





# **DESIGN OF A HIGH SPEED UART**

By

**DEEPAK KUMAR-051002**

**LAXMIKANT UPADHYAY-051018**

**RITESH PANDEY-051100**



**MAY-2009**

**Submitted in partial fulfillment of the Degree of  
Bachelors of Technology**

**DEPARTMENT OF E.C.E.**

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY  
WAKNAGHAT**



## CERTIFICATE

This is to certify that the work entitled, "**Design of a High Speed UART**" submitted by Deepak Kumar (051002), Laxmikant Upadhyay(051018) and Ritesh Pandey (051100) in fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Supervisor:



Mr. Vipin Balyan

Department of ECE

Jaypee University of Information Technology

Waknaghat, Himachal Pradesh



Prof. Sunil V. Bhooshan

H.O.D. (ECE)

JUIT, Waknaghat


Himachal Pradesh

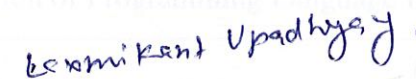
## ACKNOWLEDGEMENT


We would like to take the opportunity to express our deep sense of **acknowledgment** to our project guide Mr. Vipin Baliyan, Department of ECE, whose help, **stimulating suggestions** and encouragement helped us in all the stages of the project. His **overly enthusiasm and his view** for providing 'Only high quality work and not less has made a deep impression on us.

We are also thankful to Prof. Sunil Bhooshan and other staff members who co-operated with us in learning about the working of the software and guiding us throughout the semester.

Last, but not the least, we would like to thank everyone who has contributed for successful completion of our project.

  
Deepak Kumar

  
Laxmikant Upadhyay

  
Ritesh Pandey



## TABLE OF CONTENTS

• List of Figures.....	1
• List of Abbreviations.....	3
• Abstract.....	5
• Chapter 1: Introduction.....	6
➤ 1.1: Serial Transmission.....	6
➤ 1.2: The UART.....	7
• Chapter 2: Description of Hardware Used.....	12
➤ 2.1: Virtex 4.....	12
• Chapter 3: Description of Programming Language Used.....	14
➤ 3.1: HDL.....	14
➤ 3.2: VHDL.....	14
➤ 3.3: Programming Concepts.....	17
• Chapter 4: Tools Used.....	20
➤ 4.1: Xilinx ISE.....	20
➤ 4.2: ModelSim.....	21
• Chapter 5: Results and Simulations.....	23
➤ 5.1: Transmitter.....	23
➤ 5.2: Receiver.....	26

➤ 5.3: Baud-Rate Generator.....	29
➤ 5.4: Main UART.....	32
• Conclusion.....	45
• Bibliography.....	46
• Figure 1.1: Standard Data Serial Encoding.....	
• Figure 1.2: Communication b/w two Computers.....	
• Figure 1.3: UART Components.....	
• Figure 4.1: Xilinx ISE.....	21
• Figure 4.2: ModelSim.....	23
• Figure 5.1: SM Chart of Transmitter.....	24
• Figure 5.2: RTL Schematic of Transmitter.....	24
• Figure 5.3: Technology Schematic 1 of Transmitter.....	24
• Figure 5.4: Technology Schematic 2 of Transmitter.....	25
• Figure 5.5: Design Summary of Transmitter.....	25
• Figure 5.6: SM Chart of Receiver.....	27
• Figure 5.7: RTL Schematic of Receiver.....	28
• Figure 5.8: Technology Schematic of Receiver.....	28
• Figure 5.9: Design Summary of Receiver.....	29
• Figure 5.10: Baud-Rate Generator.....	30
• Figure 5.11: RTL Schematic of Baud-Rate Generator.....	30
• Figure 5.12: Technology Schematic of Baud-Rate Generator.....	31
• Figure 5.13: Design Summary of Baud-Rate Generator.....	31
• Figure 5.14: RTL Schematic of UART.....	32
• Figure 5.15: Technology Schematic 1 of UART.....	33
• Figure 5.16: Technology Schematic 2 of UART.....	33



## LIST OF FIGURES

• Figure 1.1: Standard Data Serial Encoding.....	7
• Figure 1.2: Communication b/w two Computers.....	8
• Figure 1.3: UART Components.....	9
• Figure 4.1: Xilinx ISE.....	21
• Figure 4.2: ModelSim.....	22
• Figure 5.1: SM chart of Transmitter.....	23
• Figure 5.2: RTL Schematic of Transmitter.....	24
• Figure 5.3: Technology Schematic 1 of Transmitter.....	24
• Figure 5.4: Technology Schematic 2 of Transmitter.....	25
• Figure 5.5: Design Summary of Transmitter.....	25
• Figure 5.6: SM Chart of Receiver.....	27
• Figure 5.7: RTL Schematic of Receiver.....	28
• Figure 5.8: Technology Schematic of Receiver.....	28
• Figure 5.9: Design Summary of Receiver.....	29
• Figure 5.10: Baud-Rate Generator.....	30
• Figure 5.11: RTL Schematic of Baud-Rate Generator.....	30
• Figure 5.12: Technology Schematic of Baud-Rate Generator.....	31
• Figure 5.13: Design Summary of Baud-Rate Generator.....	31
• Figure 5.14: RTL Schematic of UART.....	32
• Figure 5.15: Technology Schematic 1 of UART.....	33
• Figure 5.16: Technology Schematic 2 of UART.....	33

- Figure 5.17: Technology Schematic 3 of UART.....34
- Figure 5.18: Design Summary of UART.....34
- Figure 5.19.....43
- Figure 5.20.....44
- Figure 5.21.....44

- CPU.....Central Processing Unit
- DoD.....Department of Defense
- PE.....Parity Error
- FPGA.....Field Programmable Gate Array
- HDL.....Hardware Description Language
- IEEE.....Institute of Electrical and Electronics Engineers
- LSH.....Least Significant Bit
- OE.....Output Enable
- RDR.....Receive Data Register
- RDRI.....Receive Data Register Full
- RIE.....Receive Interrupt Enable
- RIF.....Receive Interrupt Flag
- SCCR.....Serial Communications Control Register
- SCSR.....Serial Communications Status Register
- SFR.....System File Register
- TDR.....Transmit Data Register
- TDRE.....Transmit Data Register Empty
- TIE.....Transmit Interrupt Enable



## LIST OF ABBREVIATIONS

- ANSI.....American National Standard Information
- ASCII.....American Standard Code for Information Interchange
- ASMBL.....Advanced Silicon Modular Block
- CPU.....Central Processing Units
- DoD.....Department of Defense
- FE.....Framing Error
- FPGA.....Field Programmable Gate Arrays
- HDL.....Hardware Description Language
- IEEE.....Institution of Electrical and Electronics Engineers
- LSB.....Least Significant Bit
- OE.....Overrun Error
- RDR.....Receive Data Register
- RDRF.....Receive Data Register Full
- RIE.....Receive Interrupt Enable
- RIE.....Receive Interrupt Enable
- SCCR.....Serial Communications Control Register
- SCSR.....Serial Communications Status Register
- SoC.....System on Chip
- TDR.....Transmit Data Register
- TDRE.....Transmit Data Register Empty
- TIE.....Transmit Interrupt Enable

- TSR.....Transmit Shift Register
- UART.....Universal Asynchronous Receiver Transmitter
- VHDL.....VHSIC Hardware Description Language
- VHSIC.....Very High Speed Integrated Circuits



## CHAPTER 1: INTRODUCTION

### ABSTRACT

The project focuses on the design of high speed 8 bit UART. The project starts with describing the soul element of serial data communication, "THE UART" using VHDL. While designing the code we have used Bottom to Top Strategy (we firstly did the coding of Transmitter, Receiver and Baud-Rate Generator and finally interconnected all three blocks). In our project, we have focused on reducing the use of unnecessary hardware components from UART by removing the unwanted signals (on which execution process does not depend ) from sensitivity list in our code and we have also avoided slower VHDL command (example- while, wait for t ns etc.) so that we can increase the speed of UART. The project gives the whole details (flowchart, code, RTL schematics, technology semantics, design summary, synthesis report and simulation waveform) about each block of the UART separately and also of the whole UART code combined. The synthesis report of the UART shows that the speed of UART is higher and hardware components (no. of flip-flops and logic cells) required for the implementation is lesser in comparison to latest updated 8 bit UART.

**1.1.2 Asynchronous Serial Transmission:** Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which is used to synchronize the sending and receiving units.

When a word is given to the UART for Asynchronous transmission, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. After the Start Bit, the individual bits of the word of data are sent, with the least Significant Bit (LSB) being sent first. Each bit in the transmission is represented by a voltage level. The receiver "looks" at the wire at approximately half the period of the clock used to clock the transmitter to determine if the bit is a 1 or a 0. The voltage level may be 1 or 0 and the receiver has "looked" at the value of the bit. The receiver only knows when the clock wire begins transmitting the next bit of

## CHAPTER 1: INTRODUCTION

### 1.1 SERIAL TRANSMISSION:

Serial Transmission of data means sending the data bit by bit i.e. one bit at a time. There are two basic forms of serial transmission. Synchronous and Asynchronous Serial Transmission.

**1.1.1 Synchronous Serial Transmission:** Synchronous serial transmission requires that the sender and receiver share a clock with one another, or that the sender provide a strobe or other timing signal so that the receiver knows when to “read” the next bit of the data. In most forms of serial Synchronous communication, if there is no data available at a given instant to transmit, a fill character must be sent instead so that data is always being transmitted. Synchronous communication is usually more efficient because only data bits are transmitted between sender and receiver, and synchronous communication can be more costly if extra wiring and circuits are required to share a clock signal between the sender and receiver.

**1.1.2 Asynchronous Serial Transmission:** Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which is used to synchronize the sending and receiving units.

When a word is given to the UART for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. After the Start Bit, the individual bits of the word of data are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits, and the receiver “looks” at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a 1 or a 0. The sender does not know when the receiver has “looked” at the value of the bit. The sender only knows when the clock says to begin transmitting the next bit of



the word. When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter. When the receiver has received all of the bits in the data word, it may check for the Parity Bits (both sender and receiver must agree on whether a Parity Bit is to be used), and then the receiver looks for a Stop Bit. If the Stop Bit does not appear when it is supposed to, the UART considers the entire word to be garbled and will report a Framing Error to the host processor when the data word is read. The usual cause of a Framing Error is that the sender and receiver clocks were not running at the same speed, or that the signal was interrupted. Regardless of whether the data was received correctly or not, the UART automatically discards the Start, Parity and Stop bits. If the sender and receiver are configured identically, these bits are not passed to the host.

If another word is ready for transmission, the Start Bit for the new word can be sent as soon as the Stop Bit for the previous word has been sent. Because asynchronous data is “self synchronizing”, if there is no data to transmit, the transmission line can be idle.

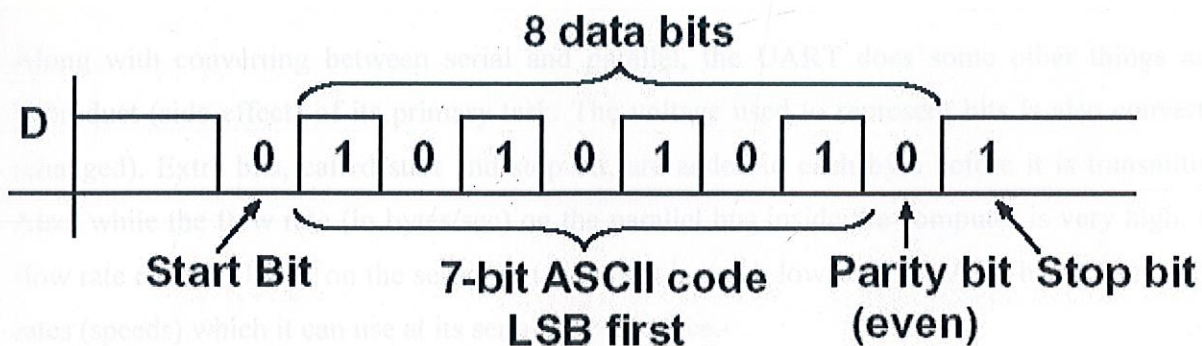


Figure 1.1: Standard Data serial encoding.

**1.2 THE UART:** The UART is a key component of the serial communication sub-system of a computer. It takes bytes of data and transmits the individual bits in a sequential way. At the destination end, a second UART re-assembles the individual bits into complete bytes.



The UART's purpose is to convert bytes from the PC's parallel bus to a serial bit-stream. The cable going out of the serial port is serial and has only one wire for each direction of flow. The serial port sends out a stream of bits, one bit at a time. Conversely, the bit stream that enters the serial port via the external cable is converted to parallel bytes that the computer can understand. UARTs deal with data in byte sized pieces, which is conveniently also the size of ASCII characters.

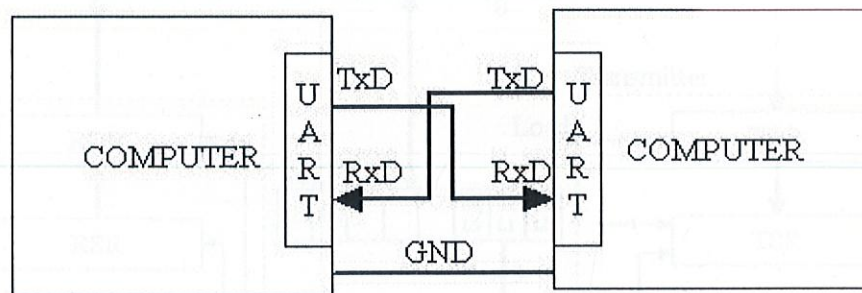


Figure 1.2: Communication between two computers.

Along with converting between serial and parallel, the UART does some other things as a byproduct (side effect) of its primary task. The voltage used to represent bits is also converted (changed). Extra bits, called start and stop bit, are added to each byte before it is transmitted. Also, while the flow rate (in bytes/sec) on the parallel bus inside the computer is very high, the flow rate out the UART on the serial port side of it is much lower. The UART has a fixed set of rates (speeds) which it can use at its serial port interface.

In addition to this, a UART will usually provide additional circuits for signals that can be used to indicate the state of the transmission media, and to regulate the flow of data in the event that the remote device is not prepared to accept more data. For example, when the device connected to the UART is a modem, the modem may report the presence of a carrier on the phone line while the computer may be able to instruct the modem to reset itself or to not take calls by raising or lowering one more of these extra signals.



### 1.2.1 UART COMPONENTS:

A UART is composed of four main components: the receiver, the transmitter, the baud rate generator and the UART registers. We suppose that the UART is connected to a microcontroller by a data bus and an address bus, to allow the CPU to read and write the register of the UART. Refer to figure 3 for the discussion of the following components.

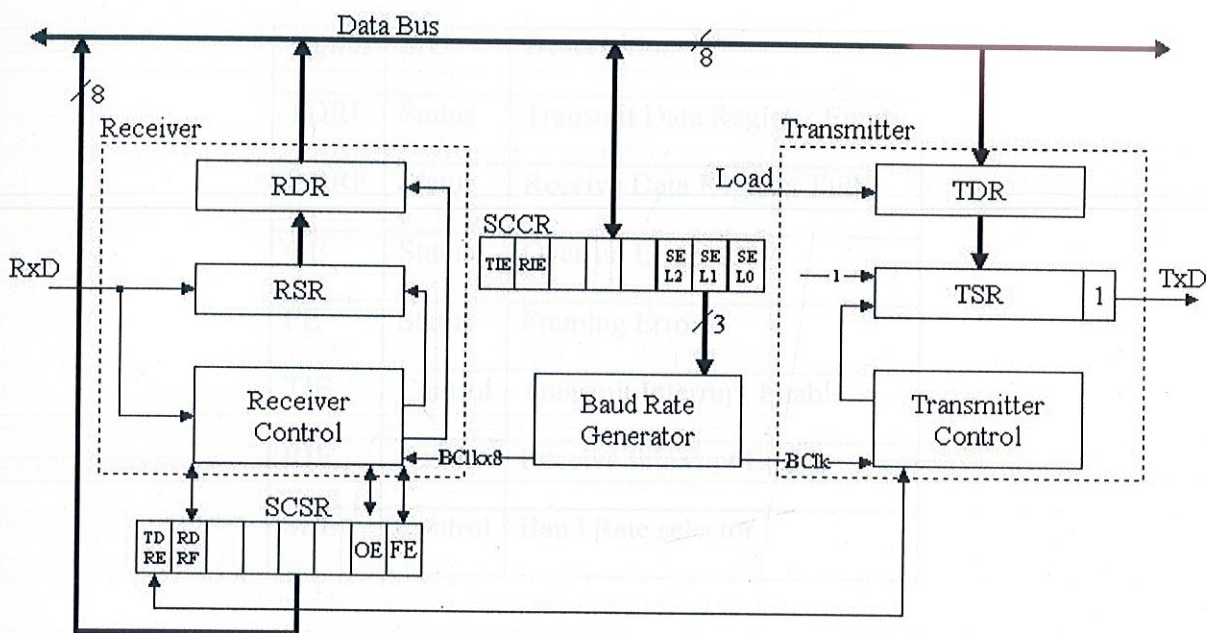


Figure 1.3: The UART Components.

#### 1.2.1.1 UART REGISTERS:

There usually exists four registers for each UART: The Receive Data Register (RDR), the Transmit Data Register (TDR), the Serial Communications Control Register (SCCR), and the Serial Communications Status Register (SCSR).

RDR is an 8-bit register which receives the data from an 8-bit shift register (RSR) and contains the serially received byte. TDR is also an 8-bit register which transmits the data from the application to an 8-bit shift register (TSR) and contains the byte that has to be serially transmitted. SCCR is an 8-bit register which contains the UART control signals: TIE, RIE and

SEL [2:0] and SCSR is an 8-bit register which contains the UART status signals: TDRE, RDRF, OE and FE.

The TSR (Transmit Shift Register) receives the output of the TDR, and the RSR (Receive Shift Register) provides the inputs to RDR. The status and control signals are defined as shown in table 1.

<b>Signal</b>	<b>S/C</b>	<b>Description</b>
TDRE	Status	Transmit Data Register Empty
RDRF	Status	Receive Data Register Full
OE	Status	Overrun Error
FE	Status	Framing Error
TIE	Control	Transmit Interrupt Enable
RIE	Control	Receive Interrupt Enable
SEL	Control	Baud Rate selector

Table 1: UART Status and Control Signals.

#### **1.2.1.2 TRANSMITTER:**

The UART transmitter consists of register TDR and TSR and the transmitter control. The status bit TDRE in the SCSR is asserted by the controller when TDR is empty.

#### **1.2.1.3 RECEIVER:**

The UART receiver consists of registers RDR and RSR and the receiver controller. The status bit RDRF in the SCSR is asserted by the controller when RDR is full.



#### 1.2.1.4 BAUD-RATE GENERATOR:

The baud rate generator is programmable by utilizing the three control bits (SEL[2 : 0]) in SCCR. Since we are using three bits, we have the choice of 8 baud rates. The Baud-Rate corresponding to each select input are given in Table 2.

SEL[2:0]	Baud-Rate ( <i>Bclk</i> )
000	38462
001	19231
010	9615
011	4808
100	2404
101	1202
110	601
111	300.5

Table 2: Frequencies of *Bclk*

#### 1.2.2 APPLICATIONS OF UART:

The first UART was designed by Gordon Bell. The first single chip UART was designed in 1971 by a company called Western Digital. Since then, the UART has been modified and redesigned to suit the need of the hour. UARTs can be found in most of the microprocessor based devices, be it a computer or PDA or mobile phones. Various applications of the UART are:

- 1.) The basic job of UART is to convert data from parallel to serial for transmission and from serial to parallel for reception.
- 2.) The UART usually provides additional circuits for signals that can be used to indicate the state of the transmission media.
- 3.) The UART also provides circuits to regulate the flow of data in the event that the remote device is not prepared to accept more data.
- 4.) It adds a parity bit (if it's been selected) on outbound transmissions and checks the parity of incoming bytes (if selected) and discards the parity bit.
- 5.) It adds start and stop delineators on outbound and strips them from inbound transmissions.
- 6.) It handles interrupts from the keyboard and mouse.



## CHAPTER 2: DESCRIPTION OF HARDWARE USED

The hardware used during our project was Virtex 4. The Virtex series of FPGAs have integrated features such as wired and wireless infrastructure equipment, advanced medical equipment, test and measurement, and defense systems. In addition to FPGA logic, the Virtex series includes embedded fixed function hardware for commonly used functions such as multipliers, memories, serial transceivers and microprocessor cores.

**2.1 VIRTEX 4:** The Virtex-4 series was introduced in 2004 and was manufactured on a 1.2V, 90-nm, triple-oxide process technology. The Virtex-4 family introduced the new Advanced Silicon Modular Block (ASMBL) architecture enabling FPGA platforms with a combination of features to support logic (LX), embedded processing and connectivity (FX), digital signal processing (SX). As part of the family of RealView Development Boards, Logic Tiles enable system-on-chip (SoC) developers to prototype complete systems, prove custom IP, develop and test device drivers for custom IP. Logic Tiles may be stacked to provide additional capacity. High performance and high pin-count interconnect allows large design prototyping.

**2.1.1 MEMORY SYSTEM:** Because of the limited number of I/O of Virtex-4 FPGAs, the Logic Tile does not contain on-chip RAM. The FPGA configuration image is stored in a Flash device. The configuration Flash can store up to 2 FPGA images.

### **2.1.2 I/O:**

- 1.) Logic Tile Header connectors on top & bottom of each tile.
- 2.) 395 interconnect pins to tile above.
- 3.) 395 interconnect pins to tile below.
- 4.) 128 interconnect pins common to tiles above and below.
- 5.) Option to fold over some signals to increase I/O to tile below.
- 6.) 8 DIP switches and 8 general purpose LEDs.
- 7.) Pushbutton.



### **2.1.3 SCALABILITY:**

- 1.) Logic Tiles may be stacked.
- 2.) Logic Tiles for Virtex-4 devices are mechanically and electrically compatible with Integrator Interface Modules (IM-LT1 and IM-LT3). However, example FPGA images are only provided for the Emulation Baseboard and Platform Baseboard configuration.
- 3.) Logic Tiles interface directly to the Emulation Baseboard and the Platform Baseboard for ARM926EJ-S, without an adapter.

### **2.1.4 FPGA DESIGN SOFTWARE AND PROGRAMMING TOOLS:**

- 1.) Xilinx Virtex-4 compatible FPGA synthesis tools may be used with Logic Tiles.
- 2.) The Logic Tile configuration Flash can be programmed with Multi-ICE, RealView ICE with the USB debugger on the Platform Baseboard for ARM926EJ-S and the Emulation Baseboard.

## CHAPTER 3: DESCRIPTION OF PROGRAMMING LANGUAGE USED

The programming language that we have used for the successful completion of our project is a HDL ( Hardware Description Language) called VHDL.

**3.1 HARDWARE DESCRIPTION LANGUAGE (HDL):** A **Hardware Description Language** or **HDL** is any language from a class of computer languages and/or programming languages of electronic circuits. It can describe the circuit's operation, its design an organization, and tests to verify its operation by means of simulation.

HDLs are used to write executable specifications of some piece of hardware. A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this executability that gives HDLs the illusion of being programming languages.

**3.2 VHDL:** VHDL is acronym for VHSIC Hardware Description Language. It is a hardware description language that can be used to model a digital system at many levels of abstraction, ranging from the algorithm level to the gate level. The complexity of the digital system being modeled could from that of a simple gate to a digital electronic system or in between. The digital system can also be described hierarchically. Timing can also be explicitly modeled in the same description. The VHDL language can be regarded as an integrated amalgamation of the following language:

- Sequential language
- Concurrent language
- Net -list language
- Timing specifications
- Waveform generation language



Therefore, the language has constructs that enable one to express the concurrent or sequential behavior of a digital system with or without timings. It also allows one to model the system as an interaction of components. Test waveforms can also be generated by using the same construction. All the above constructs may be combined to provide a comprehensive description of the system in single model.

The language not only defines the syntax but also defines a very clear simulation semantics for language can verify using VHDL simulator. It is strongly typed language and is often verbose to write. It inherits many of its features. Especially the sequential language part, from the Ada Programming language.

**3.2.1 HISTORY:** The requirements for the language were first generated in 1981 under the VHSIC program. In this program, a number of US companies were involved in designing VHSIC chips for the Department of Defense (DoD). At that time, most of the companies were using different hardware description languages to describe and develop their integrated circuits. As a result, different vendors could not effectively design with one another. Also different vendors provided DoD with description of their chips in different HDL. Reprocurment and re-use was also a big issue. Thus a need for standardized HDL for the design, documentation and verification of digital system was generated.

A team of three companies, IBM, Texas instruments and Intermetrics were first awarded the contract by DoD to develop a version of language in 1983. Version 7.2 of VHDL was developed and released to the public in 1985. There was strong industry participation throughout the VHDL language development process, especially from the companies that needed to make the language an industry – wide standard. Consequently the language was transferred to IEEE for standardization in 1986. After a substantial enhancement to the language made by a team of industry, university and DoD representatives, the language was standardized by the IEEE in Dec 1987; this version of the language come to be known as the IEEE Std 1076-1987. The official description appears in the IEEE Std. VHDL language Reference Manual (LRM) available from IEEE. The language has also being recognized as an America National Standards institute (ANSI) standard.



**3.2.2 CAPABILITIES:** The following are major capabilities that are provided along with the features that differentiate it from other hardware description languages.

- The language can be used as an exchange medium between chip vendor and CAD tools users. Different chip vendors can provide tool users can use it to capture the behavior of the design at a high level of the abstraction for the functional simulation.
- The language can also be used as a communication medium between CAD and CAE tools. For example, a schematic capture program may be used to generate a VHDL description for the design, which can be used as an input to a simulation program.
- The language supports the hierarchy, that is, in a digital system can be modeled as a set of interconnected components; each component, in turn, modeled as a set of interconnected sub-components.
- The language supports, flexible design methodologies: top-down, bottom-up or mixed.
- The language is not technology specific, but is capable of supporting technology specific features. It can also support the various hardware technologies; for example, you may define new logic types and new components; you may also specific attributes. By being technology independent the same model can be synthesized into different vendor libraries.
- It can support both synchronous and asynchronous timing models.
- Various digital modeling techniques such as finite-state machine description, algorithm description, and the Boolean equation can be modeled using the language.
- The language is publicly available, human readable, machine readable, and above all, it is not proprietary.
- It is an IEEE and ANSI standard; therefore models described using these languages are portable. The government also has a strong interest in maintaining this as a standard so that re-procurement and second sourcing may become easier.
- The language supports three basic different description styles: structural, dataflow, and behavioral. A design may also be expressed in any combination of these three descriptive styles.



**3.3 PROGRAMMING CONCEPTS:** The purpose of VHDL description is to provide a model for digital circuits and systems. This abstract view of the real physical circuit is referred to as an entity. An entity normally consists of five basic elements or design units:

1. Entity declaration
2. Architecture body
3. Configuration declaration
4. Package declaration
5. Package body

**3.3.1 ENTITY DECLARATION:** An entity declaration specifies the name of the entity and its interface. Signals, which are used for communication with the surrounding modules, are called ports. In general, the entity declaration has the following format:

**SYNTAX:**

**entity** [ entity\_name ] **is**

**port**([declaration(types, constants, signals)])

**end** [entity name];

**Ports** are of four modes:

**Mode in:** The port can be read within the entity and its architecture.

**Mode out:** This port can only be written.

**Mode inout:** This port can be read or written. This is useful for modeling bus system.

**3.3.2 ARCHITECTURE BODY:** Following the entity declaration the second important component of a VHDL description is the Architecture. This is where the functionality and the internal implementation of the module is described.

### SYNTAX:

**architecture** [architecture\_name] **of** [entity\_name] **is**

[arch\_declarative\_part]

**begin**

[arch\_statement\_part]

**end** [architecture\_name];

The architecture specifies the implementation of the entity. A label architecture\_name must be assigned to the architecture. In case there are multiple architecture associated with one entity this label is then used within a configuration statement to bind one particular architecture to its entity. The architecture block consists of two parts: the arch\_declarative\_part before the begin and subprogram is defined. The actual model description is done in the statement part. The statements in the arch\_statement\_part are executed concurrently, or in parallel however, during the simulation of VHDL description, all concurrent statement are executed on a processor, which processes all the statements sequentially. Therefore, a special simulation algorithm is used to achieve a virtual concurrent processing.

**3.3.3 CONFIGURATION DECLARATION:** A configuration declaration is used to select one of the possibly many architecture bodies that an entity may have, and to bind components, used to represent structure in that architecture body, to entities represented by an entity-architecture pair or by a configuration, which reside in a design library. There are no behavioral or simulation semantics associated with a configuration declaration. It merely specifies a binding that is used to build a configuration for an entity. These bindings are performed during the elaboration phase of simulation when the entire design to be simulated is being assembled. Having a configuration for the entity, the configuration can then be simulated.



**3.3.4 PACKAGE DECLARATION:** A package declaration is used to store a set of common declarations, such as components, types, procedures, and functions. These declarations can then be imported into other design units using a use clause.

**3.3.5 PACKAGE BODY:** A package body is used to store the definitions of functions and procedures that were declared in the corresponding package declaration, and also the complete constant declarations for any deferred constants that appear in the package declaration. Therefore, a package body is always associated with a package declaration. Furthermore, a package declaration can have at most one package body associated with it.

## CHAPTER 4: TOOLS USED

To check our VHDL codes, we have used Xilinx and Modelsim to generate the respective waveforms and to check for various syntax errors.

**4.1 XILINX ISE:** Xilinx is the world's largest supplier of programmable logic devices, the inventor of the field programmable gate array (FPGA) and the first semiconductor company with a fabless manufacturing model.

Founded in Silicon Valley in 1984 and headquartered in San Jose, California, U.S.A.; Dublin, Ireland; Singapore; and Tokyo, Japan, the company has corporate offices throughout North America, Asia and Europe.

The programmable logic device market has been led by Xilinx since the late 1990s. Over the years, Xilinx has fueled an aggressive expansion to India, Asia and Europe – regions Xilinx representatives have described as high-growth areas for the business.

Xilinx has two main FPGA families: the high-performance Virtex series and the high-volume Spartan series, with a cheaper Easy Path option for ramping to volume production. It also manufactures two CPLD lines, the Cool Runner and the 9500 series. Each model series has been released in multiple generations since its launch.

The latest Virtex-6 and Spartan-6 FPGA families are said to consume 50 percent less power, cost 20 percent less, and have up to twice the logic capacity of previous generations of FPGAs.

The following is a screen shot taken of the software we have used.



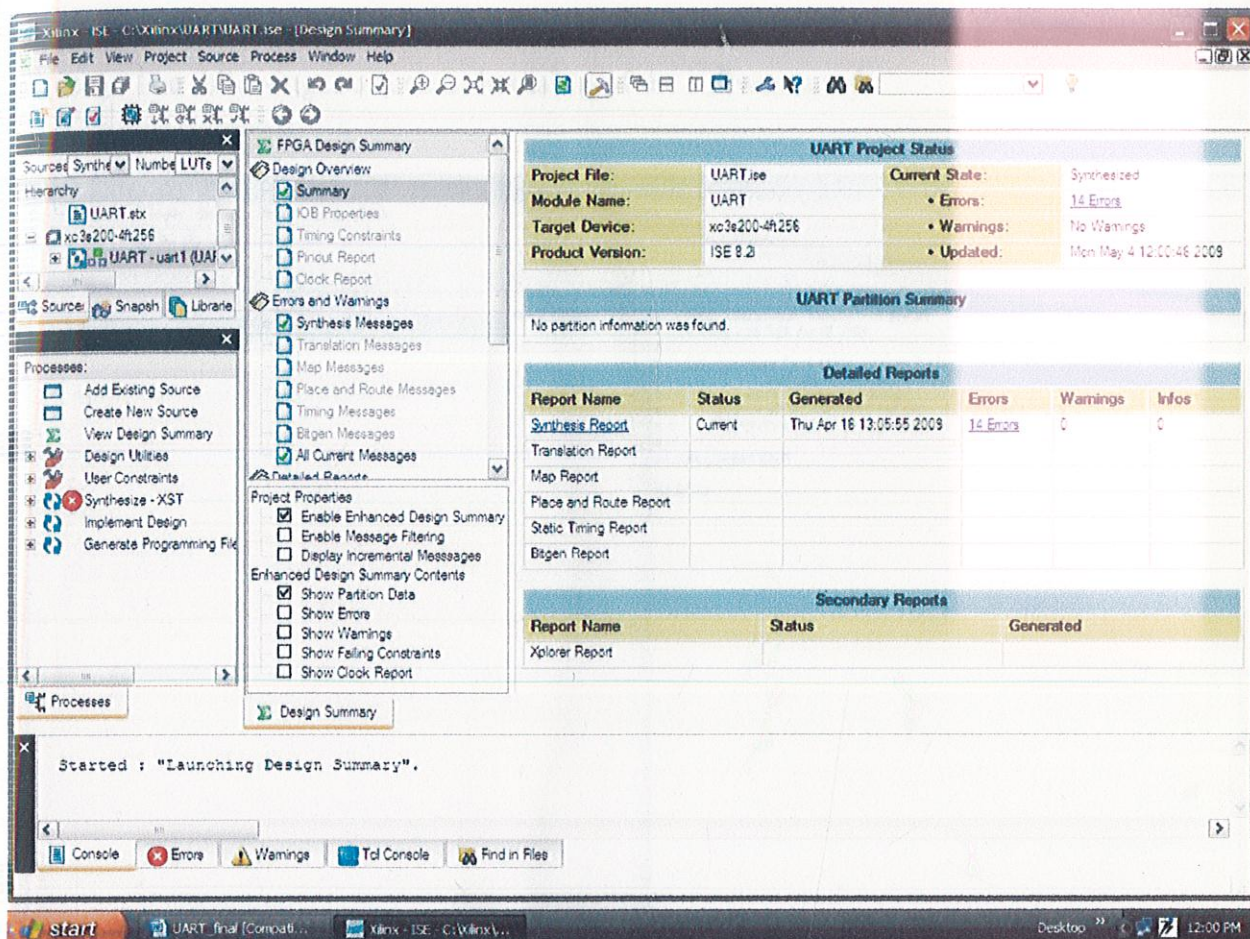


Figure 4.1: Xilinx ISE

**4.2 MODELSIM:** ModelSim is an industry standard simulator used in many semiconductor companies worldwide for their HDL simulation needs. It provides complete support for all Verilog features, in contrast to the limitations imposed by the MAX+PLUS II simulator. The advanced features of ModelSim are so well integrated into the simulator that you will not be overwhelmed while using the software for your basic simulation needs. These features include advanced debugging techniques, integrated Tcl/Tk scripting etc., which could turn out to be very useful in case you decide on doing your senior design projects in this area. While MAX+PLUS II excels as an integrated simulationsynthesis tool for basic learning purposes, it is no match for hands on experience on an industry standard HDL simulator when the purpose of the lab sessions



is to get you familiar with HDL simulations and prepare you for a possible future career in the industry. The following is a screen shot taken of the software used.

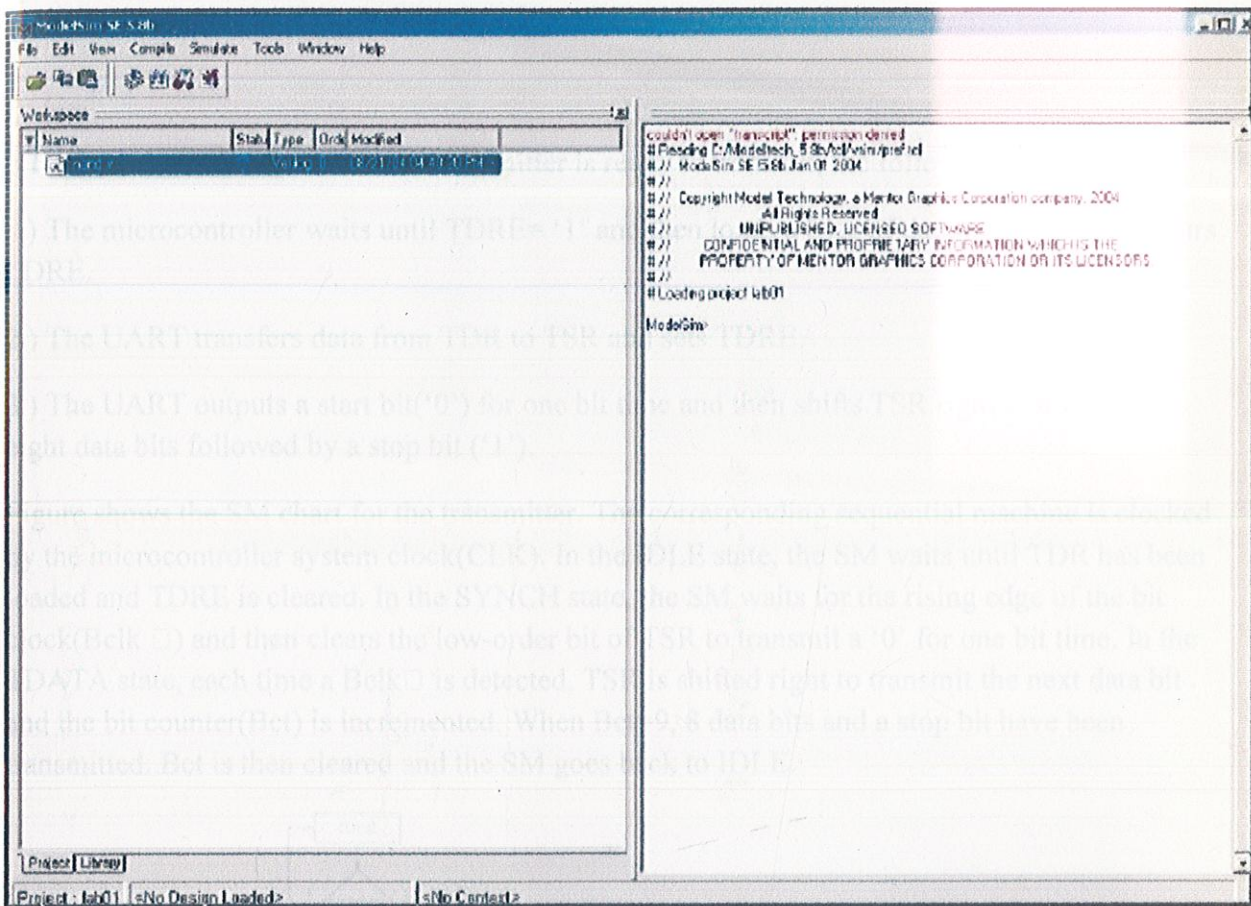


Figure 4.2: ModelSim



## CHAPTER 5: RESULTS AND SIMULATIONS

**5.1 TRANSMITTER:** When the transmitter is ready to transmit, the following occurs:

- 1.) The microcontroller waits until TDRE= '1' and then loads a byte of data into TDR and clears TDRE.
- 2.) The UART transfers data from TDR to TSR and sets TDRE.
- 3.) The UART outputs a start bit('0') for one bit time and then shifts TSR right to transmit the eight data bits followed by a stop bit ('1').

Figure shows the SM chart for the transmitter. The corresponding sequential machine is clocked by the microcontroller system clock(CLK). In the IDLE state, the SM waits until TDR has been loaded and TDRE is cleared. In the SYNCH state, the SM waits for the rising edge of the bit clock(Bclk □) and then clears the low-order bit of TSR to transmit a '0' for one bit time. In the TDATA state, each time a Bclk□ is detected, TSR is shifted right to transmit the next data bit and the bit counter(Bct) is incremented. When Bct=9, 8 data bits and a stop bit have been transmitted. Bct is then cleared and the SM goes back to IDLE.

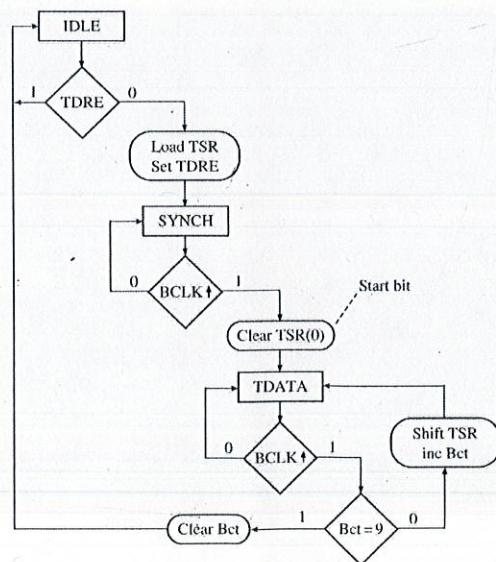


Figure 5.1: SM chart of Transmitter



### 5.1.1 RTL SCHEMATIC:

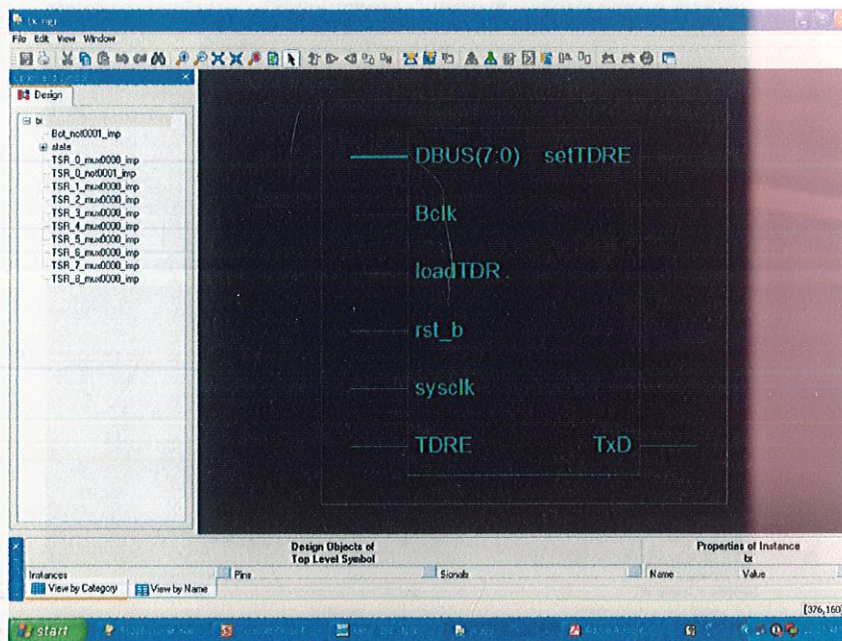


Figure 5.2: RTL schematic of transmitter

### 5.1.2 TECHONOLGY SCHEMATIC:

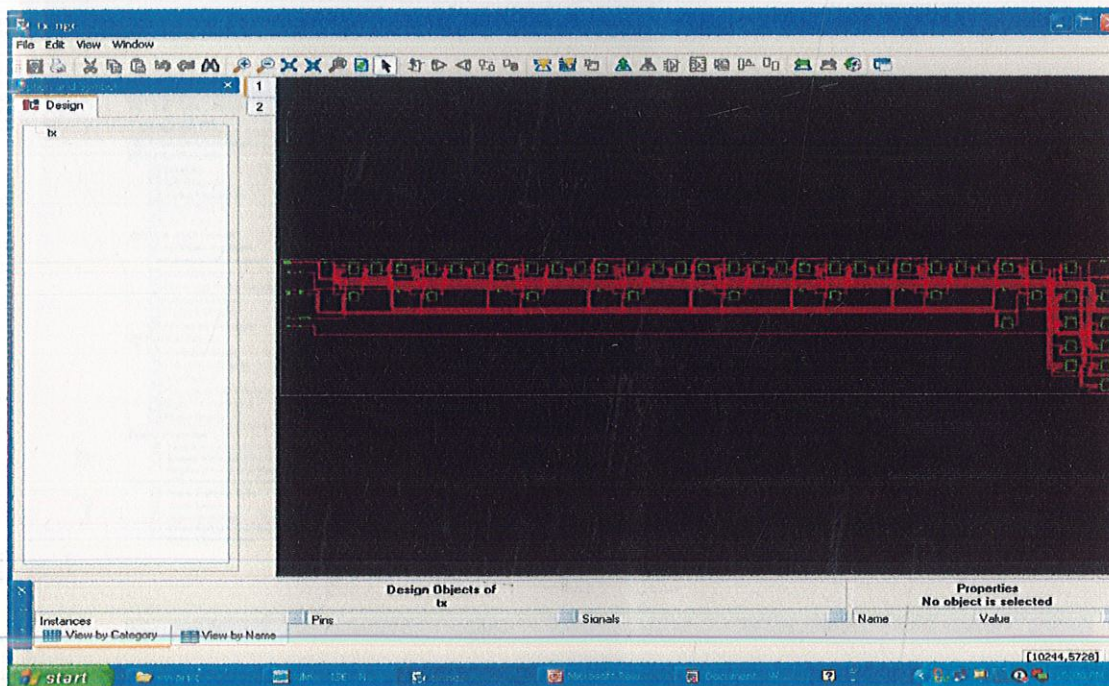


Figure 5.3: Technology Schematic 1 of Transmitter



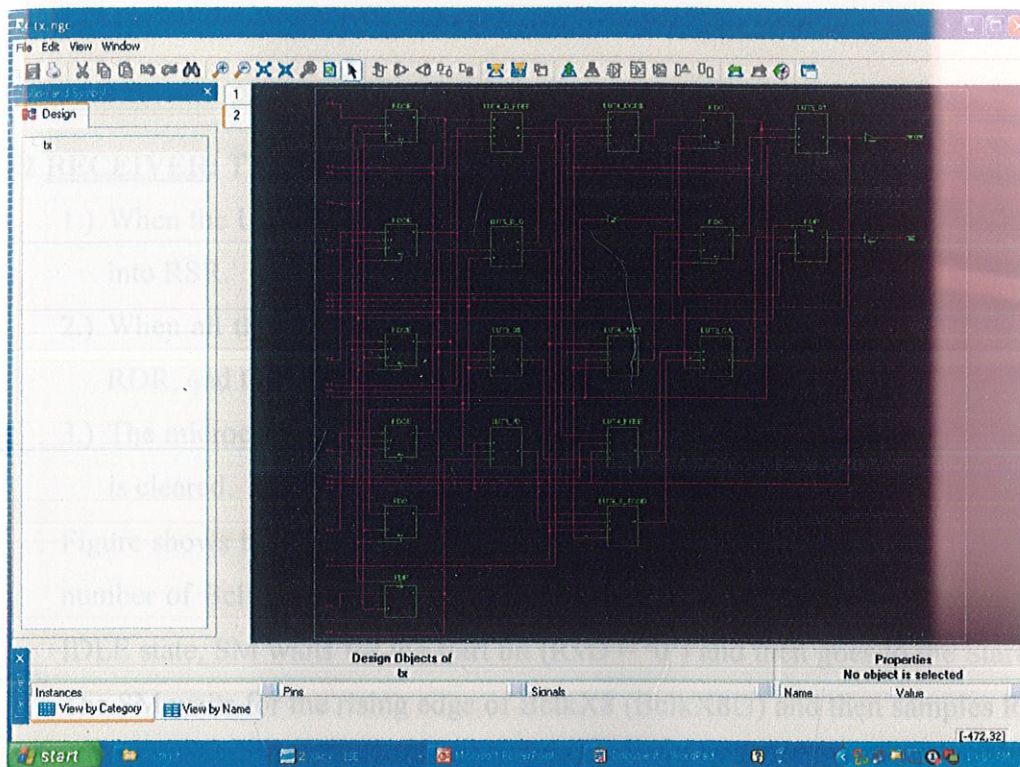


Figure 5.4: Technology Schematic 2 of Transmitter

### 5.1.3 DESIGN SUMMARY:

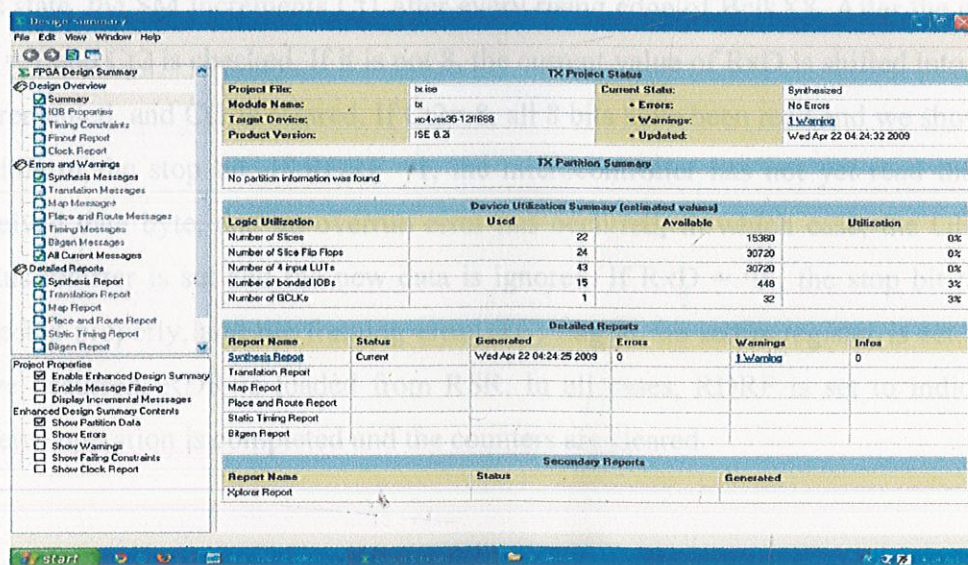


Figure 5.5: Design Summary of Transmitter



**5.2 RECEIVER:** The operation of the UART receiver is as follows:

- 1.) When the UART detects a start bit, it reads in the remaining bits serially and shifts them into RSR.
- 2.) When all the data bits and the stop bit have been received, the RSR is loaded into the RDR, and the RDRF flag in the SCSR is set.
- 3.) The microcontroller checks the RDRF flag, and if it is set, the RDR is read and the flag is cleared.

Figure shows the SM chart for the UART receiver. Two counters are used. Ct1 counts the number of BclkX8 clocks. Ct2 counts the number of bits received after the start bit. In the IDLE state, SM waits for the start bit ( $RxD = '0'$ ) and then goes to the Start Detected State. The SM waits for the rising edge of BclkX8 ( $BclkX8 \uparrow$ ) and then samples RxD again. Since the start bit should be '0' for eight BclkX8 clocks, we should read '0'. Ct1 is still 0, so Ct1 is incremented and the SM waits for BclkX8  $\uparrow$ . If  $RxD = '1'$ , this is an error condition and the SM clears Ct1 and resets to the IDLE state. Otherwise, the SM keeps looping. When RxD is 0 for the fourth time, Ct1=3, so Ct1 is cleared and the state goes to Receive Data. In this state, the SM increments Ct1 after every rising edge of BclkX8. After the eighth clock, Ct1= 7 and Ct2 is checked. If it is not 8, the current value of RxD is shifted into RSR, Ct2 is incremented, and Ct1 is cleared. If Ct2= 8, all 8 bits have been read and we should be in the middle of the stop bit. If RDRF =1, the microcontroller has not yet read the previously received data byte, and an overrun error has occurred, in which case, the OE flag in the status register is set and the new data is ignored. If  $RxD = '0'$ , the stop bit has not been detected properly, and the framing error (FE) flag in the status register is set. If no errors have occurred, RDR is loaded from RSR. In all cases, RDRF is set to indicate that the receive operation is completed and the counters are cleared.



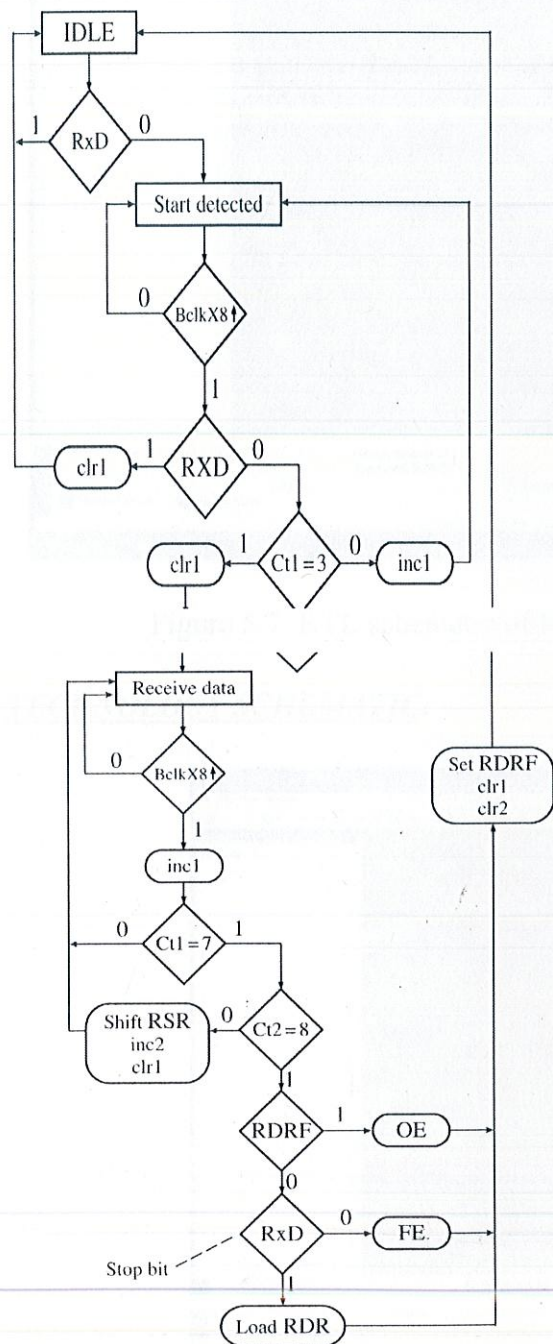


Figure 5.6: SM chart of Receiver



### 5.2.1 RTL SCHEMATIC:

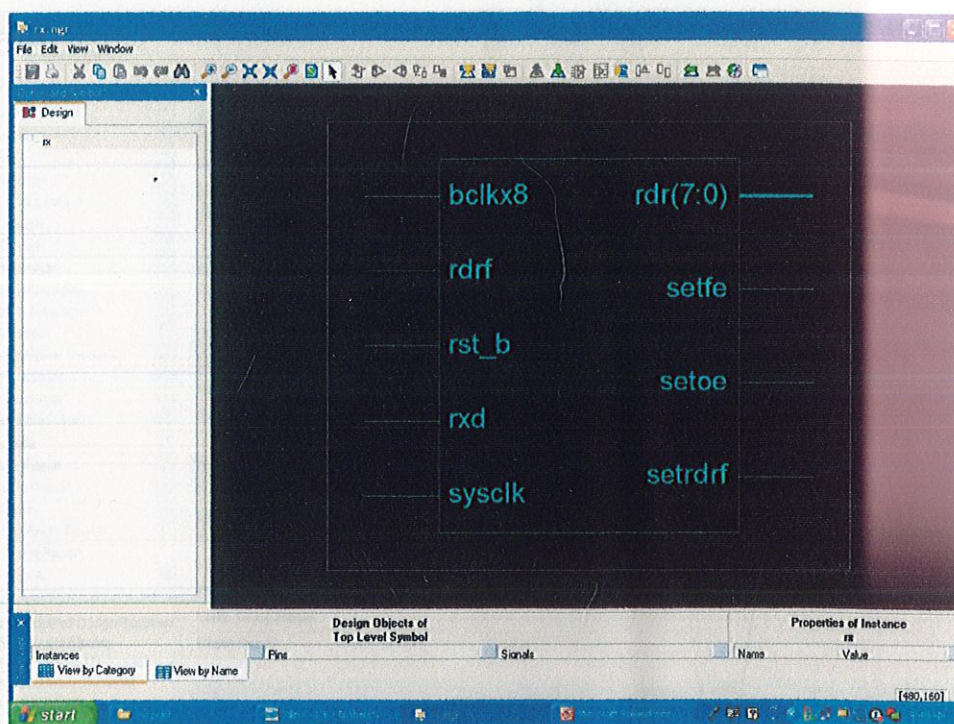


Figure 5.7: RTL schematic of Receiver

### 5.2.2 TECHNOLOGY SCHEMATIC:

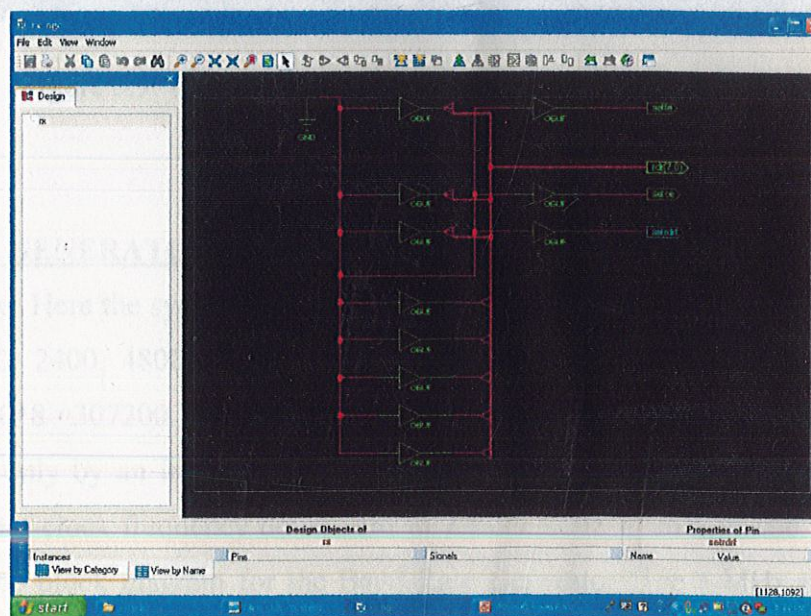


Figure 5.8: Technology schematic of Receiver



### 5.2.3 DESIGN SUMMARY:

**UART\_RECEIVER Project Status**

Project File:	uart_receiver.isc	Current State:	Synthesized
Module Name:	uart_receiver	• Errors:	No Errors
Target Device:	xc4vsx35-12H668	• Warnings:	13 Warnings
Product Version:	ISE 8.2i	• Updated:	Wed Apr 22 03:16:18 2009

**UART\_RECEIVER Partition Summary**

No partition information was found.

**Device Utilization Summary (estimated values)**

Logic Utilization	Used	Available	Utilization
Number of Slices	0	15360	0%
Number of bonded I/Os	11	448	2%

**Detailed Reports**

Report Name	Status	Generated	Errors	Warnings	Infos
<a href="#">Synthesis Report</a>	Current	Wed Apr 22 03:16:08 2009	0	13 Warnings	10 Infos
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Bigen Report					

**Secondary Reports**

Report Name	Status	Generated
Xplorer Report		

**Project Properties**

- ☒ Enable Enhanced Design Summary
- ☐ Enable Message Filtering
- ☐ Display Incremental Messages

**Enhanced Design Summary Contents**

- ☒ Show Partition Data
- ☐ Show Errors
- ☐ Show Warnings
- ☐ Show Failing Constraints
- ☐ Show Clock Report

Figure 5.9: Design Summary of Receiver

**5.3 BAUD-RATE GENERATOR:** Three bits in the SCCR are used to select any one of the eight Baud rates. Here the system clock frequency is taken as 8MHz and we want Baud rates 300, 600, 1200, 2400, 4800, 9600, 19200 and 38400. The maximum BclkX8 frequency needed is  $38400 \times 8 = 307200$ . To get this frequency, we should divide 8 MHz by 26.04. Since we can divide only by an integer, we need to either accept a small error in Baud rate or adjust the system clock frequency downward to 7.9877 MHz to compensate. The following figure shows the block diagram for the Baud-Rate Generator. The 8-MHz system clock is first divided by 13 using a counter. This counter output goes to an 8-bit binary counter. The



outputs of the flip-flops in this counter correspond to divide by 2, divide by 4, ... and divide by 256. One of these outputs is selected by a multiplexer. The MUX select inputs come from the lower 3 bits of the SCCR. The MUX output corresponds to BlkX8, which is further divided by 8 to give Bclk. The frequencies generated are given in Table 2.

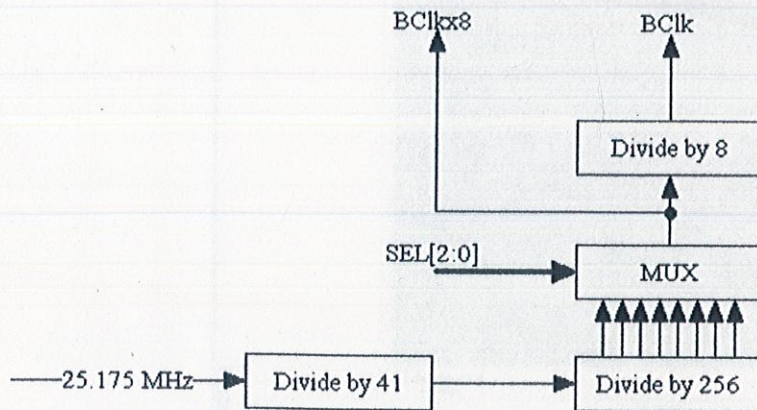


Figure 5.10: Baud-Rate Generator

### 5.3.1 RTL SCHEMATIC:

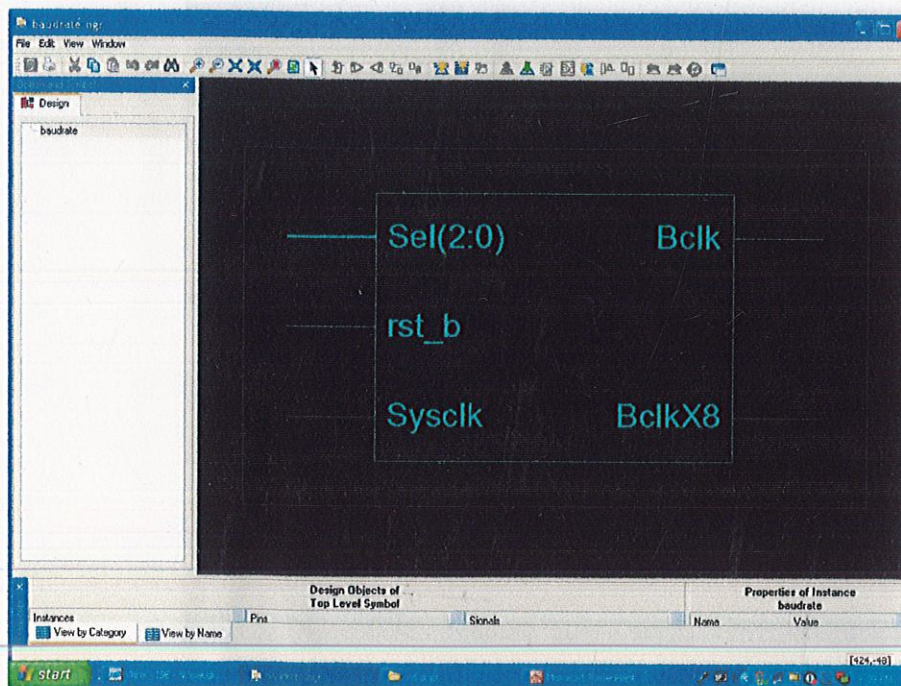


Figure 5.11: RTL Schematic of Baud-rate generator



### 5.3.2 TECHNOLOGY SCHEMATIC:

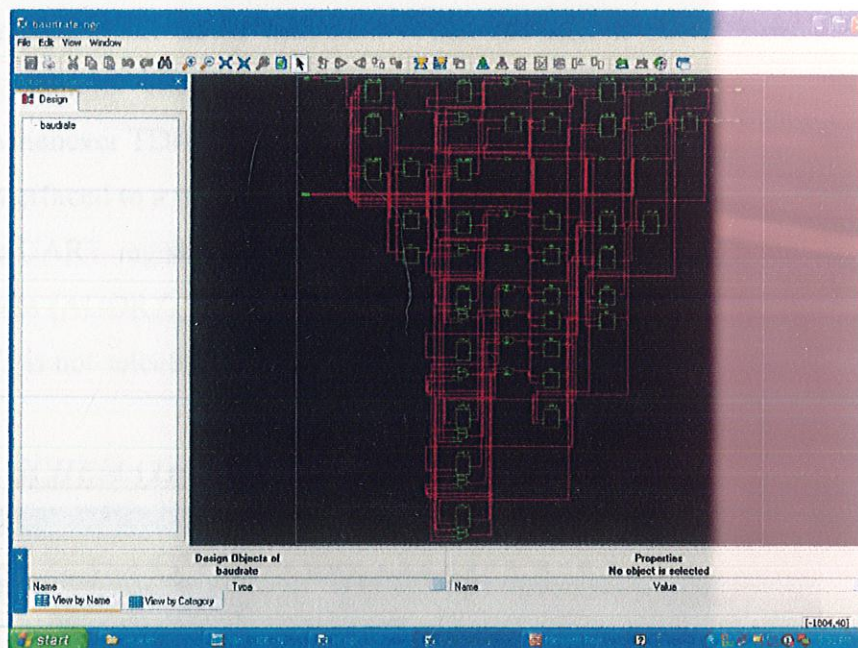


Figure 5.12: Technology Schematic of Baud-Rate Generator

### 5.3.3 DESIGN SUMMARY:

CLK_DEVIDER Project Status			
Project File:	clk_devider.tce	Current State:	Synthesized
Module Name:	clk_devider	• Errors:	No Errors
Target Device:	xo4vxc05-128568	• Warnings:	1 Warning
Product Version:	ISE 8.2i	• Updated:	Wed Apr 22 04:15:42 2009

CLK_DEVIDER Partition Summary			
No partition information was found.			

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	10	15300	0%
Number of Slice Flip Flops	15	30720	0%
Number of 4 input LUTs	10	30720	0%
Number of bonded IOBs	6	448	1%
Number of BCLKs	1	32	3%

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Wed Apr 22 18:37:13 2009	0	1 Warning	2 Infos
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Brgon Report					

Secondary Reports		
Report Name	Status	Generated
Module Report		

Figure 5.13: Design Summary of Baud rate generator



**5.4 MAIN UART:** SCI\_IRQ is an interrupt signal that interrupts the CPU when the UART receiver or transmitter needs attention. When the RIE is set in SCCR, SCI\_IRQ is generated whenever RDRE or OE is '1'. When TIE is set in SCCR, SCI\_IRQ is generated whenever TDRE is '1'.

The UART is interfaced to a microcontroller address and data buses so that the CPU can read and write to the UART registers when the UART is selected by SCIsel = '1'. The last two bits of the address (ADDR2), together with R\_W signal, are used for the register selection. When the UART is not selected for reading, the data bus is driven to high-Z.

#### 5.4.1 RTL SCHEMATIC:

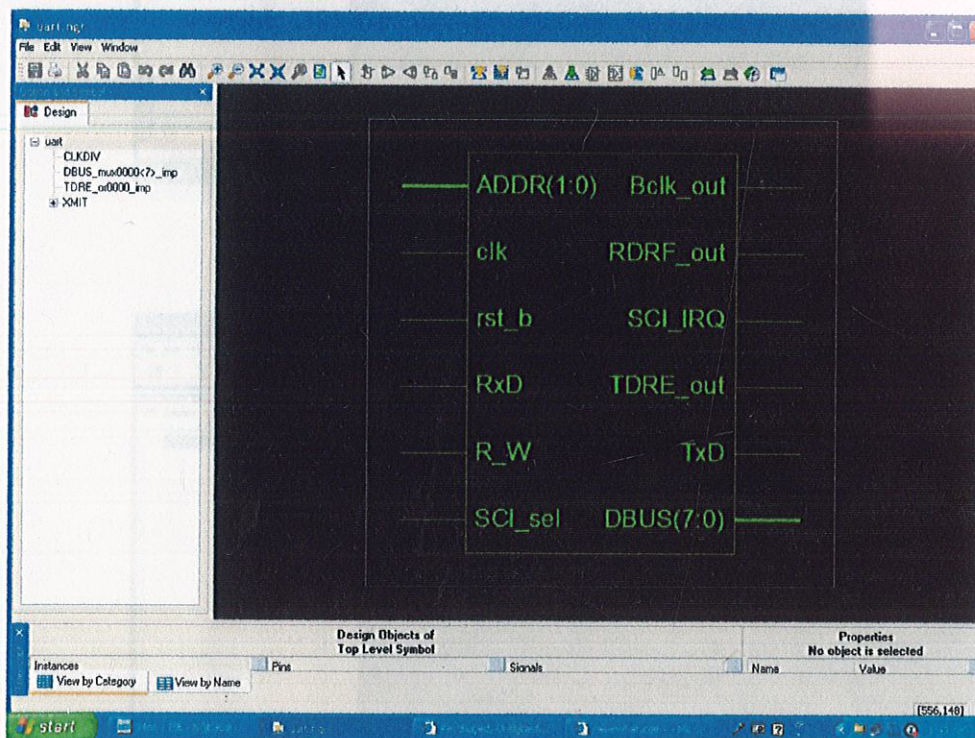


Figure 5.14: RTL Schematic of UART



### 5.4.2 TECHNOLOGY SCHEMATIC:

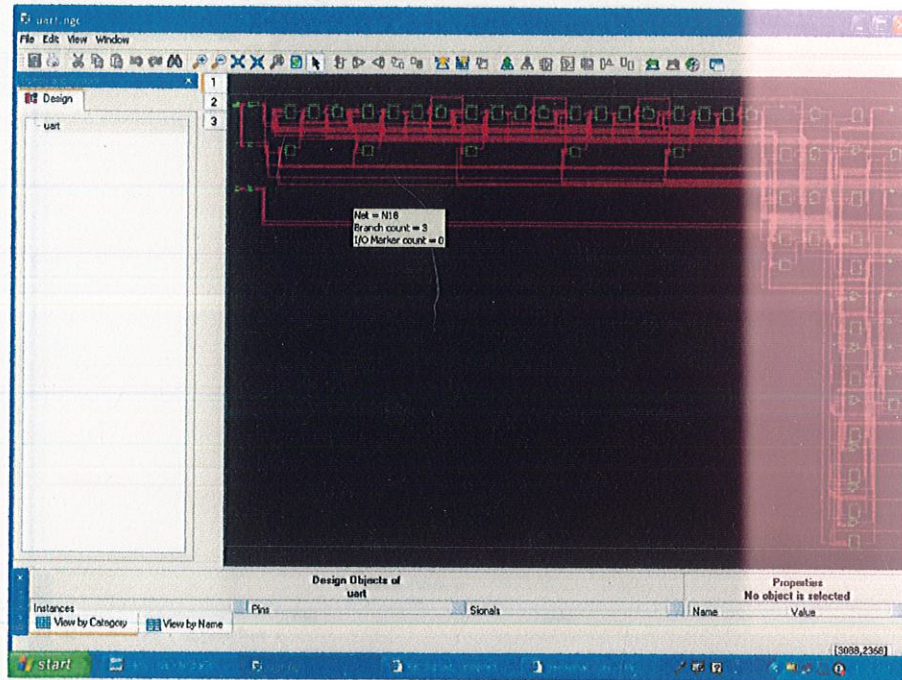


Figure 5.15: Technology Schematic 1 of UART

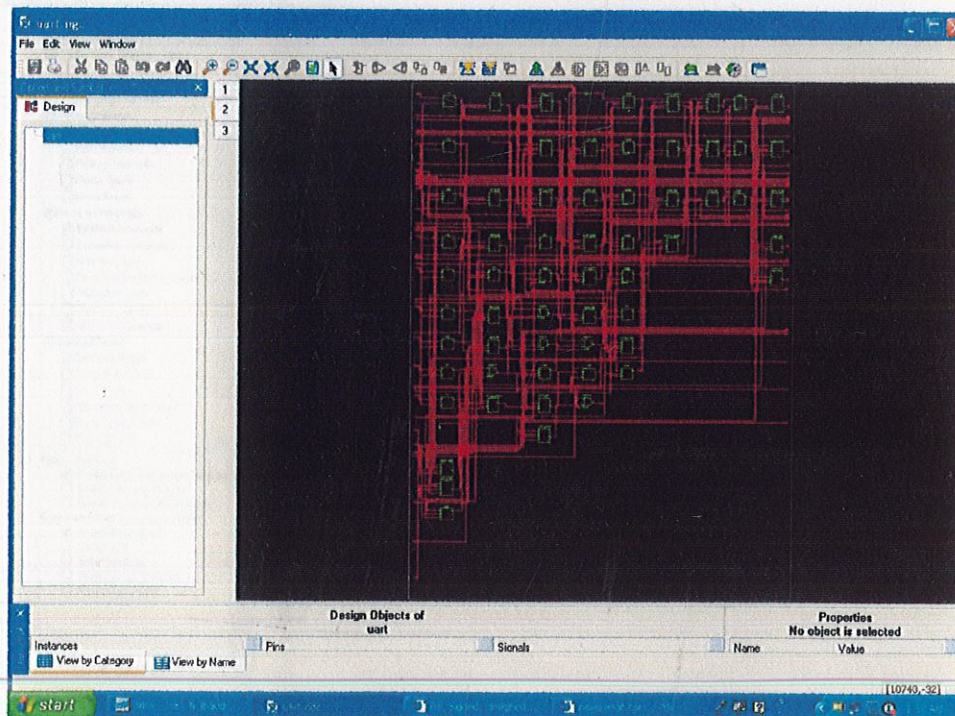


Figure 5.16: Technology Schematic 2 of UART



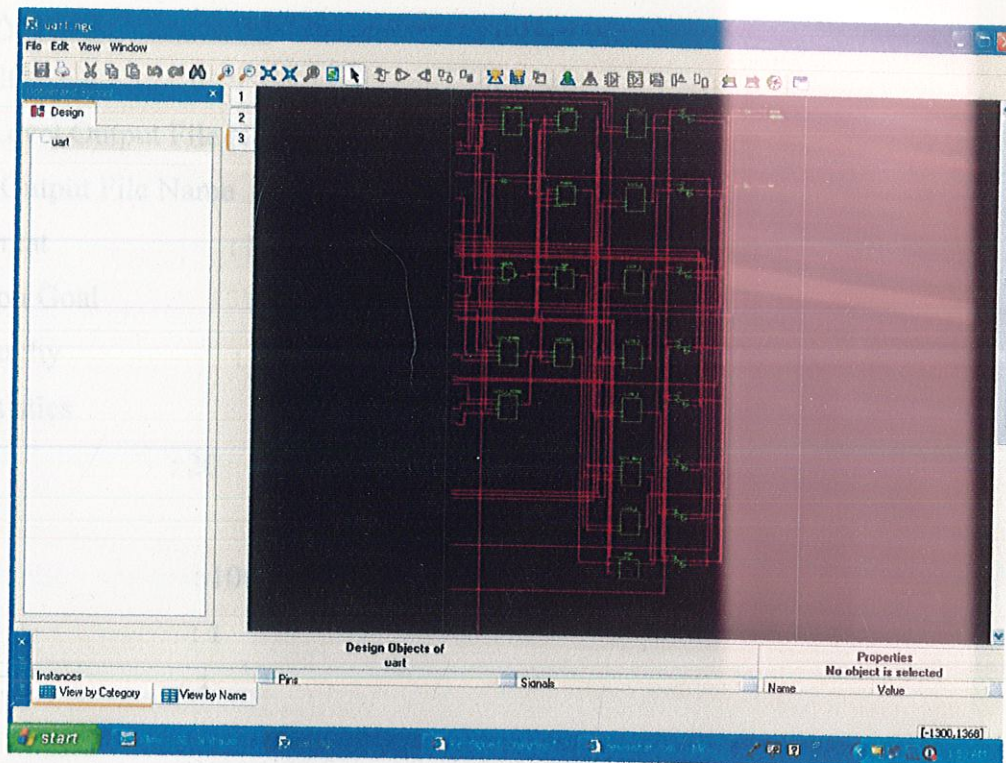


Figure 5.17: Technology Schematic 3 of UART

#### 5.4.3 DESIGN SUMMARY:

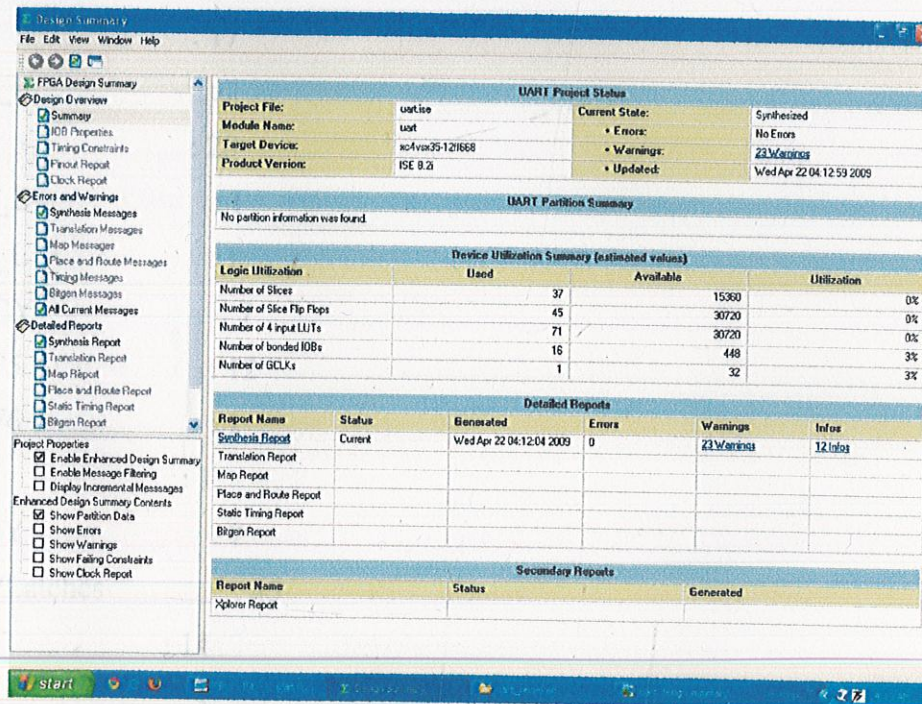


Figure 5.18: Design Summary of UART



#### 5.4.4 SYNTHESIS REPORT:

##### Final Results

RTL Top Level Output File Name : uart.ngr

Top Level Output File Name : uart

Output Format : NGC

Optimization Goal : Speed

Keep Hierarchy : NO

##### Design Statistics

# IOs : 20

##### Cell Usage :

# BELS : 100

# GND : 1

# INV : 3

# LUT1 : 7

# LUT2 : 12

# LUT2\_D : 1

# LUT3 : 17

# LUT3\_L : 1

# LUT4 : 28

# LUT4\_D : 1

# LUT4\_L : 1

# MUXCY : 7

# MUXF5 : 12

# MUXF6 : 1

# VCC : 1

# XORCY : 7

# FlipFlops/Latches : 45

# FD : 14

# FDC : 3

# FDCE : 6



```

# FDE : 11
# FDP : 10
# FDR : 1
# Clock Buffers : 1
# BUFGP : 1
# IO Buffers : 15
# IBUF : 5
# IOBUF : 8
# OBUF : 2

```

---

Device utilization summary:

---

Selected Device: 4vsx35ff668-12

```

Number of Slices : 37 out of 15360 0%
Number of Slice Flip Flops : 45 out of 30720 0%
Number of 4 input LUTs : 71 out of 30720 0%
Number of IOs : 20
Number of bonded IOBs : 16 out of 448 3%
Number of GCLKs : 1 out of 32 3%

```

---

## TIMING REPORT

### Clock Information:

```

-----+-----+-----+
Clock Signal | Clock buffer (FF name) | Load |
-----+-----+-----+
clk | BUFGP | 34 |

```



BclkX8(CLKDIV/Mmux\_BclkX8\_f6:O) | NONE(\*) (CLKDIV/ctr3\_0) | 3 |  
 CLKDIV/ctr1\_3 | NONE(CLKDIV/ctr2\_2) | 8 |

-----+-----+-----+

(\*) This 1 clock signal(s) are generated by combinatorial logic,  
 and XST is not able to identify which are the primary clock signals.

Please use the CLOCK\_SIGNAL constraint to specify the clock signal(s) generated by combinatorial logic.

INFO:Xst:2169 - HDL ADVISOR - Some clock signals were not automatically buffered by XST with BUFG/BUFR resources. Please use the buffer\_type constraint in order to insert these buffers to the clock signals to help prevent skew problems.

Destination Clock: clk rising

#### Asynchronous Control Signals Information:

-----+-----+-----+

Control Signal	Buffer(FF name)	Load
----------------	-----------------	------

-----+-----+-----+

XMIT/rst_b_inv(XMIT/rst_b_inv1_INV_0:O)	NONE(XMIT/TSR_8)	19
---	------------------	----

-----+-----+-----+

#### Timing Summary:

-----

Speed Grade: -12

Minimum period: 2.325ns (Maximum Frequency: 430.135MHz)

Minimum input arrival time before clock: 3.201ns

Maximum output required time after clock: 4.741ns

Maximum combinational path delay: 5.204ns

#### Timing Detail:

-----

All values displayed in nanoseconds (ns)



=====

Timing constraint: Default period analysis for Clock 'clk'

Clock period: 2.325ns (frequency: 430.135MHz)

Total number of paths / destination ports: 170 / 24

-----

Delay: 2.325ns (Levels of Logic = 3)

Source: XMIT/state\_FFd1 (FF)

Destination: XMIT/TSR\_0 (FF)

Source Clock: clk rising

Destination Clock: clk rising

Data Path: XMIT/state\_FFd1 to XMIT/TSR\_0

	Gate	Net				
Cell:in->out	fanout	Delay	Delay	Logical Name	(Net Name)	
-----						
FDC:C->Q	16	0.272	0.767	XMIT/state_FFd1	(XMIT/state_FFd1)	
LUT3:I0->O	1	0.147	0.000	XMIT/TSR_0_rstpot_SW0_F	(N138)	
MUXF5:I0->O	1	0.291	0.403	XMIT/TSR_0_rstpot_SW0	(N73)	
LUT4:I3->O	1	0.147	0.000	XMIT/TSR_0_rstpot	(N72)	
FDP:D		0.297		XMIT/TSR_0		

-----

Total 2.325ns (1.154ns logic, 1.171ns route)  
(49.6% logic, 50.4% route)

-----

Timing constraint: Default period analysis for Clock 'BelkX8'

Clock period: 1.636ns (frequency: 611.135MHz)

Total number of paths / destination ports: 6 / 3

-----

Delay: 1.636ns (Levels of Logic = 0)



Source: CLKDIV/ctr3\_0 (FF)  
 Destination: CLKDIV/ctr3\_0 (FF)  
 Source Clock: BclkX8 rising  
 Destination Clock: BclkX8 rising

Data Path: CLKDIV/ctr3\_0 to CLKDIV/ctr3\_0

	Gate	Net
Cell:in->out	fanout	Delay Delay Logical Name (Net Name)
-----		
FDR:C->Q	3	0.272 0.406 CLKDIV/ctr3_0 (CLKDIV/ctr3_0)
FDR:R	0.958	CLKDIV/ctr3_0
-----		

Total 1.636ns (1.230ns logic, 0.406ns route)  
 (75.2% logic, 24.8% route)

Timing constraint: Default period analysis for Clock 'CLKDIV/ctr1\_3'

Clock period: 2.054ns (frequency: 486.855MHz)

Total number of paths / destination ports: 36 / 8

Delay: 2.054ns (Levels of Logic = 9)

Source: CLKDIV/ctr2\_0 (FF)

Destination: CLKDIV/ctr2\_7 (FF)

Source Clock: CLKDIV/ctr1\_3 rising

Destination Clock: CLKDIV/ctr1\_3 rising

Data Path: CLKDIV/ctr2\_0 to CLKDIV/ctr2\_7

	Gate	Net
Cell:in->out	fanout	Delay Delay Logical Name (Net Name)
-----		



FD:C->Q	2	0.272	0.408	CLKDIV/ctr2_0 (CLKDIV/ctr2_0)
INV:I->O	2	0.322	0.000	CLKDIV/Mcount_ctr2_lut<0>_INV_0
(Result<0>1)				
MUXCY:S->O	1	0.278	0.000	CLKDIV/Mcount_ctr2_cy<0>
(CLKDIV/Mcount_ctr2_cy<0>)				
MUXCY:CI->O	1	0.034	0.000	CLKDIV/Mcount_ctr2_cy<1>
(CLKDIV/Mcount_ctr2_cy<1>)				
MUXCY:CI->O	1	0.034	0.000	CLKDIV/Mcount_ctr2_cy<2>
(CLKDIV/Mcount_ctr2_cy<2>)				
MUXCY:CI->O	1	0.034	0.000	CLKDIV/Mcount_ctr2_cy<3>
(CLKDIV/Mcount_ctr2_cy<3>)				
MUXCY:CI->O	1	0.034	0.000	CLKDIV/Mcount_ctr2_cy<4>
(CLKDIV/Mcount_ctr2_cy<4>)				
MUXCY:CI->O	1	0.034	0.000	CLKDIV/Mcount_ctr2_cy<5>
(CLKDIV/Mcount_ctr2_cy<5>)				
MUXCY:CI->O	0	0.034	0.000	CLKDIV/Mcount_ctr2_cy<6>
(CLKDIV/Mcount_ctr2_cy<6>)				
XORCY:CI->O	1	0.273	0.000	CLKDIV/Mcount_ctr2_xor<7> (Result<7>)
FD:D	0.297	CLKDIV/ctr2_7		

-----

Total                    2.054ns (1.646ns logic, 0.408ns route)

                              (80.1% logic, 19.9% route)

=====

Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'

Total number of paths / destination ports: 80 / 27

-----

Offset:                3.201ns (Levels of Logic = 3)

Source:                R\_W (PAD)

Destination:        XMIT/TDR\_0 (FF)



Destination Clock: clk rising

Data Path: R\_W to XMIT/TDR\_0

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name (Net Name)	
-----					
IBUF:I->O	3	0.754	0.581	R_W_IBUF (R_W_IBUF)	
LUT4:I0->O	2	0.147	0.543	_and00031 (loadTDR)	
LUT2:I1->O	8	0.147	0.467	XMIT/_and00001 (XMIT/_and0000)	
FDE:CE		0.562		XMIT/TDR_0	
-----					
Total		3.201ns (1.610ns logic, 1.591ns route)			
		(50.3% logic, 49.7% route)			

Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'

Total number of paths / destination ports: 9 / 7

Offset: 4.741ns (Levels of Logic = 2)

Source: TDRE (FF)

Destination: SCI\_IRQ (PAD)

Source Clock: clk rising

Data Path: TDRE to SCI\_IRQ

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name (Net Name)	
-----					
FDP:C->Q	13	0.272	0.673	TDRE (TDRE)	
LUT2:I1->O	1	0.147	0.394	_or00041 (SCI_IRQ_OBUF)	
OBUF:I->O		3.255		SCI_IRQ_OBUF (SCI_IRQ)	



-----  
Total                    4.741ns (3.674ns logic, 1.067ns route)  
                              (77.5% logic, 22.5% route)

=====

Timing constraint: Default path analysis

Total number of paths / destination ports: 22 / 8

-----

Delay:                5.204ns (Levels of Logic = 3)

Source:              R\_W (PAD)

Destination:        DBUS<7> (PAD)

Data Path: R\_W to DBUS<7>

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical	Name (Net Name)
-----					
IBUF:I->O	3	0.754	0.581	R_W_IBUF	(R_W_IBUF)
LUT2:I0->O	8	0.147	0.467	SCI_Read_inv1	(SCI_Read_inv)
IOBUF:T->IO		3.255		DBUS_1_IOBUF	(DBUS<1>)

-----

Total                    5.204ns (4.156ns logic, 1.048ns route)  
                              (79.9% logic, 20.1% route)

=====

CPU : 18.69 / 20.08 s | Elapsed : 19.00 / 20.00 s

Total memory usage is 242880 kilobytes

Number of errors : 0 ( 0 filtered)

Number of warnings : 23 ( 0 filtered)



Number of infos : 12 ( 0 filtered)

#### 5.4.5 MODEILSIM SIMULATIONS RESULTS:

Following Results clearly represent the function of UART,

ADDR2	R_W	Action
00	1	DBUS $\square$ RDR
00	0	TDR $\square$ DBUS
01	1	DBUS $\square$ SCSR
01	0	DBUS $\square$ hi-Z
1-	1	DBUS $\square$ SCCR
1-	0	SCCR $\square$ DBUS

Table 3



Figure 5.19



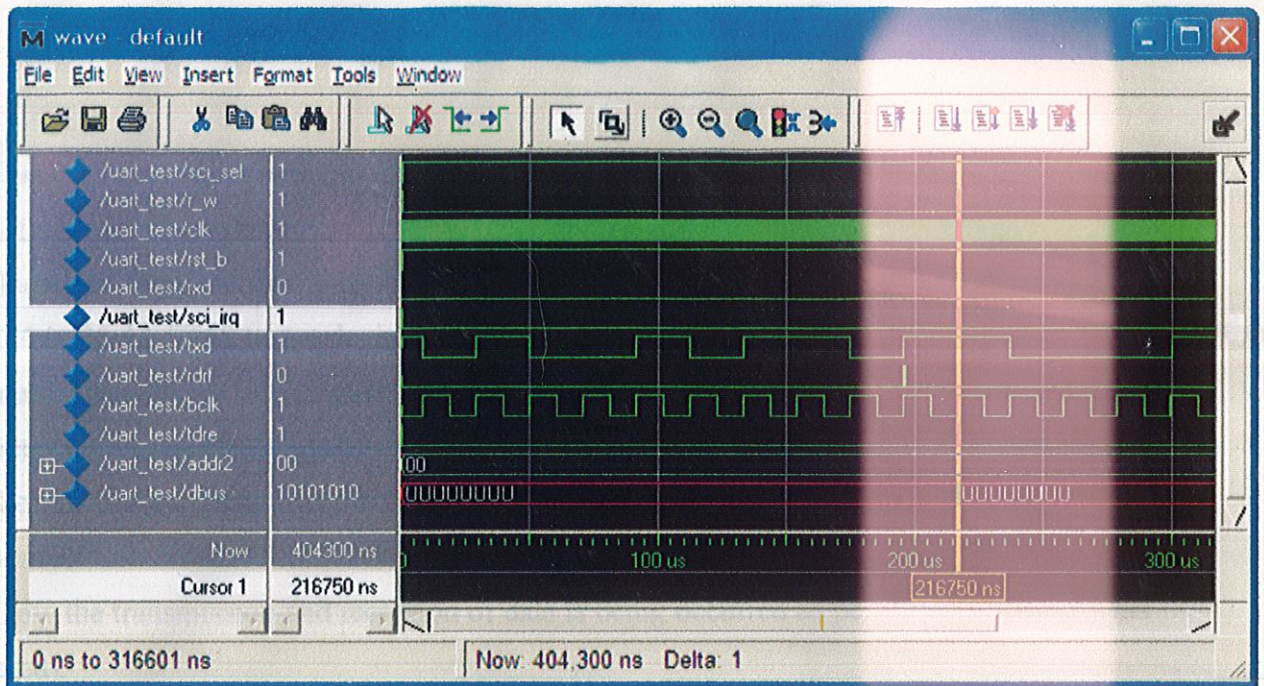


Figure 5.20

Future Prospects: We can modify UART to USART for synchronous data transmission.



Figure 5.21



## CONCLUSION

The software developed for the 8bit UART is synthesized using Xilinx and Modelsim software using vertex4 as a family. The resulting implementation requires only 71 flip-flops (less than the latest updated 8bit UART which uses 78 flip-flops) and fits into xc4vsx35-12-ff668 target device. The synthesis report shows that the speed grade is -12 means that the device is guaranteed to send a signal from an input pin through to an output pin in under  $(10 - 20\% \text{ of } 10)\text{ns} = 9.8 \text{ ns}$ . The timing detail of the report shows that the maximum frequency is 430.135MHz (corresponding to delay = 2.325ns). The simulation waveform of UART clearly represents the characteristics of UART design (how the transmission and reception of data is being occurred as per different flag signals used).

**Future Prospects:** We can modify UART to USART for synchronous data transmission also. Due to time constraints, we could not further pursue the modification of the UART.



## BIBLIOGRAPHY

1. Roth, Jr., Charles *Digital Systems Design Using VHDL*. Thomson Asia Pvt. Ltd 2005
2. Bhasker, J *A VHDL Primer*. Prentice Hall of India 2005
3. Ashenden, Peter J *The Designer's Guide to VHDL*. Harcourt India Pvt. Ltd. 2002
4. Mano, Morris *Digital Logic and Circuit Design*. Prentice Hall of India 2005
5. [www.ieeexplore.com](http://www.ieeexplore.com)
6. [www.nuigalway.ie/cs/staff/software/modelsim.html](http://www.nuigalway.ie/cs/staff/software/modelsim.html)
7. [www.en.wikipedia.org/wiki/Xilinx](http://www.en.wikipedia.org/wiki/Xilinx)
8. [www.arm.com/products/DevTools/LT-XC4VLX160-200.html](http://www.arm.com/products/DevTools/LT-XC4VLX160-200.html)