IMPACT: International Journal of Research in
Engineering & Technology (IMPACT: IJRET)
ISSN(E): 2321-8843; ISSN(P): 2347-4599
Vol. 2, Issue 8, Aug 2014, 23-32

FPGA IMPLEMENTATION OF ADVANCED UART CONTROLLER USING VHDL

ARPITA TIWARI[1], RAVI MOHAN[2] & DIVYANSHU RAO[3]

[1]Research Scholar, Shri Ram Institute of Technology, Jabalpur, Madhya Pradesh, India

[2]Head, Professor, Shri Ram Institute of Technology, Jabalpur, Madhya Pradesh, India

[3]Professor, Shri Ram Institute of Technology, Jabalpur, Madhya Pradesh, India

ABSTRACT

As the specific interface chip (ASICs) will cause waste of resources and increased cost. Particularly in the field of electronic design, SOC technology is recently becoming increasingly mature. This situation results in the requirement of realizing the whole system function in a single or a very few chips. Universal Asynchronous Receiver Transmitter (UART) is a kind of serial communication protocol. In parallel communication the cost as well as complexity of the system increases due to simultaneous transmission of data bits on multiple wires. Serial communication alleviates this drawback of parallel communication and emerges effectively in many applications for long distance communication as it reduces the signal distortion because of its simple structure. The UART implemented with VHDL language can be integrated into the FPGA to achieve compact, stable and reliable data transmission. This paper presents implementation of advanced UART with configurable baud rate.

KEYWORDS: Universal Asynchronous Receiver Transmitter, Serial Receiver, Interrupt Identification Register

INTRODUCTION

Universal Asynchronous Receiver Transmitter (UART) is a kind of serial communication protocol; mostly used for short-distance, low speed, low-cost data exchange between computer and peripherals. Serial communication reduces the distortion of a signal, therefore makes data transfer between two systems separated in great distance possible. It is generally connected between a processor and a peripheral, to the processor the UART appears as an 8-bit read/write parallel port. The UART is a full-duplex, asynchronous communication channel that communicates with peripheral devices and personal computers through protocols, such as RS-232, RS-485. This UART reference design contains a Receiver and a Transmitter. The receiver performs serial-to-parallel conversion on the asynchronous data frame received from the serial data input SIN. The transmitter performs parallel-to-serial conversion on the 8-bit data received from the CPU. In order to synchronize the asynchronous serial data and to insure the data integrity, Start, Parity and Stop bits are added to the serial data. An example of the UART frame format is shown in Figure 1 below.

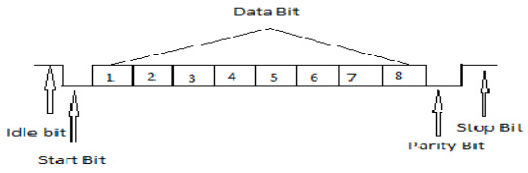

Figure 1: UART Frame Format

In recent years the researchers have proposed various UART designs like automatic baud rate synchronizing capability[2], recursive running sum filter to remove noisy samples[2], predictable timing behavior to allow the

integration of nodes with imprecise clocks in time-triggered real-time systems[4], integration of only core functions into a FPGA chip to achieve compact, stable and reliable data transmission to avoid waste of resources and decrease cost[1], programmable logic to enable interfacing between asynchronous communications protocols and DSP having synchronous serial ports.

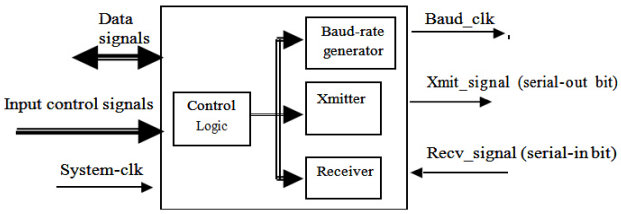

Figure 2: Block Diagram of UART

Basic UART communication needs only two signal lines (RXD, TXD) to complete full-duplex data communication. TXD is the transmit side, the output of UART; RXD is the receiver, the input of UART. UART's basic features are: There are two states in the signal line, using logic 1 (high) and logic 0 (low) to distinguish respectively. For example, when the transmitter is idle, the data line is in the high logic state. Otherwise when a word is given to the UART for asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. After the Start Bit, the individual data bits of the word are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits, and the receiver "looks" at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a 1 or a 0. When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter. When the receiver has received all of the bits in the data word, it may check for the Parity Bits (both sender and receiver must agree on whether a Parity Bit is to be used), and then the receiver looks for a Stop Bit.

Regardless of whether the data was received correctly or not, the UART automatically discards the Start, Parity and Stop bits. If the sender and receiver are configured identically, these bits are not passed to the host. If another word is ready for transmission, the Start Bit for the new word can be sent as soon as the Stop Bit for the previous word has been sent. Because asynchronous data are "self synchronizing", if there are no data to transmit, the transmission line can be idle.

In this paper a more efficient and stable design of UART is proposed. The proposed design of UART satisfies the system requirements of high integration, stabilization, low bit error rate, and low cost. It also supports configurable baud rate generator with data length of 8 bits per frame. The configurable baud rate generator has been implemented on cyclone II FPGA to specify the baud rate of input and accordingly input data can verified with start bit and stop bit.

IMPLEMENTATION

UART Transmitter

The serial transmitter section consists of an 8-bit Transmitter Hold Register (THR) and transmitter Shift

Register (TSR). There are two ways to indicate the status of THR: an independent TxRDY output pin or the THRE flag in the Line Status Register (LSR). When the THR is empty, pin TxRDYn will be low active, and the THR empty flag in LSR will be set to a logic 1. Only when the THR is empty, can a write operation be performed to transfer the data from CPU to THR without trashing the previous data. After the data is loaded in THR, the THR empty flag in LSR will be reset to logic 0, and pin TxRDYn will go inactive high. The serial data transmission will be automatically enabled after the data is loaded into THR. First a Start bit (logic 0) is transmitted and the data in THR is parallel loaded to TSR automatically. Then the Data bits will be shifted out of TSR with certain word lengths defined in Line Control Register (LCR) followed by the Parity bit if parity is enabled. Finally, the Stop bit (logic 1) is generated to indicate the end of the frame. After a frame is fully transmitted, another frame will be transmitted immediately if THR is not empty (due to a THR write occurring during the first frame of transmission). This automatic sequencing causes the frames to be transmitted back-to-back which increases the transmission bandwidth. When no transmission is taking place, the SOUT pin is held in the high state.

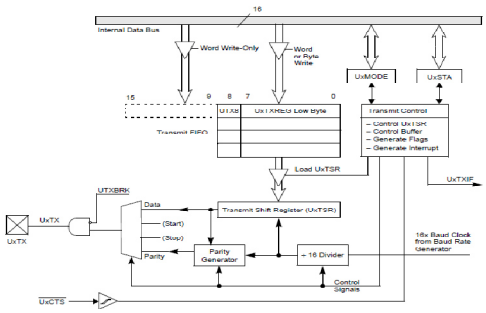

Figure 3: UART Transmitter Block Diagram

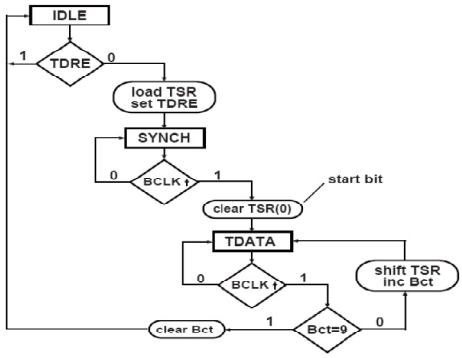

Figure 4: SM Chart of UART Transmitter

The behaviour of the transmitter is controlled by the FSM (Finite State Machine) shown in Figure:

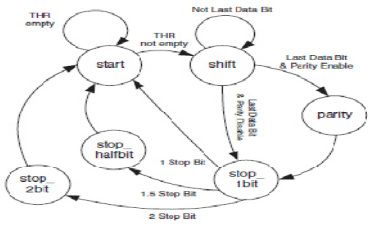

Figure 5: Transmitter State Machine

<start>:

When the UART is reset by MR pin, the transmitter FSM will be reset to this state. When in this state, the transmitter is waiting to assert the Start bit. A Start bit will be asserted as soon as the THR is not empty. Once a low SOUT (Start bit) is asserted, the FSM will switch to <shift> state.

<shift>:

When the FSM is in this state, it's waiting for the last (most significant) Data bit to be shifted out. After the last Data bit is shifted out, the FSM will switch to <parity> state if parity is enabled. Otherwise, it will switch to <stop_1bit> state.

<parity>:

When the FSM is in this state, the last Data bit is still in transmission. When the transmission is complete, the FSM will assert the Parity bit. Once the Parity bit is asserted, the FSM switch to the <stop_1bit> state.

<stop_1bit>:

No matter if the Stop bit is configured to be 1, 1.5 or 2 bits long, the FSM will always switch to this state, wait for a baud clock cycle, and then assert the Stop bit(s). For 1 Stop bit, the FSM switches back to <start> state and waits to assert the Start bit of another frame. For 1.5 Stop bit, it switches to <stop_halfbit> state and stays there for just half baud clock cycle before switching to <start> state. For 2 Stop bits, it switches to <stop_2bit> state then switches back to <start> state. Note that the Stop bit(s) is asserted at the time when the FSM is leaving the <stop_1_bit> state.

<stop_halfbit>:

This state is for 5-bit Data bits with 1.5 Stop bit. The FSM will stay in this state for only half baud clock cycle and then switch to <start> state.

<stop_2bit>:

When the FSM is in this state, the first Stop bit is in transmission. It waits for a baud clock cycle, then asserts the second Stop bit and switches to the <start> state.

Setup for UART Transmit

Steps to follow when setting up a transmission:

- Initialize the UxBRG register for the appropriate baud rate.
- Set the number of data bits, number of Stop bits and parity selection by writing to the PDSEL<1:0> (UxMODE<2:1>) and STSEL (UxMODE<0>) bits.
- If transmit interrupts are desired, set the UxTXIE control bit in the corresponding Interrupt Enable Control register (IECx). Specify the interrupt priority for the transmit interrupt using the UxTXIP<2:0> control bits in the corresponding Interrupt Priority Control register (IPCx). Also, select the Transmit Interrupt mode by writing the UTXISEL<1:0> (UxSTA<15,13>) bits.
- Enable the UART module by setting the UARTEN (UxMODE<15>) bit.

- Enable the transmission by setting the UTXEN (UxSTA<10>) bit, which will also set the UxTXIF bit. The UxTXIF bit should be cleared in the software routine that services the UART transmit interrupt. The operation of the UxTXIF bit is controlled by the UTXISEL<1:0> control bits.
- Load data to the UxTXREG register (starts transmission). If 9-bit transmission has been selected, load a word. If 8-bit transmission is used, load a byte. Data can be loaded into the buffer until the UTXBF status bit (UxSTA<9>) is set.

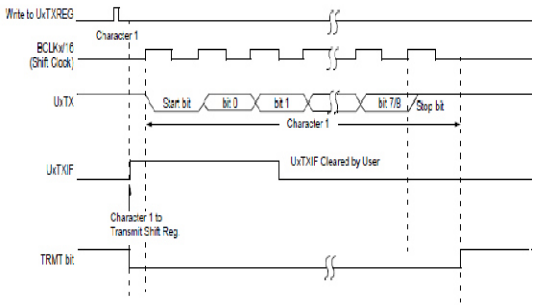

Figure 6: Transmission (8-Bit or 9-Bit Data)

UART Receiver

The serial receiver section also contains an 8-bit Receiver Buffer Register (RBR) and Receiver Shift Register (RSR). The status of RBR can be provided by either independent pin RxRDYn or the Receiver Data available flag (DR) in LSR. Since the serial frame is synchronous to the receiving clock, a high to low transition of SIN pin will be treated as the Start bit of a frame. However, in order to avoid receiving a incorrect data due to SIN signal noise, the False Start Bit Detection feature is implemented in the design which requires the Start bit to be low at least 50% of the receiving baud rate clock cycle.

Once a valid 8 Clk16X clocks Start bit is received, the Data bits and Parity bit will be sampled every 16 Clk16X clocks (the receiving baud rate). If the Start bit is exactly 16 Clk16X clocks long, each of the following bits will be sampled at the center of the bit itself. LSR will be updated to show the received frame status when any of the line errors (Overrun error, Parity error, Framing error, Break) is detected.

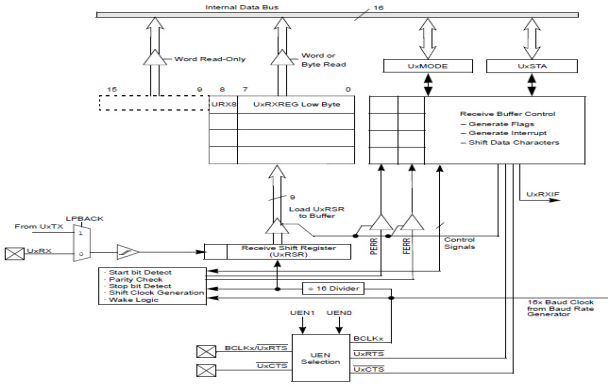

Figure 7: UART Receiver Block Diagram

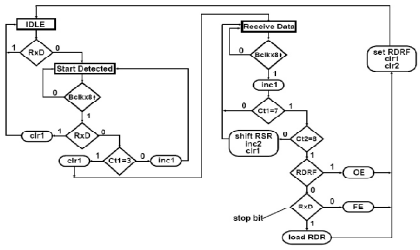

Figure 8: SM Chart of UART Receiver

Whenever the Framing error is detected, the UART assumes that the error was due to the Start bit of the following frame and tries to resynchronize it. To do this, it samples the Start bit twice based on the Clk16X clock. If both samples of the SIN are low, the UART will take in the following frame's Data bits after the 8 Clk16X clocks Start bit is sampled. The resynchronization will not occur for the Framing error caused by the Break. When the data is available in RBR, the RxRDYn will be low active, and the Receiver Data available flag (DR) in LSR will be set to logic 1 to inform the CPU that the data is ready to be read. The behavior of the receiver is controlled by the FSM shown in Figure:

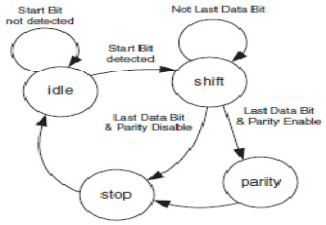

Figure 9: Receiver State Machine

<idle>:

When the MR pin resets the UART, the receiver FSM will be reset to this state. When in this state, it's waiting for SIN to be changed from high to low and stay low for 8 Clk16X clocks to be considered a valid Start bit. Once a valid Start bit is detected, the FSM will switch to <shift> state.

<shift>:

When the FSM is in this state, it waits 16 Clk16X clocks for each Data bit to shift into RSR. After the last Data bit is shifted in, the FSM will switch to <parity> state if parity is enabled. Otherwise, it will switch to <stop> state.

<parity>:

When the FSM is in this state, it waits for 16 Clk16X clocks and then samples the Parity bit. Once the Parity bit is sampled, the FSM switch to the <stop> state.

<stop>:

No matter if the Stop bit length is configured to be 1, 1.5 or 2 bits long, the FSM will always wait for 16 Clk16X clocks and then sample the Stop bit. As long as a logic 1 is sampled at the Stop bit, the Framing error flag (FE) in LSR will not be set.

Setup for UART Reception

Steps to follow when setting up a reception:

- Initialize the UxBRG register for the appropriate baud rate.
- Set the number of data bits, number of Stop bits and parity selection by writing to the PDSEL<1:0> (UxMODE<2:1>) and STSEL (UxMODE<0>) bits.
- If interrupts are desired, then set the UxRXIE bit in the corresponding Interrupt Enable Control (IECx) register. Specify the interrupt priority for the interrupt using the UxRXIP<2:0> control bits in the corresponding Interrupt Priority Control register (IPCx). Also, select the Receive Interrupt mode by writing to the URXISEL<1:0> (UxSTA<7:6>) bits.
- Enable the UART module by setting the UARTEN (UxMODE<15>) bit.
- Receive interrupts will depend on the URXISEL<1:0> control bit settings. If receive interrupts are not enabled, the user can poll the URXDA bit.
- Read data from the receive buffer. If 9-bit transmission has been selected, read a word; otherwise, read a byte. The URXDA status bit (UxSTA<0>) will be set whenever data is available in the buffer.

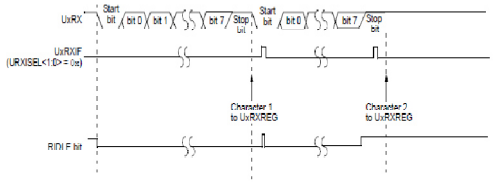

Figure 10: UART Reception

Interrupt

The UART common interrupt request pin INTR will go high active when any of the interrupt conditions is matched and enabled by Interrupt Enable Register (IER).

The UART prioritizes interrupts into four levels to minimize external software interaction, and records these in the Interrupt Identification Register (IIR). The four levels of interrupt conditions in order of priority are Receiver Line Status; Received Data Ready; Transmitter Holding Register Empty; and MODEM Status. Performing a read cycle on IIR freezes all interrupts and indicates the highest priority pending interrupt to the CPU. A lower level interrupt may be seen at next IIR reading. The behaviour of the interrupt is controlled by the FSM shown in Figure 11:

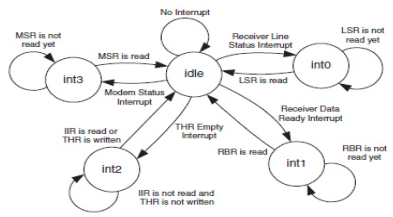

Figure 11: Interrupt State Machine

<idle>:

When the MR pin resets the UART, the interrupt FSM will be reset to this state. When in this state, it is waiting for the enabled interrupt conditions to be true and then switches to the interrupt state with highest priority.

<int0>:

The FSM switches to this state when the highest priority level interrupt occurs. It stays at this state until the LSR is read.

<int1>:

The FSM switches to this state when the second priority level interrupt occurs. It stays at this state until the RBR is read.

<int2>:

The FSM switches to this state when the third priority level interrupt occurs. It stays at this state until the IIR is read or after THR is written.

<int3>:

The FSM switches to this state when the fourth priority level interrupt occurs. It stays at this state until the MSR is read.

The interrupts continue to be generated as long as the corresponding enable bit in IER is set and the corresponding interrupt condition is matched. Since the interrupt state machine is running at Clk16X, the INTR pin might be low for a Clk16X clock and then return to high again.

CONCLUSIONS AND RESULTS

This thesis has introduced universal asynchronous serial protocols, described the role of UARTs, defined the basic conditions for successful data transmission, and specifically addressed the application as the T=0/T=1 protocol in the ISO7816 standard.

The motivation for UART implementation on FPGA is the increased flexibility compared to hardware UART components, and potential cost-savings by eliminating a discrete hardware component.

To this end, UART implementation on FPGA meeting the requirements has been proposed in this thesis, derived as a combination of the common implementation patterns. The proposed implementation uses edge-sensitive GPIO interrupts for start-bit detection, and disables interrupts for the subsequent bits within the frame. Furthermore, the algorithm detects the edge transition in order to synchronise the receiver clock with the sender clock, i.e. compensating for the interrupt latency jitter.

The Device utilization summary for the transmitter and receiver is given as follows.

Table 1: a_serial_transmitter

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	7	4656	0%

Table 1: Contd.,

Number of Slice Flip	12	9312	0%
Number of 4 Input LUTs	7	9312	0%
Number of Bonded IOBs	12	232	5%
Number of GCLKs	1	24	4%

Transmitter utilization summary for the FPGA Device

Table 2: a_serial_receiver

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	19	4656	0%
Number of Slice Flip Flops	29	9312	0%
Number of 4 Input LUTs	17	9312	0%
Number of Bonded IOBs	13	232	5%
Number of GCLKs	2	24	8%

Receiver utilization summary for the FPGA Device

As the presented implementation relies primarily on the interrupt latency to be within certain boundaries, the interrupt latency was measured under various conditions in order to determine the operational conditions for which error-free UART operation is to be expected. Finally the actual UART implementation was bench-marked with respect to error rates and network responsiveness at various communication rates. The data analysis has shown, that basically the stated goals have been attained in principle. The results would have been significantly worse, possibly resulting in having missed the stated goals with the current implementation.

FUTURE WORK

As the implementation presented served the purpose of designing the hardware of advanced UART. As we have been observed that, it suffers from various limitations. Eliminating those limitations is regarded as future work, and there is also room left for various other optimisations and improvements.

The Presented implementation does not make use of minimum latency offset compensation in order to shift the sampling point into the sampling zone where the data signal is guaranteed by the specification to be reliable. Implementing the offset compensation would improve the Design conformance with respect to worst-case timing scenarios.

REFERENCES

1. Fang Yi-yuan, Chen Xue-jun. "Design and Simulation of UART Serial Communication Module Based on VHDL," Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on, vol., no., pp.1-4, 28-29 May 2011.
2. Himanshu Patel, Sanjay Trivedi R, Neelkanthan V. R. Gujraty. "A Robust UART Architecture Based on Recursive Running Sum Filter for Better Noise Performance," VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on, vol., no., pp.819-823, Jan. 2007.

3. Gallo R, Delvai M, Elmenreich W, Steininger A. "Revision and verification of an enhanced UART," Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on, vol., no., pp. 315- 318, 22-24 Sept. 2004.
4. Elmenreich W, Delvai M. "Time-triggered communication with UARTs," Factory Communication Systems, 2002. 4th IEEE International Workshop on, vol., no., pp. 97- 104, 2002J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., Vol. 2.Oxford: Clarendon, 1892, pp.68–73.
5. Palnitkar S. (2006). A Guide to Digital Design and Synthesis (2nd ed). India: Dorling Kindersley Pvt. Ltd.
6. Naresh Patel, Vatsalkumar Patel and Vikaskumar Patel. “VHDL Implementation of UART with Status Register” 2012 International Conference on Communication Systems and Network Technologies (IEEE).
7. Hazim Kamal Ansari, Asad Suhail Farooqi. “Design Of High Speed Uart For Programming FPGA” International Journal Of Engineering And Computer Science Volume1 Issue 1 Oct 2012.
8. Liakot Ali, Roslina Sidek, Ishak Aris, Alauddin Mohd. Ali, Bambang Sunaryo Suparjo.”Design of a micro-UART for Soc application”, 2004, Elsevier Ltd.
9. Ananya Chakraborty, Surbhi, Sukanya Gupta, Swati Deshkar, Pradeep Kumar Jaisal. “Design of UART (Universal Asynchronous Receiver Transmitter) using VHDL”, IJCST Vol. 3, Issue 1, Jan. - March 2012.
10. Hazim Kamal Ansari, Asad Suhail Farooqi.“Design Of High Speed Uart For Programming FPGA” International Journal Of Engineering And Computer Science Volume1 Issue 1 Oct 2012.
11. Naresh patel, Vatsalkumar Patel and Vikaskumar Patel. “VHDL Implementation of UART with Status Register” 2012 International Conference on Communication Systems and Network Technologies (IEEE).