CHAPTER 3

UART with BIST

3.0 Introduction

There are two ways a computer can transfer data. The two ways are parallel data communication and serial data communication. In parallel data communication, often eight or more lines of wire conductors are used to transfer data to two devices. For that reason, lot of data can be transferred in a short time by using many parallel wires. However parallel communication fails to transfer data in a greater length (more than few feet away) since long cables diminish and even distort signals. To transfer data in much greater length, serial data communication is used. Serial data communication made communication between two systems located at distances of hundreds of feet and millions of miles possible.

Serial data is transmitted via its serial port. A serial port is one of the most universal parts of a computer. It is a connector where serial line is attached and connected to peripheral devices such as mouse, modem, printer and even to another computer. In contrast to parallel communication, these peripheral devices communicate using a serial bit stream protocol (where data is sent one bit at a time). The serial port is usually connected to an integrated circuit called a Universal Asynchronous Receive/Transmit (UART) which handles the conversion between serial and parallel data.

To fulfill the needs of most customers who are expecting the designer to include testability feature, this chapter will direct the reader to the implementation of Built-In-Self-Test (BIST) technique to a UART design. Using communication between two PCs as example, the discussion will begin with answering the question on how does the UART works. Following the question, the architecture of UART and BIST will be described. The discussion will continue with the insertion of BIST circuitry that will allow efficient test coverage to the UART design. At the end of this chapter, the features of the proposed UART design will be presented.

3.1 Universal Asynchronous Receive/Transmit (UART)

Figure 3.1 shows how the UART receives a byte of parallel data and converts it to a sequence of voltage to represent 0s and 1s on a single wire (serial). To transfer data on a telephone line, the data must be converted from 0s and 1s to audio tones or sounds (the audio tones are sinusoidal shaped signals). This conversion is performed by a peripheral device called a modem (modulator/demodulator). The modem takes the signal on the single wire and converts it to sounds. At the other end, the modem converts the sound back to voltages, and another UART converts the stream of 0s and 1s back to bytes of parallel data.

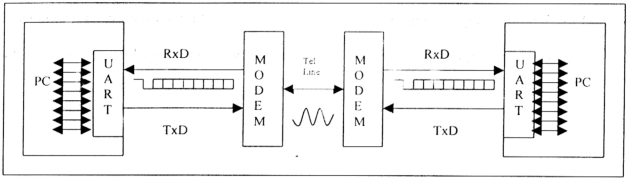

Figure 3.1: Serial Data Transmission and Receive

3.2 UART Architecture

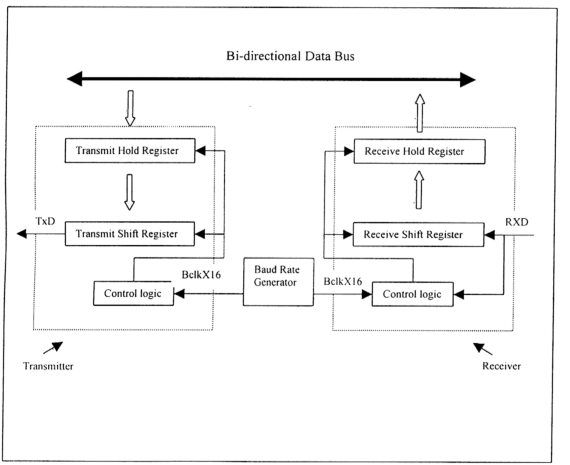

Figure 3.2: UART block diagram

Figure 3.2 shows a UART block diagram, which consists of two independent modules [Oelsner, 2000] (the transmitter and the receiver). Each module implements its own function as a transmitter or a receiver. Both the transmitter and receiver modules can be combined as a top-level design and writes or reads data all through bi-directional CPU interface. They also shared bi-directional data bus, system clock and reset lines. However, they have separate inputs and outputs for most of their control lines. They also have separate unsynchronized clock signals.

In this section, the reader will be provided with the description of the UART's transmitter followed by the description of the UART's receiver. Next, the discussion will continue with the description of a baud rate generator. The discussion conducted in this section will provide the reader with the basis on how the UART is designed.

3.2.1 The Transmitter

The transmitter's module consists of Transmitter Holding Register (THR) and Transmitter Shift Register (TSR). Both the THR and TSR are controlled by a control-logic to perform their functions. THR holds the contents of parallel data (i.e. 5-, 6-, 7-, 8-bits) from data bus and TSR on the other hand shifts out the contents of the THR. Before any transmission is conducted, a high reset forces the transmitter into idle mode. The transmitter then waits for a new data to be written to the THR. A new data is detected when the transmitter is not in the idle mode. If a new data is detected, the transmitter will enter load mode. In the load

mode, the contents of THR are loaded into TSR and at the same time, the transmitter's output is asserted with a low start bit. After the contents of the THR are successfully loaded, the data will enter shift mode. In this mode, data from TSR is shifted to the transmitter's output. The shifting is controlled by 16X clock, which transmits one bit every 16-clock pulse. Furthermore, a flag bit in a status register will be set to true in the shift mode. The computer can read the flag bit to see if the UART is ready to transmit another byte. The shifted data is then transmitted as serial data frames at the transmitter's output. During the transmission, a parity bit and stop bit will be generated depending on the users chosen mode. The transmitter sequence is summarized in **Figure 3.3**.

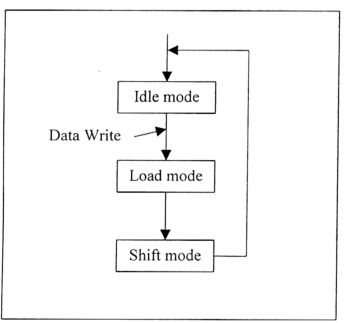

Figure 3.3: Transmitter sequence of UART

3.2.2 The Receiver

As the transmitter, the receiver also has two registers, Receiver Holding Register (RHR) and Receiver Shift Register (RSR). RHR is a register (i.e. 5-, 6-, 7-, 8-bits) that holds the contents of data received at the receiver's input. RSR is a register used for shifting the data at the receiver's input. Similar to the transmitter, these registers are also controlled by a control-logic. However, the receiver has more complicated architecture. This is due to the responsibility of catching a data transmitted by another transmitter on an asynchronous bus (which probably has a clock with different phase and potentially a bit different in period). Before any data is received, active high-reset forces the receiver to idle mode. In the idle mode, the receiver waits for the receiver's input to go low. If falling edge is detected, the receiver will enter 'hunt' mode and searches for a valid start bit. The valid start bit can be recognized by assuring that the signal stays low at least half a bit period by using a 16X high-speed clock. If valid start bit is detected, the receiver will enable the clock used and synchronized it to the center of the start bit. Then, the serial data will enter shift mode where it will be shifted from the receiver's input to the RSR. If an invalid start bit is detected, the receiver will return to 'idle' mode. During reception of a data frame, various parity and error checks (framing and overrun error) are performed. When the receiver finished shifting and a complete data frame has been received, the receiver will return to 'idle' mode. At the same time, the contents of RSR will be loaded into the RHR.

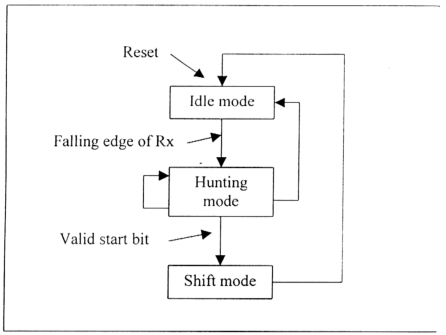

Figure 3.4: Receiver sequence

3.2.3 Baud Rate Generator

A programmable baud rate generator is capable of dividing the timing reference clock input by divisors of 1 to (2^{16} - 1), and generates 16 times the actual required baud rate. The clock is controlled by the Baud Rate Select registers (BRSEL0 and BRSEL1); together form a 16-bits integer 'N'. BRSEL1 is the most significant byte of the integer 'N' and BRSEL0 is the least significant byte. The following equation gives the baud rate for any value of N that can be programmed into BRSEL1 and BRSEL0:

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

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:

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

= BRSEL1& BRSEL0 (in hexadecimal)

The size of the accumulator is controlled by a generic "accum_width" and the user can change this as required. Larger width accumulators will require larger values of N and may mean that some higher speed baud rates cannot be generated.

The architecture of the UART has been presented in this chapter consists of two independent modules (the transmitter and receiver) and a programmable baud rate generator. To improve its test capability a modification has to be made to the UART design to incorporate BIST technique. A general BIST architecture is to add a Test Pattern Generator (TPG), Output Data Compactor (ODC) and Test Controller (TC) hardware blocks to the design to be tested (the UART). In the following section,

detailed design for test (DFT) methodology will be made to create a clearer picture on how to implement BIST technique to a UART design.

3.3 BIST Consideration

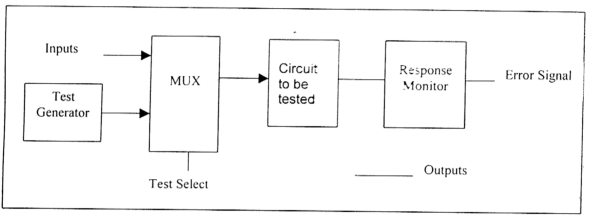

Figure 3.5: Generic BIST

This section introduces the design for test steps and the rules that should be followed to implement BIST technique to the UART design. A general method for using BIST is illustrated in **figure 3.5**. The first step to implement BIST is to identify the possible way to run self-test process by modifying the UART circuitry. The UART circuitry should consists of Test Pattern Generator (TPG) and Output Data Compactor (ODC). A modification also has to be made to the UART design to create a Test Controller (TC) circuit. The TC will select test modes to invoke an on-chip TPG (which will apply test patterns to the UART circuit). The resulting output is compressed by the ODC then observed by the response monitor. The response monitor will produce an error signal if an incorrect output pattern is detected.

A description of Linear Feedback Shift Register (LFSR) and Multiple Input Signature Register (MISR) will be carried out in this section. The implementation of BIST technique to the UART design will use these registers to serve as Test Pattern Generator (TPG) and Output Data Compactor (ODC).

3.3.1 Linear Feedback Shift Register (LFSR)

A Linear Feedback Shift Register (LFSR) is a Test Pattern Generator (TPG) approach that does not depend on the availability of an instruction process. The LFSR is constructed by performing exclusive-OR on the outputs of two or more of the flip-flops together and feeding those outputs back into the input of one of the flip-flops (**Figure 3.6**).

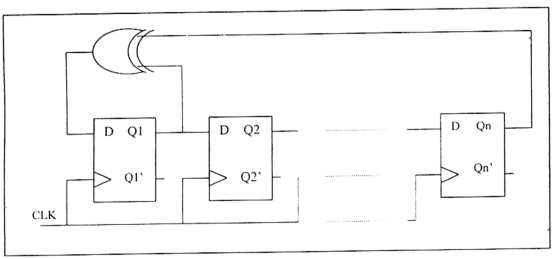

Figure 3.6: n-bits Linear Feedback Shift Register

LFSR makes extremely good Pseudo Random Pattern Generator (PRPG) [Petersom, 1972]. PRPG is a test pattern that has no obvious order and has certain randomness properties. PRPG is obviously very useful for BIST, since they can generate all possible patterns at the input of each sub-circuit with small logic circuitry. Pseudo-random pattern of 1s and 0s are initialized by feeding a "seed" value to the circuitry using a clock pulse. The "seed" can be anything except all 0s (all 0s "seed" will cause the LFSR to produce all 0 patterns).

A maximal-length LFSR produces the maximum number of PRPG patterns possible and has a pattern count equal to $2^n - 1$ (where n is the number of register elements in the LFSR). Because there is no way to predict mathematically if an LFSR will be maximal length, Peterson and Weldon [Petersom, 1972] have compiled tables of maximal-length LFSRs to which designers may refer. **Table 3.1** shows some feedback combination that will generate all 2^n-1 bit patterns for LFSRs with length in the range n=4 to 32.

Table 3.1: Feedback Combination to Generate LFSR

N	Feedback
4,6,7	$Q_1 \oplus Q_n$
5	$Q_2 \oplus Q_5$
8	$Q_2 \oplus Q_3 \oplus Q_4 \oplus Q_8$
12	$Q_1 \oplus Q_4 \oplus Q_6 \oplus Q_{12}$
14,16	$Q_3 \oplus Q_4 \oplus Q_5 \oplus Q_n$
24	$Q_1 \oplus Q_2 \oplus Q_7 \oplus Q_{24}$
32	$Q_1 \oplus Q_2 \oplus Q_{22} \oplus Q_{32}$

3.3.2 Multiple Input Signature Register (MISR)

The MISR is nothing more than a LFSR. MISR can be constructed by modifying a LFSR by adding exclusive-OR gates between the shift register. An n-bit MISR is shown in **Figure 3.7**. The test data (Z1, Z2, Z3,..., Zn) is XORed into the register with each clock, and the results represent a signature that can be compared with the signature for a known correctly functioning component. This type of signature analysis will catch many, but not all-possible errors. An n bit signature registers maps all possible input streams into one of the 2^n possible signatures. One of this is the correct signature, and the other indicates that error has occurred.

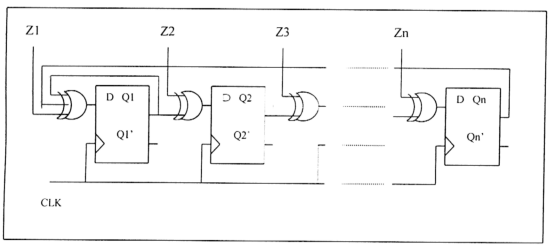

Figure 3.7: n stage LFSR Configured as a MISR

The correct signature can be obtained in two ways:

- **The golden chip approach**

  This approach takes the correct signature by performing the self-test operation on chips, which have passed all other forms of manufacturing and functional test.

- **Simulation of the entire self test sequence**

  This approach will use a simulation tool to simulate the correct signature.

Due to the availability and reliability of simulation tools, the simulation of the entire self-test sequence approach will be used in this thesis.

3.4 UART with BILBO Register and Tester

The problem of testing sequential network is simplified by observing the state of all the flip-flops instead of just observing the outputs. For each state of the flip-flops and for each input combination, the network outputs need to be verified and goes to the correct next state. One approach would be to connect the output of each flip-flop within the IC being tested to one of the IC pins. Since the number of pins on the IC is limited, this approach is not practical. The solution to the question is by arranging flip-flops to form a shift register. The state of the flip-flop will be shifted out bit-by-bit using a single serial-output pin on the IC. This is called scan path testing.

BILBO is a scan register that can be modified to serve as state register, pattern generator, signature register, or shift register. In summary the BILBO operating modes are presented as follows:

B1B2	Operating Mode
00	Shift register
01	LFSR/PRPG
10	Normal
11	MISR

Figure 3.8: BILBO Operating Modes

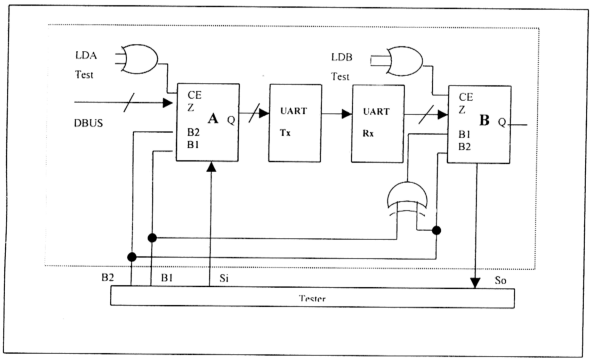

Figure 3.9: UART with BILBO Register and Tester

Figure 3.9 illustrates how to apply BILBO registers to test the UART design. In this structure, "Register A" and "Register B" may be configured by mode control

("bilbo_mode") signal to act as either a shift register, a test pattern generator (PRPG), normal application mode function (normal) or a data compressor (MISR). The test starts with the initialization of the BILBO by applying a "seed" to its serial-in (si) pin. The initialization can be obtained by configuring BILBO's operating mode ("bilbo_mode") to "00" (shift register mode). Following the initialization, the bilbo_mode is set to "01" so that "Register A" is configured as LFSR ("bilbo_mode" = "01") and Register B as MISR ("bilbo_mode" = "11" (Note: XOR force "01" to "11")).

"Register A" (LFSR) produces 8-bits pseudo random pattern data in parallel. Then, the parallel data is feed to the UART's transmitter. The UART converts the pseudo random parallel data to serial data. The serial data is then looped back to its receiver to create an internal diagnostic capability. The UART's receiver converts the serial data back to parallel and will be accepted by "Register B" (MISR). A signature will be produced after 255 clock iterations (8 data bits produce 2^8 = 256 PRPG) and completes the test. The signature is then scanned out from serial output (so) pin by configuring bilbo_mode to "00". Following the scan, the signature is then compared with the correct signature achieved from the simulation of the entire self-test sequence approach in a tester. If the signature produced by MISR is similar to the correct signature, it can be concluded that the UART is working properly.

The implementation of BIST technique in UART design has been discussed thoroughly in this section. Both UART and BIST behavior will be described using VHISC hardware Description Language (VHDL) in the coming chapter. Before

proceeding to the VHDL implementation, it is appropriate to preempt with some brief features of the proposed UART.

3.5 UART Features

In this thesis, the UART will be designed using National Semiconductor and [National, 1995][Martin, 1989] QuickLogic [Oelsner, 2000] standards and freely distributed modules of Generic UART [Harvey, 1999] with some modification to suite the implementation of BIST technique. It will capable and compatible of running 16450 software. The UART will be included with a programmable baud rate generator that capable of dividing the timing reference clock input by divisors of 1 to (2^{16} - 1), and generates 16X clock for driving the internal transmitter logic. The 16X high-speed clock also will be used for sampling data (at the center of a bit) to reliably captures the bit stream to overcome the tricky part of framing error.

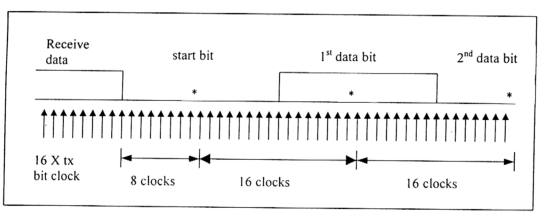

Figure 3.10: Sampling data received with 16 clocks faster than transmitter bit clock

The UART will also equipped with fully programmable serial interface characteristics. It can:

- transmits or receives 5-, 6-, 7- or 8-bits characters,
- capable to generate and detect even, odd, stick and no parity bit,
- generates 1- or 2- stop bits and
- generates baud rate (DC to 1.5M baud)

Other feature that will be provided by the UART is false start bit detection. The false start bit detection will prevent any noise received by the receiver to be interpreted as the data transmitted by the other UART. The UART also will be equipped with line break generation and detection to stop current data from being transmitted or received by the UART. The break occurs when the line is held at logic '0' for a time of one character.

The UART has complete modem control functions (i.e. CTS, RTS, DSR, DTR, RI and DCD), and a processor-interrupt system. Interrupts can be programmed to the user's requirements, minimizing the computation required to handle the communications link. The summary of the modem control functions is:

- DTR and DSR to indicate that the DTE (PC) and DCE (modem) are alive and well
- RTS and CTS control the flow of data

- RTS asserts when DTE wants to send data
- CTS asserts when DCE is ready and has room to accept the data in response to RTS. If no room, CTS does not activate and the DTE will reassert DTR.

Note:

CTS – Clear To Send	RI – Ring Indicator
RTS – Request To Send	DCD – Data Carrier Detect
DSR – Data Set Ready	DTE – Data Terminal Equipment
DTR – Data Terminal Ready	DCE – Data Communication Equipment

The CPU can read complete status of the UART at any time during the functional operation. Status information reports include the type and condition of the transfer-operations being performed by the UART also the status information of error conditions (parity, overrun, framing, or break interrupt). The designed UART will have a nominal voltage supply of 5.0V and low gates count (about 3000 gates).

An internal diagnostic capability features loop-back-control for communication link-fault-isolation. The loop-back will loop back the transmitted data to the receiver to test the functionality of the UART. The internal diagnostic capability also features Built-in-Self-Test (BIST) technique, which allows the UART circuit to test itself.

This chapter has described both the proposed UART and BIST technique that can be implemented together to improve the UART test capability. Following this

discussion, the extensive use of CAD tools and language-based design to describe the structure and behavior of digital electronic hardware designs will be presented in the next chapter. The suitability of VHDL to design BIST will be investigated. In addition, UART with BIST pins and registers descriptions will be presented throughout the rest of the chapter.