CHAPTER 5

SYNTHESIS AND SIMULATION

5.0 Introduction

"Chapter 4: VHDL Implementation" has pointed out to the reader on how to manipulate and store the logic characteristics of a Field Programmable Gate Array (FPGA) using "Xilinx Foudation Series 2i" software. In that chapter, general descriptions of design implementation stage and verification stage have been carried out. In this chapter, again the discussion on the design implementation stage and verification stage will be carried out but the focus will be targeted to the UART with BIST design. The UART with BIST design (which has been written in VHDL) will be proven by adequately captures the implementation intention by using "Logic Simulator" simulation tools. A test bench provided by the "Logic Simulator" simulation tools will be used to feed inputs to the designed logic gates and generate the output waveforms to verify the theoretical characteristics of the designed UART.

5.1 Simulation

Before simulation process is conducted, VHDL codes have to be analyzed by "Xilinx Foundation Series 2i (XSE2i)" HDL editor. The Hardware Description Language (HDL) editor will check the syntax error of the VHDL codes. Then the codes

will be added to XSE2i Project Manager. The VHDL source codes are free from syntax errors if a green checked mark appears in XSE2i Project Manager. Following the syntax error checks, XSE2i Project Manager synthesis tools will transform the circuit defined in VHDL to a gate level definition. This gate level definition will be simulated to verify the intended UART with BIST design by using Logic Simulator and Active CAD 3.1. Active CAD 3.1 is a script editor provided by ALDEC to generate input signals to the designed UART. The script editor will produce the desired input and will generate the outputs of the synthesized design in a waveform viewer. Then, the generated output signals will be verified with theoretical value. Script files that have been created by using Active CAD 3.1 script editor are listed in **Table 5.1**.

Table 5.1: Script Files of UART with BIST.

Script files	Description
Txnorm_tb.cmd	**Transmitter normal mode** - Transmits 8-bits data from parallel to serial
Tx5bit_tb.cmd	**Transmitter 5-bits mode** - Transmits 5-bits parallel data to serial
Txbrk_tb.cmd	**Transmitter break interrupt** - Shows the effect of break interrupt on a transmitter
Txpty_tb.cmd	**Transmitter even parity mode** - Transmits 8-bits data from parallel to serial with even parity bit
Rxnorm_tb.cmd	**Receiver normal mode** - Receives 8-bits serial data and converts the data to parallel
Rx5bit_tb.cmd	**Receiver 5-bits mode** - Receives 5-bits serial data and converts the data to parallel
Rxerr_tb.cmd	**Receiver Error**

	- Shows the detection of error in a receiver (i.e. frame, parity and overrun error)
Bist_tb.cmd	**BIST mode** - Shows how the LFSR produces PRPG and how the MISR compresses the data to create a signature.

The script files in **Table 5.1** are files used for simulating the characteristics and the behaviors of the designed UART. The simulation will show the reader how the UART transmits and receives data in a communication line. The simulation will also conducts a test using the embedded BIST feature of the designed UART. To create a clearer view of the UART's verification stage, the critical part of the generated waveform will be presented and discussed. Before advancing to the waveform section, it is appropriate to explain the UART's frame format in order to understand the UART characteristics.

5.2 UART Frame Format

It is difficult for the receiver to make sense of the data coming in if all the data transferred were 0s and 1s. The problem is solved if the sender and receiver agree on a set of rules, a protocol on how the data is packed, how many bits constitute a character, and when data begins and ends. A common UART frame consists of:

- a start bit
- data bits (i.e. 5, 6, 7 or 8-bits)

- a programmable ninth data bit (if data bits = 8)
- and stop bit (1 or 2)

An active low start bit informs the UART's receiver that a new sequence of data is on its way. Following the low start bit, the frame format is succeeded with the information data bits, which may be 5, 6, 7 or 8-bits long. The information data is transmitted in a format that begins with Least Significant Bit (LSB) and ends with Most Significant Bit (MSB). The programmable bit, which comes after the information data bits, is usually used to assign the parity bit. The parity bit can be configured to even parity, odd parity or no parity mode. Next to the parity bit, comes a trailing high stop-bit which indicates the end of a data frame. **Figure 5.1** shows the UART frame format.

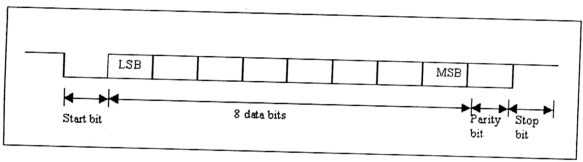

Figure 5.1: UART Frame Format

The UART frame format has been explained in this section. In the coming section, the reader will be presented with the simulated waveform results of the designed UART. The simulation is divided into three categories, which consist of transmitter simulation, receiver simulation and BIST simulation. The discussion covered in this

chapter is much related to the previous "**Chapter 4:** VHDL Implementation". Therefore, the reader should refer to the chapter for pins and registers description to grasp the meaning of the following discussion.

5.3 Transmitter Simulation

5.3.1 Transmitter 8-Bits Data Transmission

Txnorm_tb.cmd is the script file for 8-bits data transmission simulation. The simulation will show the transmission of 8-bits UART frame format with 1 stop bit and without a parity bit. The transmission will be performed at 115.2kbps. To start the simulation, the host clock (hclk) must be set according to the clock that will be used in the actual hardware.

The simulation will be using 40MHz clock, which is equal to 25ns (1/40MHz = 25ns) period. The clock rate was chosen because of the capability of the Logic Simulator to make a fast simulation and therefore reducing the need to wait for the long simulation to complete. Each address (ADDR[3:0]) for choosing the required registers is selected using chip select (CS).

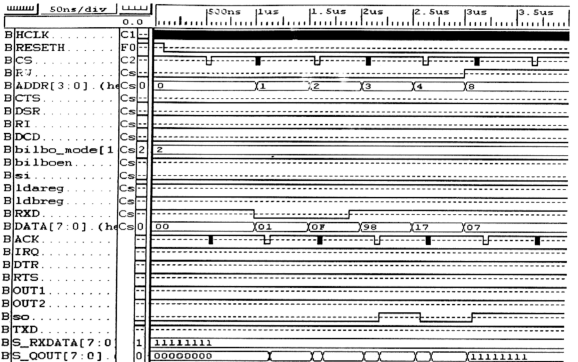

Figure 5.2: 8-bits Data Transmission Setup

In **Figure 5.2,** ADDR[3:0] = '0_H' will select Line Control Register 1 (LCR1). The LCR1 will be set with DATA[7:0] which acts as a communicator between the UART and the CPU. When DATA[7:0] is selected as "00_H", the LCR will translate the input and specify the UART with 8-bits mode, 1 stop bit and no parity bit. The selected data also disable both stick parity bit and break interrupt.

Table 5.2: ADDR[3:0] and DATA[7:0] corresponds to the simulation result.

ADDR[3:0]	DATA[7:0]	Description
1_H (LCR2)	01_H	- Specify the UART's register as transmitter - Normal channel mode - Disable other bits

2_H (MCR)	$0F_H$	- Force DTR, RTS, Out1 and Out2 to '1'
3_H (BRSEL0)	98_H	- Baud Rate Select 0 using 40MHz clock for 115.2kbps data transmission
4_H (BRSEL1)	17_H	- Baud Rate Select 1 using 40MHz clock for 115.2kbps data transmission
8_H (HOLD)	07_H	- Parallel data of Transmitter Holding Register (THR) (to be sent as serial data)

Following the LCR, the function of ADDR[3:0] and DATA[7:0] corresponds to the simulation result are described in **Table 5.2.**. BRSEL or baud rate select is a programmable baud rate generator used for selecting the baud rate transmission of the UART using the specified clock (40 MHz). The following equation gives the calculation of the baud rate for any value of N to be programmed into BRSEL1 and BRSEL0:

$$\mathbf{N = (16 \times 2^n \times BaudRate / Freq_{HCLK})}$$

= BRSEL1& BRSEL0 (in hexadecimal)

Where:

Baud rate = bit/sec.

$Freq_{HCLK}$ = Frequency of HCLK in Hertz

N = decimal value to program into BRSEL1 & BRSEL0

n = accumulator width in bits

Solving for N when a maximum baud rate of 115,200 baud is to be used, accum_size generic = 16 (i.e. accumulator is 17 bits wide) and a host clock frequency of 40MHz:

$$N = (16 \times 2^{17} \times 115200 / 40 \times 10^{6}) = 6039.798$$

$$\approx 6040_{d} \text{ (in decimal)}$$

$$\approx 1798_{h} \text{ (in hexadecimal)}$$

The value of BRSEL0 is 98 and BRSEL1 is 17 (8-bits wide each).

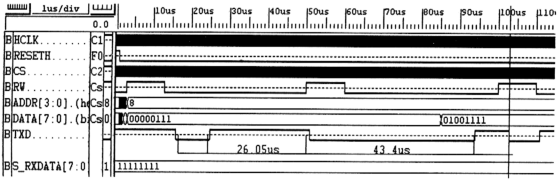

Figure 5.3: Serial 8-bits Data Transmission at TXD

Figure 5.3 shows the signal results of the 8-bits data ("00000111_{B}") transmission via DATA[7:0]. The transmitted serial data can be observed at TXD (1 low start bit, 8 data bits (LSB to MSB), and 1 high stop bit). The 3-bits high data are equal to 26.05us therefore 1 data bit is equal to 26.05us/3 = 8.68us. From this information, the baud rate can be calculated as 1/8.68us, which is equal to 115,207. The result achieved is almost

the same as 115.2k (the value of the calculated baud rate select (BRSEL) using paper calculation).

5.3.2 Transmitter 5-Bits Data Transmission

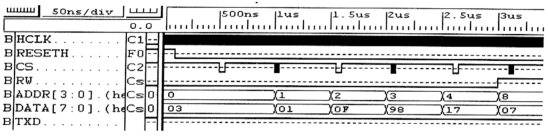

Figure 5.4: 5-bits Data Transmission Setup

Tx5bit_tb.cmd is the script file of 5-bits data transmission. The 5-bits transmission setup is almost the same as the 8-bits data transmission. The only difference is located at DATA[7:0]. ADDR[3:0] = '0_H' will select Line Control Register 1 (LCR1), then the LCR1 is set with DATA[7:0] = "03_H". This allows LCR1 to choose 5-bits mode data transmission.

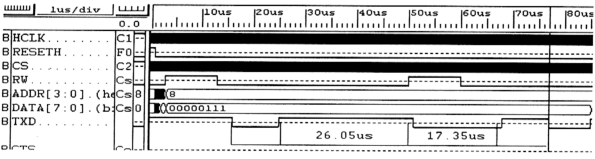

Figure 5.5: Serial 5-bits Data Transmission at TXD

Figure 5.5 shows the result of the 5-bits UART transmission where three of the MSB DATA[7:0] are ignored. The TXD has transmitted 5-bits serial data with 1 start bit, 5 data bits (LSB first) and 1 stop bit. The transmitted data bits consist of 3-bits high (26.05us) and 2-bits low (17.35us) data. The 2-bits low data at the end of TXD is equal to 17.35us, which mean 1 bit is equal to 17.35us/2 = 8.675us. Therefore, the baud rate selected is 1/8.675us = 115.273k (almost the same as 115.2k, the value of the calculated baud rate select (BRSEL)).

5.3.3 Transmitter Break Interrupt

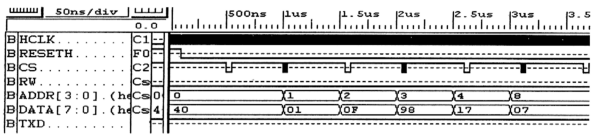

Figure 5.6: Transmitter Break Interrupt Setup

Break control is set up as the previous simulation. The difference only can be seen at DATA[7:0] = "40_H". Txbrk_tb.cmd sets DATA[7:0] with ADDR[3:0] = '0_H'. This enables the break control to force the serial output (TXD) to '0'.

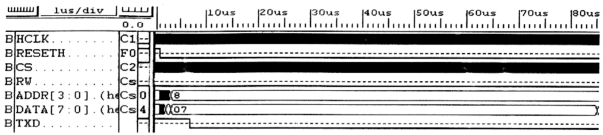

Figure 5.7: Transmitter Break Interrupt at TXD

From **Figure 5.7,** it is observed that TXD always '0' even the parallel data transmitted at DATA[7:0] is "07_H" or "00000111_B".

5.3.4 Transmitter Even Parity Mode

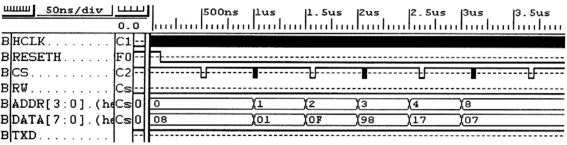

Figure 5.8: Transmitter Even Parity Setup

In Txpty_tb.cmd, the DATA[7:0] is set to "08_H" and specified with ADDR[3:0] = '0_H'. The rest of the setup is similar with the previous simulation.

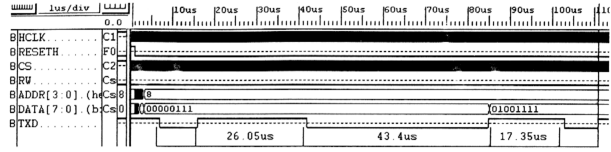

Figure 5.9: Transmitter Even Parity Data Transmission at TXD

Figure 5.9 shows the transmitted serial data (TXD) consists of 1 low start bit, 8 data bits (3 high = 26.05us, 5 low = 43.4us), 1 parity bit and 1 stop bit. The right most of the transmitted (TXD) data are 2 bits (17.35us) high (1 high parity bit and 1 high stop bit). The result shows that even number of high bits (3 high data bits + 1 parity bit = 4 (even)) has been transmitted. This proves the theoretical value of even parity mode. (Note: 1-bit ≈ 8.68us)

5.3.5 Transmitter CTS Enable (txctsen)

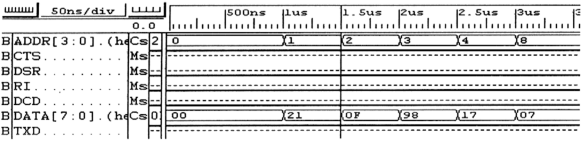

Figure 5.10: Transmitter CTS Enable Setup

The function of Transmitter CTS Enable (txctsen) is to enable Clear To Send (CTS) in order to transmit serial data (at TXD) when CTS = '0'. The data transmission (TXD) will stay high if CTS = '1'. **Figure 5.10** shows the setup of the Transmitter CTS Enable. The setup is similar to the 8-bits data transmission, only ADDR[3:0] = '1_H' is set with DATA[7:0] = "21_H" or "00100001_B". This setup will enable CTS (bit 5 of DATA[7:0]) at LCR2. **Figure 5.11** and **Figure 5.12** show the simulation results of the serial data transmission (TXD) when CTS = '1' and CTS='0'.

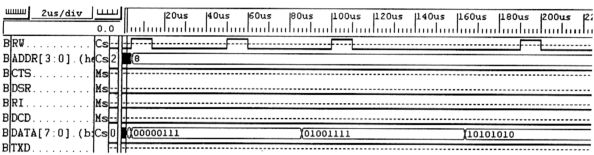

Figure 5.11: TXD when CTS = '1' and tcstsen is enable

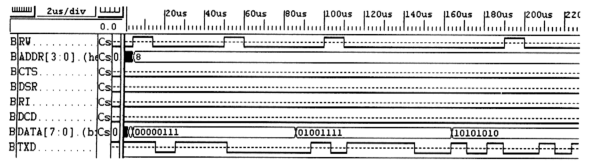

Figure 5.12: TXD when CTS = '0' and tcstsen is enable

As can be observed in **Figure 5.11** and **Figure 5.12**, the serial data is transmitted at TXD when CTS = '0' and stays high if CTS = '1'.

5.3.6 Interrupt Mask Register (INTMSK)

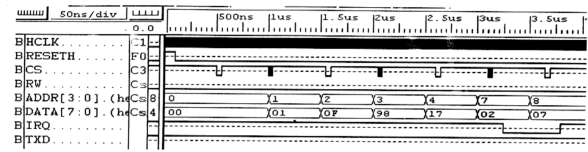

Figure 5.13: Interrupt Mask Register setup

The setup of Interrupt Mask Register needs one extra step from the 8-bits data transmission setup. The extra step sets ADDR[3:0] = "7_H" with DATA[7:0] = "02_H" or "00000010_B". The setup will enable TXRDY Interrupt Mask of INTMSK (ADDR[3:0] = "7_H"). An interrupt request (irq) will be activated when Transmitter Holding Register (THR) is full. The high irq requests the transmitter's register to write out its contents. It can be observed that the irq (**Figure 5.14**) will become low then high when rw is alternated between '1' and '0' but stays low when rw is high. The irq stays low because a high rw forces the CPU to read the UART holding register.

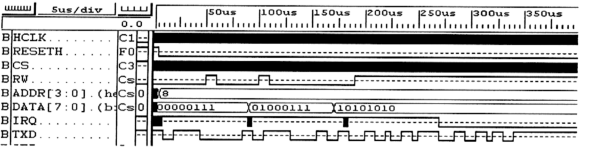

Figure 5.14: Interrupt Request (IRQ) when TXRDY Interrupt Mask is enabled

In the "Transmitter Simulation" section, the reader has been presented with the simulated waveforms of the UART's transmitter. The basic principles of transmitting 8 and 5-bits data in a communication line has shown how data is transmitted using a UART. The section also presented the line break generation, parity mode, interrupt mask register and transmitter CTS enable. The feature provided by the UART is to assure the data will be transmitted correctly. In the following section, the receiver simulation will be conducted. As the transmitter, simulated waveform of the UART's receiver will be presented.

5.4 Receiver Simulation

5.4.1 Receiver 8-bits Mode

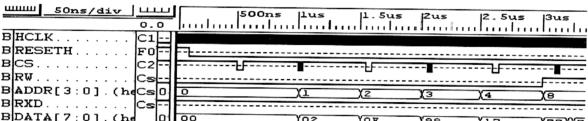

Figure 5.15: Receiver 8-bits Setup

Rxnorm_tb.cmd will configure the designed UART to a receiver 8-bits mode. The setup for the receiver 8-bits mode is shown in **Figure 5.15**. At 3us, DATA [7:0] is manually disconnected by selecting Stimulator Mode => Disconnected from the pop-up menus of the "Logic Simulator" (using right-click of a mouse that has been pointed to

the DATA[7:0] stimulator (Cs)). The disconnection will force the bi-directional DATA[7:0] to high impedance and prevents new inputs from entering DATA[7:0] pins. Then, the disconnection will force the bi-directional DATA[7:0] to show the output results of the simulated logic gates.

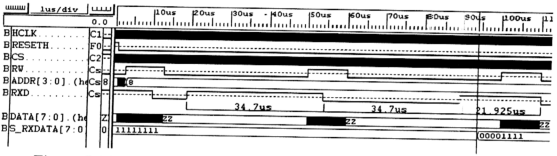

Figure 5.16: 8-bits Data Received at DATA[7:0] when Data Output Enabled. Internal Data Received at S_RXDATA.

Figure 5.16 shows how 8-bits serial data from RXD is received. The data consist of 4 high bits and 4 low bits (1 bit = 8.675us). The data is then converted to parallel S_RXDATA at 93.01us. S_RXDATA is the internal parallel data received at the receiver. The output of the received parallel data is then routed to DATA[7:0] output's pin. The parallel data will be activated when the data output is enabled (s_dataoe = '0'). The received data can be observed between the high impedance "ZZ_H" at DATA[7:0] (if the time/div is widened).

5.4.2 Receiver 5-bits Mode

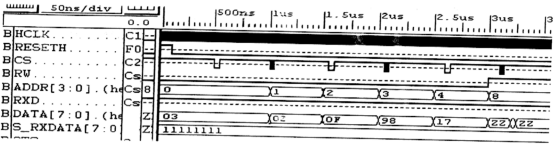

Figure 5.17: Receiver 5-bits Setup

The setup for the receiver 5-bits mode is quite similar to the receiver 8-bits mode. For the receiver 5-bits mode, "Rx5bit_tb.cmd" will specify the DATA[7:0] = "03_H" at ADDR[3:0] = '0_H'.

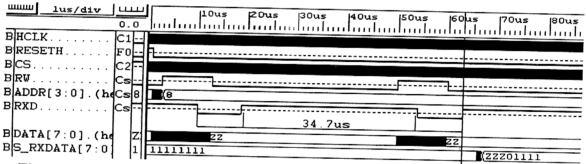

Figure 5.18: 5-bits Data Received at DATA[7:0] when Data Output Enabled. Internal Data Received at S_RXDATA.

The serial data received at RXD is 4 bits high (34.75us) and 1 bit low. The received serial data is then converted to internal parallel data ("01111_B"). The internal parallel data can be observed at S_RXDATA. The "ZZZZ" (at the MSB of S_RXDATA) is high impedance data that will be ignored by the output pin since the receiver is in 5-bits mode. The output bi-directional DATA[7:0] pin then reads as "XXXX01111", where X can be ignored.

5.4.3 Receiver Error and LSR

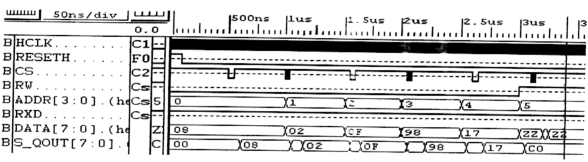

Figure 5.19: Receiver Error and LSR setup

In this section, the simulation of the internal diagnostic capabilities of the UART will be conducted. The received data at RXD is purposely made to show the simulation of parity, overrun and framing error and to prove the internal diagnostic capabilities in the system designed.

Rxerr_tb.cmd will specify the ADDR[3:0] and DATA[7:0] as the previous simulation. However, there are two changes made to the script file. The script file specifies ADDR[3:0] = '0' with DATA[7:0] = "08_H" or "00001000_B", forcing the normal UART's receiver mode to even parity mode. Instead of using ADDR[3:0] = '8_H' (holding register), ADDR[3:0] = '5_H' (LSR mode) is specified in Rxerr_tb.cmd. At 3us (when LSR starts) the DATA[7:0] is manually disconnected using Stimulator Mode => Disconnected.

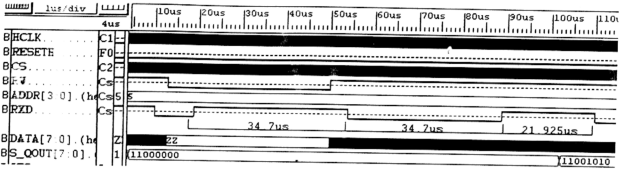

Figure 5.20: Results of Receiver Error and LSR

Figure 5.20 shows the results of the simulated logic gates at 5us to 115us. The first S_QOUT[7:0] is observed as "11000000_B" which indicates that the THR and TSR are full (txempty='1', txrdy='1'). THR and TSR will always full since LCR2 is in receiver mode, therefore they will not affected. At 101.7us, it can be observed that S_QOUT[7:0] has changed to "11001010". The second bit (high) from right shows that the receiver shift register (RSR) is full (rxrdy = '1'). The 4th bit (high) from right shows that there is a parity error (pty_err = '1'). The parity error occurred because the data received at RXD has a high parity bit although the parity bit was specified as even parity mode. 4 high data bits plus 1 high parity bit is equal to 5 (odd) high bits (result of odd parity mode). Therefore, the parity error bit acknowledges the user that there is a parity error in the receiver's register. For even parity mode, an even (e.g. 2, 4, 6) number of high bits should be received.

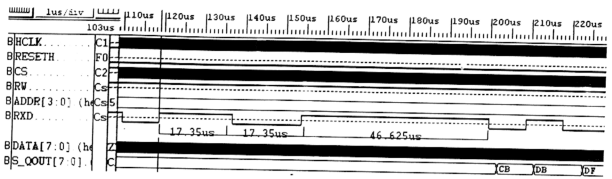

Figure 5.21: Results of Receiver Error and LSR (continued)

Figure 5.21 is the continuation result of **Figure 5.20** taken from 110us to 220us. S_QOUT[7:0] result has changed to "CB_H" or "11001011_B". The first bit (high) from right shows that the "Receiver Hold Register" (RHR) is full (rxfull = '1'). This occurs because there is a new data byte received at RXD before RSR is emptied or read out.

After the result "CB_H" is achieved, S_QOUT[7:0] changes again to "DB_H" or "11011011_B". The high 5th bit from right shows that a frame error has occurred (frm_err = '1'). This is because the data received at RXD has been purposely set to 1 start bit, 2 high data bits followed by 2 low data bits then 4 high data bits, 1 parity bit, and 1 stop bit. From the setup, the stop bit should be received at 197us (Note: (119 + 8.675X9)us ≈ 197us). However, the stop bit received is not high at least half a bit period, therefore the frame error occurred. Frame error will be activated if there is an invalid stop bit or the stop bit received is equal to '0'.

The S_QOUT[7:0] = "DF_H" or "11011111_B" shows that an overrun error has occurred. The overrun error occurred because there is a third byte received at RXD, although the RSR and RHR are still full.

The simulation done in "Receiver Simulation" section has proven the theory on how data are received by the UART's receiver. This section has also presented the line status register simulation, which reports the types and conditions of the transfer operation performed by the UART. The status register also reports the status information of parity, overrun and framing error conditions. In the coming section, the focus will be on the test capability feature which has been incorporated to the UART design. The simulated waveform of Built-In-Self-Test technique in a UART design will be presented and discussed.

5.5 BIST Simulation

5.5.1 BIST Mode

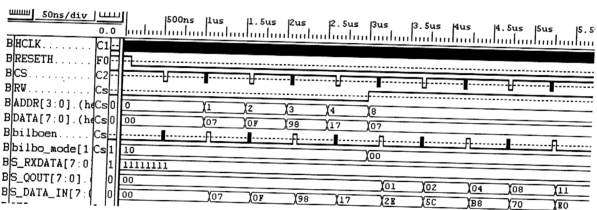

Figure 5.22: BIST Setup

The script file Bist_tb.cmd will be used to test 8-bits data with BIST. The detailed discussion on how the UART was implemented with BIST architecture has been carried out in **Chapter 3** under sub-topic "UART with BILBO Register and Tester". In this particular simulation case, the UART should be set to internal loop back mode (chanmode = "01_b"). This mode will be used to test both the transmitter and receiver of the UART. The mode will loop-back the serial data and transmit the data back to the receiver.

As before, the setup for ADDR[3:0] and DATA[7:0] are made. The ADDR[3:0] = '1_H' is set with DATA[7:0] = "07_H" or "00000111_B". The first 2-bits (high) from right of DATA[7:0] are set to activate both the receiver and transmitter (txen = '1', rxen = '1'). The 4th and 3rd bit from right enable the channel mode to internal loop-back mode (chanmode = "01_b").

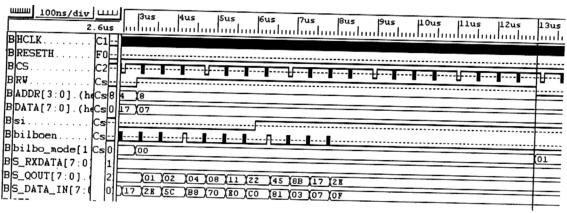

Figure 5.23: BILBO Act as Shift Register

To start the test, the BILBO_MODE (before 3us) is set to "10_B" to operate in normal mode. From 3us to 13us, the BILBO_MODE is set to "00_B" and acts as a shift register. BILBO shift register then shifts the value of S_DATA_IN from right to left (LSB to MSB) by using a serial in (si) pin (**Figure 5.23**). The shifting is conducted to initialize the LFSR and MISR with "seed". S_DATA_IN then pushes the data to S_QOUT after 2-clock delay. **Table 5.3** shows S_QOUT and S_DATA_IN in binary. As can be observed at the end of **Table 5.3,** the LFSR is initialized to "$0F_H$" and MISR is initialized to "$2E_H$".

Table 5.3: S_QOUT and S_DATA_IN in binary

S_QOUT (HEX)	S_QOUT (BINARY)	S_DATA_IN (HEX)	S_DATA_IN (BINARY)
01	00000001	2E	**0**0101110
02	00000010	5C	**0**1011100
04	00000100	B8	**1**0111000
08	00001000	70	**0**1110000
11	00010001	E0	**1**1100000
22	00100010	C0	**1**1000000
45	01000101	81	**1**0000001
8B	10001011	03	**0**0000011
17	00010111	07	00000111
2E	00101110	0F	00001111

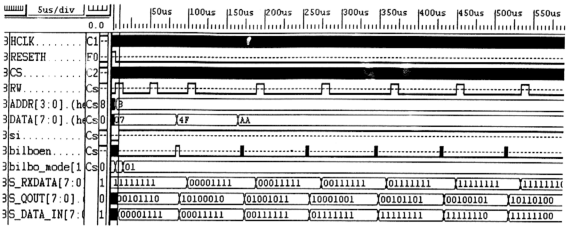

Figure 5.24: BILBO Act as LFSR and MISR

Figure 5.24 shows that the BILBO_MODE is set to "01_B". In this mode, the BILBO at the transmitter is configured as LFSR and BILBO at the receiver is configured as MISR. During BILBO_MODE = "01_B" DATA[7:0] can be ignored.

S_DATA_IN acts as a LFSR that produces parallel pseudo random pattern (PRPG) signals to the UART's transmitter. These parallel signals are then converted to serial data in a communication line and will be looped back to the receiver. The receiver then converts the data back to parallel and forwards it to S_RXDATA. S_QOUT (MISR) compresses all the received pseudo random parallel data (S_RXDATA) into one signature. The produced signature is then compared with the correct signature.

S_DATA_IN (PRPG) is achieved by XOR-ing bit 1,2,3 and 7 (MSB...LSB, b7...b0) and then the XORed result is placed to bit 0 (LSB). The other remaining bits (b6...b0) are then shifted to the left.

The S_QOUT (MISR) is achieved as S_DATA_IN except that the produced data (PRPG) is then XORed with S_RXDATA. The examples below show how the simulated logic gates on **Figure 5.24** produced the signal result.

Example 1 "00101110" (S_DATA_IN) XORed and shift will produce "01011101" then "01011101" XORed with "11111111" (S_RXDATA) will produce "10100010"

Example 2 "10100010" (S_DATA_IN) XORed and shift will produce "01000100" then "01000100" XORed with "00001111" (S_RXDATA) will produce "01001011"

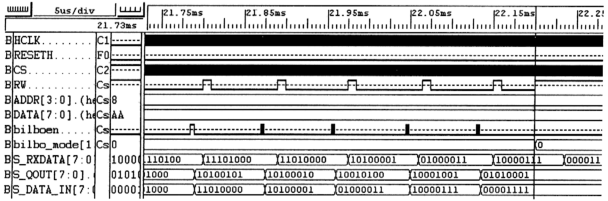

Figure 5.25: Final Value of MISR (S_QOUT) to be shifted out as serial signature

The process of feeding the transmitter with pseudo random data and compressing it with MISR is complete after 255 clocks iteration. The final result of MISR (S_QOUT) in **Figure 5.25** can be observed as "01010001_B". However, S_QOUT (MISR) is the internal data of the designed UART. Therefore, there should be a method to send out the signature without sacrificing extra observance output pins. The signature is shifted out at serial data out (so) output's pin. To shift out the signature, BILBO_MODE is set back to "00_B" and acts as a shift register. In **Figure 5.26,** bilboen signal enables the signature to be transmitted as a serial out data (so). As can be observed, 'so' is transmitted as the following sequence: 1 low bit, 1 high, 1 low, 1 high, 3 low and 1 high. The result is the serial data of "01010001_B", which is equal to the value of MISR at S_QOUT. The signature produced is also similar with the correct signature achieved from the simulation of the entire self test sequence approach using C programming (**Appendix 1**).

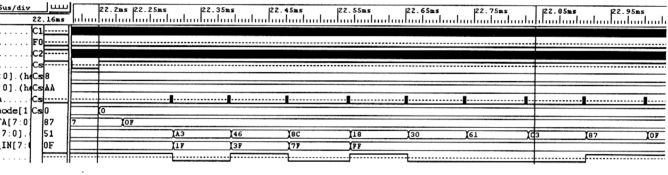

Figure 5.26: Serial Data Out Signature at "so"

The simulated waveforms presented in this chapter have proven the reliability of the VHDL implementation to describe the characteristics and the architecture of the

designed UART with embedded BIST. Furthermore, the simulated waveforms have shown the observer how fast a test result can be achieved by using the Built-In-Self-Test technique. The test as shown in **Figure 5.26** is completed at 22.2ms using 40 MHz clock speed.

In order to further the verification to a real FPGA, the following chapter will be focusing on the implementation of the designed UART in a hardware level. The hardware test of the FPGA will be conducted using XS40 test board and XSTOOLs provided by XESS Corporation.