**1. Introduction**

A **Universal Asynchronous Receiver/Transmitter**, abbreviated **UART**, is a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485. The *universal* designation indicates that the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling etc.) are handled by a driver circuit external to the UART.

A UART is usually an individual or part of an integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers. A dual UART, or **DUART**, combines two UARTs into a single chip. Many modern ICs now come with a UART that can also communicate synchronously; these devices are called **USARTs** (universal synchronous/asynchronous receiver/transmitter).

* 1. **Transmitting and receiving serial data**

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 a shift register, 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 less costly than parallel transmission through multiple wires.

The UART usually does not directly generate or receive the external signals used between different items of equipment. Separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels. External signals may be of many different forms. Examples of standards for voltage signaling are RS-232, RS-422 and RS-485 from the EIA. Historically, current (in current loops) was used in telegraph circuits. Some signaling schemes do not use electrical wires. Examples of such are optical fiber, IrDA (infrared), and (wireless) Bluetooth in its Serial Port Profile (SPP). Some signaling schemes use modulation of a carrier signal (with or without wires). Examples are modulation of audio signals with phone line modems, RF modulation with data radios, and the DC-LIN for power line communication.

**1.2 Transmitter**

Transmission operation is simpler since it is under the control of the transmitting system. As soon as data is deposited in the shift register after completion of the previous character, the UART hardware generates a start bit, shifts the required number of data bits out to the line, generates and appends the parity bit (if used), and appends the stop bits. Since transmission of a single character may take a long time relative to CPU speeds, the UART will maintain a flag showing busy status so that the host system does not deposit a new character for transmission until the previous one has been completed; this may also be done with an interrupt. Since full-duplex operation requires characters to be sent and received at the same time, UARTs use two different shift registers for transmitted characters and received characters.

**1.3 UART in modems**

Modems for personal computers that plug into a motherboard slot must also include the UART function on the card. The original 8250 UART chip shipped with the IBM personal computer had a one character buffer for the receiver and the transmitter each, which meant that communications software performed poorly at speeds above 9600 bits/second, especially if operating under a multitasking system or if handling interrupts from disk controllers. High-speed modems used UARTs that were compatible with the original chip but which included additional FIFO buffers, giving software additional time to respond to incoming data.

A look at the performance requirements at high bit rates shows why the 16, 32, 64 or 128 byte FIFO is a necessity. The Microsoft specification for a DOS system requires that interrupts not be disabled for more than 1 millisecond at a time. Some hard disk drives and video controllers violate this specification. 9600 bit/s will deliver a character approximately every millisecond, so a 1 byte FIFO should be sufficient at this rate on a DOS system which meets the maximum interrupt disable timing. Rates above this may receive a new character before the old one has been fetched, and thus the old character will be lost. This is referred to as an overrun error and results in one or more lost characters.

**1.4 Design using HDL**

**1.4.1 Verilog HDL**

**Verilog**, standardized as **IEEE 1364**, is a [hardware description language](http://en.wikipedia.org/wiki/Hardware_description_language) (HDL) used to model electronic systems. It is most commonly used in the design and verification of [digital circuits](http://en.wikipedia.org/wiki/Digital_electronics) at the [register-transfer level](http://en.wikipedia.org/wiki/Register-transfer_level) of [abstraction](http://en.wikipedia.org/wiki/Abstraction_%28computer_science%29). It is also used in the verification of [analog circuits](http://en.wikipedia.org/wiki/Analogue_electronics) and [mixed-signal circuits](http://en.wikipedia.org/wiki/Mixed-signal_integrated_circuit).

Hardware description languages such as Verilog differ from software [programming languages](http://en.wikipedia.org/wiki/Programming_language) because they include ways of describing the propagation of time and signal dependencies (sensitivity). There are two types of assignment operators; a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form. At the time of Verilog's introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical [schematic capture](http://en.wikipedia.org/wiki/Schematic_capture) software and specially written software programs to document and [simulate electronic circuits](http://en.wikipedia.org/wiki/Electronic_circuit_simulation).

The designers of Verilog wanted a language with syntax similar to the [C programming language](http://en.wikipedia.org/wiki/C_%28programming_language%29), which was already widely used in engineering software development. Like C, Verilog is [case-sensitive](http://en.wikipedia.org/wiki/Case-sensitive) and has a basic [preprocessor](http://en.wikipedia.org/wiki/Preprocessor) (though less sophisticated than that of ANSI C/C++). Its [control flow](http://en.wikipedia.org/wiki/Control_flow) [keywords](http://en.wikipedia.org/wiki/Keyword_%28computer_programming%29) (if/else, for, while, case, etc.) are equivalent, and its [operator precedence](http://en.wikipedia.org/wiki/Operator_precedence) is compatible. Syntactic differences include variable declaration (Verilog requires bit-widths on net/reg types[[*clarification needed*](http://en.wikipedia.org/wiki/Wikipedia:Please_clarify)]), demarcation of procedural blocks (begin/end instead of curly braces {}), and many other minor differences.

A Verilog design consists of a hierarchy of modules. Modules encapsulate *design hierarchy*, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a [dataflow language](http://en.wikipedia.org/wiki/Dataflow_language).

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

A subset of statements in the Verilog language are synthesizable. Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a [netlist](http://en.wikipedia.org/wiki/Netlist), a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific [FPGA](http://en.wikipedia.org/wiki/FPGA) or [VLSI](http://en.wikipedia.org/wiki/VLSI) technology. Further manipulations to the netlist ultimately lead to a circuit fabrication blueprint (such as a [photo mask set](http://en.wikipedia.org/wiki/Mask_set) for an [ASIC](http://en.wikipedia.org/wiki/Application-specific_integrated_circuit) or a [bitstream](http://en.wikipedia.org/wiki/Bitstream) file for an [FPGA](http://en.wikipedia.org/wiki/FPGA)).

**1.4.2 Verilog Design Flow**

The following design flow describes the procedure for behavioral simulation, synthesis, and implementation of Verilog designs containing CORE Generator™ modules using the following vendor tools:

|  |  |
| --- | --- |
| **Function** | **Tools** |
| Synthesis | Xilinx® XST  Synopsys FPGA Compiler II  Mentor Graphics Precision Synthesis  Synopsys Synplify |
| Simulation | ModelSim/VLOG  Cadence Verilog-XL  Cadence NC-Verilog  Synopsys VCS/VCSi |
|  |  |

Figure 1.1: Vendor Tools

**Logic Synthesis**

* Design described in a Hardware Description Language (HDL)

Verilog, VHDL

**Simulation to check for correct functionality**

* Simulation semantics of language

**Synthesis tool**

* Identifies logic and state elements
* Technology-independent optimizations (state assignment, logic

Minimization)

* Map logic elements to target technology (standard cell library)
* Technology-dependent optimizations (multi-level optimization,

gate strengths, etc.

**Features of Verilog**

* A concurrent language (syntax similar to C)
* Provides a way to specify concurrent activities
* Allows timing specification

**Applications of Verilog**

* Description of design at a higher level
* Development of formal models
* System documentation
* Simulation to uncover errors (bugs) in design
* Synthesis of designs
* Design reuse

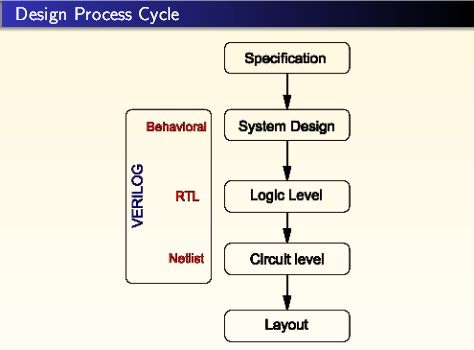
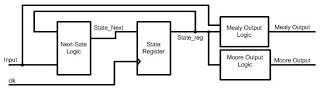


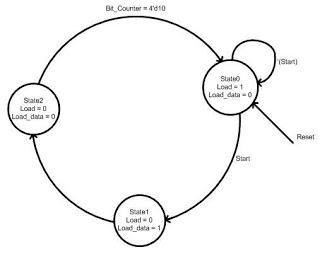
Figure 1.2: Design Process Cycle

**Design and Implementation of UART Transmitter in Verilog HDL and FPGAs**

Every digital system is composed of two parts : 1. Finite State Machine (FSM) 2. Data Path.FSM controls the program flow, while a datapath performs different operations under control of FSM.  
An FSM is used for modeling of a systen that transits among a finite number of internal states. The transitions of the system depend on the external input and current state. Unlike a regular sequential circuit, the state transitions of an FSM do not exhibit a simple, repetitive pattern. Its next-state logic is usually constructed from scratch and is sometimes known as "random" logic. This is different from the next-state logic of a regular sequential circuit, which is composed mostly of "structured components, such as incrementors and shifters. In practice, the main application of an FSM is to act as the controller of a large digital system, which examines the external commands and status and activates proper control signals to control operation of a data path, which is usually composed of regular sequential components. This is known as an FSMD (finite state machine with data path).  
The basic block diagram of an FSM is shown in Figure1 below. It consists of a state register, next-state logic, and output logic. An FSM is known as a Moore machine if the output is only a function of state, and is known as a Mealy machine if the output is a function of state and external input. Both types of output may exist in a complex FSM, and we simply refer to it as containing a Moore output and a Mealy output. The Moore and Mealy outputs are similar but not identical. Understanding their subtle differences is the key for controller design.

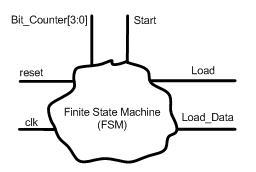


**Figure1.3: Block Diagram of FSM**                  
  
  
 In this code the UART Transmitter has been implemented based on Moore State Machine. The state diagram of UART Transmitter is shown in the Figure 2.

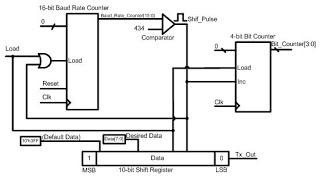


**Figure 1.4: State Diagram**

The state machine and data path for UART Transmitter are shown in Figure 1.5(a) and Figure 1.5(b) respectively.



**Figure 1.5(a) : State Machine  for UART Transmitter**



**Figure 1.5(b): Data path for UART Transmitter**

Though this implementation is not doing much except for continuously sending a character 'U' to the serial port of PC,however,this code is actually a demonstration of how a UART Transmitter can be implemented on FPGA using Verilog HDL.

**1.5 SPARTAN-3E FPGA**

Xilinx has two main FPGA families: the high-performance Virtex series and the high-volume Spartan series. The Virtex series of FPGAs have integrated features such as wired and wireless infrastructure equipment, advanced medical equipment, test and

measurement, and defense systems. The Spartan series targets applications with a low-power footprint, extreme cost sensitivity and high-volume such as displays, set-top boxes, wireless routers and other applications. The Spartan-6 family is built on a 45-

nanometer (nm), 9-metal layer, and dual-oxide process technology. It is a low-cost solution for automotive, wireless communications, flat-panel display and video surveillance applications. The Spartan-3A consumes more than 70-90 percent less power in suspend mode and 40-50 percent less for static power compared to standard devices.

In addition, the integration of dedicated DSP circuitry in theSpartan series has inherent power advantages of approximately25 percent over competing low-power FPGAs. Of these two at present, the Spartan-3e of Spartan series is used to implement the

Multi-Channel UART Controller. The Spartan-3E family of Field-Programmable Gate Arrays(FPGAs) is specifically designed to meet the needs of high volume,cost-sensitive consumer electronic applications. The Spartan-3E family builds on the success of the earlier Spartan-3 family byincreasing the amount of logic per I/O, significantly reducing the cost per logic cell [1]. New features improve system performance and reduce the cost of configuration.

**Features of Spatarn-3E**

1. Very low cost, high-performance logic solution for highvolume,consumer-oriented applications

2. Proven advanced 90-nanometer process technology, Multivoltage,multi-standard Select IO™ interface pins

3. Enhanced Double Data Rate (DDR) support, Abundant,flexible logic resources

4. Efficient wide multiplexers, wide logic, Fast look-ahead carrylogic

5. Eight global clocks plus eight additional clocks per each half of device, plus abundant low-skew routing Configuration interface to industry-standard PROMs

6. Fully compliant 32-/64-bit 33 MHz PCI support (66MHz in some devices)

**2. History of UART**

Some early telegraph schemes used variable-length pulses (as in Morse code) and rotating clockwork mechanisms to transmit alphabetic characters. The first UART-like devices (with fixed-length pulses) were rotating mechanical switches (*commutators*). Various character codes using 5, 6, 7, or 8 data bits became common in teleprinters and later as computer peripherals. Gordon Bell designed the UART for the PDP series of computers. The teletypewriter made an excellent general-purpose I/O device for a small computer. To reduce costs, including wiring and back-plane costs, these computers also pioneered flow control using XON and XOFF characters rather than hardware wires.

Western Digital made the first single-chip UART WD1402A around 1971; this was an early example of a medium scale integrated circuit. Another popular chip was a SCN2651 from the Signetics 2650 family.

An example of an early 1980s UART was the National Semiconductor 8250. In the 1990s, newer UARTs were developed with on-chip buffers. This allowed higher transmission speed without data loss and without requiring such frequent attention from the computer. For example, the popular National Semiconductor 16550 has a 16 byte FIFO, and spawned many variants, including the *16C550, 16C650, 16C750, and 16C850*.

Depending on the manufacturer, different terms are used to identify devices that perform the UART functions. Intel called their 8251 device a "Programmable Communication Interface". MOS Technology 6551 was known under the name "Asynchronous Communications Interface Adapter" (ACIA). The term "Serial Communications Interface" (SCI) was first used at Motorola around 1975 to refer to their start-stop asynchronous serial interface device, which others were calling a UART. Zilog manufactured a number of Serial Communication Controllers or SCCs.

**3. Data transmission using UART**

The CPU communicates with the UART by reading or writing one of eight bytes called *ports*. A computer system normally has more than one UART, so the port addresses depend on the particular UART being accessed. Each UART is associated with a different *base address*, and a particular port is specified by adding a specific *index* to that base address. The index for a particular port is independent of the UART, so we can characterize the ports by indices 0 through 7.

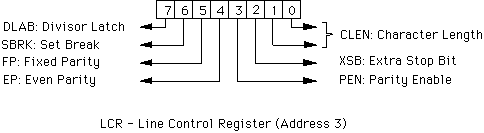
Some of the UART ports can only be read, others can only be written, and both accesses are possible on some. Even when both accesses are allowed, however, they may be unrelated. For example, the UART has a data-in buffer register and a data-out buffer register. Both of these registers, each of which holds one byte, are accessed via port 0. If the UART has assembled a byte from the bits it has received, then the CPU can get that byte by reading port 0. Similarly, if the UART has completely disposed of a byte then the CPU can output another byte by writing it to port 0. Notice that reading and writing are totally unrelated -- if the CPU writes a byte to port 0 and then immediately reads from that port, it will *not* get the byte it wrote.

## 3.1 Data Transmission Control

The controllable characteristics of the data transmission are:

* Baud rate
* Number of information bits per character
* Type of parity checking
* Number of stop bits
* Breaking the transmission

These characteristics are controlled by the line control register (LCR, accessed via port 3).



**Figure 3.1: LCR**

**Baud rate**

The baud rate is established by storing the value 115200/(baud rate) into a 16-bit register inside the UART. When the DLAB bit of LCR is 1, the least-significant byte of this register can be accessed via port 0 (using symbol DLL), and the most-significant byte can be accessed via port 1 (using symbol DLM). When DLAB bit of LCR is0, the data-in and data-out buffer registers are accessed via port 0 (using symbols RBR and THR as discussed above) and the interrupt enable register IER is accessed via port 1.

**Number of information bits per character**

CLEN encodes the number of information bits in each character:

* CLEN=0 for 5 information bits
* CLEN=1 for 6 information bits
* CLEN=2 for 7 information bits
* CLEN=3 for 8 information bits

**Type of parity checking**

If PEN=0 then the UART neither generates a parity bit for outgoing characters nor checks parity on incoming characters. If PEN=1 and FP=0 then even parity is generated and checked if EP=1, and odd parity is generated if EP=0. Finally, if PEN=1 and FP=1 then the generated parity bit is equal to EP, and the parity bit of an incoming character must be equal to EP.

**Number of stop bits**

XSB specifies the number of stop bits transmitted with each serial character. If XSB=0 then one stop bit is generated in the transmitted data. Otherwise 1.5 stop bits are generated for characters with 5 information bits and 2 stop bits are generated for all other characters. The receiver checks the first stop bit only, regardless of the number of stop bits selected.

**Breaking the transmission**

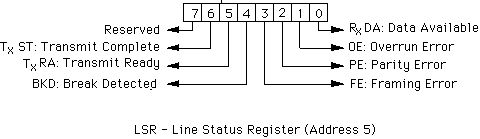
When SBRK=1, the serial output line is held in the spacing state; no characters are transmitted. This condition is detected by the UART at the other end of the line, and causes BKD=1 in the line status register port 5 of that UART.

**3.2 Data Transmission Status**

The status of the data transmission involves

* Availability of an input character
* Completion of character output
* Errors
* Break detection

This status is reported in the line status register (LSR, accessed via port 5). All bits in this register are reset to 0 when the register is read, except as noted below.



**Figure 3.2: LSR**

**Availability of an input character**

RxDA=1 when a character is available in the data-in buffer register.

**Completion of character output**

TxRA=1 when the data-out buffer register does *not* contain a character, and TxST=1 when transmission of all characters is complete. These bits remain 1 if no transmission is in progress when the line status register is read.

**Errors**

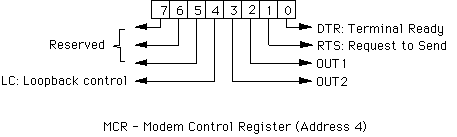
PE=1 indicates a parity error, OE=1 an overrun error, and FE=1 a framing error.

**Break detection**

BKD=1 if the input line has been held in a spacing condition for two or more character times.

**3.3 Modem Handshaking Control**

An protocol called RS-232C describes how a computer and a modem should interact to ensure that they agree on who is ready to do what. The modem control register (MCR, accessed via port 4), is used to control the outgoing signals used for this protocol. It also allows the machine to perform self-tests without using the modem at all.



**Figure 3.3: MCR**

**Handshaking protocol**

The computer should set DTR=1 when it is ready for communication. (Setting DTR=0, for example, will cause a modem to hang up the telephone line at the end of a transmission.)

RTS=1 indicates that the computer desires to transmit information. Setting RTS=0 would indicate that the modem should turn the line around when using half-duplex mode. For full-duplex mode, RTS should be permanently 1.

**Loopback control**

It is useful to be able to test communication software without having to have another device to communicate with. If LC=1 then transmitted characters are sent directly to the receiver -- they ``loop back'' within the UART itself. When LC=0, characters are transmitted normally over the serial line.

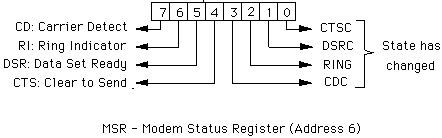
**Miscellaneous control bits**

OUT1 is used only by specialized hardware (like the Hayes SmartModem internal board). The meaning of this bit is determined by the hardware using it. For example, if the computer has a Hayes SmartModem board then OUT1=1 resets the modem.

OUT2 controls interrupt servicing for the UART: OUT2=0 blocks the interrupt generated by the UART. This allows the user to prevent UART interrupts from reaching the computer's priority interrupt controller, while still generating them within the UART.

**3.4 Modem Handshaking Status**

The modem status register (MSR, accessed via port 6), is used to sense incoming signals used for the handshaking RS-232C protocol. All bits in this register are reset to 0 when the register is read, except as noted below.



**Figure 3.4: MSR**

**Current State**

DSR=1 indicates that the modem is ready for communication. This bit remains 1 after the register is read if the modem remains ready.

CTS=1 indicates that the computer is allowed to transmit information. In half-duplex mode (Section 9-1), the modem responds to a signal RTS=1 from the computer by turning the line around and then setting CTS=1. This bit remains 1 after the register is read if the computer is still allowed to transmit information.

CD=1 if the modem believes that there actually is an incoming signal. For example, if a computer is communicating with a remote terminal over a telephone line, and the modem detects that the other party has hung up, it will set CD=0. This bit remains 1 after the register is read if the modem still believes that there actually is an incoming signal.

The modem sets RI=1 when it detects a ringing signal on the telephone line. Thus RI=1 is a request for service from a remote site.

**State Changes**

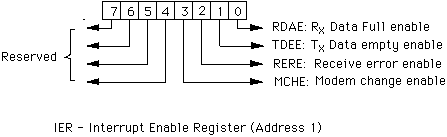
The remaining bits in the register are set to 1 when a state change is detected. Each of these bits is associated with one of the state bits mentioned above, and becomes 1 when that state bit changes. Thus these bits allow the computer to determine which state bits have changed since the last time the register was read, without having to keep the old values and compare them.

**3.5 Interrupt Behavior**

The UART is capable of generating an interrupt when any one of a number of situations arises. These situations are grouped into four classes:

1. Receive machine error or break condition
2. Receive data register full
3. Transmit data register empty
4. Change in the state of the modem input pins

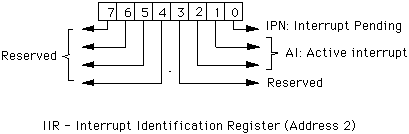
Interrupts in each of these classes must be explicitly enabled by writing a 1 to the appropriate bit in the interrupt enable register (IER, accessed via port 1).



**Figure 3.5: IER**

In addition to enabling the particular kind of interrupt, interrupt servicing for the UART must be requested by setting bit OUT2 in the modem control register (MCR, accessed via port 4).

If an interrupt occurs, the class of interrupt can be determined by reading the interrupt identification register (IIR, accessed via port 2).



**Figure 3.6: IIR**

IPN=0 if an interrupt is pending. Thus the AI field of the interrupt identification register is relevant only if IPN=0.

The interrupt classes are prioritized, with AI=0 being the lowest and AI=3 the highest priority:

* AI=3 for a receive machine error or break condition
* AI=2 when received data is available
* AI=1 when the transmit register is empty
* AI=0 when there is a change in the state of the modem input pins

If the interrupt identification register indicates that there was a receive machine error or break condition then the program must examine the line status register to determine the precise condition. Similarly, if the interrupt identification register indicates that there was a change in the state of the modem input pins the program must determine the change by reading the modem status register.

Only the highest priority interrupt appears in the interrupt identification register at any time. After dealing with a UART interrupt, therefore, the program must read the interrupt identification register again in order to determine whether a lower-priority interrupt also occurred. This cycle stops when IPN=1.

**4 .The RS232-C and V.24 Standards**

### In most computer systems, the UART is connected to circuitry that generates signals that comply with the EIA RS232-C specification. There is also a CCITT standard named V.24 that mirrors the specifications included in RS232-C.

#### 4.1. RS232-C Bit Assignments (Marks and Spaces)

In RS232-C, a value of 1 is called a Mark and a value of 0 is called a Space. When a communication line is idle, the line is said to be “Marking”, or transmitting continuous 1 values.

The Start bit always has a value of 0 (a Space). The Stop Bit always has a value of 1 (a Mark). This means that there will always be a Mark (1) to Space (0) transition on the line at the start of every word, even when multiple word are transmitted back to back. This guarantees that sender and receiver can resynchronize their clocks regardless of the content of the data bits that are being transmitted.

The idle time between Stop and Start bits does not have to be an exact multiple (including zero) of the bit rate of the communication link, but most UARTs are designed this way for simplicity.

In RS232-C, the "Marking" signal (a 1) is represented by a voltage between -2 VDC and -12 VDC, and a "Spacing" signal (a 0) is represented by a voltage between 0 and +12 VDC. The transmitter is supposed to send +12 VDC or -12 VDC, and the receiver is supposed to allow for some voltage loss in long cables. Some transmitters in low power devices (like portable computers) sometimes use only +5 VDC and -5 VDC, but these values are still acceptable to a RS232-C receiver, provided that the cable lengths are short.

#### 4.2. RS232-C Break Signal

RS232-C also specifies a signal called a Break, which is caused by sending continuous Spacing values (no Start or Stop bits). When there is no electricity present on the data circuit, the line is considered to be sending Break.

The Break signal must be of a duration longer than the time it takes to send a complete byte plus Start, Stop and Parity bits. Most UARTs can distinguish between a Framing Error and a Break, but if the UART cannot do this, the Framing Error detection can be used to identify Breaks.

In the days of teleprinters, when numerous printers around the country were wired in series (such as news services), any unit could cause a Break by temporarily opening the entire circuit so that no current flowed. This was used to allow a location with urgent news to interrupt some other location that was currently sending information.

In modern systems there are two types of Break signals. If the Break is longer than 1.6 seconds, it is considered a "Modem Break", and some modems can be programmed to terminate the conversation and go on-hook or enter the modems' command mode when the modem detects this signal. If the Break is smaller than 1.6 seconds, it signifies a Data Break and it is up to the remote computer to respond to this signal. Sometimes this form of Break is used as an Attention or Interrupt signal and sometimes is accepted as a substitute for the ASCII CONTROL-C character.

Marks and Spaces are also equivalent to “Holes” and “No Holes” in paper tape systems.

### Note:

Breaks cannot be generated from paper tape or from any other byte value, since bytes are always sent with Start and Stop bit. The UART is usually capable of generating the continuous Spacing signal in response to a special command from the host processor.

#### 4.3. RS232-C DTE and DCE Devices

The RS232-C specification defines two types of equipment: the Data Terminal Equipment (DTE) and the Data Carrier Equipment (DCE). Usually, the DTE device is the terminal (or computer), and the DCE is a modem. Across the phone line at the other end of a conversation, the receiving modem is also a DCE device and the computer that is connected to that modem is a DTE device. The DCE device receives signals on the pins that the DTE device transmits on, and vice versa.

When two devices that are both DTE or both DCE must be connected together without a modem or a similar media translator between them, a NULL modem must be used. The NULL modem electrically re-arranges the cabling so that the transmitter output is connected to the receiver input on the other device, and vice versa. Similar translations are performed on all of the control signals so that each device will see what it thinks are DCE (or DTE) signals from the other device.

The number of signals generated by the DTE and DCE devices are not symmetrical. The DTE device generates fewer signals for the DCE device than the DTE device receives from the DCE.

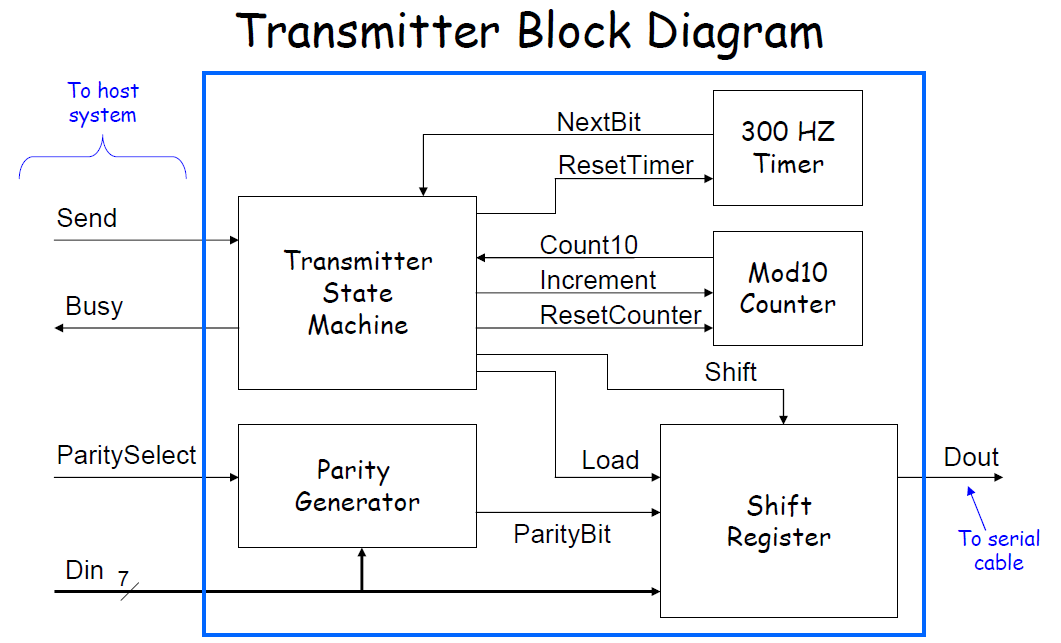
#### 4.4. RS232-C Pin Assignments

The EIA RS232-C specification (and the ITU equivalent, V.24) calls for a twenty-five pin connector (usually a DB25) and defines the purpose of most of the pins in that connector.

In the IBM Personal Computer and similar systems, a subset of RS232-C signals is provided via nine pin connectors (DB9). The signals that are not included on the PC connector deal mainly with synchronous operation, and this transmission mode is not supported by the UART that IBM selected for use in the IBM PC.

Depending on the computer manufacturer, a DB25, a DB9, or both types of connector may be used for RS232-C communications. (The IBM PC also uses a DB25 connector for the parallel printer interface which causes some confusion).

**5. Block Diagram of UART of UART as Transmitter**

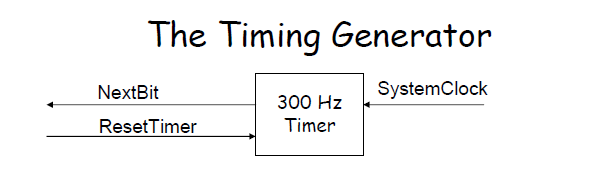
****

**Figure 5.1: Block Diagram of UART as Transmitter**

**5.1 Timing Generator**

Itdivides system clock down to 300 Hz. Output of the timing generator is NextBit signal to state machine. It goes high for one system clock cycle 300 times a second.

It is simply a Mod (fclk/300) resettable counter where NextBit is the rollover signal. More sophisticated UARTs have programmable timing generators for different baud rates.

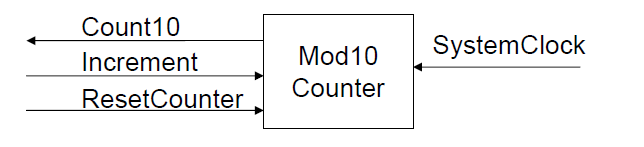


**Figure 5.2: The Timing Generator**

**5.2 Mod10 Counter**

Mod10 Counter resets to 0 on command from state machine and increments on command from state machine. It counts from 0 to 9, then rolls over to 0 .

Mod10 counter informs state machine when it’s going to roll over from 9 back to 0 (signal Count10).

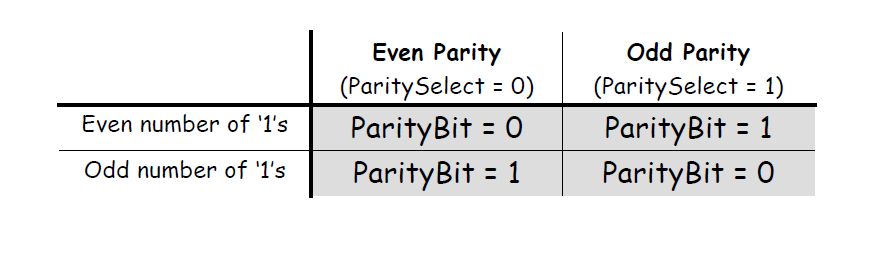


**Figure 5.3: Mod10 Counter**

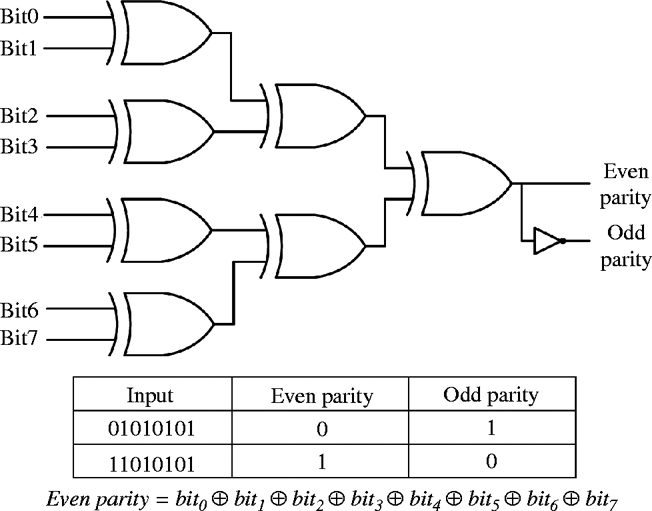
**5.3 Parity Generator**

It is a combinational circuit. Parity Generator generates ParityBit according to value of Din[6:0] and ParitySelect input.

The value of ParityBit is the bit needed to make the number of 1’s even (if even parity) or odd (if odd parity).



**Figure 5.4 ParityBit Table**



**Figure 5.5: 8 bit Parity Generator**

**5.4 Shift Register**

We use a Standard Parallel-In/Serial-Out (PISO) shift register.

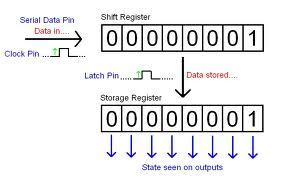
It has 4 operations:

– Do nothing

– Load parallel data from Din

– Shift right

– Reset



**Figure 5.6: Shift register**

• When it loads:

– Have it load ‘0’ for the start bit on the right (LSB)

– Have it load the parity bit on the left (MSB)

– Have it load 7 data bits in the middle

• When it shifts:

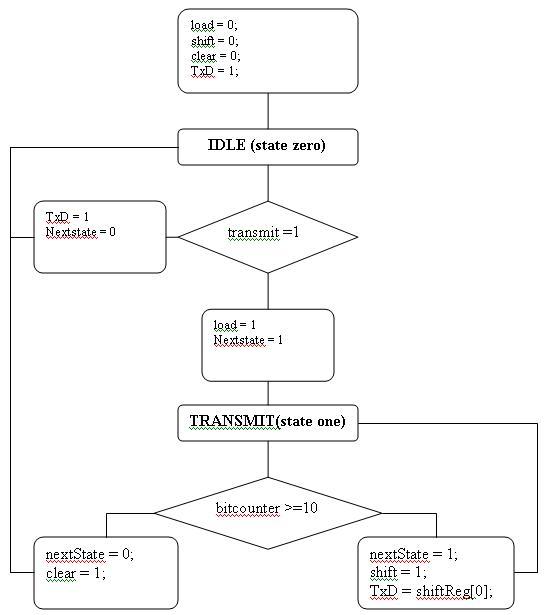
– Have it shift ‘1’ into the left so a stop bit is sent at the end

• When it resets:

– Have it load all 1’s so that its default output is a ‘1’ (line idle value)

**5.6. State Diagrams**

Here is the state diagram. Anything to the left of an if statement ( the diamond shaped box) means TRUE and to the right or bottom of the if statement means FALSE.



**Figure 5.6: State Diagram of UART as Transmitter**

The transmitter starts in the zero state, IDLE. Of course, the transmitter cannot begin in the transmitting state. This is done by setting your initial state register value to zero. So what is happening in these boxes?

The first box shows what happens when the program first starts running. Because no other boxes below reference back to it, this will only happen when your program starts. Load, shift and clear are set to zero. This is because we do not want to shift the shiftregister when the program is idling ( not transmitting) and send the wrong information over the serial line. We also do not want to set the bitcounter to zero as it will not let us increase it when we are transmitting, and we do not have any data to load from just yet.

In UART communications, the line is held high to show that nothing is being transmitted . When the line goes low, both receiver and transmitter know that transmission is about to begin. This is why our start bit is always equal to zero. When communication is ready to end, the line goes high again and the transmission stops. This is why the end bit is always equal to one.

The diagram tells us the following:

We move to the IDLE state and see if the input transmits is 1. This input can be as simple as a physical switch we press to tell the program we are going to transmit. If the transmit is zero, we go back to the IDLE state and keep waiting until that switch is pressed. If it is pressed, then transmit is one and the program moves to the next state, TRANSMIT.

Transmit just makes sure that the bitcounter is not over 11. Remember, we have 11 bits, the start bit, 8 data bits, 1 parity bit and a stop bit. When the bitcounter reaches 11, it means that it has completed its task of transmitting. Shift is left high until then.We can see that by the left statement of if (bitcounter>=10).

Remember in Verilog we count the zero state, so 10 actually means 11. When we reached our goal of an 11 bit transmission, the clear must go high; so to clear the bitcounter back to zero and get ready for the next transmission. Also the next state must go back to IDLE so that it can wait for the next time you push the button to transmit data again.

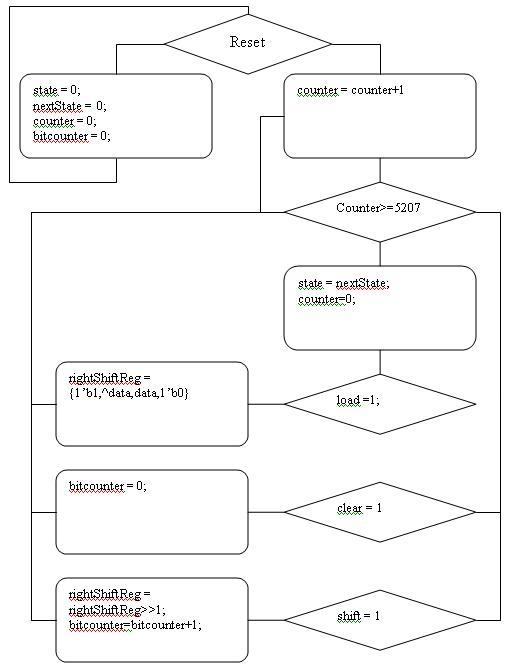
If bitcounter is not greater or equal to 10, naturally we must stay in TRANSMIT mode and therefore the nextState will be TRANSMIT (1 in this case). Also, we must put shift to 1 because we want to shift our shiftregister so the next bit can be shifted onto our TxD output pin to be transmitted. This is done in           TxD = shiftreg[0].

NOTE - Transmission begins from the least significant and as bits are shifted, one by one is sent until the most significant. This means that the receiver must account for this and know how to re-arange the data so that it is received in the correct order.

The state stays in TRANSMIT until it is equal to 10 and goes back to IDLE.

But wait.... where is the counter we need and all that other stuff? We are not done yet!

This is just a state diagram of what we are supposed to do. Everything must run from the FPGA clock, therefore another "state diagram" must be created that lets us run the program from the clock and perform all of this smoothly. So here it goes:



**Figure 5.7: State Diagram of UART with clock**

First the system is reset. This can be a physical switch that sets everything to factory settings. This is in the case the program deadlocks you can have the option to reset the system without reprogramming the FPGA. After this, counter starts counting at each positive edge of the FPGA clock.

When the counter reaches 5208, it is ready to do something. 5208 is the number calculated to get a 9600 baud rate depending on our FPGA clock speed. In other words, the counter is used to slow down the 50MHZ FPGA clock for our program to work at 9600 baud. Also, above says 5027 plus zero state = 5208.

At each time that the counter reaches 5208 we set state to nextState. NextState is always changed in the above state diagram, Figure 2. The reason for doing this is to make the program synchronous to the clock ( or counter). Therefore the transition from IDLE to TRANSMIT and vise versa only happens at each 5208 count of the counter.

Counter is also reset each time it reaches 5208 so that it can start to count up to 5208 again and do the same thing over and over again.

**Note** - this state diagram shows if statements for shift and clear unconnected. Actually shift, clear and load are connected to the block statement that load is connected to so they run simultaneously.

Saying this, At each time the counter reaches 5208 it will check is load =1? shift = 1? clear = 1?

If load = 1 the right shift register loads the data that is to be transmitted. This is why, once the state goes from IDLE to TRANSMIT, load goes high. It ensures that data is loaded into the register and is ready to be transmitted.

When shift is high, rightshiftreg = rightshiftreg>>1. This is a shift operator in verilog and shifts the data right one time. Bitcounter is also incremented once. This tells the program that one bit was already sent.

When clear is high, we have finished transmitting and bitcounter goes back to zero; waiting for another new transmission.

**6.RESULTS**

**Summary report:**

Release 10.1 - xst K.31 (nt)

Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

--> Parameter TMPDIR set to C:/uartproject/xst/projnav.tmp

Total REAL time to Xst completion: 1.00 secs

Total CPU time to Xst completion: 0.08 secs

--> Parameter xsthdpdir set to C:/uartproject/xst

Total REAL time to Xst completion: 1.00 secs

Total CPU time to Xst completion: 0.08 secs

--> Reading design: uartmain.prj

TABLE OF CONTENTS

1) Synthesis Options Summary

2) HDL Compilation

3) Design Hierarchy Analysis

4) HDL Analysis

5) HDL Synthesis

5.1) HDL Synthesis Report

6) Advanced HDL Synthesis

6.1) Advanced HDL Synthesis Report

7) Low Level Synthesis

8) Partition Report

9) Final Report

9.1) Device utilization summary

9.2) Partition Resource Summary

9.3) TIMING REPORT

=========================================================================

\* Synthesis Options Summary \*

=========================================================================

---- Source Parameters

Input File Name : "uartmain.prj"

Input Format : mixed

Ignore Synthesis Constraint File : NO

---- Target Parameters

Output File Name : "uartmain"

Output Format : NGC

Target Device : xc3s1200e-5-fg400

---- Source Options

Top Module Name : uartmain

Automatic FSM Extraction : YES

FSM Encoding Algorithm : Auto

Safe Implementation : No

FSM Style : lut

RAM Extraction : Yes

RAM Style : Auto

ROM Extraction : Yes

Mux Style : Auto

Decoder Extraction : YES

Priority Encoder Extraction : YES

Shift Register Extraction : YES

Logical Shifter Extraction : YES

XOR Collapsing : YES

ROM Style : Auto

Mux Extraction : YES

Resource Sharing : YES

Asynchronous To Synchronous : NO

Multiplier Style : auto

Automatic Register Balancing : No

---- Target Options

Add IO Buffers : YES

Global Maximum Fanout : 500

Add Generic Clock Buffer(BUFG) : 24

Register Duplication : YES

Slice Packing : YES

Optimize Instantiated Primitives : NO

Use Clock Enable : Yes

Use Synchronous Set : Yes

Use Synchronous Reset : Yes

Pack IO Registers into IOBs : auto

Equivalent register Removal : YES

---- General Options

Optimization Goal : Speed

Optimization Effort : 1

Library Search Order : uartmain.lso

Keep Hierarchy : NO

Netlist Hierarchy : as\_optimized

RTL Output : Yes

Global Optimization : AllClockNets

Read Cores : YES

Write Timing Constraints : NO

Cross Clock Analysis : NO

Hierarchy Separator : /

Bus Delimiter : <>

Case Specifier : maintain

Slice Utilization Ratio : 100

BRAM Utilization Ratio : 100

Verilog 2001 : YES

Auto BRAM Packing : NO

Slice Utilization Ratio Delta : 5

=========================================================================

=========================================================================

\* HDL Compilation \*

=========================================================================

Compiling verilog file "/../Documents and Settings/VNR/Desktop/timer.v" in library work

Compiling verilog file "/../Documents and Settings/VNR/Desktop/statemachine.v" in library work

Module <timer> compiled

Compiling verilog file "/../Documents and Settings/VNR/Desktop/shiftregister.v" in library work

Module <statemachine> compiled

Compiling verilog file "/../Documents and Settings/VNR/Desktop/paritygenerator.v" in library work

Module <shiftregister> compiled

Compiling verilog file "/../Documents and Settings/VNR/Desktop/mod10ctr.v" in library work

Module <paritygen> compiled

Compiling verilog file "/../Documents and Settings/VNR/Desktop/uart.v" in library work

Module <mod10counter> compiled

Module <uartmain> compiled

No errors in compilation

Analysis of file <"uartmain.prj"> succeeded.

=========================================================================

\* Design Hierarchy Analysis \*

=========================================================================

Analyzing hierarchy for module <uartmain> in library <work>.

Analyzing hierarchy for module <paritygen> in library <work>.

Analyzing hierarchy for module <mod10counter> in library <work> with parameters.

n = "1001"

Analyzing hierarchy for module <timer> in library <work>.

Analyzing hierarchy for module <statemachine> in library <work>.

Analyzing hierarchy for module <shiftregister> in library <work>.

=========================================================================

\* HDL Analysis \*

=========================================================================

Analyzing top module <uartmain>.

Module <uartmain> is correct for synthesis.

Analyzing module <paritygen> in library <work>.

Module <paritygen> is correct for synthesis.

Analyzing module <mod10counter> in library <work>.

n = 4'b1001

Module <mod10counter> is correct for synthesis.

Analyzing module <timer> in library <work>.

Module <timer> is correct for synthesis.

Analyzing module <statemachine> in library <work>.

Module <statemachine> is correct for synthesis.

Analyzing module <shiftregister> in library <work>.

Module <shiftregister> is correct for synthesis.

Synthesis report:

=========================================================================

\* HDL Synthesis \*

=========================================================================

Performing bidirectional port resolution...

INFO:Xst:2679 - Register <load> in unit <statemachine> has a constant value of 1 during circuit operation. The register is replaced by logic.

INFO:Xst:2679 - Register <rsttimer> in unit <statemachine> has a constant value of 1 during circuit operation. The register is replaced by logic.

INFO:Xst:2679 - Register <startbit> in unit <shiftregister> has a constant value of 0 during circuit operation. The register is replaced by logic.

INFO:Xst:2679 - Register <stopbit> in unit <shiftregister> has a constant value of 1 during circuit operation. The register is replaced by logic.

Synthesizing Unit <paritygen>.

Related source file is "/../Documents and Settings/VNR/Desktop/paritygenerator.v".

Found 1-bit xor8 for signal <parity>.

Summary:

inferred 1 Xor(s).

Unit <paritygen> synthesized.

Synthesizing Unit <mod10counter>.

Related source file is "/../Documents and Settings/VNR/Desktop/mod10ctr.v".

Found 1-bit register for signal <count10>.

Found 4-bit up counter for signal <q>.

Found 4-bit comparator greatequal for signal <q$cmp\_ge0000> created at line 23.

Summary:

inferred 1 Counter(s).

inferred 1 D-type flip-flop(s).

inferred 1 Comparator(s).

Unit <mod10counter> synthesized.

Synthesizing Unit <timer>.

Related source file is "/../Documents and Settings/VNR/Desktop/timer.v".

Found 1-bit register for signal <nextbit>.

Found 9-bit up counter for signal <q>.

Summary:

inferred 1 Counter(s).

inferred 1 D-type flip-flop(s).

Unit <timer> synthesized.

Synthesizing Unit <statemachine>.

Related source file is "/../Documents and Settings/VNR/Desktop/statemachine.v".

Register <shift> equivalent to <busy> has been removed

Register <start> equivalent to <busy> has been removed

Found 1-bit register for signal <busy>.

Summary:

inferred 1 D-type flip-flop(s).

Unit <statemachine> synthesized.

Synthesizing Unit <shiftregister>.

Related source file is "/../Documents and Settings/VNR/Desktop/shiftregister.v".

Found 1-bit register for signal <dout>.

Found 10-bit register for signal <dreg>.

Summary:

inferred 11 D-type flip-flop(s).

Unit <shiftregister> synthesized.

Synthesizing Unit <uartmain>.

Related source file is "/../Documents and Settings/VNR/Desktop/uart.v".

Unit <uartmain> synthesized.

=========================================================================

HDL Synthesis Report

Macro Statistics

# Counters : 2

4-bit up counter : 1

9-bit up counter : 1

# Registers : 14

1-bit register : 14

# Comparators : 1

4-bit comparator greatequal : 1

# Xors : 1

1-bit xor8 : 1

=========================================================================

=========================================================================

\* Advanced HDL Synthesis \*

=========================================================================

Loading device for application Rf\_Device from file '3s1200e.nph' in environment C:\Xilinx\10.1\ISE.

WARNING:Xst:1710 - FF/Latch <nextbit> (without init value) has a constant value of 0 in block <a3>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:1710 - FF/Latch <busy> (without init value) has a constant value of 0 in block <a4>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:1710 - FF/Latch <dreg\_0> (without init value) has a constant value of 0 in block <a5>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch <dout> (without init value) has a constant value of 0 in block <a5>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:2677 - Node <dreg\_7> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_9> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_8> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_4> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_6> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_5> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_3> of sequential type is unconnected in block <a5>.

WARNING:Xst:2677 - Node <dreg\_2> of sequential type is unconnected in block <a5>.

=========================================================================

Advanced HDL Synthesis Report

Macro Statistics

# Counters : 2

4-bit up counter : 1

9-bit up counter : 1

# Registers : 14

Flip-Flops : 14

# Comparators : 1

4-bit comparator greatequal : 1

# Xors : 1

1-bit xor8 : 1

=========================================================================

=========================================================================

\* Low Level Synthesis \*

=========================================================================

WARNING:Xst:1710 - FF/Latch <a3/nextbit> (without init value) has a constant value of 0 in block <uartmain>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch <a4/busy> (without init value) has a constant value of 0 in block <uartmain>. This FF/Latch will be trimmed during the optimization process.

Optimizing unit <uartmain> ...

Optimizing unit <shiftregister> ...

WARNING:Xst:1710 - FF/Latch <a5/dreg\_0> (without init value) has a constant value of 0 in block <uartmain>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch <a5/dout> (without init value) has a constant value of 0 in block <uartmain>. This FF/Latch will be trimmed during the optimization process.

WARNING:Xst:2677 - Node <a5/dreg\_2> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_3> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_5> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_6> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_4> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_8> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_9> of sequential type is unconnected in block <uartmain>.

WARNING:Xst:2677 - Node <a5/dreg\_7> of sequential type is unconnected in block <uartmain>.

Mapping all equations...

Building and optimizing final netlist ...

Found area constraint ratio of 100 (+ 5) on block uartmain, actual ratio is 0.

Final Macro Processing ...

=========================================================================

Final Register Report

Found no macro

=========================================================================

=========================================================================

\* Partition Report \*

=========================================================================

Partition Implementation Status

-------------------------------

No Partitions were found in this design.

-------------------------------

=========================================================================

\* Final Report \*

=========================================================================

Final Results

RTL Top Level Output File Name : uartmain.ngr

Top Level Output File Name : uartmain

Output Format : NGC

Optimization Goal : Speed

Keep Hierarchy : NO

Design Statistics

# IOs : 13

Cell Usage :

# BELS : 1

# GND : 1

# IO Buffers : 2

# OBUF : 2

=========================================================================

Device utilization summary:

---------------------------

Selected Device : 3s1200efg400-5

Number of Slices: 0 out of 8672 0%

Number of IOs: 13

Number of bonded IOBs: 2 out of 304 0%

---------------------------

Partition Resource Summary:

---------------------------

No Partitions were found in this design.

---------------------------

=========================================================================

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT

GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

------------------

No clock signals found in this design

Asynchronous Control Signals Information:

----------------------------------------

No asynchronous control signals found in this design

Timing Summary:

---------------

Speed Grade: -5

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: No path found

Timing Detail:

--------------

All values displayed in nanoseconds (ns)

=========================================================================

Total REAL time to Xst completion: 4.00 secs

Total CPU time to Xst completion: 3.41 secs

-->

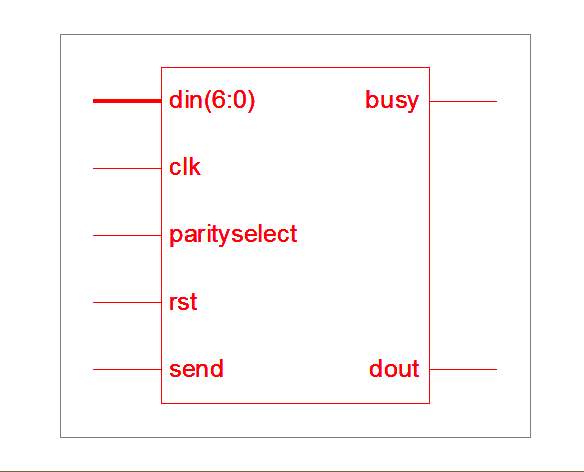
Total memory usage is 165704 kilobytes

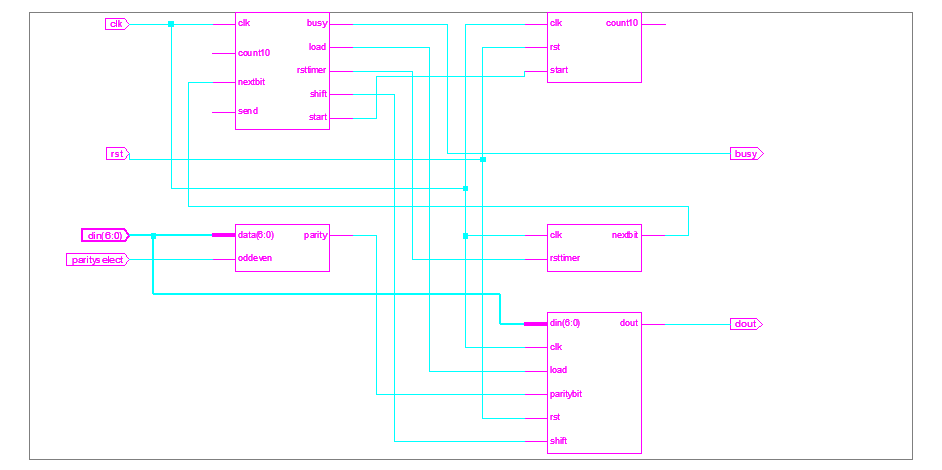
Number of errors : 0 ( 0 filtered)

Number of warnings : 24 ( 0 filtered)

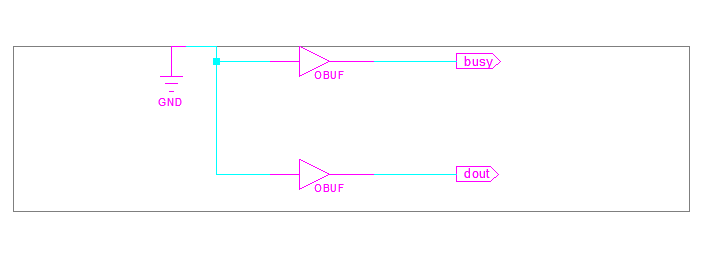
Number of infos : 4 ( 0 filtered)

**RTL SCHEMATIC:**

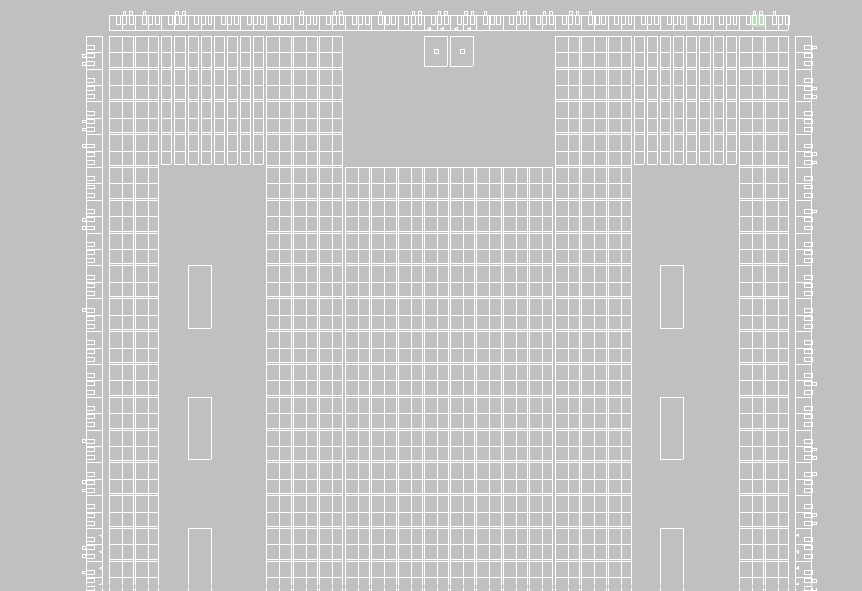


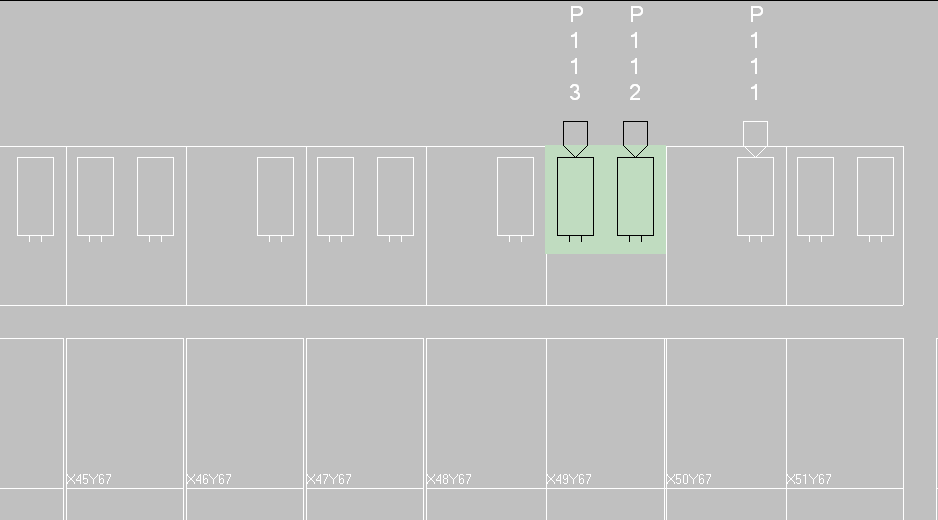


**Uart rtl schematic**

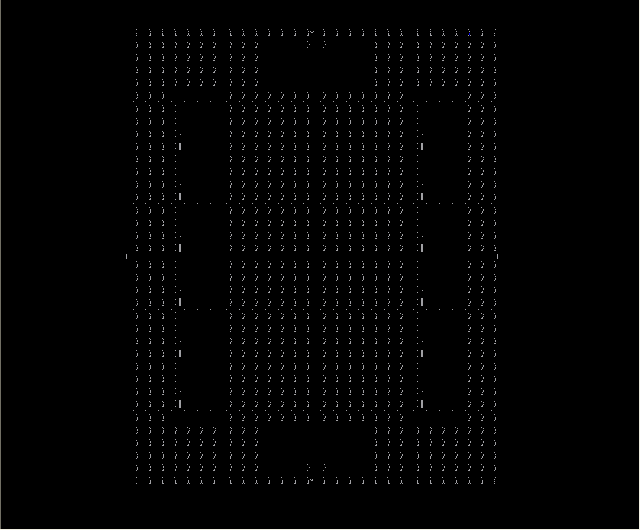


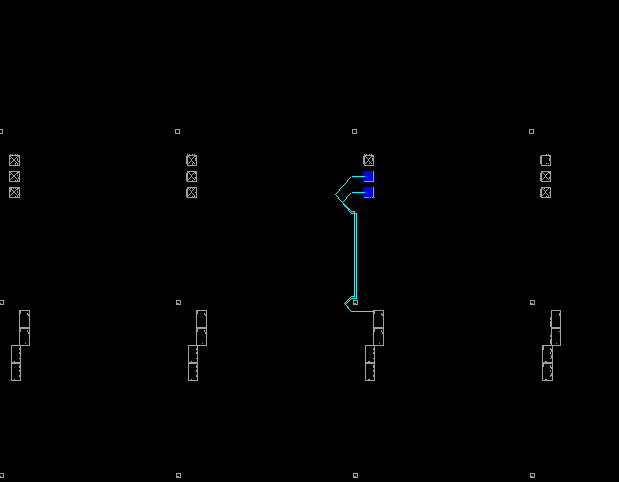
**Technology schematic**



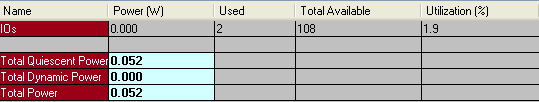


**Placed design(floor planner)**

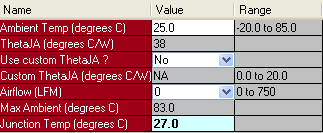




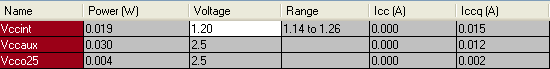
**Routed Design(FPGA editor)**



Xpower analyzer

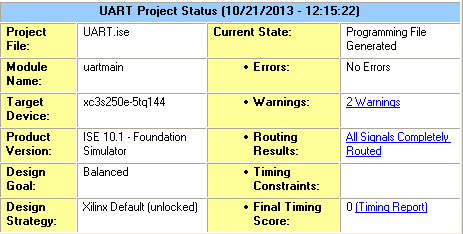


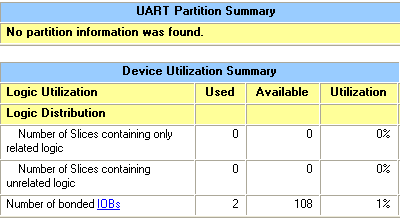
**Thermal information**

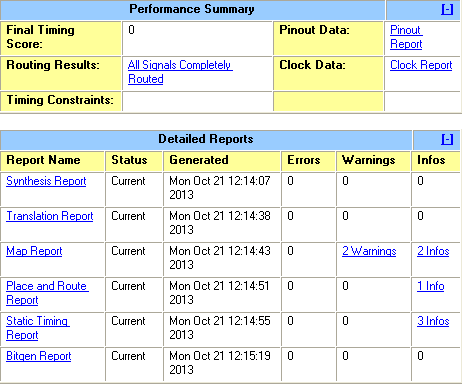


Voltage source information

DESIGN SUMMARY







**7.Modules and output graphs**

**Implementation using Verilog :**

**7.1 Main uart module**

module uart(busy,din,send,parityselect,dout,clk,rst);

input send,clk,rst,parityselect;

input [6:0]din;

output dout,busy;

wire count10,paritybit,load,shift,start,nextbit;

shiftregister a1(clk,rst,shift,load,din,paritybit,dout);

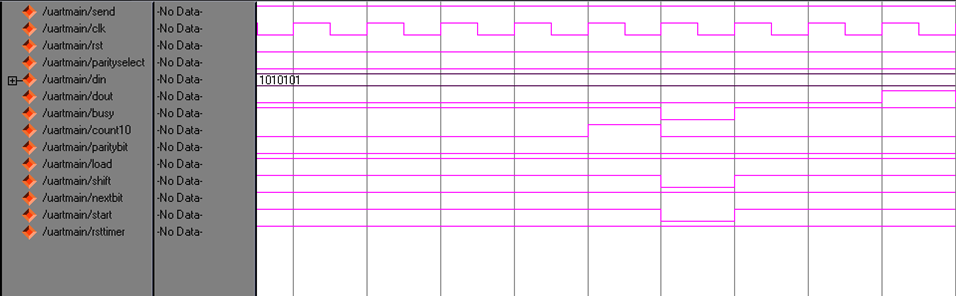
paritygen a2(din,parityselect,paritybit);

mod10counter a3(rst,start,clk,count10);

statemachine a4(send,clk,nextbit,count10,busy,start,load,shift,rsttimer);

timer a5(clk,rsttimer,nextbit);

endmodule



**Figure 7.1: UART main program output waveform**

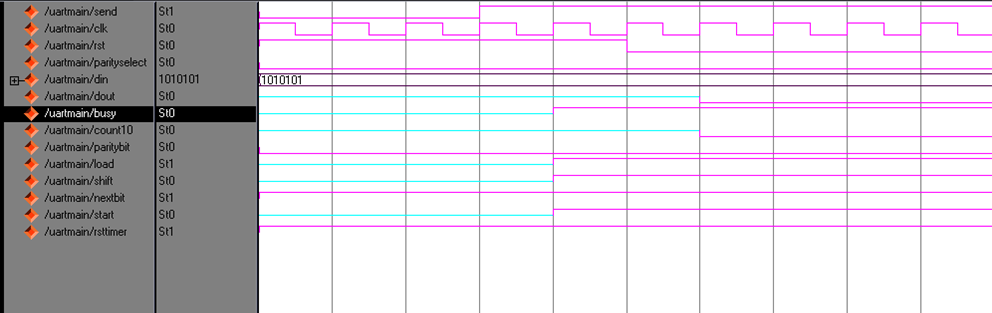


Figure 7.1.1:

**7.2 State Machine**

module statemachine(send,clk,nextbit,count10,busy,start,load,shift,rsttimer);

input send,count10,nextbit,clk;

output busy,load,shift,start,rsttimer;

reg busy,load,shift,start,rsttimer;

always@(posedge clk)

if(nextbit == 1'b1)

begin

if(send == 1'b1)

begin

if(count10 == 1'b1)

begin

busy = 1'b0;

load = 1'b1;

shift = 1'b0;

start = 1'b0;

end

else

begin

busy = 1'b1;

load = 1'b1;

shift= 1'b1;

start =1'b1;

end

end

end

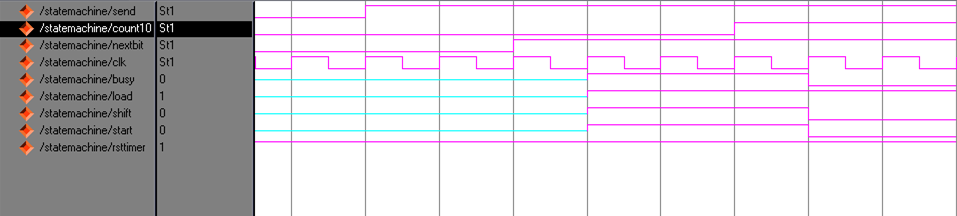
else

begin

rsttimer = 1'b1;

end

endmodule



**Figure 7.2: Output waveforms for State machine**

**7.3 Mod10 Counter**

module mod10counter(rst,start,clk,count10 );

input start,rst,clk;

output count10;

reg [3:0]q;

reg count10;

parameter n=4'b1001;

always@(posedge clk)

if(rst)

begin

q=4'b0000;

end

else

begin

if(start==1'b1)

begin

if(q==4'b1001)

begin

count10 = 1'b1;

q=4'b0000;

end

else

begin

q=(q>=n)?4'b000:q+1'b1;

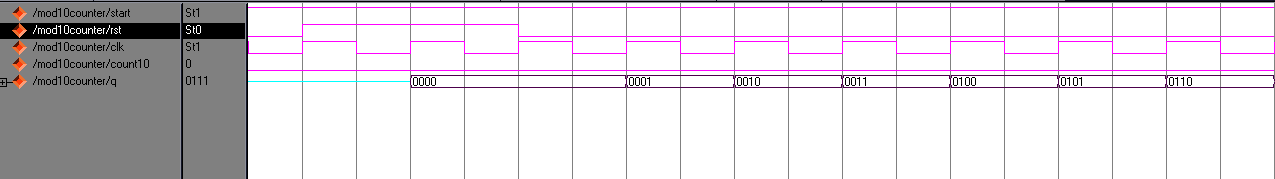
count10 = 1'b0;

end

end

end

endmodule



**Figure 7.3: Output waveforms for Mod10 Counter**

**7.4 Parity Generator**

module paritygen(data, oddeven, parity);

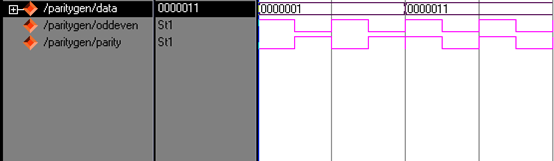
input [6:0] data;

input oddeven;

output parity;

assign parity = (^data) ^ oddeven;

endmodule



**Figure 7.4: Parity Generator output waveforms**

**7.5 Shift Register**

module shiftregister(clk,rst,shift,load,din,parityselect,dout);

input rst,clk,shift,load,parityselect;

input [6:0]din;

output dout;

reg dout;

reg [9:0]dreg;

reg stopbit,startbit;

always@(posedge clk)

if(rst==1'b1)

begin

dreg=4'h0;

startbit= 1'b0;

stopbit = 1'b1;

end

else

begin

if(load==1'b1 & shift == 1'b0)

begin

dreg[7:1] = din;

dreg[9] = stopbit;

dreg[8] = parityselect;

dreg[0] = startbit;

end

else

if(shift == 1'b1)

begin

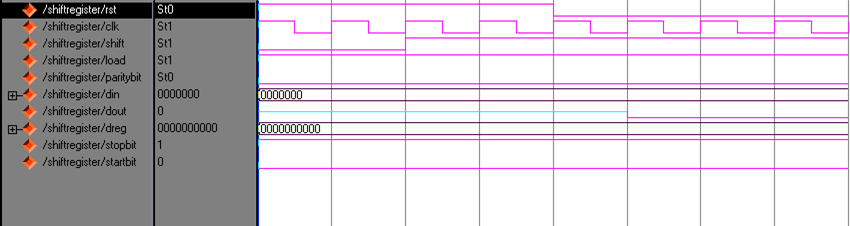
dout = dreg[0];

dreg=dreg>>1;

end

end

endmodule



**Figure 7.5: Shift register output waveforms**

**7.6 Timer**

module timer(clk,rsttimer,nextbit);

input rsttimer,clk;

output nextbit;

reg nextbit;

reg [8:0]q;

always@(posedge clk)

if(rsttimer)

begin

q=9'b000000000;

end

else

begin

if(q==9'b100101100)

begin

nextbit = 1'b0;

end

else

begin

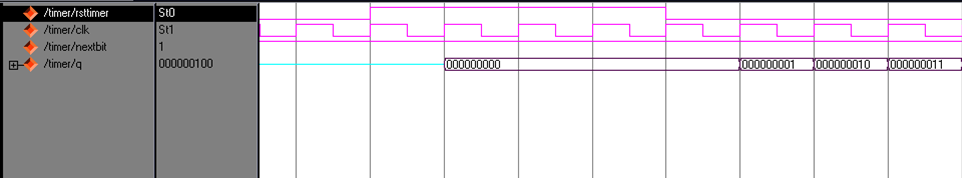
q=q+1;

nextbit = 1'b1;

end

end

endmodule



**Figure 7.6: Timer output waveforms**

**8.Applications**

A UART is a useful component for controlling asynchronous (without a separate clock line) serial buses. It can be used via a level converter to talk to the RS232 serial port of a computer. This is not, however, the only application. It can also be used in a circuit to communicate with peripherals, or over other types of cables (such at RS485 with a differential driver) to connect to other circuits over quite long distances.

Transmitting and receiving UARTs must be set for the same bit speed, character length, parity, and stop bits for proper operation. The receiving UART may detect some mismatched settings and set a "framing error" flag bit for the host system; in exceptional cases the receiving UART will produce an erratic stream of mutilated characters and transfer them to the host system.

Typical serial ports used with personal computers connected to modems use eight data bits, no parity, and one stop bit; for this configuration the number of ASCII characters per second equals the bit rate divided by 10.

Some very low-cost home computers or embedded systems dispensed with a UART and used the CPU to sample the state of an input port or directly manipulate an output port for data transmission. While very CPU-intensive, since the CPU timing was critical, these schemes avoided the purchase of a UART chip. The technique was known as a bit-banging serial port.

**9.Drawbacks**

The word “asynchronous” means that there is not clock which synchronizes the two communicating devices. The advantage is that no additional clock wire between sender/receiver is needed, the disadvantage is that both devices need the same configuration (bit-/baud-rate, control bits) and both devices need to run at the same speed.

The timing dependency is one of the big drawbacks of UART, and the solution is **USART**, for Universal Synchronous/Asynchronous Receiver Transmitter. This can do UART, but also a synchronous protocol. In synchronous there's not only data, but also a clock transmitted. With each bit a clock pulse tells the receiver it should latch that bit. Synchronous protocols either need a higher bandwidth, like in the case of Manchester encoding, or an extra wire for the clock, like SPI and I2C.

**Unreliable** – That’s why it only good for application with repeated transmission and when data integrity is not that critical.

**Slow** – With baud rate of 1200bps, and a packet of data which consists of 10 bytes of data, we only have a useful data rate of 12 bytes per second.

**Unidirectional** – The data can only be sent from the Transmitter to the Receiver but not vice versa.

Nevertheless, the low cost RF module is still capable of handling some basic wireless communication. This can be seen in our daily life application such as central locking system for automobile, remote control for auto gate, wireless doorbell and etc.

**10. Conclusion**

The design of UART as Transmitter is made using Verilog HDL and is verified by using MODEL SIM Tool. This tool helps us to identify whether the design working properly or not.

The design made, have delay code which will be used while implementing it in FPGA board. Only one request will be carried out in one clock cycle period. The timer is used to provide 20 seconds of delay when closing the elevator door. But in our design, timer is working properly in simulation and not synthesized in XILINX Tool. So we consider the timer operation is carried out by using switch.

FPGAs provide very handy prototyping. The implementation of design does not take much time and hence the design can go in some cycles of iterations for optimization. The main design report is updated at every step and individual step reports are also available for analysis.

The UART as Transmitter was successfully implemented, with tuning of the timing constraints to meet the desired targets.Total REAL time to Xst completion is 1.00 secs.Total CPU time to Xst completion is 0.08 secs. Total memory usage is 165704 kilobytes.

FPGAs are built from a programmable fabric of logic cells that emulate the user-defined functionality. These results to increase in the production cost of FPGAs in comparison to that of ASICs. FPGAs are only available in certain sizes. So, in case of just a single gate or block required a higher size of FPGA board is to be taken. For this reason, area minimization my not always be a factor, again leading to new design approaches. But still, FPGAs are finding more and more application in the wide area of semiconductor industry applications, due the advantages of rapid prototyping.

**11. FUTURE SCOPE**

A UART as Transmitter implementation on an FPGA is cost effective. Due to the reprogram-ability feature of FPGA, it is possible to rewrite or modify the code for a UART. If the complete usage of chip area is of concern then it is better

to go for an ASIC where the required amount of chip area is used and it is mainly used, where there is no need to rewrite the code for a specific application. Hence depending upon the application requirements i.e. where there is chip utilization and no need of

reprogram-ability it is better to go for ASIC devices.

Although not required for the ISO 7816 protocol, especially given that only one wire is used for both data transmission directions, extending the algorithm presented in this

Thesis to the generic case of full-duplex operation might be desirable for other protocols.

Serial communication on PC compatibles started with the 8250 UART in the XT. In the years after, new family members were introduced like the 8250A and 8250B revisions and the 16450. The last one was first implemented in the AT. The higher bus speed in this computer could not be reached by the 8250 series. The differences between these first UART series were rather minor. The most important property changed with each new release was the maximum allowed speed at the processor bus side.   
   
 The 16450 was capable of handling a communication speed of 38.4 kbs without problems. The demand for higher speeds led to the development of newer series which would be able to release the main processor from some of its tasks. The main problem with the original series was the need to perform a software action for each single byte to transmit or receive. To overcome this problem, the 16550 was released which contained two on-board FIFO buffers, each capable of storing 16 bytes. One buffer for incoming, and one buffer for outgoing bytes. 

The 16550 chip contained a firmware bug which made it impossible to use the buffers. The 16550A which appeared soon after was the first UART which was able to use its FIFO buffers. This made it possible to increase maximum reliable communication speeds to 115.2 kbs. This speed was necessary to use effectively modems with on-board compression. A further enhancement introduced with the 16550 was the ability to use DMA, direct memory access for the data transfer. Two pins were redefined for this purpose. DMA transfer is not used with most applications. Only special serial I/O boards with a high number of ports contain sometimes the necessary extra circuitry to make this feature work. The 16550A is the most common UART at this moment. Newer versions are under development, including the 16650 which contains two 32 byte FIFO's and on board support for software flow control. Texas Instruments is developing the 16750 which contains 64 byte FIFO's.

**12. BIBLIOGRAPHY**

**Ieee papers:**

* Fang YI-yuan: Coll. Of Electrical and electronics engineering ,Shanghai university of Eng .Sci., Shanghai, China;Chen Xue-jun
* Frank Durda Serial and UART Tutorial. [uhclem@FreeBSD.org](mailto:uhclem@FreeBSD.org)
* Features, design tools and application domains of FPGAs –Rodriguez-Andina J.J; Vigo univ, Vigo;Mourne M.J; Valdes;M.D
* Design of high speed UART –Norhuzaimin, J. ;Dept of electron.,Univ Malasiya Sarawak, Kota Samarahan; Maimun, H.H
* Project based learning experience in VHDL digital electronic circuit design-Machado, F. ; Dept. de Tecnol. Electron., Univ. Rey Juan Carlos, Madrid, Spain; Borromeo, S.; Malpica, N.

**Websites:**

* http://en.wikipedia.org/wiki/Universal\_asynchronous\_receiver/transmitter
* http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter
* http://www.freebsd.org/doc/en/articles/serial-uart/
* http://www.ece301.com/fpga-projects/52-uart-txd.html
* http://www.engr.siu.edu/~haibo/ece428/notes/ece428\_uart.pdf
* http://www.ehow.com/how\_5667684\_create-uart-transmitter-verilog-hdl.html

**Books:**

* HDL Chip design: A practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs Using VHDL or Verilog Doone Publications 1998, Douglas J. Smith
* A verilog HDL Primer, Second edition Star Galaxy Publishing 1999, J. Bhaskar.