Journal of Information Systems and Communication
ISSN: 0976-8742 & E-ISSN: 0976-8750, Volume 3, Issue 1, 2012, pp.- 265-268.
Available online at http://www.bioinfo.in/contents.php?id=45

UART WITH AUTOMATIC BAUD RATE GENERATOR AND FREQUENCY DIVIDER

BHAVNA MAHURE[1]*AND RAHUL TANWAR[2]

[1]Department of Electronics & Communication, JNU, Jaipur, Raj., India
[2]Department of Computer Science & Information Technology, JNIT, Jaipur, Raj., India
*Corresponding Author: Email- bhavnamahure@gmail.com, rahul123tanwar@gmail.com

Received: January 12, 2012; Accepted: February 15, 2012

Abstract- This paper concentrates on developing a serial communication protocol including bus automatic baud rate detection with selection and bit synchronization, frequency division according to the input clock. All modules are designed using Verilog programming language and implemented on Xilinx Spartan-3 FPGA development board.

In the result and simulation part, this paper will focus on baud rate generation at different frequencies and check the receive data with error free. Besides, in the Baud Rate Generator part, the Baud Rate Generator is incorporated into the UART design before the overall design is synthesized. The role of frequency divider here we can use this at those places where we require lower frequency to operate the functionality. This frequency divider will automatically adjusted according to requirements. The simulated waveforms at different frequencies between 150 to 38400 at 50 MHz clock cycle. The simulated waveforms in this paper have proven the reliability of the HDL implementation to describe the characteristics and the architecture of the design UART with baud rate generator.

Keywords- UART, Oversampling, Frequency Divider, Baud Rate, Baud Clock, Super Sampling, Baud Rate Generator

Citation: Bhavna Mahure and Rahul Tanwar (2012) UART with Automatic Baud Rate Generator and Frequency Divider. Journal of Information Systems and Communication, ISSN: 0976-8742 & E-ISSN: 0976-8750, Volume 3, Issue 1, pp.- 265-268.

Introduction

In several control systems, UART a kind of serial communication circuit is used widely. A universal asynchronous receive/transmit (UART) is an integrated circuit which plays the most important role in serial communication. It handles the conversion between serial and parallel data. Serial communication reduces the distortion of a signal, therefore makes data transfer between two systems separated in great distance possible [2]. A Universal Asynchronous Receiver Transmitter includes a transmitter and a receiver. The transmitter is essentially a special shift register that loads data in parallel and then shifts it out bit by bit at a specific rate. The receiver, on the other hand, shifts in data bit by bit and then reassembles the data. UART transmitter controls transmission by fetching a data word in parallel format and directing the UART to transmit it in a serial format. Likewise, the Receiver must detect transmission, receive the data in serial format, strip of the start and stop bits, and store the data word in a parallel format.

Since the UART is asynchronous in working, the receiver does not know when the data will come, so receiver generate local clock in order to synchronize to transmitter whenever start bit is received. Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. The transmitter and receiver agree on timing parameters in advance and special bits are added to each word which is used to synchronize the sending and receiving units [7].

When a word is given to the UART for Asynchronous transmission, a bit called the “Start Bit” is added to the beginning of each word that is to be transmitted. The Star Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. After the Start Bit, the individual bits of the word of data are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits, and the receiver “looks” at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a 1 or a 0. 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. Then at least one Stop Bit is sent by the transmitter. Because asynchronous data is "self synchronous", if there is no data to transmit, the transmission line can be idle [7].

The UART performs serial-to-parallel conversions on data received from a peripheral device and parallel-to-serial conversion on data received from the CPU. The CPU can read the UART status at any time. The UART includes control capability and a processor interrupt system that can be tailored to minimize software management of the communications link. The UART includes a programmable baud generator capable of dividing the UART input clock by divisors from 1 to 65535 and producing a 16 reference clock for the internal transmitter and receiver logic [3].

Implementation Process

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 [1].

Oversampling Procedure

The most commonly used sampling rate is 16 times the baud rate, which means that each serial bit is sampled 16 times. Assume that the communication uses N data bits and M stop bits. The oversampling scheme works as follows:

- Wait until the incoming signal becomes 0, the beginning of the start bit, and then start the sampling tick counter.
- When the counter reaches 7, the incoming signal reaches the middle point of the start bit. Clear the counter to 0 and restart.
- When the counter reaches 15, the incoming signal progresses for one bit and reaches the middle of the first data bit. Retrieve its value, shift it into a register, and restart the counter.
- Repeat step 3 N-1 more times to retrieve the remaining data bits.
- If the optional parity bit is used, repeat step 3 one time to obtain the parity bit.
- Repeat step 3 M more times to obtain the stop bits.

The oversampling scheme basically performs the function of a clock signal. Instead of using the rising edge to indicate when the input signal is valid, it utilizes sampling ticks to estimate the middle point of each bit. While the receiver has no information about the exact onset time of the start bit, the estimation can be off by at most the subsequent data bit retrievals are off by at most & from the middle point as well. Because of the oversampling, the baud rate can be only a small fraction of the system clock rate, and thus this scheme is not appropriate for a high data rate [6].

Role of Automatic Baud Rate Detection

The most commonly used number of data bits of a serial connection is eight, which corresponds to a byte. When a regular ASCII code is used in communication, only seven LSBs are used and the MSB is 0. If the UART is configured as 8 data bits, 1 stop bit, and no parity bit, the received word is in the form of 0-dddd-ddd-0-1, in which d is a data bit and can be 0 or 1. Assume that there is sufficient time between the first word and subsequent transmissions. In this scheme, the transmitting system first sends an ASCII code for rate detection and then resumes normal operation afterward. The receiving subsystem uses the first word to determine a baud rate and then uses this rate for the baud rate generator for the remaining transmission. Assume that the UART configuration is 8 data bits, 1 stop bit, and no parity bit, and the baud rate can be 4800, 9600, or 19,200 baud. The revised UART receiver should have two operation modes. It is initially in the "detection mode" and waits for the first word. After the word is received and the baud rate is determined, the receiver enters "normal mode" and the UART operates in a regular fashion.

The approach we use is to set the receive baud rate to 9600 baud and wait for a character to be received from the terminal. The time taken to transmit each bit, T, depends on the baud rate. Thus, in the time taken to transmit a single bit at 9600 baud, two bits could be transmitted at 19200 baud. Similarly, if two bits are transmitted at 9600 baud, only a single bit can be transmitted in the same time at 4800 baud. In the detection mode a carriage return character (0x0D) is transmitted by the transmitter and this is received at 9600 baud rate. A transmitted character 0x0D will be received differently from the receiver, if the transmitter transmitting the character other then 9600 baud rate. According to the received data the receiver manipulates the transmitter baud rate and set itself to that baud rate [6].

Role of Frequency Divider in UART

The circuit produces Frequency Division as it now divides the input frequency by a factor of two (an octave). A frequency divider is designed using D Flip Flop. D-type Flip-Flop is as a binary divider, for Frequency Division or as a "divide-by-2" counter. Frequency Divider is dividing the frequency according to system requirement. So we can use this UART with frequency divider, no need to attach another device in that system to divide the frequency.

Implementation Language

We use here Verilog Hardware Description Language to implement the UART general functionality. The use of hardware description languages (HDLs) is becoming Increasingly common for designing and verifying FPGA designs. The Register Transfer Level (RTL) model of the UART protocol is developed using HDL and the functional simulation of the model is obtained. The transmission and reception of the various data frames are tested and verified with the modelling of a UART protocol controllers.

Using a hardware description language allows us to describe the function of the transmitter in a more behavioural manner, rather than focus on its actual implementation at the gate level. The Verilog language provides functions and tasks as constructs, analogous to software functions and procedures. A Verilog function and task are used as the equivalent to multiple lines of Verilog code, where certain inputs or signals affect certain outputs or variables [5].

Simulation Results and Discussion

The UART consists of two independent HDL modules. One module implements the transmitter, while the other module implements the receiver. The transmitter and receiver modules can be combined at the top level of the design, for any combination of transmitter and receiver channels required. Data can be written to the transmitter and read out from the receiver, all through a single 8

bit Bi-directional CPU interface. Address mapping for the transmitter and receiver channels can easily be built into the interface at the top level of the design. Both modules share a common master clock called clk. Within each module, clk is divided down to independent baud rate clocks.

UART Main Module

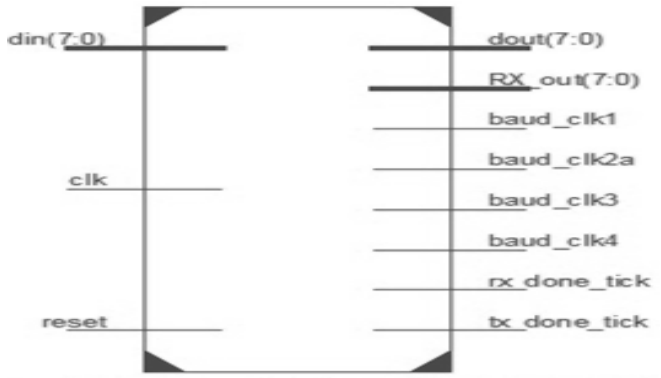

Fig. 1- UART main block

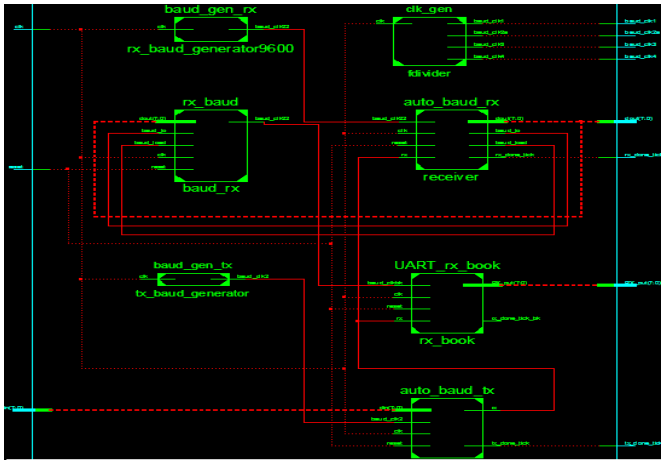

Fig. 2- RTL of UART main module

UART Pin Description

Table 1 - shows the UART functional Pin description

Symbol	Type	Description
Clk	Input	Input clock used for internal baud rate generation.
Reset	Input	Reset input signal
Din(7:0)	Input	Data input pins for transmitting 8 bit data.
Dout(7:0)	Output	Data out pins for receiving 8 bit data.
Rx_done_tick	Output	It indicates receive completed.
Tx_done_tick	Output	Indicates that transmit completed.
Baudclk22	Input	Baud Rate Clock that generates according to input clk for receiver operation.
Rx	Input	Receives serial data. rx should be held high (pulled-up), when no transmission are taking place.
Baud_lo	Output	Active low read strobe signal, used for reading out data from the receiver.
Baud_load	Output	Its 8 bit binary values, to identify transmitter baud rate value.
Thr	Reg[7:0]	8 bit transmit hold register, Used to hold the contents of data bus, when new data is written to the module.
Tsr	Reg[7:0]	8 bit transmit shift register used for shifting out the contents of Tsr.
Baudclk2	Reg	Baud rate clock used to shift the contents of the tsr register to the Tx output.
Txdatardy	Reg	txdatardy is true when transmit hold register holds new data that is ready to be transmitted.

This paper describes the software implementation of baud rate detection of an incoming data. It is used to demonstrate the detection of 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 and 38400bps. This automatic detection is useful for establishing communication link between two devices. The receiver device will be able to detect the baud rate of the transmitter side and adjust accordingly.

This protocol can be implemented on any microcontroller or any microprocessor or UART enable devices that carry an asynchronous serial port with a baud rate generator. If a data is sent based on 9600 Baud, the receiver will receive the same data if it is set at the same Baud rate. This is because the incoming data is sampled correctly, upon the activation of the start bit. The Serial bus is normally in the mark state (high level). When a space (low level) is detected, identify as a Start bit, the incoming serial data will be sampled. In which, the data is sampled on the 8th pulse of a clock with a frequency, 16 times the bit rate (Data latched at the center of bit). So I can say that when the receiver is set to 9600 Baud rate. It will receive (sample) different patterns of data from the transmitter when the transmitter is set to different baud rates that are shown in simulation results.

Here we set the clk at 50MHz frequency. In Figure 3 we have shown the waveform Baud Rate Generate at Transmitter side 38400bps according to clk frequency=50MHz. we can verify it by counts its Divisor value is 81.

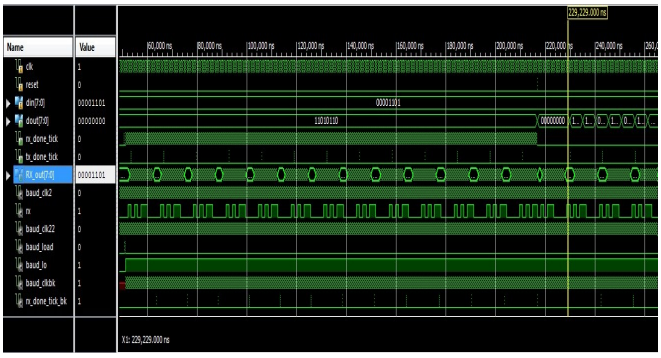

Fig. 3- Simulation waveform of transmitter baud rate generator

In Figure 4 we show the simulate waveform at baud rate=9600. First we set the clk frequency is 50 MHz. then we set the Reset input signal ='1' for loading 8 bit data "00001101" using din[7:0] pin but we want to send this data at 9600 Baud Rate to transmitter so first we have to set Reset='0' to read the data by transmitter. Then tx_baud_generator send the data at 9600bps Baud Rate to receiver. we have already set the Receiver Baudclk at 9600bps. So Receiver receives the 8 bit data "00001101" as "00001101" using dout[7:0] pin at 9600 baud rate. This 8 bit data transmit to Automatic Baud Detection Block, in which both baudclk are matched and we will receive the same data "00001101" at rx_out [7:0] and rx_done_tick signal is set to low which Indicates that new data has been received, and is ready to be read out.

Similarly In Figure 5 we are loading 8 bit data "00001101" using din[7:0] pin but we want to send this data at 2400 Baud Rate to transmitter so first we have to set Reset pin at '0' to read the data by transmitter. Then tx_baud_generator send the data at 2400 Baud Rate to receiver. we have already set the Receiver Baudclk at 9600bps. So Receiver receives the 8 bit data "00001101" as

"01111000" using dout[7:0] pin at 9600 baud rate. This 8 bit data transmit to Automatic Baud Detection Block, in which both baudclk is adjusted and we will receive the same data "00001101" at rx_out[7:0] and rx_done_tick signal is set to low which Indicates that new data has been received, and is ready to be read out.

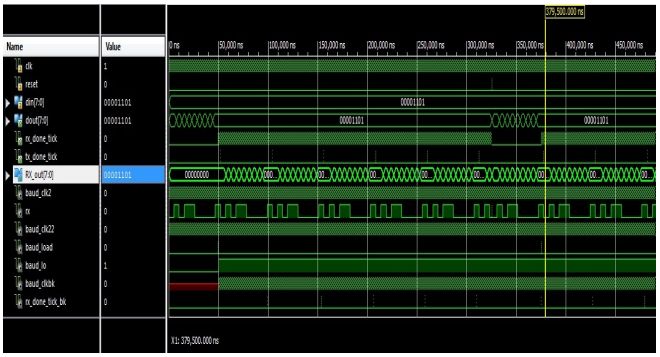

Fig. 4- Simulation waveform of transmitter baud rate at 9600 bps

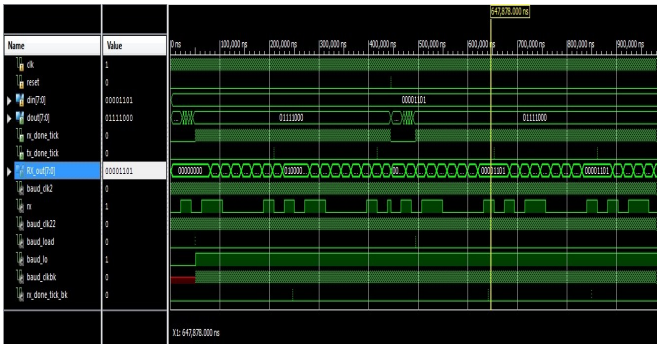

Fig. 5- Simulation waveform of transmitter baud rate at 2400 bps

In the Figure 6 we have designed frequency divider using D Flip Flop. In the above figure 'clk' is used as an input clock, when we apply any input using this clk pin then this frequency divider di

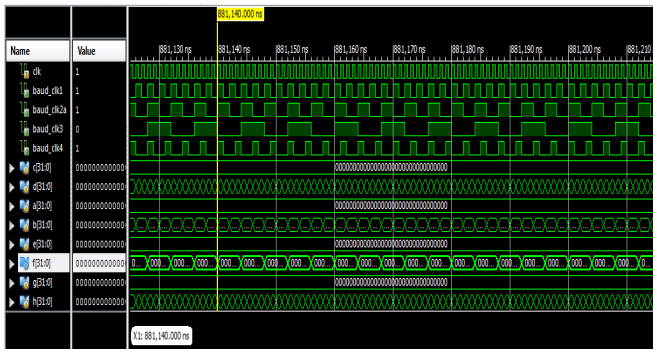

Fig. 6- Simulation waveform of frequency divider

vides the particular frequency according to an integer 'n'.In which we tested input clock 50Hz and we got output according to it in these four baud clocks as baud_clk1:f/2(25Hz) and baud_clk2:f/4 (12.5Hz) and baud_clk3:f/8(6.75Hz) and baud_clk4:f/3(16.6Hz). We can use our UART specially those microcontrollers or processors where system frequency is lower. That places we don't need to attach other lower frequency device separately.

Conclusion and Future Work

We concluded the UART Protocol as per specification has been designed in Verilog and implemented on the Spartan3 xc3s200-4ft256 device later. The UART is running absolutely fine at a clock frequency of 50 MHz. The simulation results of UART and its sub-blocks are properly working mode.

Super Sampling in UART receiver

In UART receiver a sampling is done to read the transmitted data by the middle of the received bit. The sampling ensures that a valid bit would be received. This sampling is done at the 8th Sample of the bit. If at the 8th sample, if a glitch or spike is appeared that is not a valid signal would be considered as a valid signal that is not the correct bit.

To overcome this issue we can apply a super sampling technique. In super sampling we will sample the bit at 7, 8 and at 9th sample.

References

[1] Zhichao Zhang, Wuchen Wu (2010) 2*nd International Conference on Computer Engineering and Technology*. 2(7), 331 – 334.

[2] Shouqian Yu Lili Yi Weihai Chen Zhaojin Wen (2007) *IEEE Conference on Industrial Electronics and Applications*.

[3] Norhuzaimin J. and Maimun H.H. (2005) *The Design of High Speed UART,*

[4] Raffaele Gallo, Martin Delvai, Wilfried Elmenreich, Andreas Steininger (2004) *Revision and Verification of an Enhanced UART IEEE*.

[5] Samir Palnitkar (2003) *Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition*.

[6] Pong P. Chu, *FPGA*, 215-233.

[7] TMS320DM644x DMSoC Universal Asynchronous Receiver/ Transmitter (UART) (2008) *Literature Number*: SPRUE33A.

[8] Greg Goodhue (1993) *A software duplex UART for the Philips Semiconductors*, 751-752.

[9] Mohd Yamani Idna Idris and Mashkuri Yaacob (2003) *IEEE* 1451-1454.

[10] Donald E. Thomas & Philip R. Moorby (2002) *The Verilog Hardware Description Language, Fifth Edition*.