International Journal of Multidisciplinary Engineering in Current Research
ISSN: 2456-4265, Volume 2, Issue 12, December 2017, http://ijmec.com/

Implementation of UART with BIST Technique in FPGA

THOTTARAMUDI USHA, L.V.V.DIVYA
M. Tech Scholar, Assistant Professor
Department of ECE
ISTS Women's Engineering College, Rajanagaram, Rajamahendravaram, A.P, India.

***Abstract* - Asynchronous serial communication is usually implemented by Universal Asynchronous Receiver Transmitter (UART), mostly used for short distance, low speed, low cost data exchange between processor and peripherals. UART allows full duplex serial communication link, and is used in data communication and control system. There is a need for realizing the UART function in a single or a very few chips. Further, design systems without full testability are open to the increased possibility of product failures and missed market opportunities. Also, there is a need to ensure the data transfer is error proof. This paper targets the introduction of Built-in self test (BIST) and Status register to UART, to overcome the above two constraints of testability and data integrity. The 8-bit UART with status register and BIST module is coded in Verilog HDL and synthesized and simulated using Xilinx XST and ISim version 14.4 and realized on FPGA. The results indicate that this model eliminates the need for higher end, expensive testers and thereby it can reduce the development time and cost.**

Index Terms - **UART, BIST, Error check, Status register, LFSR.**

1. INTRODUCTION

A UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices. Specifically, it provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices. As part of this interface, the UART also Converts the bytes it receives from computer along parallel circuits into single serial bit stream for outbound transmission. On inbound transmission, converts the serial bit stream into the bytes that the computer handles. 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. Adds start and stop delineators on outbound and strips them from inbound transmissions Handles interrupts from the keyboard and mouse (which are serial devices with special ports) May handle other kinds of interrupt and device management that require coordinating the computer's speed of operation with device speeds.

Serial transmission is commonly used with modems and for non-networked communication between computers, terminals and other devices.

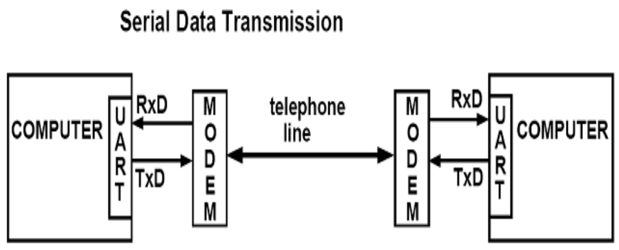


Fig.1. Serial Data Transmission

The communications links across which computers or parts of computers talk to one another may be either serial or parallel. A parallel link transmits several streams of data(perhaps representing particular bits of a stream of bytes) along multiple channels (wires, printed circuit tracks, optical fibers etc.) a serial link transmits a single stream of data. At first sight it would seem that a serial link must be inferior to a parallel one, because it can transmit less data on each clock tick. However, it is often the case that serial links can be clocked considerably faster than parallel links, and achieve a higher data rate. A number of factors allow serial to be clocked at a greater rate:

- Clock skew between different channels is not an issue (for unclocked serial links)
- A serial connection requires fewer interconnecting cables (e.g. wires/fibres) and hence occupies less space. The extra space allows for better isolation of the channel from its surroundings
- Crosstalk is less of an issue, because there are fewer conductors in proximity.

In many cases, serial is a better option because it is cheaper to implement. Many ICs have serial interfaces, as opposed to parallel ones, so that they have fewer pins and are therefore cheaper.

In telecommunications and computer science, serial communications is the process of sending data one bit at one time, sequentially, over a communications channel or computer bus. This is in contrast to parallel communications, where all the bits of each symbol are sent together. Serial communications is used for all long-haul communications and most computer networks, where the cost of cable and synchronization difficulties make parallel communications impractical. Serial computer buses are

International Journal of Multidisciplinary Engineering in Current Research
ISSN: 2456-4265, Volume 2, Issue 12, December 2017, http://ijmec.com/

becoming more common as improved technology enables them to transfer data at higher speeds.

1.2 Serial versus parallel

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port. (SPP)

1. Serial Cables can be longer than Parallel cables.
2. Serial communication don't need as many wires than parallel transmission.
3. Microcontrollers have also proven to be quite popular recently.

Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's. Only two pins are commonly used, Transmit Data (TXD) and Receive Data (RXD) compared with at least 8 pins if you use a 8 bit Parallel method (You may also require a Strobe).

There are two primary forms of serial transmission: Synchronous and Asynchronous. Depending on the modes that are supported by the hardware, the name of the communication sub-system will usually include, if it supports Asynchronous communications and if it supports Synchronous communications. Both forms are described below.

1.3 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.

A form of Synchronous transmission is used with printers and fixed disk devices in that the data is sent on one set of wires while a clock or strobe is sent on a different wire. Printers and fixed disk devices are not normally serial devices because most fixed disk interface standards send an entire word of data for each clock or strobe signal by using a separate wire for each bit of the word.

In the PC industry, these are known as Parallel devices. The standard serial communications hardware in the PC does not support Synchronous operations. This mode is described here for comparison purposes only.

1.4 Asynchronous Serial Transmission

Asynchronous serial communication has advantages of less transmission line, high reliability, and long transmission distance, therefore is widely used in data exchange between computer and peripherals. Asynchronous serial communication is usually implemented by Universal Asynchronous Receiver Transmitter (UART). UART allows full-duplex communication in serial link, thus has been widely used in the data communications and control system.

In actual applications, usually only a few key features of UART are needed. Specific interface chip will cause waste of resources and increased cost. Particularly in the field of electronic design, SOC technology is recently becoming increasingly mature.

This situation results in the requirement of realizing the whole system function in a single or a very few chips. Designers must integrate the similar function module into FPGA. This paper uses VHDL to implement the UART core functions and integrate them into a FPGA chip to achieve compact, stable and reliable data transmission, which effectively solves the above problem.

Basic UART communication needs only two signal lines (RXD, TXD) to complete full-duplex data communication. TXD is the transmit side, the output of UART; RXD is the receiver, the input of UART. UART’s basic features are:

There are two states in the signal line, using logic 1 (high) and logic 0(low) to distinguish respectively. For example, when the transmitter is idle, the data line is in the high logic state.

Otherwise 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. These two clocks must be accurate enough to not have the frequency drift by more than 10% during the transmission of the remaining bits in the word.

After the Start Bit, the individual data bits of the word 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. For example, if it takes two seconds to send each bit, the receiver will examine the signal to determine if it is a 1 or a 0 after one second has passed, then it will wait two seconds and then examine the value of the next bit, and so on.

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 are "self-synchronizing", if there are no data to transmit, the transmission line can be idle.

Asynchronous serial communication describes an asynchronous transmission protocol in which a start signal is sent prior to each byte, character or code word and a stop signal is sent after each code word.

The start signal serves to prepare the receiving mechanism for the reception and registration of a symbol and the stop signal serves to bring the receiving mechanism to rest in preparation for the reception of the next symbol. A common kind of start-stop transmission is ASCII over RS-232, for example for use in teletypewriter operation.

In the diagram, a start bit is sent, followed by eight data bits, no parity bit and one stop bit, for a 10-bit character frame. The number of data and formatting bits, and the transmission speed, must be pre-agreed by the communicating parties.After the stop bit, the line may remain idle indefinitely, or another character may immediately be started.

1.5 Bits, Baud and Symbols

Baud is a measurement of transmission speed in asynchronous communication. Because of advances in modem communication technology, this term is frequently misused when describing the data rates in newer devices.

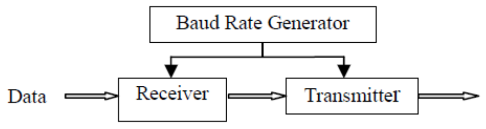
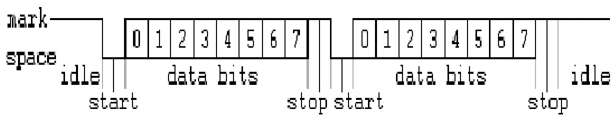


Figure 2. UART Module

Gigure.3. A



synchronous Serial Transmission

1.5.1 Baud Rate Generator

Baud Rate Generator is actually a frequency divider. The baud rate frequency factor can be calculated according to a given system clock frequency (oscillator clock) and the requested baud rate.

The calculated baud rate frequency factor is used as the divider factor. In this design, the frequency clock produced by the baud rate generator is not the baud rate clock, but 16 times the baud rate clock. The purpose is to precisely sample the asynchronous serial data at the receiver.

Assume that the system clock is 32MHz, baud rate is 9600bps, and then the output clock frequency of baud rate generator should be 16 * 9600Hz. Therefore the frequency coefficient (M) of the baud rate generator is:

$M = 32MHz/16*9600Hz = 208$

When the UART receives serial data, it is very critical to determine where to sample the data information. The ideal time for sampling is at the middle point of each serial data bit. In this design, the receive clock frequency is designed to be 16 times the baud rate, therefore, each data width received by UART is 16 times the receive clock cycle.

2. RECEIVER MODULE

During the UART reception, the serial data and the receiving clock are asynchronous, so it is very important to correctly determine the start bit of a frame data. The receiver module receives data from RXD pin. RXD jumps into logic 0 from logic 1 can be regarded as the beginning of a data frame. When the UART receiver module is reset, it has been waiting the RXD level to jump.

The start bit is identified by detecting RXD level changes from high to low. In order to avoid the misjudgment of the start bit caused by noise, a start bit error detect function is added in this design, which requires the received low level in RXD at least over 50% of the baud rate to be able to determine the start bit arrives. Since the receive clock frequency is 16 times the baud rate in the design, the RXD low level lasts at least 8 receiving clock cycles is considered start bit arrives.

Once the start bit been identified, from the next bit, begin to count the rising edge of the baud clock, and sample RXD when counting. Each sampled value of the logic level is deposited in the register rbuf [7, 0] by order. When the count equals 8, all the data bits are surely received, also the 8 serial bits are converted into a byte parallel data.

The serial receiver module includes receiving, serial and parallel transform, and receive caching, etc. In this paper we use finite state machine to design, shown in Fig. 3

The state machine includes five states: R_START (waiting for the start bit), R_CENTER (find midpoint), R_WAIT (waiting for the sampling), R_SAMPLE (sampling), and R_STOP (receiving stop bit).

R_START Status: When the UART receiver is reset, the receiver state machine will be in this state. In this state, the state machine has been waiting for the RXD level to jump over from logic 1 to logic 0, i.e. the start bit.

This alerts the beginning of a new data frame. Once the start bit is identified, the state machine will be transferred to R_CENTER state. In Fig. 3, RXD_SYNC is a synchronization signal of RXD. Because when sampling logic 1 or logic 0, we do not want the detected signal to be unstable. So we do not directly detect RXD signal, but detect the synchronization signal RXD_SYNC.

R_CENTER Status: For asynchronous serial signal, in order to detect the correct signal each time, and minimize the total error in the later data bits detection. Obviously, it is the most ideal to detect at the middle of each bit.

In this state, the task is to find the midpoint of each bit through the start bit. The method is by counting the number of bclkr (the receiving clock frequency generated by the baud rate generator) (RCNT16 isthe counter of bclkr).

In addition, the start bit detected in the R_START may not be a really start bit, it may be an occasional interference sharp pulse (negative pulse). This interference pulse cycle is very short. Therefore, the signal that maintains logic 0 over 1 / 4 bit time must be a start bit.

R_WAIT Status: When the state machine is in this state, waiting for counting bclkr to 15, then entering into R_SAMPLE to sample the data bits at the 16th bclkr.At the same time determining whether the collected data bit length has reached the data frame length (FRAMELEN). If reaches, it means the stop bits arrives. The FRAMELEN is modifiable in the design (using the Generic). In this design it is 8, which corresponds to the 8-bit data format of UART.

R_SAMPLE Status: Data bit sampling. After sampling the state machine transfers to R_WAIT state unconditionally, waits for the arrival of the next start bit.

R_STOP Status: Stop bit is either 1 or 1.5, or 2. State machine doesn't detect RXD in R_STOP, output frame receiving done signal (REC_DONE <= '1 '). After the stop bit, state machine turns back to R_START state, waiting for the next frame start bit.

The function of transmit module is to convert the sending 8-bit parallel data into serial data, adds start bit at the head of the data as well as the parity and stop bits at the end of the data.

When the UART transmit module is reset by the reset signal, the transmit module immediately enters the ready state to send. In this state, the 8-bit parallel data is read into the register txdbuf [7: 0]. The transmitter only needs to output 1 bit every 16 bclkt (the transmitting clock frequency generated by the baud rate generator) cycles.

The order follows 1 start bit, 8 data bits, 1 parity bit and 1 stop bit. The parity bit is determined according to the number of logic 1 in 8 data bits. Then the parity bit is output. Finally, logic 1 is output as the stop bit. Fig.4 shows the transmit module state diagram.

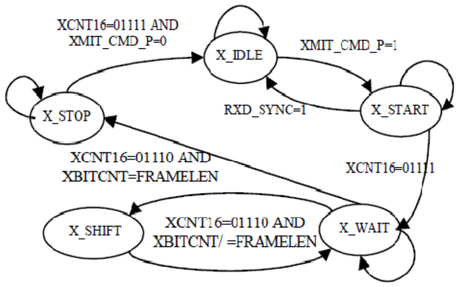


Figure 4. Transmit Module State Diagram

XMIT_CMD. XMIT_CMD_P is a processed signal of XMIT_CMD, which is a short pulse signal. Since XMIT_CMD is an external signal, outside FPGA, its pulse width is unable to be limited. If XMIT_CMD is valid, it is still valid after sending on UART data frame.

Then the UART transmitter will think by mistake that a new data transmit command has arrived, and once again start the frame transmit. Obviously the frame transmit is wrong. Here we limit the pulse width of XMIT_CMD. XMIT_CMD_P is its processed signal. When XMIT_CMD_P = '1 ', the state machine transferred to X_START, get ready to send a start bit.

X_START Status: In this state, sends a logic 0 signal to the TXD for one bit time width, the start bit. Then the state machine transferred to X_WAIT state. XCNT16 is the counter of bclkt.

X_WAIT Status: Similar with the R_WAIT of UART receive state machine.

X_SHIFT Status: In this state, the state machine realizes the parallel to serial conversion of outgoing data. Then immediately return to X_WAIT state.

X_STOP Status: Stop bit transmit state. When the data frame transmit is completed, the state machine transferred to this state, and sends 16 bclkt cycle logic 1 signal, that is, 1 stop bit. The state machine turns back to X_IDLE state after sending the stop bit, and waits for another data frame transmit command.

Traditionally, a Baud Rate represents the number of bits that are actually being sent over the media, not the amount of data that is actually moved from one DTE device to the ot her. The Baud count includes the overhead bits Start, Stop and Parity that are generated by the sending UART and removed by the receiving UART. This means that seven-bit words of data actually take 10 bits to be completely transmitted.

Therefore, a modem capable of moving 300 bits per second from one place to another can normally only move 30 7-bit words if Parity is used and one Start and Stop bit are present. If 8-bit data words are used and Parity bits are also used, the data rate falls to 27.27 words per second, because it now takes 11 bits to send the eight-bit words, and the modem still only sends 300 bits per second.

The formula for converting bytes per second into a baud rate and vice versa was simple until error-correcting modems came along. These modems receive the serial stream of bits from the UART in the host computer (even when internal modems are used the data is still frequently serialized) and converts the bits back into bytes.

These bytes are then combined into packets and sent over the phone line using a Synchronous transmission method. This means that the Stop, Start, and Parity bits added by the UART in the DTE (the computer) were removed by the modem before transmission by the sending modem. When these bytes are received by the remote modem, the remote modem adds Start, Stop and Parity bits to the words, converts them to a serial format and then sends them to the receiving UART in the remote computer, who then strips the Start, Stop and Parity bits.

The reason all these extra conversions are done is so that the two modems can perform error correction, which means that the receiving modem is able to ask the sending modem to resend a block of data that was not received with the correct checksum. This checking 12 is handled by the modems, and the DTE devices are usually unaware that the process is occurring.

Asynchronous Serial Reception

A multiplexed data communication pulse can only be in one of two states but there are many names for the two states. When on, circuit closed, low voltage, current flowing, or a logical zero, the pulse is said to be in the "space" condition. When off, circuit open, high voltage, current stopped, or a logical one, the pulse is said to be in the "mark" condition. A character code begins with the data communication circuit in the space condition. If the mark condition appears, a logical one is recorded otherwise a logical zero. There are six basic steps in receiving a serial character code into a parallel register.

1. To keep track of time, the receiver employs a clock which "ticks." When the line is in the space condition, the receiver samples the line 16 times the data rate. In other words, a data interval is equal to 16 clock ticks. In this way the receiver can determine the beginning of the start bit and "move over" to the center of the bit time for data sampling.
2. when the line goes into the mark state, declare a "looking for start bit" condition and wait one half the bit interval or eight clock ticks.
3. sample the line again and if it has not remained in the mark condition, consider this to be a spurious voltage change and go back to step one.
4. if the line was still in the mark state, then consider this a valid start bit. Shift the start bit into an eight-bit shift register and wait one bit time or 16 clock ticks.
5. after one bit time sample the line (the data should have been there for the last eight clock ticks, and should remain for eight more clock ticks). Now shift the sample into the shift register.
6. continue steps four and five seven more times. After the eighth shift, the start bit will "migrate" into a flip-flop indicating character received.
7. Go to step one.

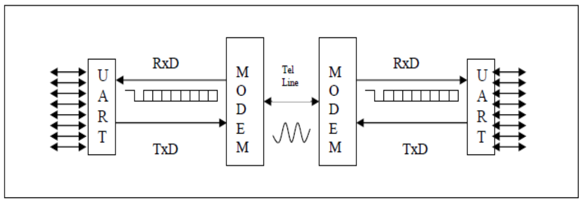


Fig. 1: Serial Data Transmission and Receive

Before the transmitter and receiver UARTs will work, they must also agree on the same values of five parameters.

1. both sides must agree on the number of bits per character.
2. the speed or Baud of the line must be the same on both sides.
3. both sides must agree to use or not use parity.
4. if parity is used, both sides must agree on using odd or even parity.
5. the number of stop bits must be agreed upon.

Having said all this, most DTEs today employ eight data bits, no parity, and one stop bit. Thus there is a rule-of-thumb that the number of characters per second is equal to the Baud divided by 10 for a typical RS-232 or RS-423 data line.

Other UART Functions

In addition to the basic job of converting data from parallel to serial for transmission and from serial to parallel on reception, 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. The function of each of these additional signals is defined in the EIA RS232-C standard.

3. SIMULATION OF MODULES

Baud Rate Generator Simulation

During simulation, the system clock frequency is set to 32MHz, and baud rate is set to 9600bps. Therefore the receiving sampling clock frequency generated by the baud rate generator is 153600Hz, which is 16 times of the baud rate. Thus the frequency coefficient of baud rate generator can be calculated, which equals 208.

Figure 5 shows the simulation result of baud rate generator. The simulation report shows that this module uses 42 logic elements （ <1% ）, 33 registers (2%), and meets timing requirement.

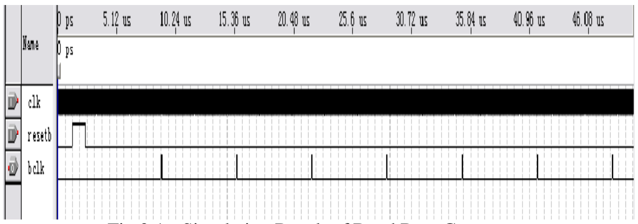


Fig 3.1-: Simulation Result of Baud Rate Generator

Receiver Simulation

During receiver simulation, the receiving sampling clock frequency generated by the baud rate generator is set to 153600 Hz, UART receiving baud rate is set to 9600bps. The input sequence is: 00110110001, including the start bit 0, parity bit 0 and 1 stop bit. The received data is stored into the register rbuf.

Fig. 6 shows the receiver module simulation diagram. The figure shows that the data in rbuf from high to low is 00110110, which is just the part of data bits of UART frame.

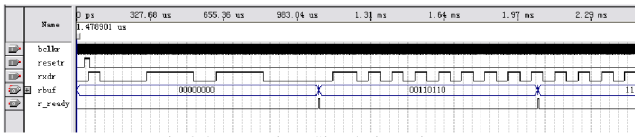


Fig 3.2 -: Receiver Simulation Diagram

Transmitter Simulation

During transmitter simulation, the sending clock frequency generated by the baud rate generator is set to 153600 Hz, and UART transmitting baud rate is set to 9600bps.

Figure 7 shows the transmitter module simulation diagram. The simulation report shows that this module uses 78 logic elements（<1%, 13 pins (4%), and meets timing requirement.

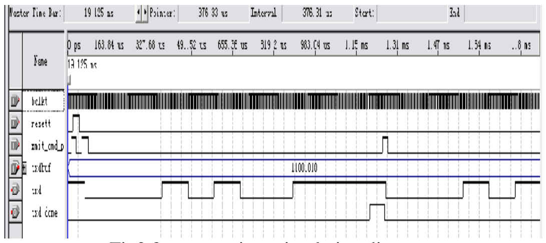


Fig3.3 -: transmitter simulation diagram

RTL of Top File

Fig.3.3 shows the RTL of UART Top File. It includes the baud rate generator, receiver, and transmitter modules.

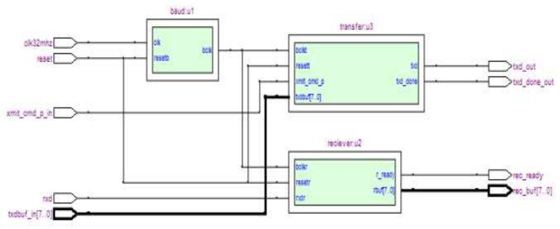
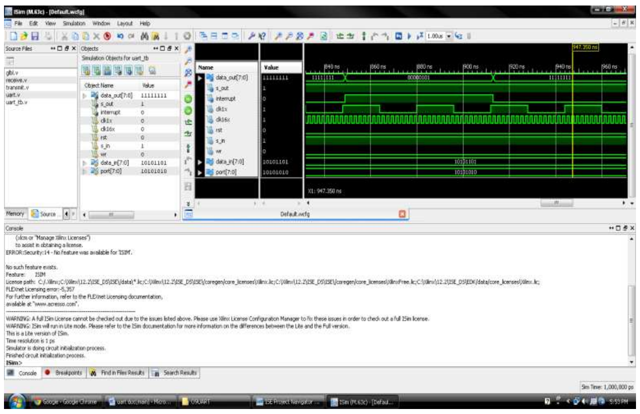


Fig 3.4 -: RTL of Top File

4. SIMULATION RESULTS

UART



```
    (xlcm or "Manage Xilinx Licenses")
    to assist in obtaining a license.
ERROR:Security:14 - No feature was available for 'ISIM'.

No such feature exists.
Feature:    ISIM
License path:  C:/.Xilinx;C:\Xilinx\12.2\ISE_DS\ISE\/data\*.lic;C:\Xilinx\12.2\ISE_DS\ISE\/coregen/core_licenses\Xilinx.lic;C:\Xilinx\12.2\ISE_DS\ISE\/coregen/core_licenses\XilinxFree.lic;C:\Xilinx\12.2\ISE_DS\EDK/data/core_licenses\Xilinx.lic;
FLEXnet Licensing error:-5,357
For further information, refer to the FLEXnet Licensing documentation,
available at "www.acresso.com".
----------------------------------------------
WARNING: A full ISim License cannot be checked out due to the issues listed above. Please use Xilinx License Configuration Manager to fix these issues in order to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

The above figure shows the simulation of the universal asynchronous reception and transmission of the data.

ADVANTAGES AND APPLICATIONS

UART has a lot of applications and advantages. The following are the applications and advantages of the UART.

ADVANTAGES

The advantages of the UART are as follows:

1. Low cost
2. Requires single transmission line
3. Low power consumption
4. Error free transmission
5. Full duplex communication

APPLICATIONS

The applications of the UART are as follows:

UART has lot of applications in emerging world, they are as follows

1. For low speed applications in short distance communication.
2. Used in data exchange between computer and peripherals.

3. Widely used in data communication systems.
4. GPS systems
5. Modems

5. CONCLUSION

The Universal Asynchronous Receiver/Transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires.

This project proposes the hardware implementation of vlsi architecture for UART that have been modified to improve performance. This project was implemented in verilog the coding is done in Verilog HDL and the FPGA synthesis is done using Xilinx Spartan library.

Now a days UART is one of the emerging field technology in the communication world and it is used in the short distance communication for low cost applications.

REFERENCES

[1]. S. Wang, **“**Generation of low power dissipation and high fault coverage patterns for scan-based BIST”, in *Proceedings of International Test Conference*, 2002, pp. 834 –843.
[2]. M. Ibrahim Abubakar, “A Built in Self Testable Bit-Slice Processor”, Faculty of Computer Science & Information Technology, University of Malaya, May 1995.
[3]. J. Turino, “RTL DFT Rule Checking – The Circuit Designer’s Secret Weapon”, *Integrated System DesignMagazine*, 2000.
[4]. A. P. Stroele, and H. J. Wunderlich, “Hardware-Optimal Test Register Insertion”, in *IEEE Transactions OnComputer-Aided Design of Integrated Circuits and Systems*, June 1998, Vol. 17, No. 6, pp. 531-539.
[5]. Z. Navabi, “VHDL Analysis and Modeling of Digital Systems”, McGraw-Hill Inc., 1991.
[6]. M. S. Michael, “A Comparison of the INS8250, NS16450 and NS16550AF Series of UARTs”, *NationalSemiconductor Application Note 493*, April 1989.
[7]. “PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs”, *National Semiconductor ApplicationNote*, June 1995.
[8]. M. S. Harvey, *Generic UART Manual*, SiliconValley, December 1999.
[9]. O. A. Petlin, and S. B. Furber, "Built-In-Self-Testing of Micropipelines” in *Advanced Research in AsynchronousCircuits and Systems*, IEEE, 1997, pp 22-29.
[10]. J. Turino, “RTL DFT Rule Checking – The Circuit Designer’s Secret Weapon”, *Integrated System DesignMagazine*, 2000.
[11]. C. H. Roth, *Digital System Design Using VHDL*, PWS Publishing Company, 1998
[12]. http://www.xess.com