted
 - txd: output of the transmitter, sends data serially to the receiver
 - tx_busy: no new data for the input of the transmitter if high, only accepts new data when low.
 - rxd: receives bits
 - rx_data: outputs the 8 data bits
 - rx valid: UART received data valid
 - framing_err: for a framing error



Metastability

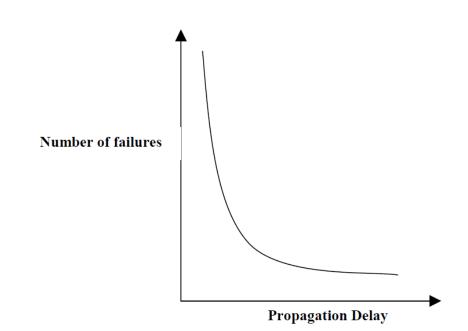
Brief overview

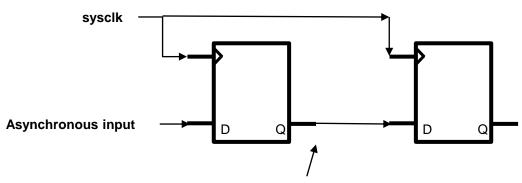
Characteristics

- A probabilistic phenomenon (random future events)
 - Can not build a bit-stable device that cannot go into a metastable state
- Occurs when two asynchronous signals combine such that their resulting output goes into an indeterminate state
- Can remain in a metastable state for an indeterminant amount of time.
 - Exponentially decreasing function, it will stay in the metastable state.

Solution

- D-type Flip Flop in series, on the same system clock.
- Increases the amount of time for the meta-state to be resolved
- Minimizes the probability that the metastability entering the main part of the circuit.





Note: in this design rxd_q[] represents the synchronizer

Key Point - Metastability

Block Diagram of parity_bit

Block diagram, Truth table

- **odd_even** decides if the **parity_bit** is going to be calculated with an odd or even configuration.
 - parity_bit will be a 1 or 0 depending on the amount of data bits
 - odd_even = 0: even parity (keep an even number of 1's)
 - odd_even = 1: odd parity (keep an odd number of 1's)
- Calculated the same with transmitter(thr[0]) or receiver (rxd_q[1])
- Purpose
 - Adding a parity bit is a form of error detection. Compares the parity from the receiver and transmitter. If they are both the same, no error, if they are different an error has occurred.

(4)bits

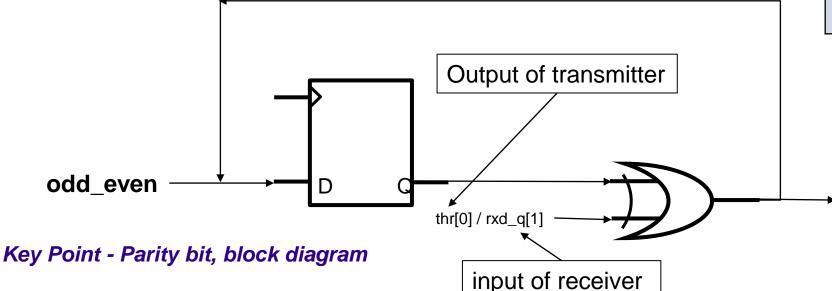
0 0 1

1 0 1

1 1 0

0 0 1

Data



Parity bit:

Even

- Even : parity bit ← 0

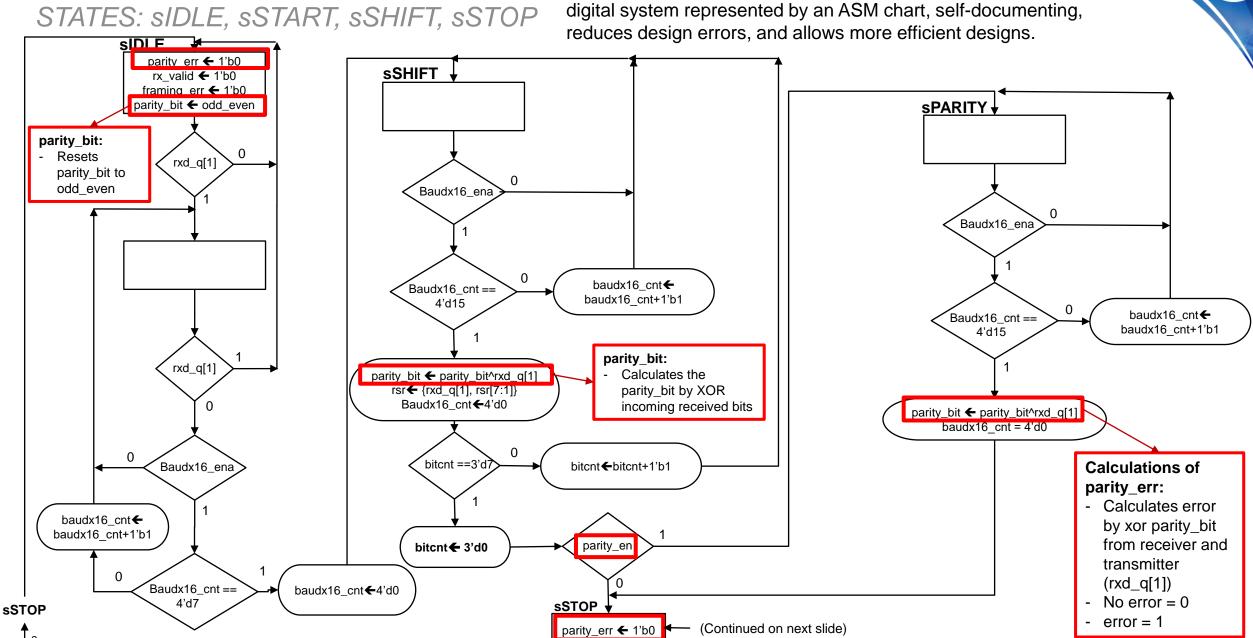
Odd

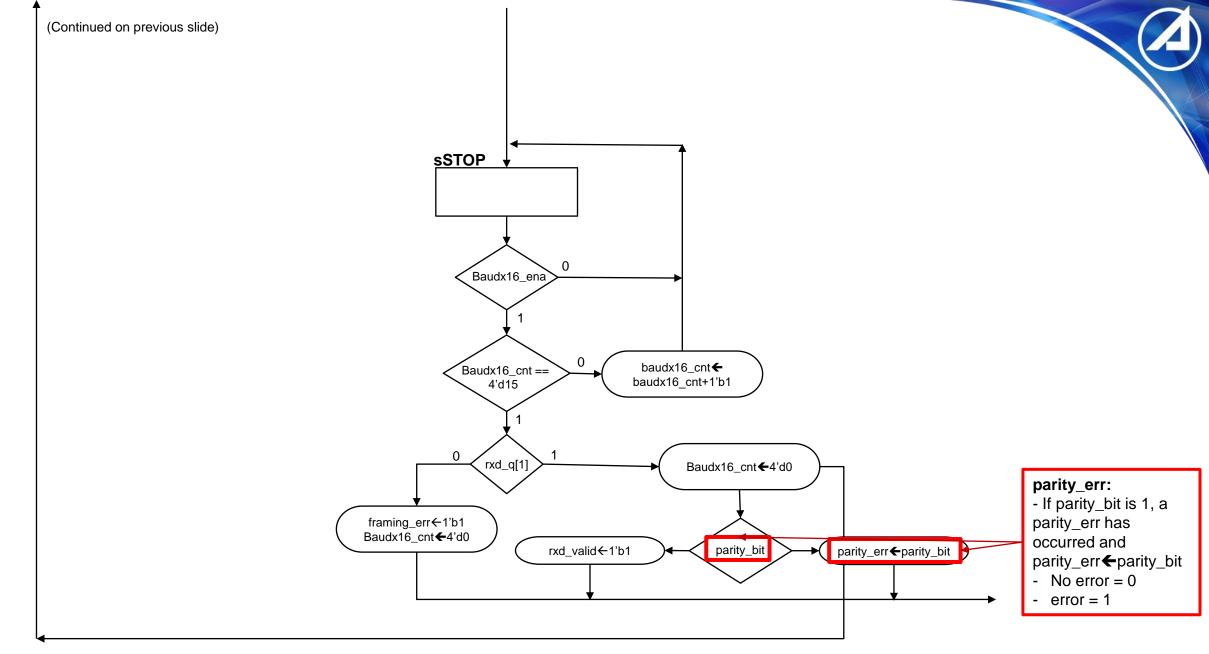
- Odd : parity bit ← 1

parity_bit

ASM CHART RECEIVER

Purpose: The ASM chart is part of the design methodology used. A powerful tool that makes easy to understand the operation of a digital system represented by an ASM chart, self-documenting, reduces design errors, and allows more efficient designs.

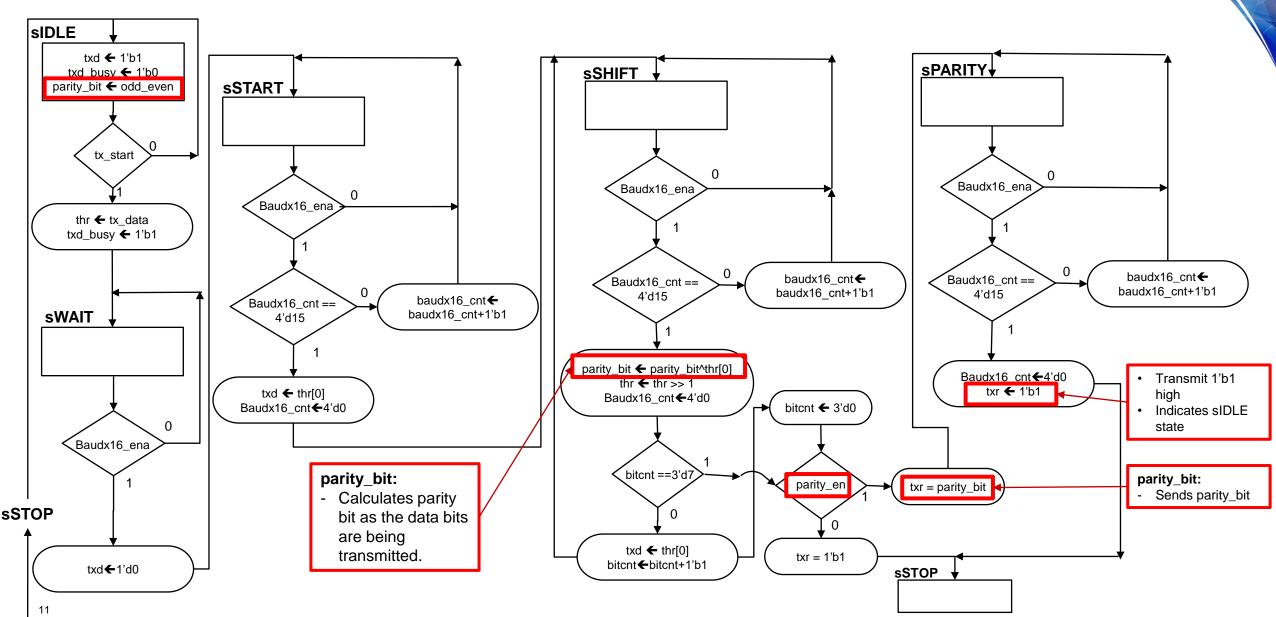


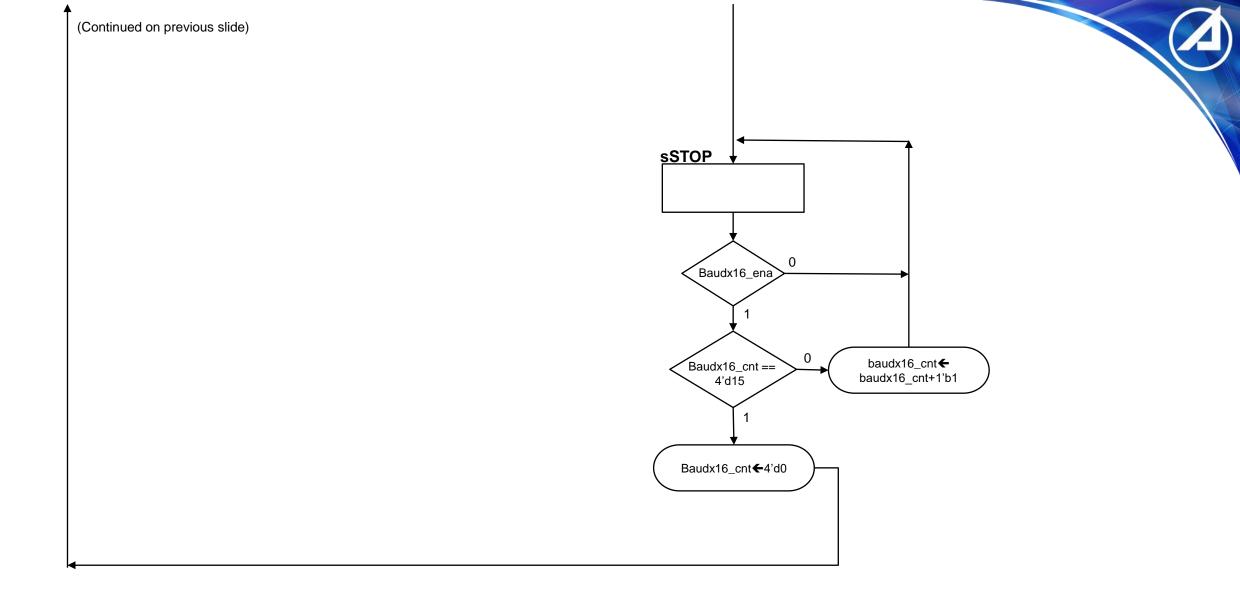


ASM CHART: RECIEVER cont.

ASM CHART Transmitter

STATES: sIDLE, sWAIT, sSTART, sSHIFT, sPARITY, sSTOP





SELF-CHECKING LOOP BACK TESTBENCH



Iterative

```
A Task that sends data to the transmitter
task sw test;
    integer i, j;
    begin
                                         Iterating through 0-255
        i = 0;
                                                                                                                                      task UART READ,
                                                                                                                                      input [DBITS-1:0] data;
                                                                                                                                          begin
            begin
                                                                                                                                          tx_start = 1'b1;
                UART READ(rx data+1)
                                                                                                                                          tx data = data;
                #(CLK BIT PERIOD*12);
                                                                                                                                          #T;
                if((tx_data) == rx_data)
                                                                                                                                          tx start = 1'b0;
                    $display(" Received! Input to transmitter = 0x%0h, Output of reciever = 0x%0h",tx data,(rx data));
                else
                    begin
                                                                                                                                      endtask
                    $display(" Not Received! Input to transmitter = 0x%0h, Output of reciever = 0x%0h",tx data,(rx data));
                    j = j+1;
                    end
            end
            $display(" Errors counted %0d out of %0d test cycles",j,i);
    end
```

Purpose: the self-checking loop back test sends data from the receiver back to transmitter while iterating through all possible combinations for 8-bits, 255 not counting 0, different values. The Self-checking happens by an output in the terminal, letting the user know if the data is received and how many errors if any.

```
Received! Input to transmitter = 0xf3, Output of reciever = 0xf3
Received! Input to transmitter = 0xf4, Output of reciever = 0xf4
Received! Input to transmitter = 0xf5, Output of reciever = 0xf5
Received! Input to transmitter = 0xf6, Output of reciever = 0xf6
Received! Input to transmitter = 0xf7, Output of reciever = 0xf7
Received! Input to transmitter = 0xf8, Output of reciever = 0xf8
Received! Input to transmitter = 0xf9, Output of reciever = 0xf8
Received! Input to transmitter = 0xf9, Output of reciever = 0xf9
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
Received! Input to transmitter = 0xf0, Output of reciever = 0xf0
```

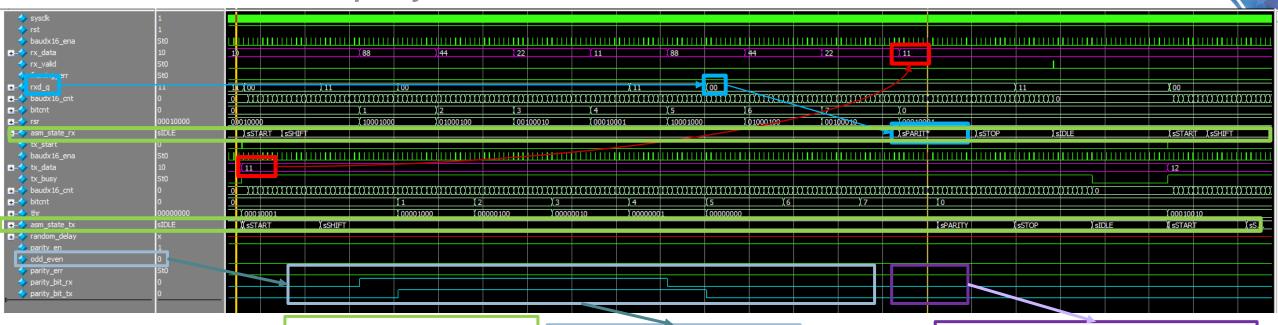
End portion of the test, iterating 0xf3 → 0xff

No errors out of 255 test cycles

endtask

Self-Checking Loop Back Test Waveform: EVEN PARITY

Iterative, Full Wave Form, parity_en = 1, odd_even = 0



The input to the transmitter and output of the receiver

- tx_data (Bottom)
- rx_data (Top)

- Receiver(top) States:
 - sIDLE
 - sSTART
 - sSHIFT
 - sPARITY
 - sSTOP
- Transmitter(bottom) States:
 - sIDLE
 - sWAIT
 - sSTART
 - sSHIFT
 - sPARITY
 - sSTOP

- parity_bit calculations
- parity_bit 0

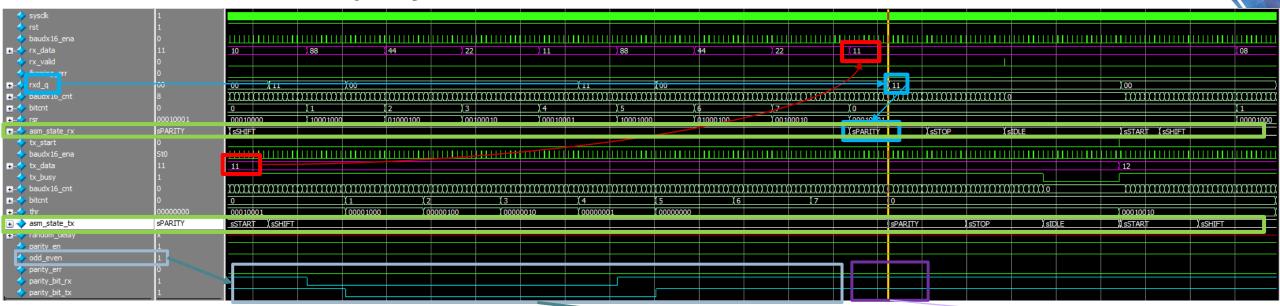
Description: shows the wave form for when the **parity_en =1** and has an **even** configuration. The results are that the circuit is working just as expected

- Calculated parity_err
 - parity_err ← 0
 - parity_bit ← 0
 - 0x11 should have a **0** parity bit for **even** configuration

Synchronizer received bit, rxd_q = 00, which is expected with an **even** parity configuration

Self-Checking Loop Back Test Waveform: ODD PARITY

Iterative, Full Wave Form, parity_en = 1, odd_even = 1



The input to the transmitter and output of the receiver

- tx_data (Bottom)
- rx_data (Top)

- Transmitter(bottom) States:
 - sIDLE
 - sWAIT
 - sSTART
 - sSHIFT
 - sPARITY
 - sSTOP
- Receiver(top) States:
 - sIDLE
 - sSTART
 - sSHIFT
 - sPARITY
 - sSTOP

- parity_bit calculations
- parity bit **←** 1

Description: shoes the wave form for when the **parity_en =1** and has an **odd** configuration. The results are that the circuit is working just as expected

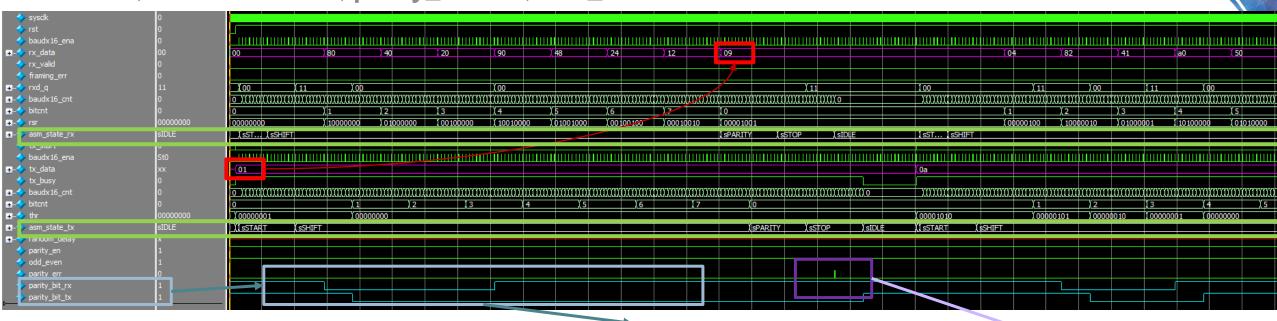
- Calculated parity_err
 - parity_err ← 0
 - parity_bit 1
 - 0x11 should have 1 parity bit for odd configuration.

Synchronizer received bit, rxd_q = 11, which is expected with an **odd** parity configuration

Self-Checking Loop Back Test Waveform: ODD PARITY ERROR TEST



Iterative, Full Wave Form, parity_en = 1, odd_even = 1



The *input* to the <u>transmitter</u> and *output* of the <u>receiver</u>

- tx_data (Bottom)
- rx_data (Top)

- Receiver(top) States:
 - sIDLE
 - sSTART
 - sSHIFT
 - sPARITY
 - sSTOP
- Transmitter(bottom) States:
 - sIDLE
 - sWAIT
 - sSTART
 - sSHIFT
 - sPARITY
 - sSTOP

- parity_bit calculations
 - parity bit 1 (error)

Description: shows the wave form for when the **parity_en =1** and has an **odd** configuration. An **error** is purposely put in the design to show that the **parity_err** will pulse. Everything in the circuit is working as expected

- Calculated parity_err
 - parity_err ← 1 for 1 pulse.
- parity_bit ← 1 (error)
 - 0x01 should have 0 parity bit for Odd configuration.
- If parity_bit ←1 , then rx_valid ←0

Self-Checking Loop Back Test Waveform: parity_en ← 0

Iterative, Full Wave Form, parity_en = 0, odd_even = 1



parity_en ← 0

- Receiver(top) States:
 - sIDLE
 - sSTART
 - sSHIFT
 - sSTOP
- Transmitter(bottom) States:
 - sIDLE
 - sWAIT
 - sSTART
 - sSHIFT
 - sSTOP

The input to the transmitter and output of the receiver

- tx_data (Bottom)
- rx_data (Top)

description:

- With parity_en ←0, skips the sPARITY state completely for both the transmitter and receiver.
- Still calculates parity_bit but the transmitter doesn't send the parity_bit.