UART

Rishabh Dubey EE [16973] M-tech

TABLE OF CONTENTS

UART

Rishabh Dubey EE [16973] M-tech

1. Introduction

The Universal Asynchronous Receiver Transmitter (UART) module is one of the serial I/O modules. The UART is a full-duplex asynchronous system that can communicate with peripheral devices, such as personal computers, RS-232 and RS-485 interfaces.

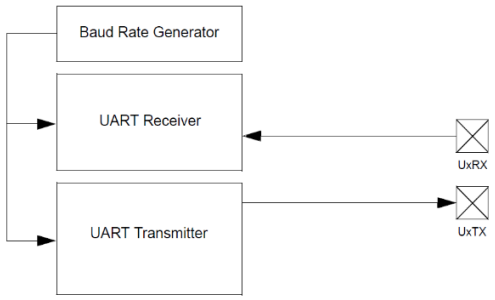
The primary features of the UART module are:

- Full-duplex 8- or 9-bit data transmission through the UTx and URx pins
- Even, Odd or No Parity options (for 8-bit data)
- One or One and a Half or two Stop bits
- Fully integrated Baud Rate Generator with 4 Standard Baud Rates.
- Baud rates {2400 ,4800, 9600, 19200} at Fsys = 50 MHz's
- 4-deep First-In-First-Out (FIFO) transmit data buffer
- 4-deep First-In-First-Out (FIFO) receive data buffer
- Parity, Buffer Overrun error detection
- Support for 9-bit mode
- Transmit and Receive Interrupts
- Loopback mode for diagnostic support
- Address Bus
- Bidirectional Data Bus

A simplified block diagram of the UART is shown in Figure-1.
The UART module consists of the key important hardware elements:

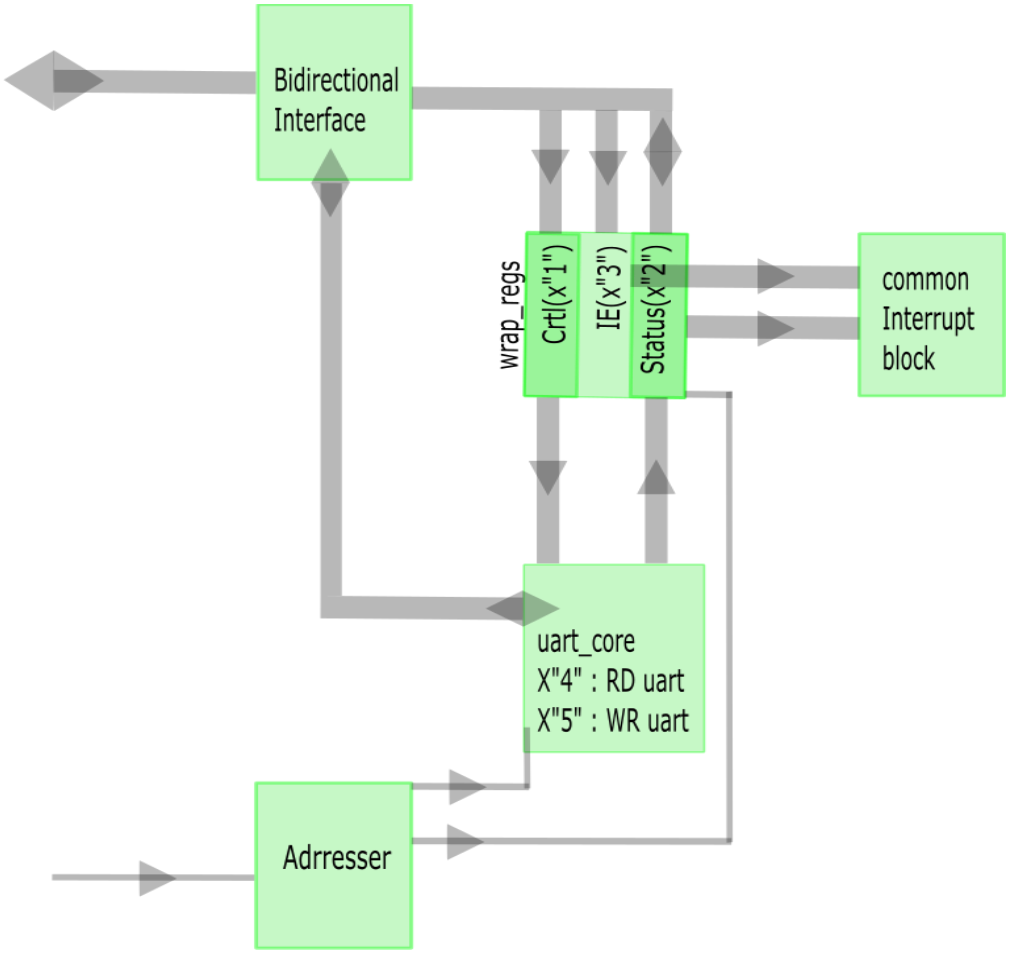
- Baud Rate Generator
- Asynchronous Transmitter
- Asynchronous Receiver

Figure 1: **UART Basic Block Diagram**



2. UART Module Bus Map

Uart Module has a 9 bit wide Bi-Directional Bus Interface for word write of octet write, The Ctrl , IE registers i.e. control and Mask register are write only status register is a Read / Write register as the Figure 2 Depicts, UART core also has Read write operation. Addresser is the module which ensures that not to modules are configured or read at the same time. Addresser takes the Offset of these registers and Uart-core, could be found in Uart-register Descriptions also shown in Diagram.



Bus Map of The UART module with its wrapper registers

Figure 2: UART Module Bus Map

3. Register Description

3.1 Register 1: Control Registers

Offset Address	x"1"
Register Type	WR
Reset Value	0x00

LOOP_BACK	STOP_BITS <1:0>	BAUD_SEL <1:0>	PARITY_TYPE<1:0>	D_BITS
Bit 7				Bit 0
WR	WR	WR	WR	WR

bit 7 **LOOP_BACK:**

UART Loopback Mode Select bit

1 = Enable Loopback mode

0 = Loopback mode is disabled

Note: Write '1' on Rx_Module Pin

bit 6-5 **STOP_BITS <1:0>:**

Stop Selection bit

10 = 2 Stop bits

01 = 1.5 Stop bits

00 / 11 = 1 Stop Bit

bit 5-4 **BAUD_SEL <1:0>:**

Baud rate bits = Baud Rate

11 = 2400

10 = 4800

01 = 9600

00 = 19200

Note: Fsys = 50 MHz's

bit 3-2 **PARITY_TYPE<1:0>:**

Parity Type bits

10 = 8-bit data, odd parity

01 = 8-bit data, even parity

00 / 11 = 8-bit data, no parity

bit 0 **D_BITS:**

Data Selection bits

1 = 9-bit data, no parity

0 = 8-bit data w/ or w/o parity

Legend:		
RD = Readable bit	WR = Writable bit	Reserved = Unimplemented bit, read as '0'
'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

3.2 Register 2: Status Registers

Offset Address	x"2"
Register Type	RD/WR
Reset Value	0x00

Reserved	RX_FIFO_INVALID_RD_WR	TX_FIFO_INVALID_RD_WR	PARITY_ERROR
Bit 7			Bit 4
	WR/RD	WR/RD	WR/RD

TX_OV_STAT	TX_NULL_STAT	RX_OV_STAT	RX_NULL_STAT
Bit 3			Bit 0
WR/RD	WR/RD	WR/RD	WR/RD

Bit 7 **Reserved:**

Write '0' to this location zero only default it will be even if programmer try to write 1 over it by mistake it will be zero only

Bit 6 **RX_FIFO_INVALID_RD_WR:**

Shows Status if any invalid read or write is asserted by program or some unknown value come Receiver FIFO

Bit 5 **TX_FIFO_INVALID_RD_WR:**

Shows Status if any invalid read or write is asserted by program or some unknown value come Transmitter FIFO

Bit 4 **PARITY_ERROR**:

Module will set this status '1' when ever a parity error will occur be it even parity or odd parity.

Bit 3 **TX_OV_STAT**:

Module will set this status '1' when a Transmitter FIFO is Full, and no more write can be done

Bit 2 **TX_NULL_STAT:**

Module will set this status '1' when a Transmitter FIFO is empty, and no more data to send.

Bit 1 **RX_OV_STAT**:

Module will set this status '1' when a Receiver FIFO is Full, and no more data can be written over it.

Bit 0 **RX_NULL_STAT:**

Module will set this status '1' when a Receiver FIFO is empty, and no more data to read.

Legend:		
RD = Readable bit	WR = Writable bit	Reserved = Unimplemented bit, read as '0'
'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

3.3 Register 3: Interrupt Enable Registers

Offset Address	**x"3"**
Register Type	**WR**
Reset Value	**0xFF**

ECI	RX_FIFO_INVALID_RD_WR_IE	TX_FIFO_INVALID_RD_WR_IE	PARITY_ERROR_IE
Bit 7			Bit 4
	WR	WR	WR

TX_OV_IE	TX_NULL _IE	RX_OV _IE	RX_NULL _IE
Bit 3			Bit 0
WR	WR	WR	WR

Bit 7 **ECI:**

Enable Common Interrupt -> Write '0' to this location to disable UART interrupt

Bit 6 **RX_FIFO_INVALID_RD_WR_IE:**

Write '0' to this location for not to generate interrupt due to incorrect read or write asserted by program or some unknown value come Receiver FIFO

Bit 5 **TX_FIFO_INVALID_RD_WR_IE:**

Write '0' to this location for not to generate interrupt due to incorrect read or write asserted by program or some unknown value come Transmitter FIFO

Bit 4 **PARITY_ERROR_IE**:

Write '0' to disable interrupt for Parity error (even /odd).

Bit 3 **TX_OV_IE**:

Write '0' to disable interrupt for Overflow of Transmitter FIFO.

Bit 2 **TX_NULL_IE:**

Write '0' to disable interrupt for Null Transmitter FIFO.

Bit 1 **RX_OV_IE**:

Write '0' to disable interrupt for Overflow of Receiver FIFO.

Bit 0 **RX_NULL_IE:**

Write '0' to disable interrupt for Null Receiver FIFO.

ADD KEYNOTE FOR PROGRAMMER

Legend:		
RD = Readable bit	WR = Writable bit	Reserved = Unimplemented bit, read as '0'
'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

3.4 Register 4: UART Receive Register

Offset Address	x"4"
Register Type	RD
Reset Value	0x00

URx8	URx<7:0>	
RD-0	RD-0	RD-0
Bit 8	Bit 7	Bit 0

Bit 8 **URX8:**

Data bit 8 of the Received Character (in 9-bit mode)

Bit 7-0 **URX<7:0>:**

Data bits 7-0 of the Received Character

3.5 Register 5: UART Transmit Register

Offset Address	x"5"
Register Type	WR
Reset Value	0xFF

UTx8	UTx<7:0>	
WR-1	WR-1	WR-1
Bit 8	Bit 7	Bit 0

Bit 8 **URX8:**

Data bit 8 of the Received Character (in 9-bit mode)

Bit 7-0 **URX<7:0>:**

Data bits 7-0 of the Received Character

4. Serial Data Transmission Protocol

A UART includes a transmitter and a receiver. The transmitter is essentially a special shift register that loads data in parallel and then shifts it out bit by bit at a specific rate. The receiver, on the other hand, shifts in data bit by bit and then reassembles the data. The serial line is' 1' when it is idle. The transmission starts with a *start bit,* which is '0', followed by *data bits* and an optional *parity bit,* and ends with *stop bits,* which are '1'.
The number of data bits can be 7, 8, 9.
The optional parity bit is used for error detection. For odd parity, it is set to '0' when the data bits have an odd number of 1's. For even parity, it is set to '0' when the data bits have an even number of 1's.
The number of stop bits can be 1, 1.5, or 2.
The transmission with 8 data bits, no parity, and 1 stop bit is shown in Figure 3 Below Note that the LSB of the data word is transmitted first.
No clock information is conveyed through the serial line. Before the transmission starts, the transmitter and receiver must agree on a set of parameters in advance, which include the baud rate (i.e., number of bits per second), the number of data bits and stop bits, and use of the parity bit. The commonly used baud rates are 2400,4800,9600, and 19,200 bauds.

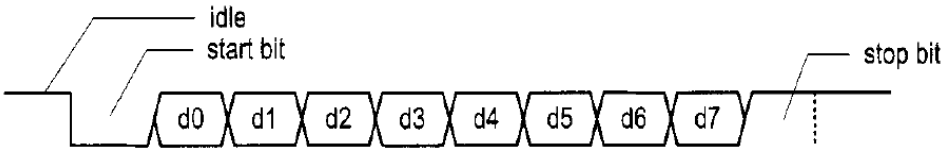


Figure 3: 8-bit data No parity 1 Stop Bit Frame

5. Oversampling procedure

Sampling rate is 16 times the baud rate, which means that each serial bit is sampled 16 times.
The oversampling scheme works as follows:
1. Wait until the incoming signal becomes '0', the beginning of the start bit, and then start the sampling tick counter.
2. When the counter reaches 7, the incoming signal reaches the middle point of the start bit. Clear the counter to 0 and restart. (We are at middle of Start bit)
3. When the counter reaches 15, the incoming signal progresses for one bit and reaches the middle of the first data bit. Retrieve its value, shift it into a register, and restart the counter. (By these glitches will go away)
4. Repeat step 3 to retrieve the remaining data bits.
5. If the optional parity bit is used, repeat step 3 one time to obtain the parity bit.
6. Same to be done to get stop bits.

6. Baud Generator

The baud rate generator generates a sampling signal whose frequency is exactly 16 times the UART's designated baud rate. 19,200 baud rates, the sampling rate must be 307,200 (i.e., 19,200*16) ticks per second. Since the system clock rate is 50 MHz

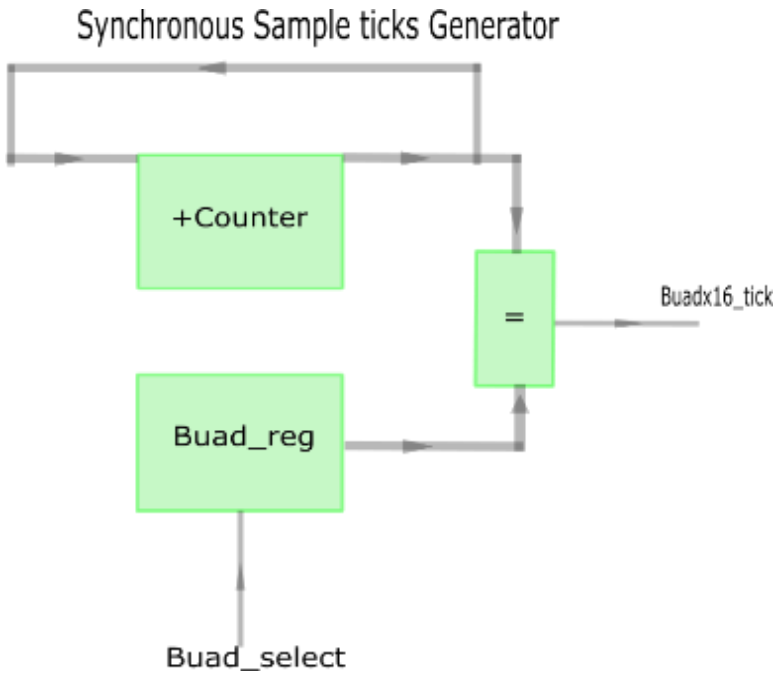
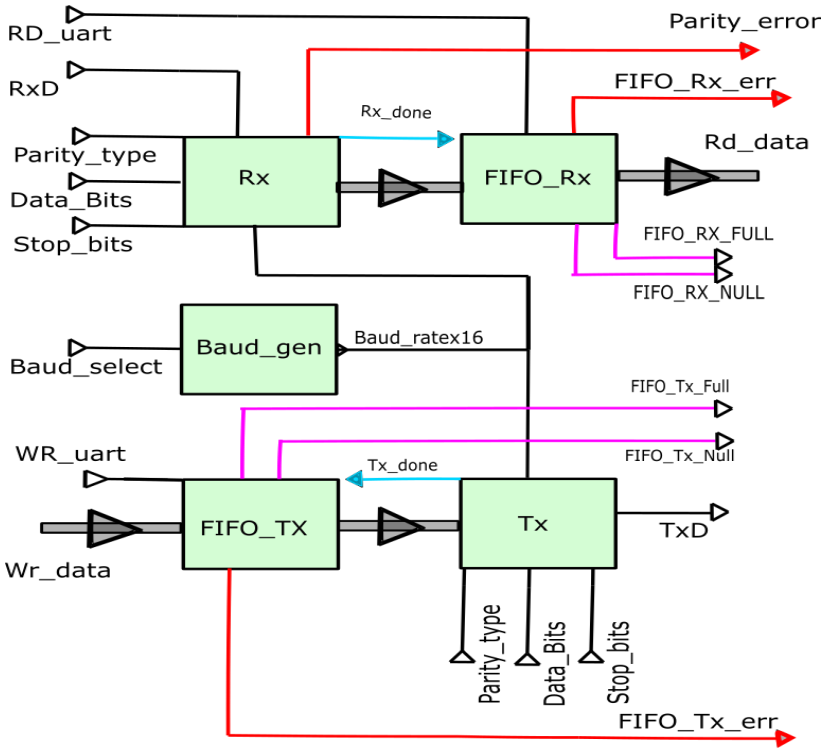


Figure 4: Sample Ticks Generator

7.UART core

The Figure 5 Shows Simplified Uart System Baud unit Is solely used to have synchronous sampling ticks which are 16 times the desired baud rate, There are 2 FIFO in the Uart Core TX_FIFFO and RX_FIFO which are connected to there corresponding peripherals i.e. Transmitter and receiver. When ever TX sends one data frame it send Tx done signal and FIFO_TX loads another Frame to it same is with RX and its FIFO when it reads one Data frame it write over its FIFO. These 4 Modules generate their own interrupts and errors and can be seen in the diagram, the purple lines are interrupts for processor and Red are error that may be generated. The Figure 6 shows implemented Design of a Uart Core.

Diagram Simplified



UART core Simplified

Figure 5: Uart Core Block Design

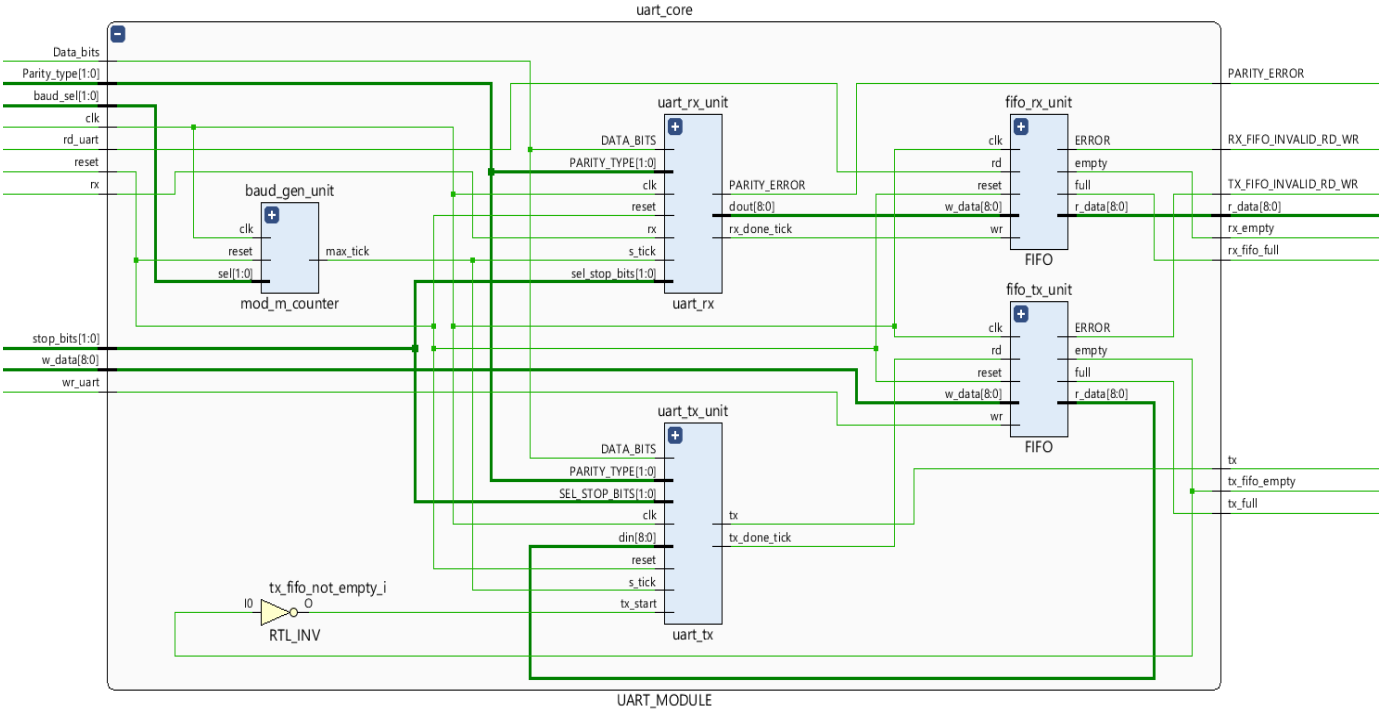


Figure 6: Uart Core implemented Design

8. UART Transmitter

A. Functional diagram

B. ASMD

C. Implemented Design

D. Description

A. Functional diagram

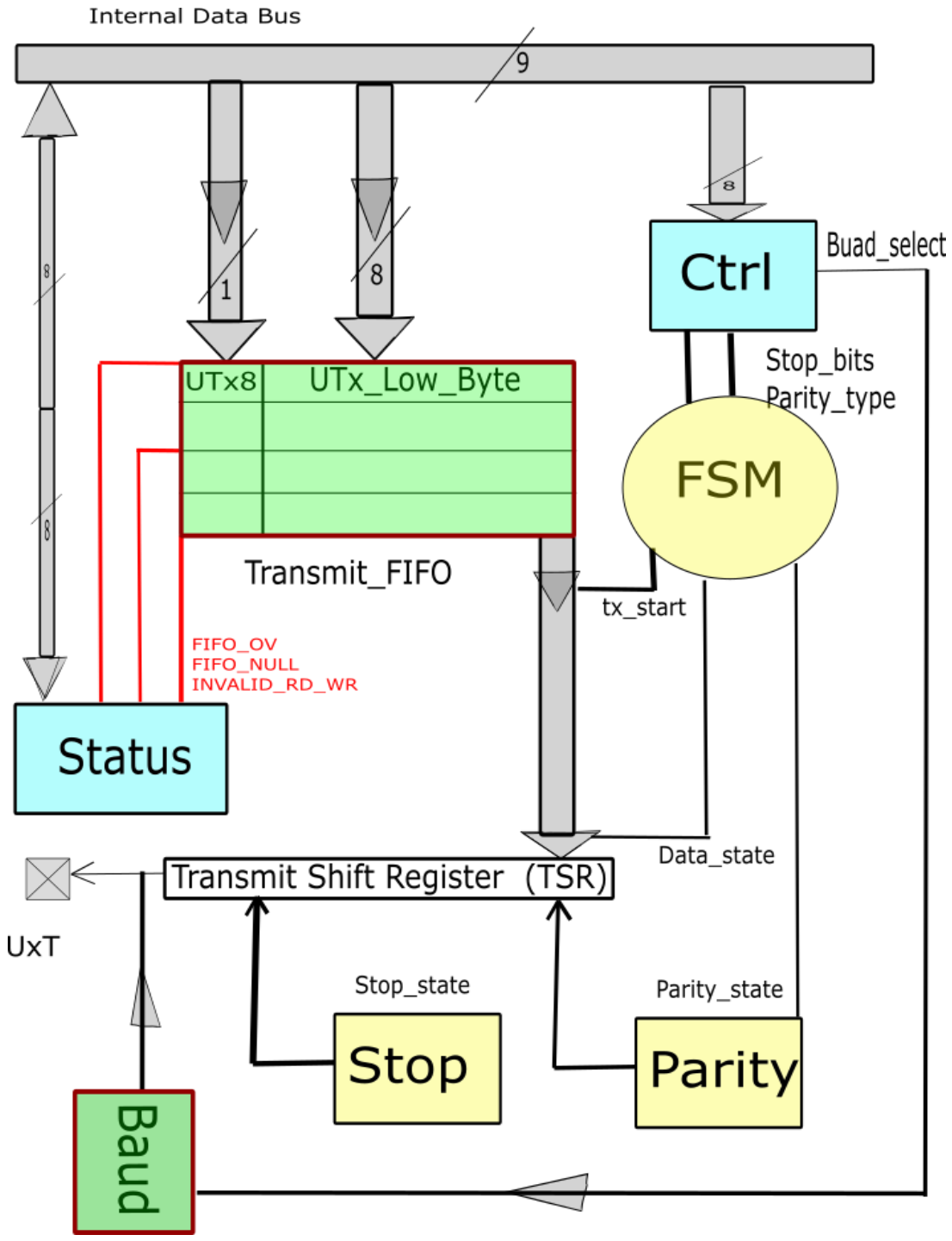


Figure 7: UART Transmitter Block Diagram

Above Figure 7 Shows the functional working of a Uart transmitter 1st Control register has to be configured of if not will work on default mode, Data is loaded to the FIFO from data bus the FIFO write has a desired address and can be seen in register specs It is understandable why the FIFO width is variable for Word write or Octet write, Once loaded data the FSM will be in idle for transmitter but the FIFO triggers the Transmitter to load value from FIFO tx-start is asserted from FIFO only by means of its output, Moving in Data State the Shift register will start sending the data with Keeping watch at sampling ticks from baud module the stop bits are asserted in stop state and parity bit in parity state, Status Register logs the values of FIFO empty or Full or invalid input asserted to FIFO.

B. ASMD (Algorithmic state machine chart) of UART Transmitter

The UART transmitter is essentially a shift register that shifts out data bits at a specific rate. The rate can be controlled by one-clock-cycle enable ticks generated by the baud rate generator. Because no oversampling is involved, the frequency of the ticks is 16 times slower than that of the UART receiver. Instead of introducing a new counter, the UART transmitter usually shares the baud rate generator of the UART receiver and uses an internal counter to keep track of the number of enable ticks. A bit is shifted out every 16 enable ticks.
The ASMD chart of the UART transmitter is shown in figure 8
After assertion of the TX-start signal, the FSMD loads the data word and then gradually progresses through the start, data, parity and stop states to shift out the corresponding bits. It signals completion by asserting the TX-done-tick signal for one clock cycle. A 1-bit buffer, TX-reg, is used to filter out any potential glitch.

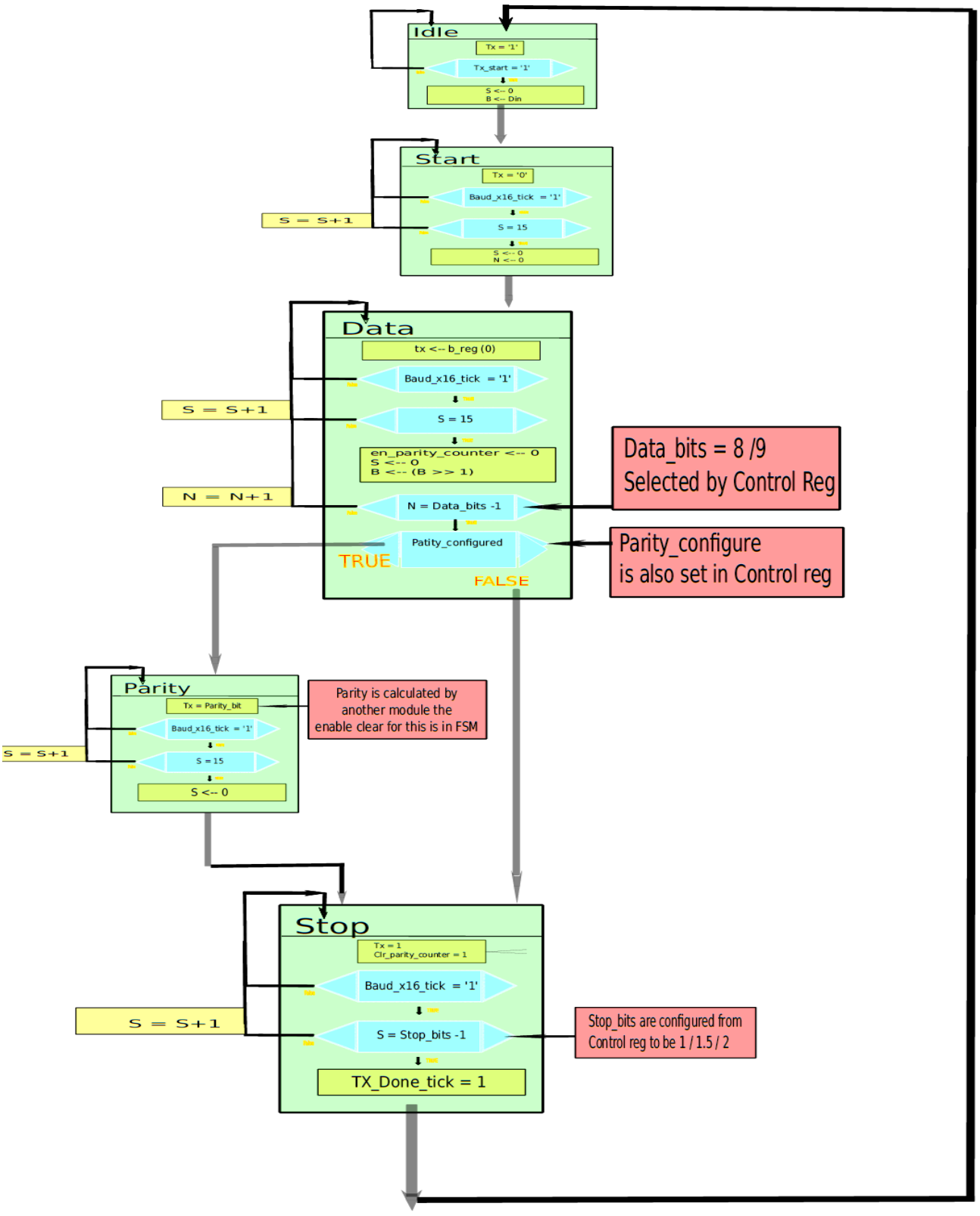


Figure 8: ASMD of UART Transmitter

C. Implemented Design

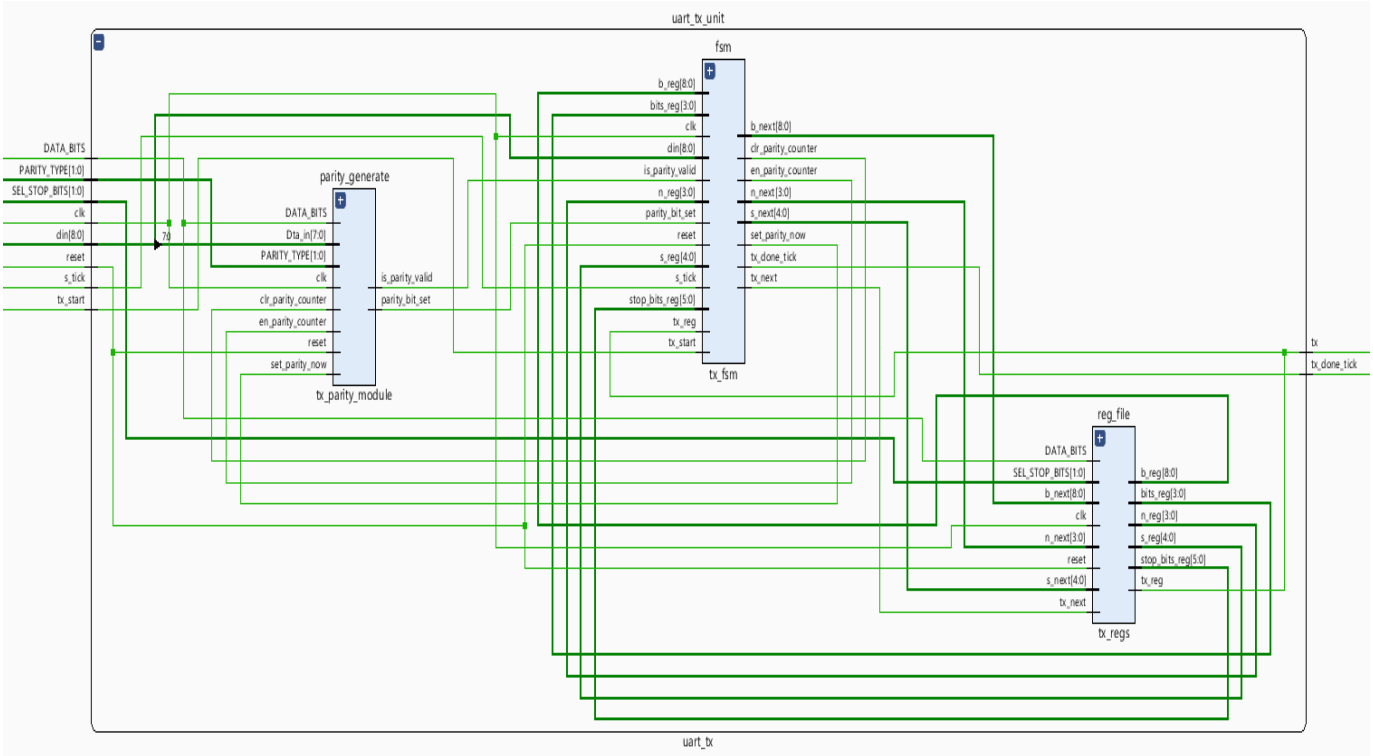


Figure 9: Implemented UART Transmitter

D. Description

Above figure 9 depicts how Transmitter is coded it consist of a FSM , register file and a parity generate module parity generate has 2 outputs is-parity-valid and parity-bit-set, is-parity-valid is used in FSM to branch between parity state and stop state, register file has stop-bits reg which is used to insert desired stop bits. Tx and Tx-done-tick are the outputs of this sub-system.

9. UART Receiver

A. Functional diagram

B. ASMD

A. Functional diagram

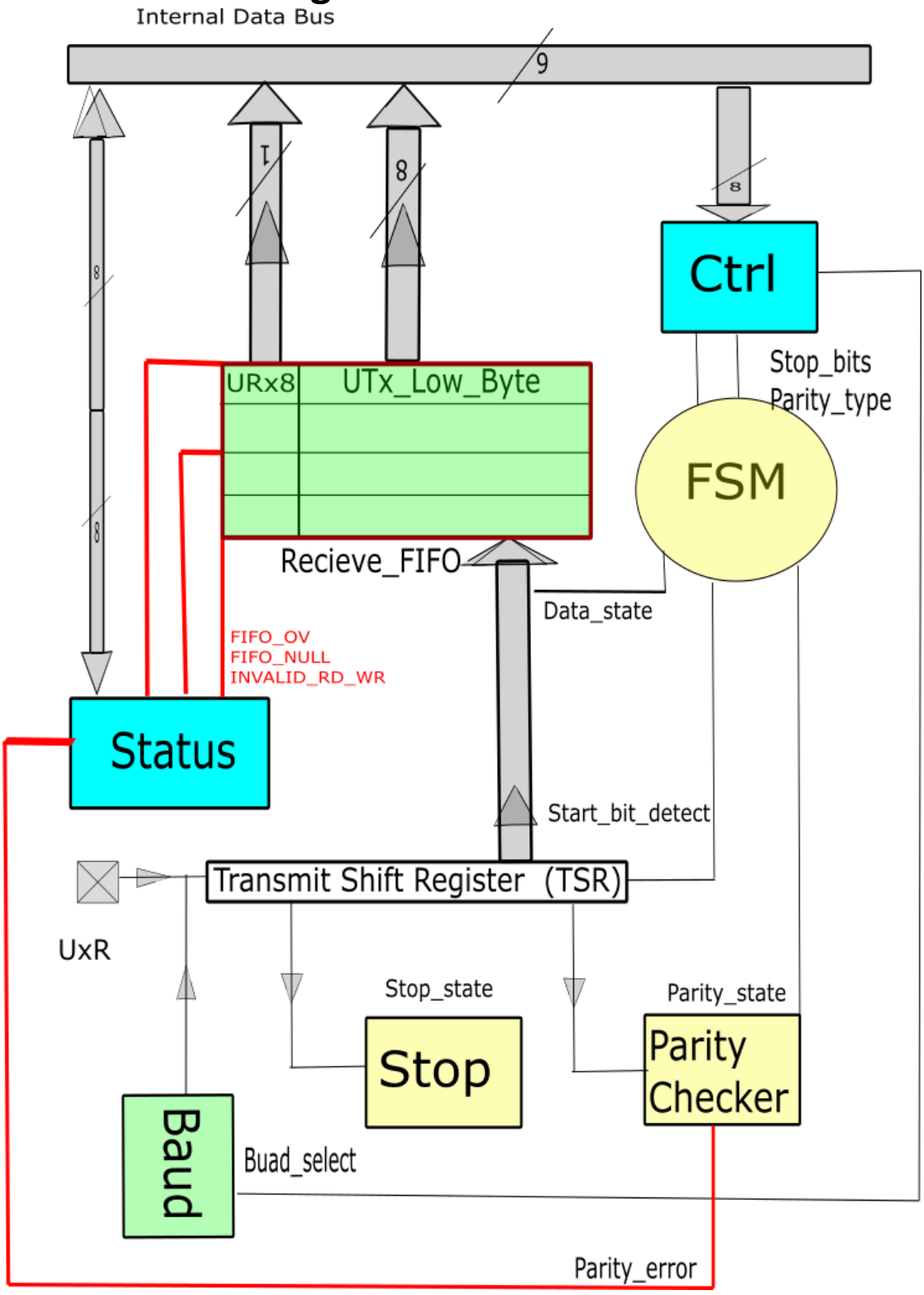


Figure 10: UART Receiver Block Diagram

Above Figure 10 Shows the functional working of a Uart receiver 1st Control register has to be configured of if not will work on default mode, in coming serial Data is loaded Shift register as soon as a start bit is detected i.e. '0' is detected at line it move to data state after certain wait till S = 7 then after assembling data receiver load it to FIFO from data bus the FIFO read has a desired address and can be seen in register specs It is understandable why the FIFO width is variable for Word read or Octet read, The FSM check that is parity is enabled if yes it moves to parity check after getting 8 bits if parity protection is not provided all the data is read by the shift reg and loaded to FIFO and Write pointer in FIFO moves 1 spot, Stop bits are also has to be passed so the FSM will wait in Stop state till the desired stop bits are passed which is configured in the Control Register. After 1 data read the Rx-done-tick is asserted, Status Register logs the values of FIFO empty or Full or invalid input asserted to FIFO.

B. ASMD

The **ASMD** chart of Uart Receiver in Figure 11
Two parameters are used in the description. The Data-length this is configured based on what is the actual data length excluding parity bit if it is applicable , Data-bit is 8 or 9 depending on what is the programmer has configured and it indicates the number of data bits, and the stop-bits indicates the number of ticks needed for the stop bits, which is 16, 24, and 32 for 1, 1.5, and 2 stop bits, respectively. Data-bits and stop-bits can be configured from control register.
The chart includes five major states: idle, start, data, parity, and stop, which represent the processing of the start bit, data bits, parity bit and stop bit. The BaudX16 = S-Tick signal is the enable tick from the baud rate generator and there are 16 ticks in a bit interval. Note that the FSMD stays in the same state unless the BaudX16 signal is asserted. There are two counters, represented by the **S** and **n** registers. The **s** register keeps track of the number of sampling ticks and counts to 7 in the start state, to 15 in the data state, to 15 in the parity state and to stop-bits in the stop state. The **n** register keeps track of the number of data bits received in the data state. The retrieved bits are shifted into and reassembled in the **b** register. A status signal, **RX-done-tick,** is included. It is asserted for one clock cycle after the receiving process is completed.

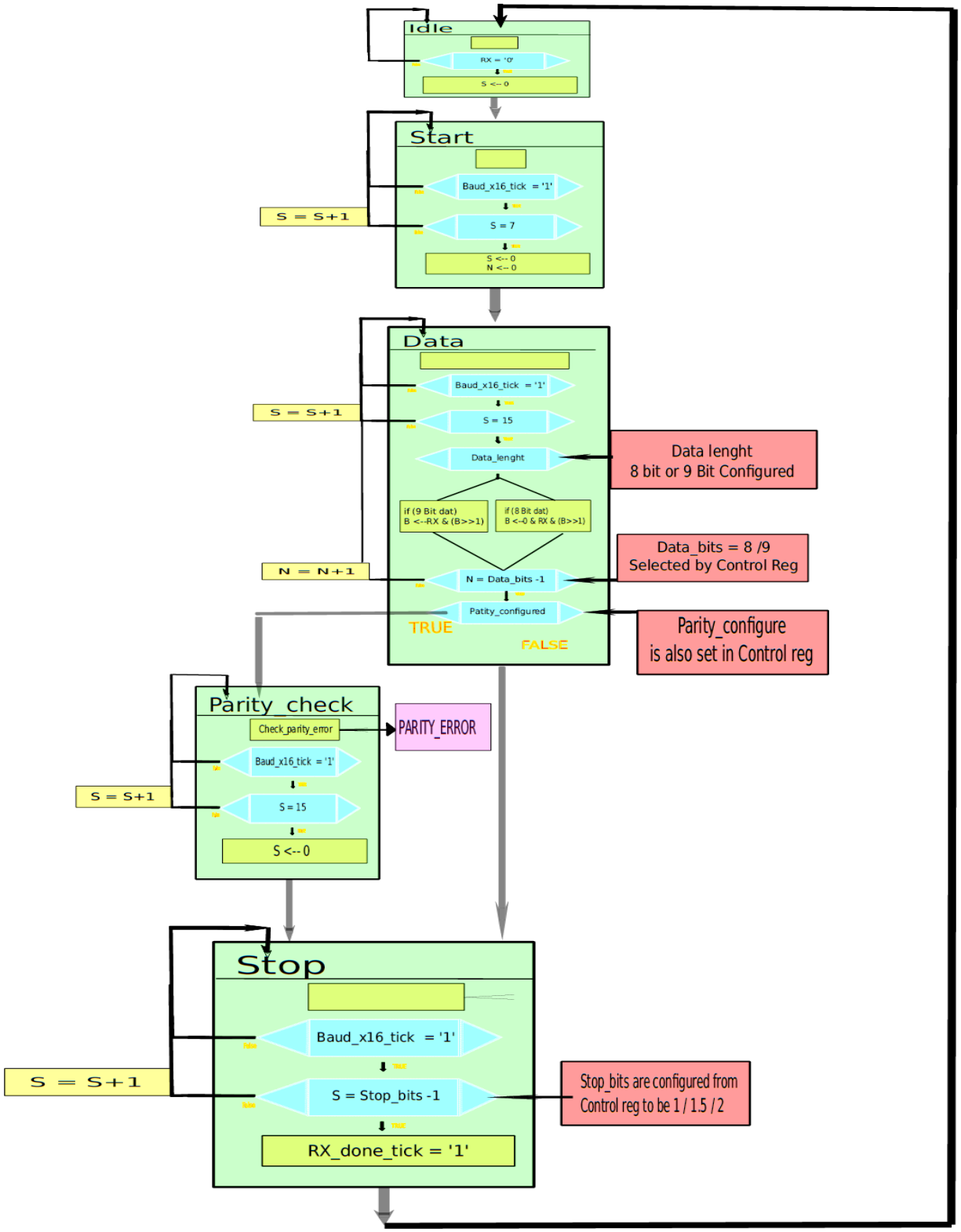


Figure 11: ASMD of UART Receiver

10. FIFO Module

A.Functional diagram and Description

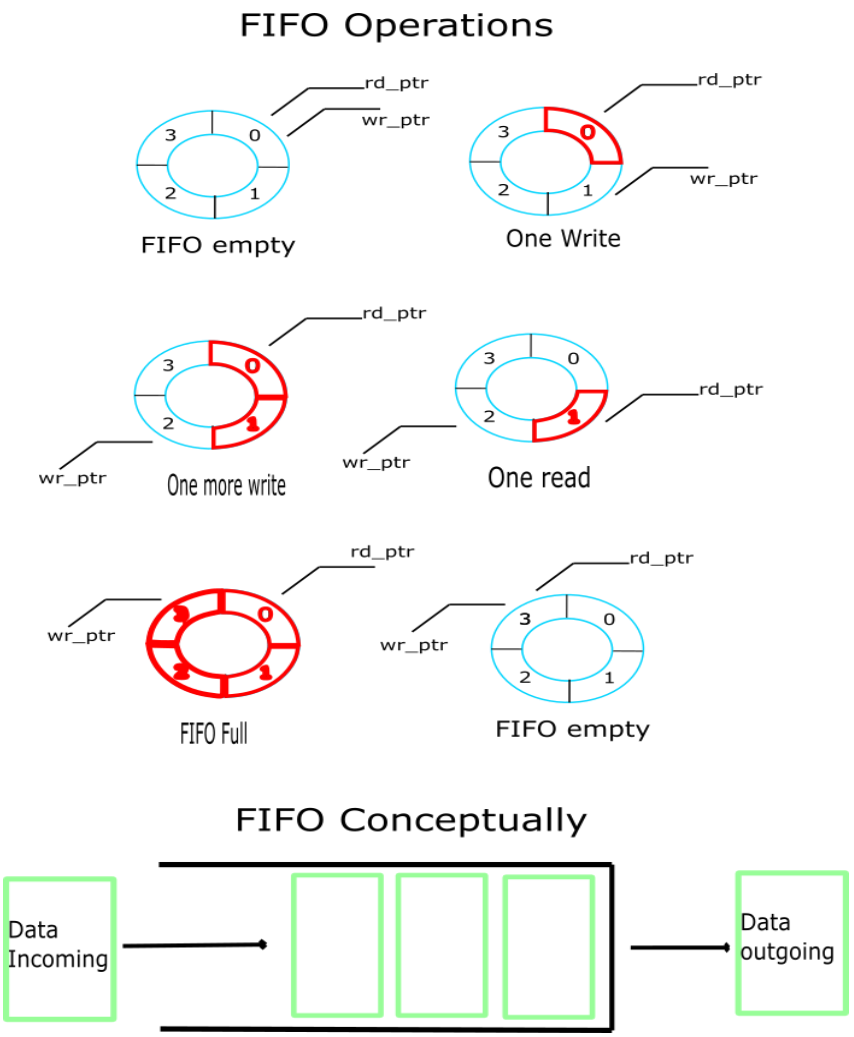


FIGURE 12: FIFO conceptual Diagram

Circular-queue-based implementation One way to implement a FIFO buffer is to add a control circuit to a register file. The registers in the register file are arranged as a circular queue with two pointers. The *write pointer* points to the head of the queue, and the *read pointer* points to the tail of the queue. The pointer advances one position for each write or read operation. A FIFO buffer usually contains two status signals, full and empty, to indicate that the FIFO is full (i.e., cannot be written) and empty (i.e., cannot be read), respectively. One of the two conditions occur when the read pointer is equal to the write pointer, as shown in Figure 12. In a large system, a UART is usually a peripheral circuit for serial data transfer. The main system checks its status periodically to retrieve and process the received word. The receiver's interface circuit has two functions. First, it provides a mechanism to signal the

availability of a ***new*** word and to prevent the received word from being retrieved multiple times. Second, it can provide buffer space between the receiver and the main system. The FIFO buffer provides more buffering space and further reduces the chance of data overrun. We can adjust the desired number of words in FIFO to accommodate the processing need of the main system. The RX-ready-tick signal is connected to the WR signal of the FIFO. When a new data word is received, the WR signal is asserted one clock cycle and the corresponding data is written to the FIFO. The main system obtains the data from FIFO's read port. After retrieving a word, it asserts the RD signal of the FIFO one clock cycle to remove the corresponding item. The empty signal of the FIFO can be used to indicate whether any received data word is available. A data-overrun error occurs when a new data word arrives, and the FIFO is full.

B. Implemented design

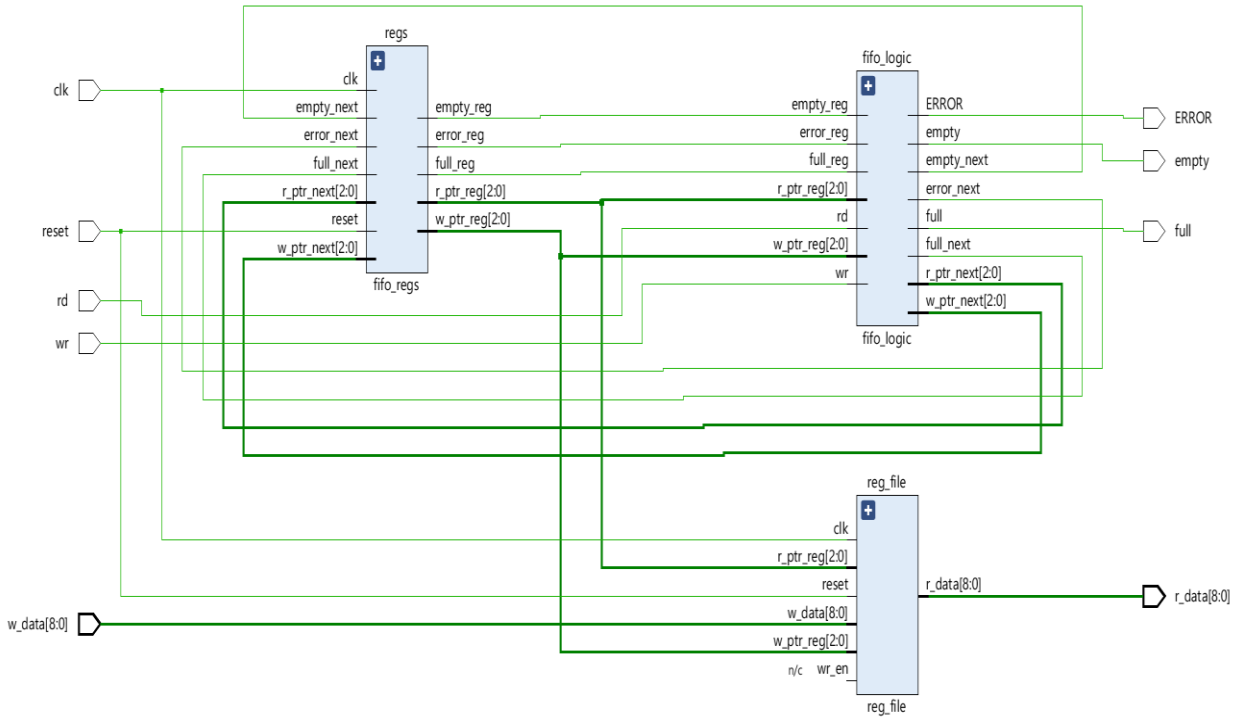


FIGURE 13: FIFO implemented

The Implemented design of a FFIO shows the 3 blocks:

- Register File
- Working Registers
- FIFO read write pointer logic

Register File is an array of Desired registers that are addressed.

11. Uart interrupt

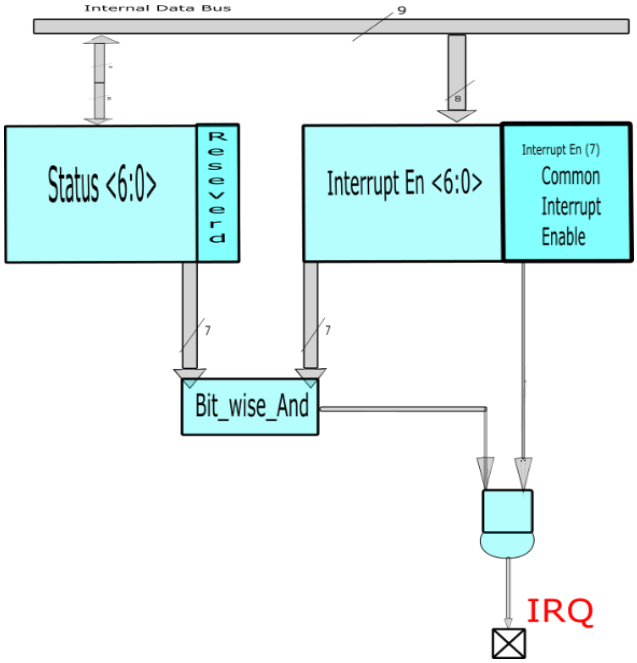


Figure 14: Interrupt Request Block

Interrupt is consolidated of all the interrupts generated withing the Uart module and then send to the processor. Reason for this being avoiding many wires for just interrupts, Processor can handle issue by reading status register.

12. Uart Implemented Design and Reports

A. Implemented Design
B. Utilization Report
C. Timing Report
D. Simulation of Implemented Design
E. Timing Constraints

A. Implemented Design

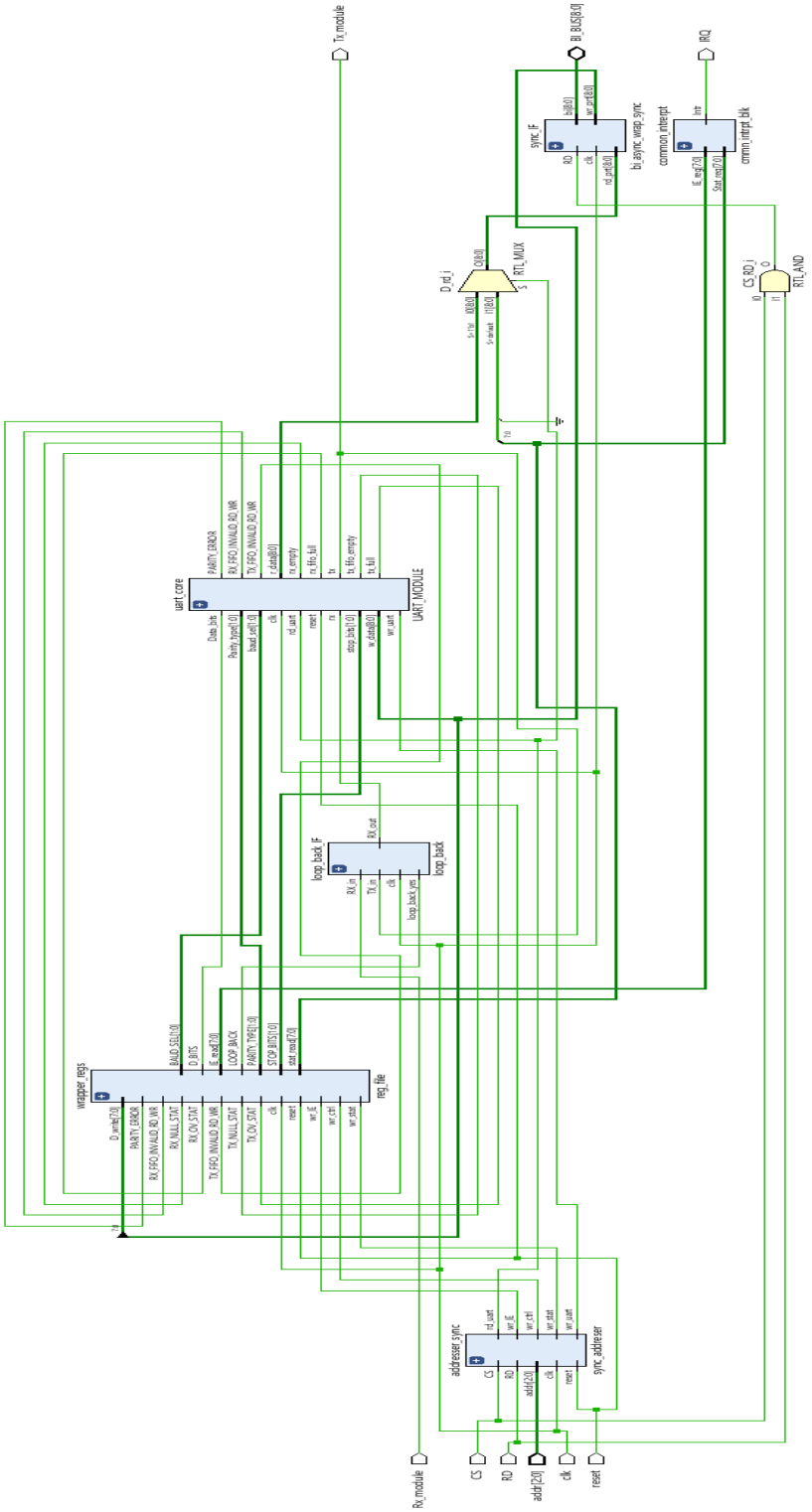


FIGURE 15: Uart Module with Wrapper around it implemented Design

B. Utilization Report

Name	Slice LUTs	Slice Regis	Slice	LUT as Log	Bonded IOB
wrap_uart	171	207	66	171	18
uart_cor	150	158	59	150	0
fifo_tx	20	42	14	20	0
fifo_rx	26	42	15	26	0
uart_rx	36	28	16	36	0
uart_tx	39	25	13	39	0
baud_g	30	21	9	30	0
wrapper	8	24	7	8	0
sync_IF (	8	19	8	8	0
addresse	5	5	2	5	0
loop_bac	0	1	1	0	0

Resource	Utilization	Available	Utilization %
LUT	171	20800	0.82
FF	207	41600	0.50
IO	18	106	16.98

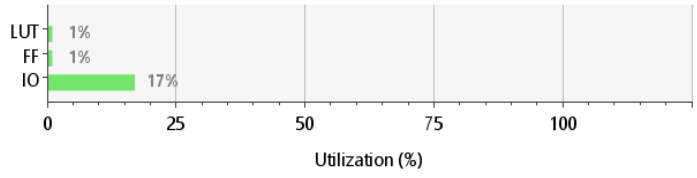


FIGURE 16: Utilization Report

C. Timing Report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.097 ns	Worst Hold Slack (WHS):	0.114 ns	Worst Pulse Width Slack (WPWS):	9.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	511	Total Number of Endpoints:	511	Total Number of Endpoints:	208

All user specified timing constraints are met.

FIGURE 17: Timing Summary

D.Simulation of Implemented Design:

D1. 9-Bit data with no parity
D2. 8-Bit data with even Parity And 2 Stop Bits
D3. 8-Bit data with odd parity
D4. Status changes and Interrupt generated

D1. 9-Bit data with no parity

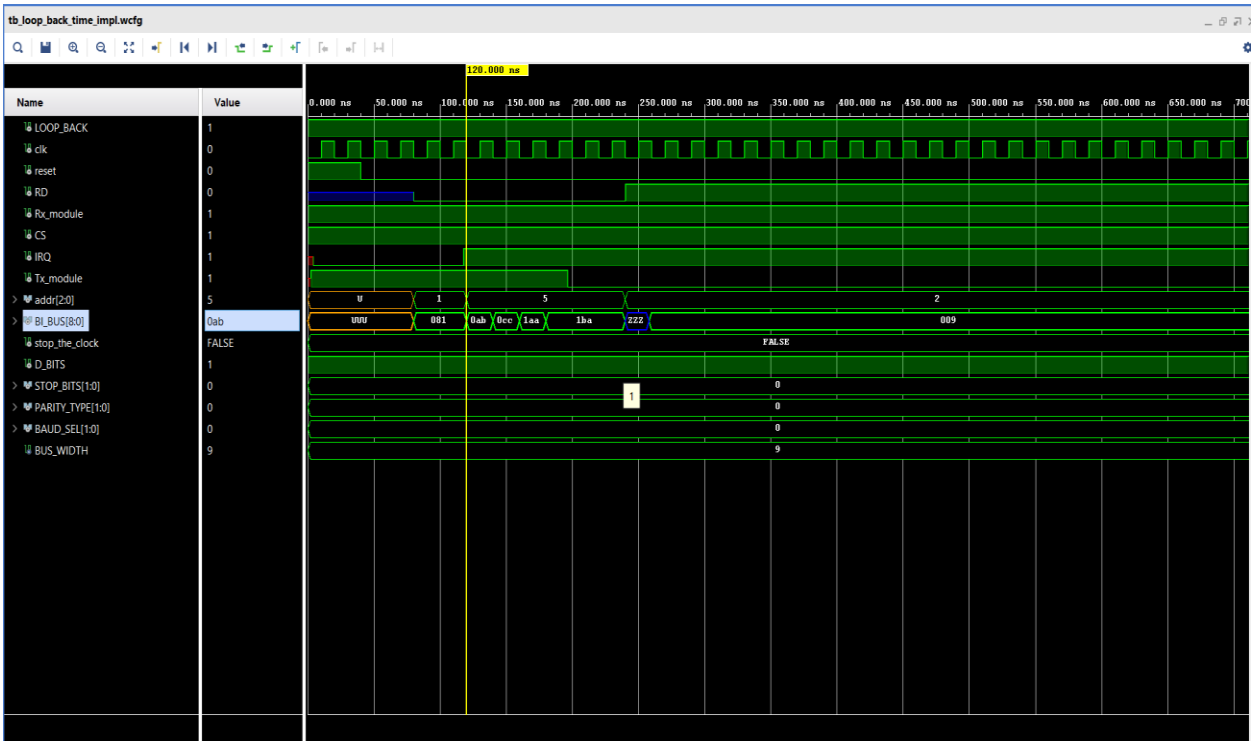


FIGURE 18: 9-Bit data with no parity data written to TX FIFO

Data written is x”0AB”, x”0CC”, x“1AA” , x“BA” , All the data will be send and the 9 bit data will be transmitted without caring about the 0 in the 1st and 2nd data. Selecting D_BITS as ‘1’ we can configure the Uart for 9-bit data transmission Parity is configured in control register but when u take 9-bit data and mistakenly do parity protection the module will ignore parity and will just send 9-bit data.
To write over Uart module the address of Uart TX register has to be passed with RD = ‘0’ and CS = ‘1’ then only you can write over Uart TX The FIFO depth is 4 and can take word or octet data as required. The simulation done is in loop back mode for this Control reg has to be configured for loop back enable and Rx of module is set to ‘1’ for initialization of Rx receiver to idle state.

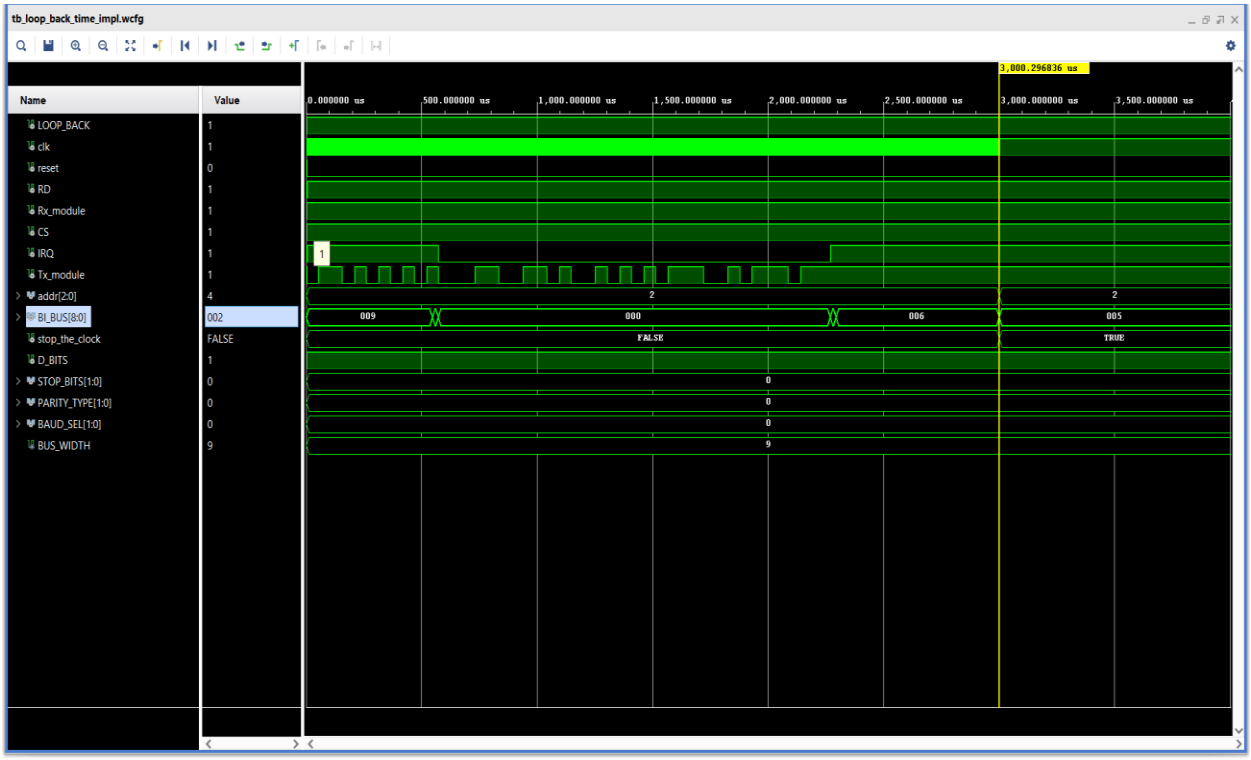


FIGURE 19: 9-Bit data Transmission

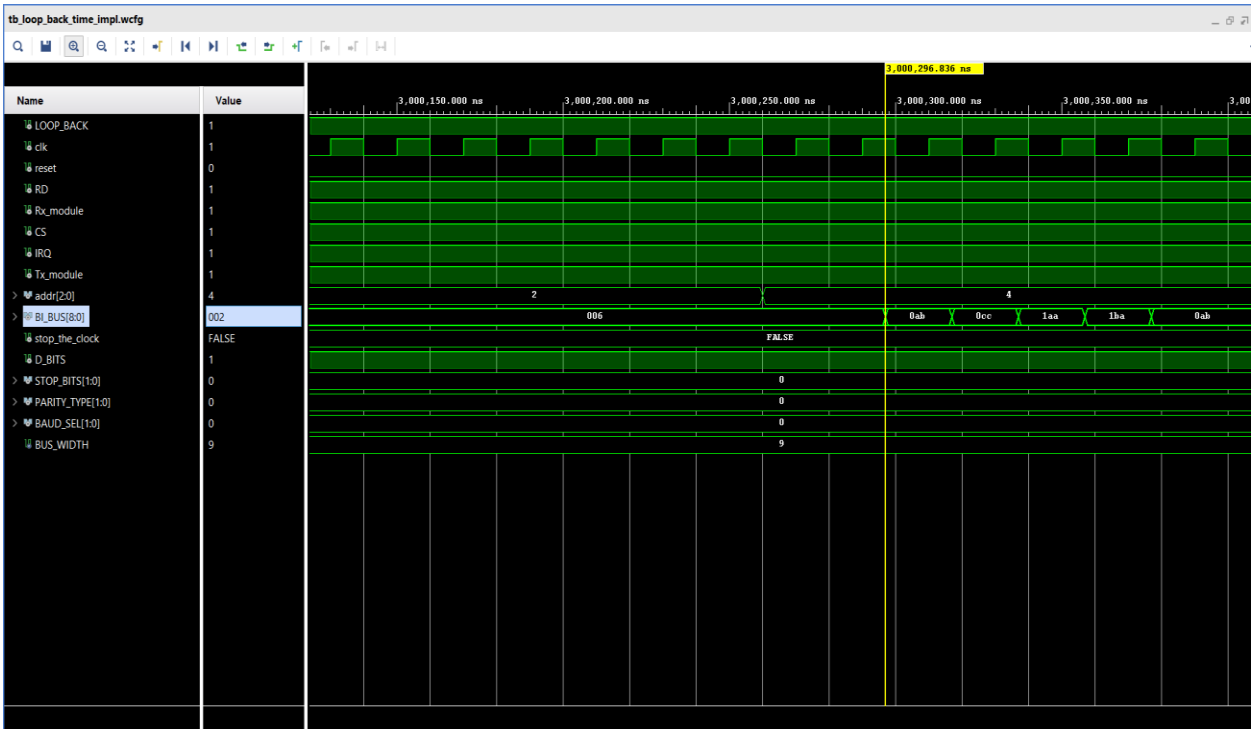


FIGURE 20: 9-Bit data read after being processed by receiver

D2. 8-Bit data with even Parity And 2 Stop Bits.

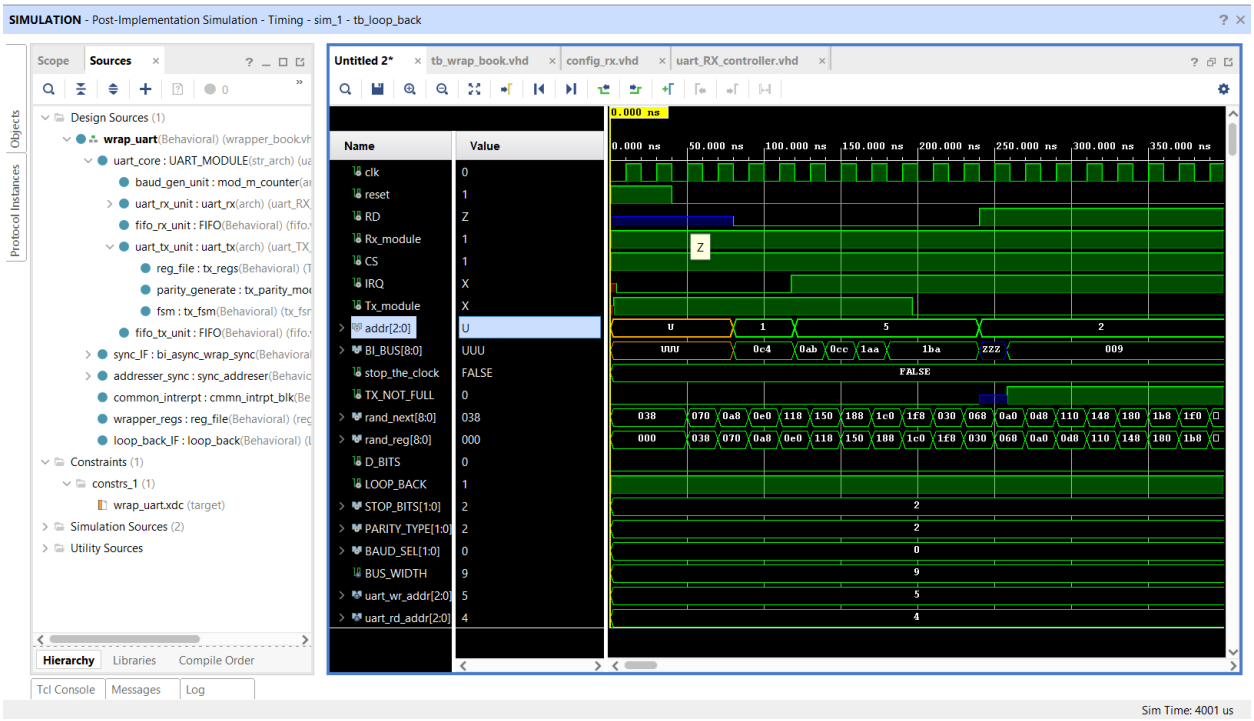


FIGURE 21: 9-Bit data written but only the 8 bits will be taken by the module due to selection of 8-bit data with even parity and 2 stop bits. The '1' in the last 2 data will be ignored only 8 bits will be taken.

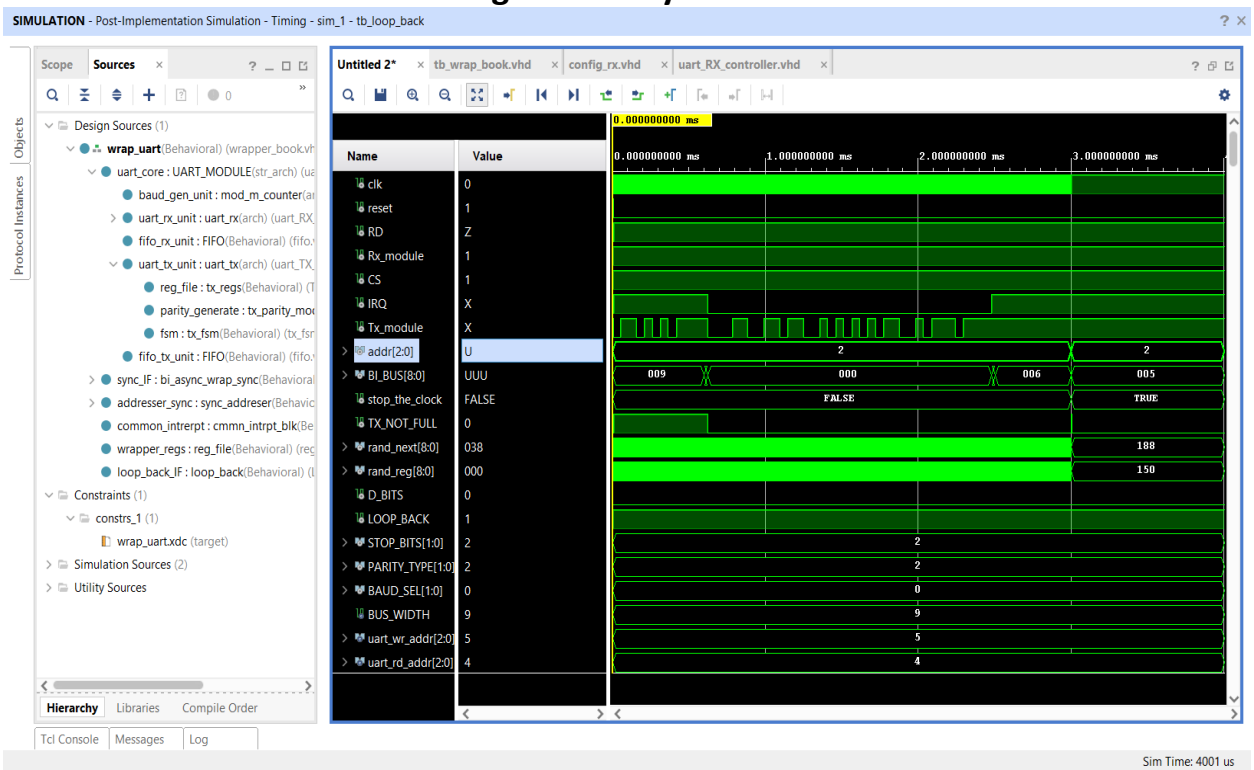


FIGURE 22: 8-Bit with even parity and 2 stop bits Transmission.

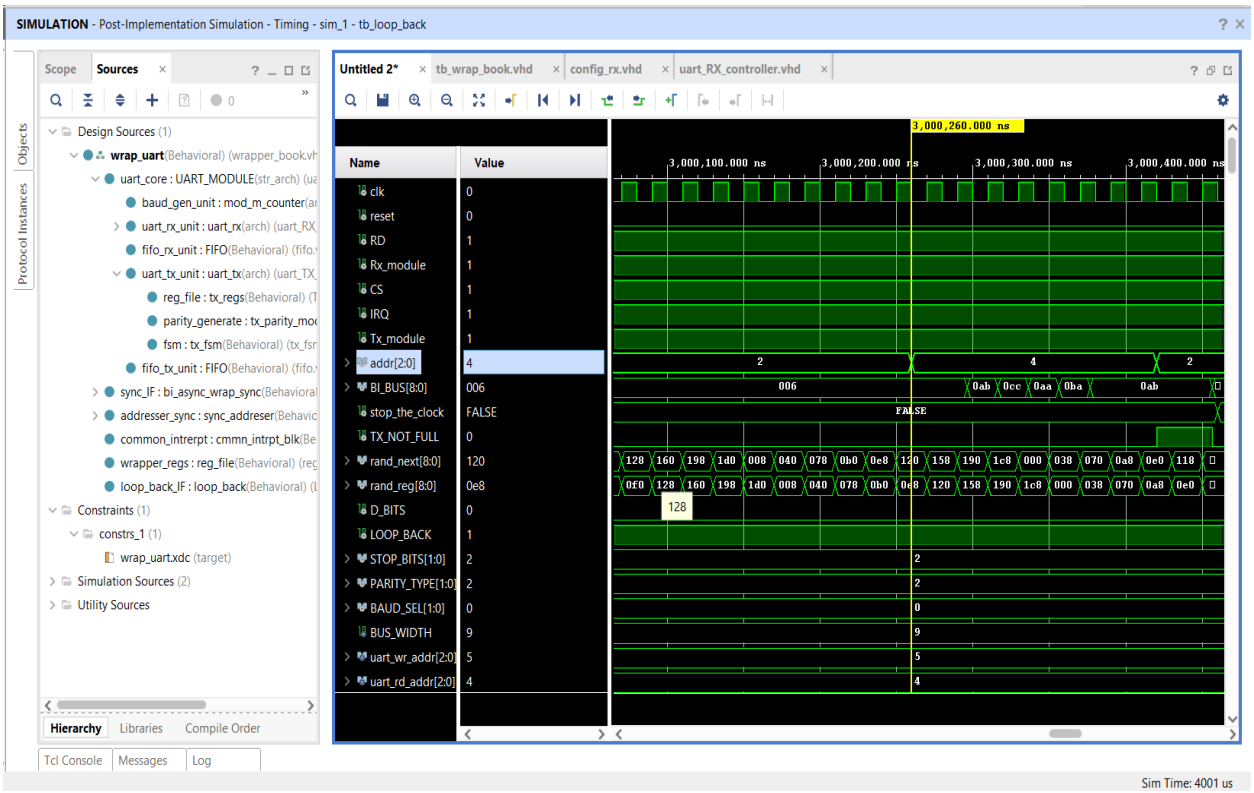


FIGURE 23: 8-Bit data with even parity read after being processed by receiver the parity error is not issued as seen in status register.

D3. 8-Bit data with odd parity

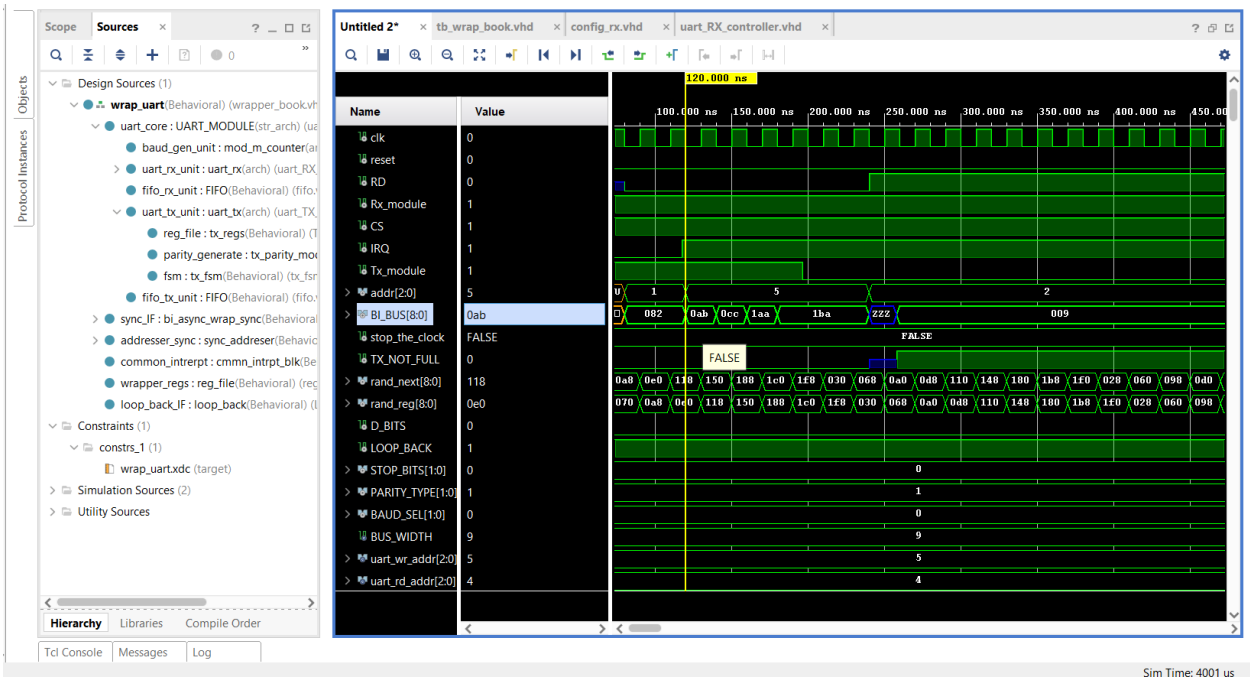


FIGURE 24: 9-Bit data written but only the 8 bits will be taken by the module due to selection of 8-bit data with odd parity and default stop bits. The '1' in the last 2 data will be ignored only 8 bits will be taken.

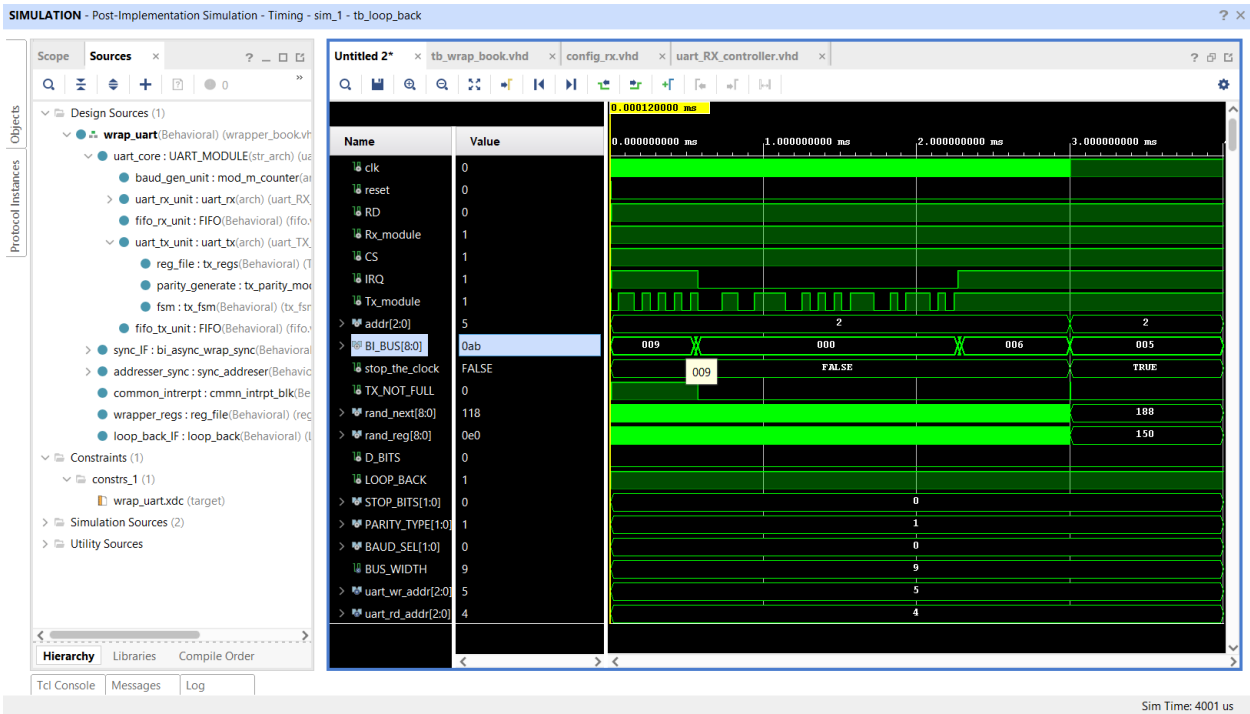


FIGURE 25: 8-Bit with odd parity Transmission.

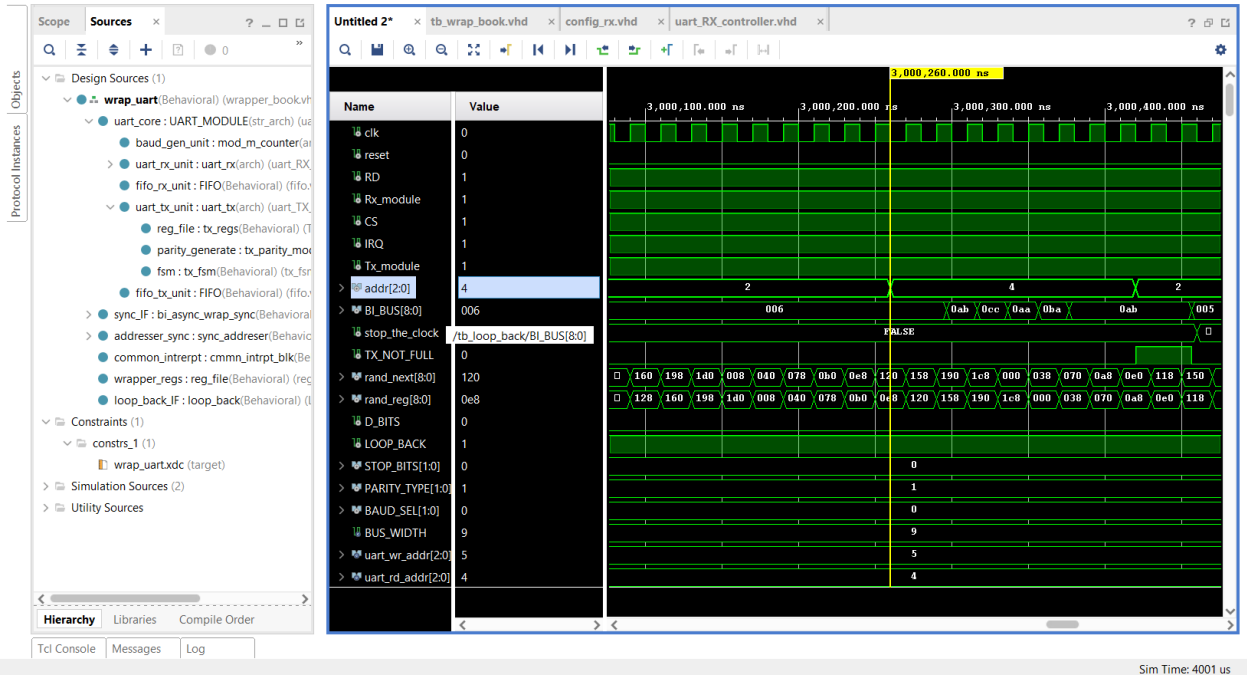


FIGURE 25: 8-Bit data with odd parity read after being processed by receiver the parity error is not issued as seen in status register.

D4. Status register and Interrupt generated

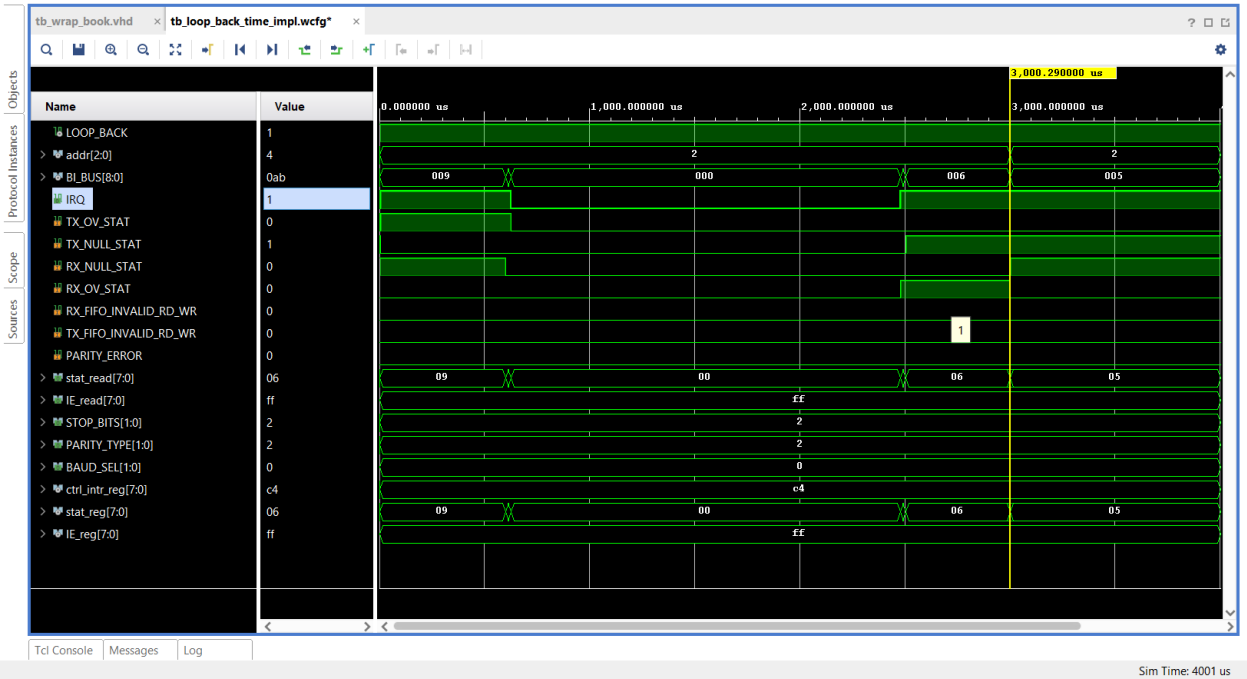


FIGURE 25: Various Fields in status register can be seen Such as full or empty of FIFO's and Common Interrupt being generated.

E. Timing Constraints

```
create_clock -period 20.000 -name clk -waveform {0.000 10.000} [get_ports clk]
set_input_delay    -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports {BI_BUS[*]}]
set_input_delay    -clock [get_clocks clk] -max  -add_delay 0.200 [get_ports {BI_BUS[*]}]
set_input_delay    -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports {addr[*]}]
set_input_delay    -clock [get_clocks clk] -max  -add_delay 0.200 [get_ports {addr[*]}]
set_input_delay    -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports CS]
set_input_delay    -clock [get_clocks clk] -max  -add_delay 0.200 [get_ports CS]
set_input_delay    -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports RD]
set_input_delay    -clock [get_clocks clk] -max  -add_delay 0.200 [get_ports RD]
set_input_delay    -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports reset]
set_input_delay    -clock [get_clocks clk] -max  -add_delay 0.200 [get_ports reset]
set_output_delay -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports {BI_BUS[*]}]
set_output_delay -clock [get_clocks clk] -max  -add_delay 0.100 [get_ports {BI_BUS[*]}]
set_output_delay -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports IRQ]
set_output_delay -clock [get_clocks clk] -max  -add_delay 0.100 [get_ports IRQ]
set_output_delay -clock [get_clocks clk] -min   -add_delay 0.000 [get_ports Tx_module]
set_output_delay -clock [get_clocks clk] -max  -add_delay 0.100 [get_ports Tx_module]
```

F. Synchronous Bi-Directional Bus

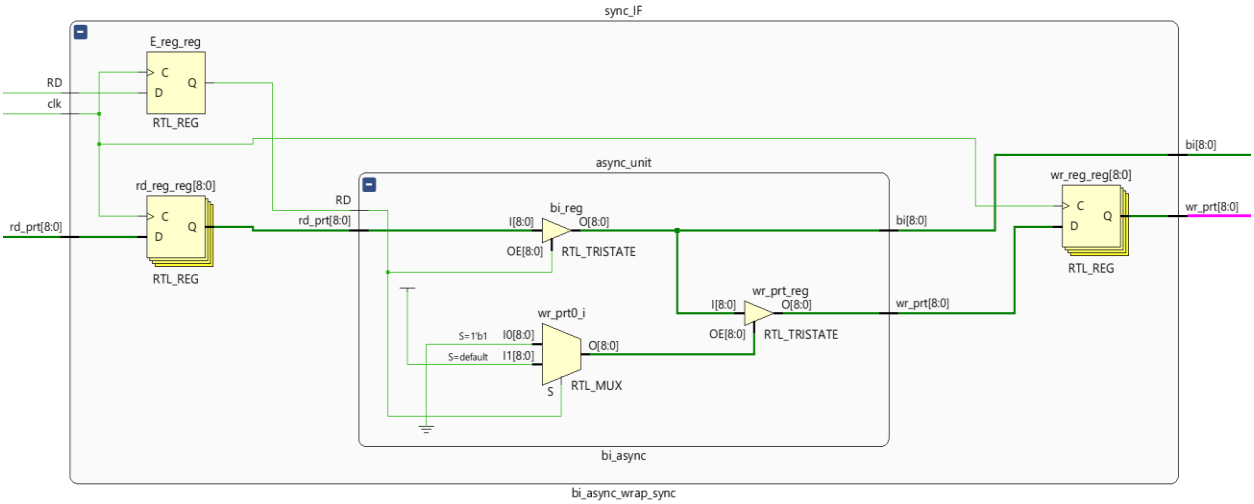


FIGURE 26: Behavioral block of a Synchronous Bi-Directional Bus, Asynchronous Bi-Directional Bus is synchronized to clock with registers this reduces glitches in Data.

13.Conclusion

The Uart Designed has many features and works with 8 or 9 data bits can be run-time configured for 8- or 9-bits parity type and stop bits can also be run-time configurable. The design has an integrated Baud Generator with four standard Baud rates supported at System frequency of 50 MHz. The unit has integrated FIFO buffers for RX and TX controllers which can be used for burst read and burst write. Parity error detection and Buffer Overrun or Buffer empty bits available in the Status register. The design has a consolidated interrupt. The Control, Status and Interrupt Enable registers to have addresses accessible through Address line and 8 bits wide each. The FIFO is configurable for word(9-bit) or Octet(8-bit) write/read by the Control register. The design also has a bi-directional data bus which is 9 bit wide but can be configured while instancing design to 16 bit wide as its Generic, The Wrapper register that is Control, Status and Interrupt Enable registers are 8 bits wide each, and can also be increased for more functionality such as breaking transmission and breaking reception these are simple functionalities and some other functionalities can also be added. The drawback of this design is that it works on the oversampling method thus, for very high transmission rates the design is not suited as 16 bits are sampled for each data bit and stop bit.

14. References

- dsPIC30F Family Reference Manual UART Section
- AXI UART 16550 v2.0 LogiCORE IP Product Guide
- Digital Systems Design Using VHDL, Second Edition : Charles H. Roth, Jr, and Lizy Kurian John
- FPGA prototyping by VHDL examples Xilinx SpartanTM-3V version Pong P. Chu
- Data Sheet for 8251 Serial Control Unit REL 1.0
- SIDSA Semiconductor Design Solutions UART 16550 IP
- PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs
- RTL HARDWARE DESIGN USING VHDL Coding for Efficiency, Portability, and Scalability PONG P. CHU
- The Designer's Guide to VHDL by P. J. Ashenden