# CHAPTER 4

# VHDL IMPLEMENTATION

## 4.0 Introduction

As mentioned in chapter 1, the objective of this thesis is to design UART chip with embedded Built-In-Self-Test technique using one of the most recent Programmable Logic Device (PLD) technology called Field Programmable Gate Array (FPGA). To bring forward on designing a programmable logic device, this chapter will take the reader to the description of "Xilinx Foundation Series 2i" CAD tools. These Computer Aided Design (CAD) tools will be extensively used throughout the design implementation process. It will be used to manipulate and store the logic characteristics of a FPGA in order to make the FPGA works as intended.

The discussion will continue with the description of VHISC Hardware Description Language (VHDL) that will be used to describe the structure and behavior of digital electronic hardware designs. The suitability of VHDL to design BIST will also be investigated and finally, the pins and registers description of a UART with embedded BIST will be presented throughout the rest of the chapter.

## 4.1 Designing a Programmable Logic Device

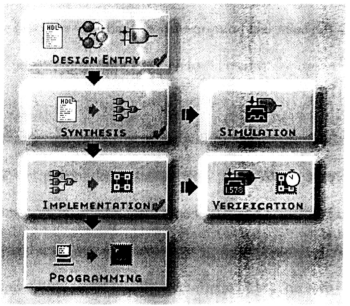


**Figure 4.1:** "Xilinx Foundation Series 2i" design flow

In this section, the description of each step involved in configuring a FPGA will be presented. This section focuses on a typical FPGA design using "Xilinx Foundation Series 2i" CAD tools. As illustrated in **Figure 4.1** the design flow consists of:

- Design entry
- Design implementation
- Verification
- Device programming

A design-entry is a tool that generates a file format for specifying a design netlist. The design-entries supported by Xilinx Foundation Series 2i are schematic entry package, finite state machine, and hardware description language (HDL). The design can also use mixed design entry (e.g. both HDL and schematic).

Following the design entry, the tools will proceed with design implementation. **Figure 4.2** shows design implementation flow of "Xilinx Foundation Series 2i" software. The design implementation tool reads "standard" netlist (i.e. **.xnf – xilinx netlist file) formats and translates it automatically. Once translated, the tool performs a design rule check and optimization on the incoming netlist. The software then partitions the design and transforms it to logic blocks that available on the device chosen at the implementation stage. After the design has been partitioned, the implementation software searches for the best location to place the logic blocks among all of the placement possibilities.

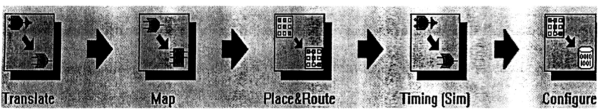


**Figure 4.2:** Design Implementation flow of Xilinx Foundation Series 2i.

The verification step consists of functional and timing simulation. Functional simulation is performed in conjunction with design entry, but before place and route. It is used to functionally simulate a design to guarantee proper functionality. Full timing

simulation must wait after the "place and route" steps. After "place and route", the software back-annotates the logic and routing delays to the netlist for simulation.

After creating a programming file, the programmable device is configured and ready for action. The actual programming method depends on the target technology. Most programmable logic technologies, including the PROMs for SRAM-based FPGAs, require some sort of a device programmer.

## 4.2 VHISC Hardware Description Language (VHDL)

Programming a programmable logic device will start with a design entry, which consists of schematic entry package, finite state machine, and hardware description language (HDL). As integrated circuit technology has become more complex, detailed design of systems at the gate and flip-flop level has become very tedious and time consuming. For this reason, use of hardware description languages in the digital design process has significantly improved in the last few years, especially for FPGA design. A hardware description language allows a digital system to be designed and debugged at a higher level before conversion to the gate and flip-flop level.

One of the most popular hardware description languages is VHISC Hardware Description Language (VHDL) [Roth, 1998]. It is used to describe and simulate the operation of a variety of digital systems, ranging in complexity from a few gates to an interconnection of many complex integrated circuits. There are many excellent hardware

description languages (HDL) were prior to VHDL but VHDL offers a number of benefits over other HDL [Navabi, 1991]. Among the advantages are:

### 4.2.1 VHDL as a Standard Language

VHDL was initiated by the United States Department of Defense in 1981 to allow a uniform method for specifying digital systems. The VHDL language has since becomes an IEEE standard, and it is widely used in industry.

### 4.2.2 Increase productivity

VHDL can increase productivity by shorten the time to market. Behavioral simulation can reduce design time by allowing design problems to be detected early on, avoiding the need to rework designs at gate level.

### 4.2.3 Better design

Behavioral simulation permits design optimization by exploring alternative architectures, resulting in better designs.

### 4.2.4 Reusability for new technology

To move a design to a new technology, a specification needs not to start from scratch or reverse-engineer. Instead, the design tree to a behavioral VHDL description can be implement in the new technology knowing that the correct functionality is preserved.

### 4.2.5 Tools independence

VHDL descriptions of hardware design and test benches are portable between design tools, and portable between design centers and project partners.

### 4.2.6 Minimum cost and time

VHDL make the most reliable design process, with minimum cost and time.

## 4.3 VHDL and BIST

It's becoming increasingly common for Design For Testability (DFT) issues to be addressed at design reviews prior to circuit tape-out approval. Previously, in the age of schematics, this often require design and test engineers to pore over mounds of pages looking for things like asynchronous set/reset circuit configurations, derived or internally generated clocks, and combinatorial and sequential feedback loops. The review inevitably occurred late in the design cycle; adversely affecting project schedules if glitches were found, and making for an uncomfortable process for the circuit designer. With today's design practices, however, schematics are mostly outdated [Turino, 2000]. Designers can take more control of the DFT review by performing DFT rule checking at the RTL (VHDL) level. However, finding DFT problems in language-based designs is still not a simple task for humans.

The acceptance of the design for test techniques has been largely due to the possibility of VHDL support to this design style. It is desirable to eventually have available a built-in-self-test approach with similarly VHDL support. The high degree of standardization makes possible to have most testability feature previously added to a design using VHDL.

## 4.4 The Design

The VHDL role to describe the behavior and the architecture of digital system and its suitability to incorporate BIST technique has been mentioned in the earlier section. To describe the behavior of digital systems in VHDL code, a designer must plan the specification of each pin and register. Therefore, the following discussion will be concentrated on the design specification of a UART with BIST pins and registers. The function of each pin and register will also be discussed in this section.

### 4.4.1 UART Pin Description

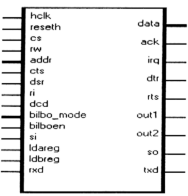


**Figure 4.3:** UART with BIST Top Level Design Symbol

**Table 4.1** will describe the function and the description of all UART pins available at the top-level design (**Figure 4.3**) of the VHDL implementation. The 24 pins UART are describe as follows:

**Table 4.1:** UART Pins Description

| Pin | In/out | Description |
|---|---|---|
| Hclk | IN | **Host Clock**<br>Clock for baud rate generation<br>(e.g. operation frequency = 8MHz therefore the time period should be 1/8MHz=125ns) |
| Reseth | IN | **Asynchronous Reset**<br>Clears all registers and control logic when input is high |
| Cs | IN | **Chip Select**<br>Enables communication between the UART and the CPU. CS is an active low signal latches address strobe for completing chip selection. |
| Rw | IN | **Read/Write**<br>'1' = CPU can read status or data from the selected UART register<br>'0' = CPU can write control or data into the selected UART register<br>(Note: cs should be active) |
| Addr | IN | **Host Address**<br>4 bit inputs, which select UART register. (Details at register's description) |
| Cts | IN | **Clear to Send**<br>When low indicates that the modem or data set is ready to exchange data |
| Dsr | IN | **Data Set Ready**<br>When low indicates that the modem or data set is ready to establish communication link with the UART. |
| Ri | IN | **Ring Indicator**<br>When low indicates that the telephone ringing signal has been received by the modem or data set |
| Dcd | IN | **Data Carrier Detect**<br>When low indicates that the modem or data set has detected a data carrier. |
| Bilbo_mode | IN | **BILBO Operating Mode**<br>"00" = shift register mode<br>"01" = PRPG mode |

| | | |
|---|---|---|
| | | "10" = normal mode<br>"11" = MISR mode |
| Bilboen | IN | **BILBO Enable**<br>'0' = disables BILBO<br>'1' = enables BILBO |
| Si | IN | **BILBO Serial In**<br>A "seed" value to initialize BILBO |
| Ldareg | IN | Load BILBO Register 'A' |
| Ldbreg | IN | Load BILBO Register 'B' |
| Rxd | IN | **Received Data**<br>Serial data input from a communication link (data from peripheral devices, modem or data set). |
| Data | INOUT | **Data Bus**<br>Provides bi-directional communication between the UART and the CPU. Data, control words and status information are transferred via the data bus. |
| Ack | OUT | **Acknowledge**<br>Active low signal, which transfers acknowledge signal to host. |
| Irq | OUT | **Interrupt Request**<br>The interrupt request will be generated for the following conditions:<br>1. Data are ready in Receiver Hold Register (RHR) (if INTMSK is enabled).<br>2. Transmitter Hold Register (THR) is empty (if transmitter and INTMSK are enabled). |
| Dtr | OUT | **Data Terminal Ready**<br>When low, informs the modem or data set that the UART is ready to establish a communication link. |
| Rts | OUT | **Request to Send**<br>When low, informs the modem or data set that the UART is ready to exchange data. |
| Out1 | OUT | **User designated output**<br>Can be set to active low by programming bit 2 of the Modem Control Register (MCR). |
| Out2 | OUT | **User designated output**<br>Can be set to active low by programming bit 3 of the Modem Control Register (MCR). |
| So | OUT | **BILBO Serial Out**<br>The serial signature produced by MISR to be compared with the correct signature |
| Txd | OUT | **Transmitter's Data**<br>Serial data output to a communication link (i.e. peripheral devices, modem and data set). |

### 4.4.2 UART Register descriptions

The system programmer may accesses any of the UART registers by simply referring to its address in **Table 4.2**. To access LCR1 for instance, the address ADDR[3:0] should be set to $0_d$ (or "$0000_b$" in binary) when chip select (CS) is active (low). If the address (ADDR[3:0]) is set to $5_d$ (or "$0101_b$" in binary), the Line Status Register (LSR) will be selected.

**Table 4.2:** UART Registers Description

| Address | Description |
|---|---|
| 0 | Line Control Register 1 (LCR1) |
| 1 | Line Control Register 2 (LCR2) |
| 2 | Modem Control Register (MCR) |
| 3 | Baud Rate Select 0 (BRSEL0) |
| 4 | Baud Rate Select 1 (BRSEL1) |
| 5 | Line Status Register (LSR) |
| 6 | Modem Status Register (MSR) |
| 7 | Interrupt Mask Register (INTMSK) |
| 8 | Holding Register (HOLD) |

#### 4.4.2.1 Line Control Register 1 (LCR1)

The line control register can be divided into two, LCR1 and LCR2. Both LCR1 and LCR2 specify the format of an asynchronous data communication used by the UART. LCR1 controls the bit modes, stop modes and the parity modes of the designed UART and enables (or disables) line break generator. LCR2 on the other hand, enables (or disables) the transmitter and the receiver. It

also can configure UART's channel mode in order to act in normal mode, internal loop mode or auto echo mode. Furthermore, LCR2 can enable Clear To Send (CTS) and Request To Send (RTS) to control the flow of data.

**Table 4.3:** LCR1 Description

| Bit | Description |
|---|---|
| 1-0 | **Bit modes (bitmode)**<br>"00" = 8-bits mode<br>"01" = 7-bits mode<br>"10" = 6-bits mode<br>"11" = 5-bits mode |
| 2 | **Stop modes (stopmode)**<br>'0' = 1 stop bit mode<br>'1' = 2 stop bits mode |
| 4-3 | **Parity modes (ptymode)**<br>"00" = no parity mode<br>"01" = even parity mode<br>(Even number of '1' is transmitted in each word)<br>"10" = odd parity mode<br>(Odd number of '1' is transmitted in each word)<br>"11" = reserved |
| 5 | **Stick Parity modes (stick)**<br>'0' = disables stick parity mode<br>'1' = enables stick parity (If even parity mode, the parity bit will be transmitted and checked as logic '0' and if odd then the parity bit will be transmitted and checked as logic '1') |
| 6 | **Break control (brk_ctrl)**<br>'0' = disables break control<br>'1' = enables break control (force serial out to '0') |
| 7 | Reserved for future use |

### 4.4.2.2 Line Control Register 2 (LCR2)

**Table 4.4:** LCR2 Description

| Bit | Description |
|---|---|
| 0 | **Transmitter enable (txen)**<br>'0' = disables transmitter<br>'1' = enables transmitter |
| 1 | **Receiver enable (rxen)**<br>'0' = disables receiver<br>'1' = enables receiver |
| 3-2 | **Channel modes (chanmode)**<br>"00" = normal mode<br>"01" = internal loop (used for UART test)<br>"10" = auto echo mode<br>"11" = reserved |
| 4 | **Receiver RTS enable (rxrtsen)**<br>'0' = disables RTS<br>'1' = enables RTS |
| 5 | **Transmitter CTS enable (txctsen)**<br>'0' = disables CTS<br>'1' = enables CTS |
| 7-6 | Reserved for future use |

### 4.4.2.3 Modem Control Register (MCR)

The Modem Control Register (MCR) transfers control signals to a modem which connected to the designed UART. The descriptions are as follows:

**Table 4.5:** Modem Control Register Description

| Bit | Description |
|---|---|
| 0 | **Force DTR (force_dtr)**<br>'1' = Forces DTR to '1' |
| 1 | **Force RTS (force_rts)**<br>'1' = Forces RTS to '1' |

| | |
|---|---|
| 2 | **Force Out1(force_out1)**<br>'1' = Forces out1 to '1' |
| 3 | **Force Out2 (force_out2)**<br>'1' = Forces out2 to '1' |
| 7-4 | Reserved |

### 4.4.2.4 Linear Status Register (LSR)

LSR provides the status information of the designed UART to the CPU. The status information reports the type and condition of the transfer operations and the status information of error conditions (overrun, parity and frame error). The description of the line status register is described in **Table 4.6**.

**Table 4.6:** Line Status Register Description

| Bit | Description |
|---|---|
| 0 | **Receiver full (rxfull)**<br>'0' = Receiver Hold Register is empty<br>'1' = Receiver Hold Register is full |
| 1 | **Receiver ready (rxrdy)**<br>'0' = Receiver Shift Register is empty<br>'1' = Receiver Shift Register is full |
| 2 | **Overrun error (ovr_err)**<br>'0' = No overrun state<br>'1'= If the Receiver Holding Register (RHR) is full and another character received at the Receiver Shift Register (RSR) |
| 3 | **Parity error (pty_err)**<br>'0' = No parity error<br>'1'= Character received with parity error |
| 4 | **Frame error (frm_err)**<br>'0' = No frame error<br>'1'= Character received does not has a valid stop bit |

| | |
|---|---|
| 5 | **Break interrupt (brk_intrpt)**<br>'0' = No break condition in the current character<br>'1'= A break condition has been reached in the current character. The break occurs when the line is held at logic 0 for a time of one character (start bit + data + parity + stop bit) |
| 6 | **Transmitter ready (txrdy)**<br>'0' = Transmitter Holding Register (THR) is empty<br>'1' = THR is full |
| 7 | **Transmitter empty (txempty)**<br>'0' = Transmitter Shift Register (TSR) and THR is empty<br>'1' = TSR and THR is full |

### 4.4.2.5 Modem Status Register (MSR)

Modem Status Register (MSR) provides modem status information of a control line.

**Table 4.7:** Modem Status Register Description

| Bit | Description |
|---|---|
| 0 | **Delta Clear To Send (delta_cts)**<br>'1' = The CTS line has changed its state |
| 1 | **Delta Data Set Ready (delta_dsr)**<br>'1' = The DSR line has changed its state |
| 2 | **Delta Ring Indicator (delta_ri)**<br>'1' = The RI line has changed its state (from low to high) |
| 3 | **Delta Data Carrier Detect (delta_dcd)**<br>'1' = The DCD line has changed its state |
| 4 | **Complement of Clear To Send (comp_cts)**<br>Complement of CTS or equals to RTS in loop back mode |
| 5 | **Complement of Data Set Ready (comp_dsr)**<br>Complement of DSR or equals to DTR in loop back mode |

| | |
|---|---|
| 6 | **Complement of Ring Indicator (comp_ri)**<br>Complement of RI or equal to Out1 in loop back mode |
| 7 | **Complement of Data Carrier Detect (comp_dcd)**<br>Complement of DCD or equal to Out2 in loop back mode |

### 4.4.2.6 Interrupt Mask Register (INTMSK)

INTMSK is used to mask interrupt request according to the description below:

**Table 4.8:** Interrupt Mask Register Description

| Bit | Description |
|---|---|
| 0 | **RXRDY Interrupt Mask**<br>'0' = masked<br>'1' = enable |
| 1 | **TXRDY Interrupt Mask**<br>'0' = masked<br>'1' = enable |
| 7-2 | Reserved (always '0') |

### 4.4.2.7 Holding Register

This register holds data in "Receiver Holding Register (RHR)" or "Transmitter Holding Register (THR)".

## 4.5 Writing VHDL Code

The designed UART pins and registers description has been presented in the previous section. From the description, the reader should have a clearer view on how to describe the functionality of the designed UART in VHDL. In this section, the reader will be presented with the VHDL description of the designed UART. The designed UART consist of 6 VHDL modules:

- Transmit.vhd
- Received.vhd
- Baudgen.vhd
- Bilbo.vhd
- Host.vhd
- Uart.vhd

**Figure 4.4** shows the hierarchical tree of UART with BIST design. The summary of each module is displayed in **Table 4.9.**

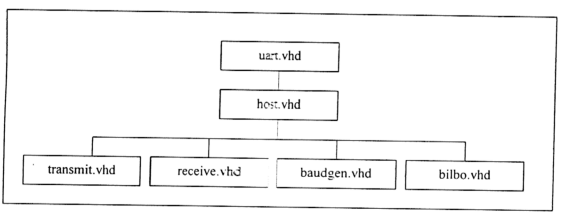


**Figure 4.4:** Hierarchical Tree of UART with BIST

**Table 4.9:** VHDL Modules

| VHDL files | Description |
|---|---|
| Transmit.vhd | **UART transmitter module**<br>- Converts parallel data to serial data.<br>- Consists of Transmitter Holding Register (THR) and Transmitter Shift Register (TSR).<br>- THR will take the parallel data and stores (the data) in its register when transmitter is enabled. TSR then loads the THR and shifts the data in serial mode.<br>- This module also generates parity and stop bit depends on the mode chosen. |
| Receive.vhd | **UART receiver module**<br>- Almost the same as transmitter's module but it catches serial data and produces parallel data from it.<br>- Receiver Shift Register (RSR) will capture the serial data. RSR will shift the data received and arranged it in parallel data format. The parallel data then will be transmitted to Receiver Holding Register (RHR).<br>- At the receiver, all kinds of error supported will be checked. A signal will be produced if an error is detected. |
| Baudgen.vhd | **Baud rate generator module**<br>- Use for specifying baud rate.<br>- This module shows how programmable baud rate generator divides any input clock by 1 to ($2^{15}$ - 1) and generates the 16X clock. |
| Bilbo.vhd | **Bilbo register**<br>- A built-in-self-test module.<br>- It can change its operating mode and acts as a shift register, PRPG, normal, or MISR. |
| Host.vhd | **Host interface control and registers**<br>- Specifies the bit position in each register so it can be easily accessed by data bus. |
| Uart.vhd | **Top-level design entity**<br>- Instantiated the other design entities. |

The suitability of VHDL to incorporate BIST features into UART design has been illustrated in this chapter. In addition, UART with BIST pins and registers descriptions also have been presented. The pins and registers description is the main criteria of writing a VHDL code that will be synthesized to obtain a Register Transfer

Level (RTL) description. This RTL will be verified in the following chapter. The timing simulation of the entire chip and the self-test process simulation results will be verified using simulation tools provided by "Xilinx Foudation Series 2i" package.