Scholars Journal of Engineering and Technology (SJET) **ISSN 2321-435X (Online)**
Sch. J. Eng. Tech., 2015; 3(6):600-608 **ISSN 2347-9523 (Print)**

(An International Publisher for Academic and Scientific Resources)
www.saspublisher.com

Research Article

Design and FPGA Implementation of UART Using Microprogrammed Controller

Mohammad Awedh, Ahmed Mueen
King Abdulaziz University, Jeddah 22254, Saudi Arabia

***Corresponding author**
Ahmed Mueen
Email: mueen123@gmail.com

Abstract: This paper presents an implementation of Universal Asynchronous Receiver-Transmitter (UART) controller based on Microprogrammed Controller on Field Programmable Gate Array (FPGA. Our design of UART is fully functional and synthesizable. It is coded using Verilog based top-down hierarchical design methodology and realized in Spartan-3E FPGA using Xilinx ISE Webpack 14.7. The implementation results demonstrate that the design can operate at a maximum clock frequency of 218.248 MHz. The maximum clock frequency of hardwired implementation of UART controller is 192.773 MHz. We also compare our Microprogrammed implementation of UART controller to the standard ROM method; our implementation uses less number of bits and hence small number of storage elements.
Keywords: Receiver Transmitter, Microprogrammed Controller, and Field Programmable Gate Array

INTRODUCTION

UART (Universal Asynchronous Receiver Transmitter) controller is a serial communication device. In several control systems, serial communication circuit is used largely .A universal asynchronous receive/transmit (UART) is an integrated circuit which plays the most important role in serial communication [9]. Serial communication is another way of communication used widely because of its simple structure and long transmission distance [6]. Serial communication is vital to computers and allows them to communicate with the low speed devices such as keyboard, mouse, modems etc [11, 7]. UARTs are used for serial communication between two devices with minimum wires. The data is sent serially, and no clock signal is sent along with it. The primary function of a UART is parallel-to-serial conversion when transmitting, and serial-to-parallel conversion when receiving. The sender and receiver have separate, unsynchronized, clock signals. In order to synchronize the asynchronous serial data and to insure the data integrity, Start and Stop bits are added to the serial data. An example of the UART frame format is shown in Figure 1.

The transmitted character is composed of an 8-bit data byte, sent LSB (least significant bit) first, preceded by a start bit (active low) and followed by a stop bit (active high). When no character is being transmitted, the line is idle (active high). The line need not go idle between characters, as it is possible for the start bit of a transmission to immediately follow the stop bit of the previous transmission.

A field-programmable gate array (FPGA) is a logic device that contains a two dimensional array of generic logic cells and programmable switches. A logic cell can be configured (i.e., programmed) to perform a simple function, and a programmable switch can be customized to provide interconnections among the logic cells. A custom design can be implemented by specifying the function of each logic cell and selectively setting the connection of each programmable switch. Once the design and synthesis are completed, we can use a simple adaptor cable to download the desired logic cell and switch configuration to the FPGA device and obtain the custom circuit.

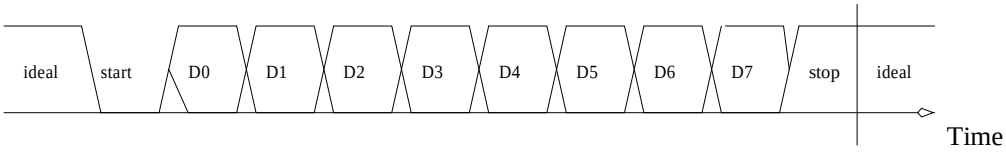

Fig.1. Transmission Frame

Top Level Design

The top-level design of UART consists of a click generator, a receiving module and a sending module. The UART Transmitter is used to transform the parallel data for output in accordance with the basic UART frame format to *serial Out* signal serial output. UART Receiver receives the serial signal on *serialIn*, and converts it into parallel data [2]. Clock generator specifically produces a local clock far higher rate than the baud rate to sample the input *serialIn* continuously to enable the receiver to maintain synchronization with the transmitter. Figure 2 shows a functional block diagram of the UART.

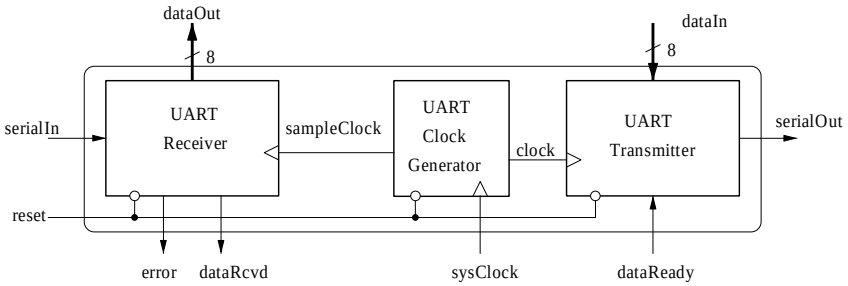

Fig.2. Functional Block Diagram

Part of a UARTs function, and the tricky part, is to ”sample” the serial input at just the right time to reliably capture the bit stream. A high-speed clock to sample the bit stream multiple times per data bit allows one to accomplish this task.

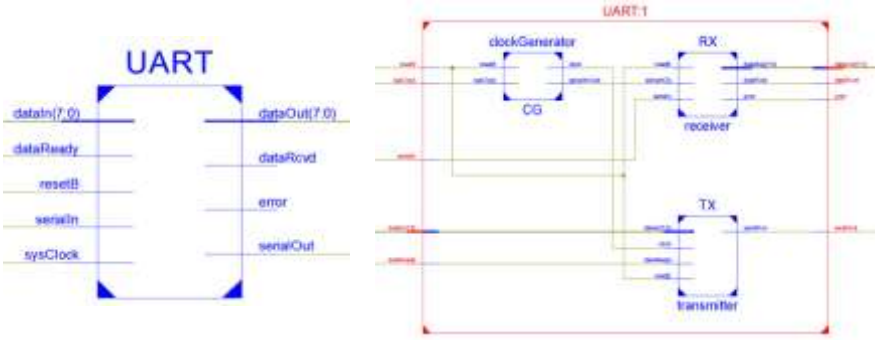

Fig.3. Top level schematic of UART

UART Transmitter

The proposed UART transmitter architecture comprises of two main building blocks which are data path unit and control unit, Figure 4. The architecture of the transmitter data path unit consists of a data register, a data shift register, and a status register, which counts the bits that are transmitted. The figure shows the input-output signals of the transmitter. The input signals are provided by the host device, and the output signals are the serial data stream and a status signal. Data is transmitted serially on the *serial Out*out put. The transmitter is ready transmit when the status signal *txDone*is asserted high. When *data Ready*is asserted high, the transmitter loads the content of the *dataIn* into *dataRegister*. *bitCnt Max* indicates the status of the bit counter in the datapath unit.

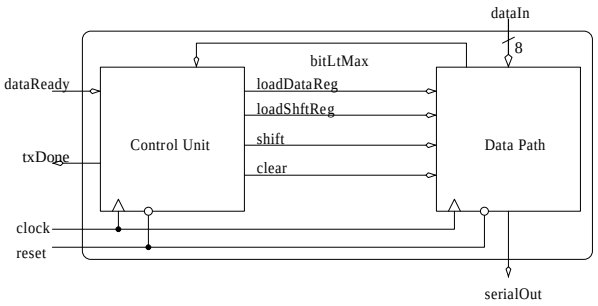

Fig-4: Functional Block Diagram of the Transmitter

The ASM chart of the state machine controlling the transmitter is shown in Figure 6. The control signals produced by the ASM chart induce state-dependent register transfers in the datapath. The assertion of *loadShiftReg* loads the contents of data register into shift register.

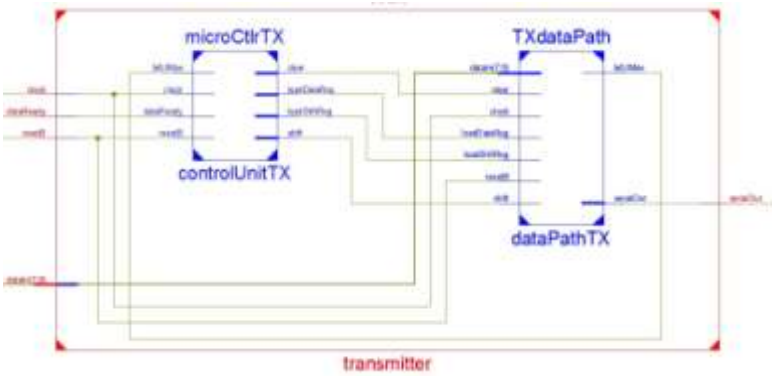

Fig.5. Top level RTL Schematic of UART Transmitter

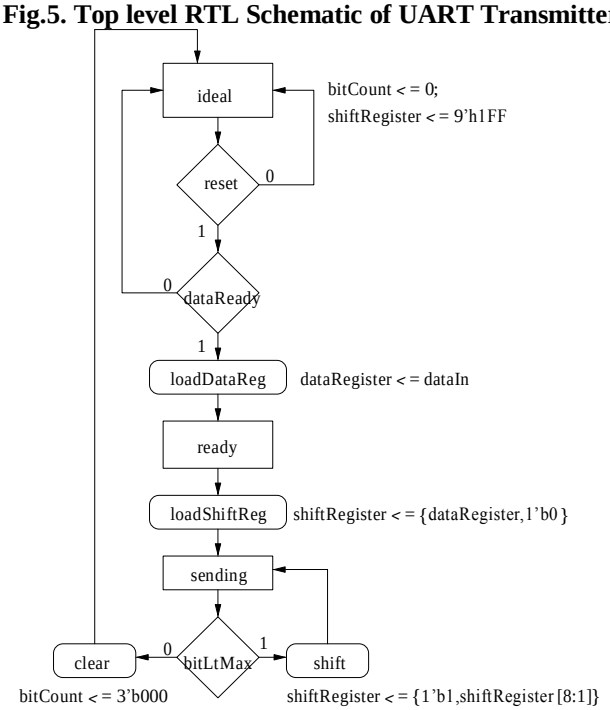

Fig.6. ASM Chart of the Transmitter Controller

UART Receiver

Figure 7 shows a functional block diagram of the UART receiver. Data is received serially on the *serialIn* input. When one byte of data has been received, it is output to the *dataOut* output bus, and the output control signal dataRcvd is asserted high for one clock period. The block is clocked with a frequency 16 times the baud rate.

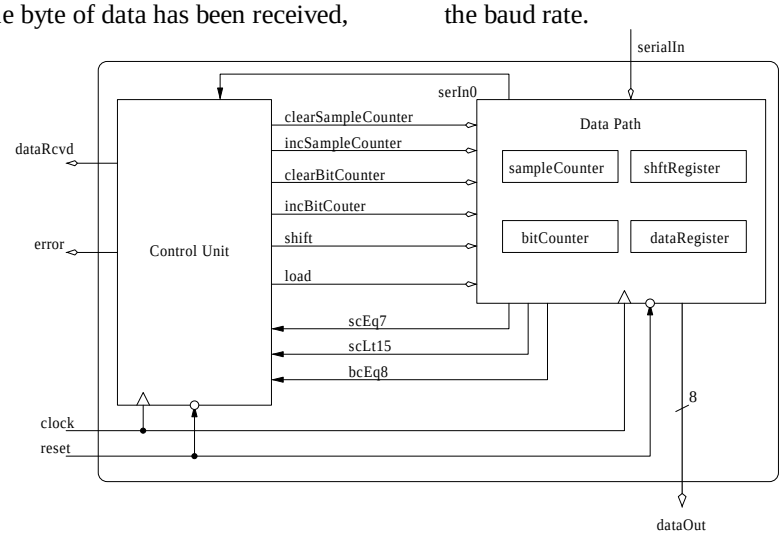

Fig.7. Functional Block Diagram of the Receiver

Although the data arrives at a standard bit rate, the data is not synchronized with the internal clock at the UART receiver. This issue of synchronization is resolved by generating a local clock at a higher frequency and using it to sample the received data in a manner that preserves the integrity of the data.

In the scheme used here, the data, assumed to be in a 10-bit format, will be sampled at a rate determined by *sample Clock*, which is generated at the receiver's host. The cycles of *sampleClock* will be counted to ensure that the data are sampled in the middle of a bit time, as shown in Figure 8.

In order to ensure the arrival of the start bit, at least half of the start bit successive samples (8 samples in our design) of value 0 are detected after the serial input data goes low.

The ASM chart of the state machine controlling the UART receiver is shown in Figure 9. The ASM block of the *starting* state determines whether the first bit is a valid start bit. The ASM block of the *receiving* state receives the remaining (8) bits which is controlled by *bcE8* status signal. The assertion of *shift* will cause the sample bit value to be loaded into the *shiftRegister*. If there is no error in receiving the data, the contents of the *shiftRegister* is loaded into the *dataRegister* and the dat aRcvd is asserted high.

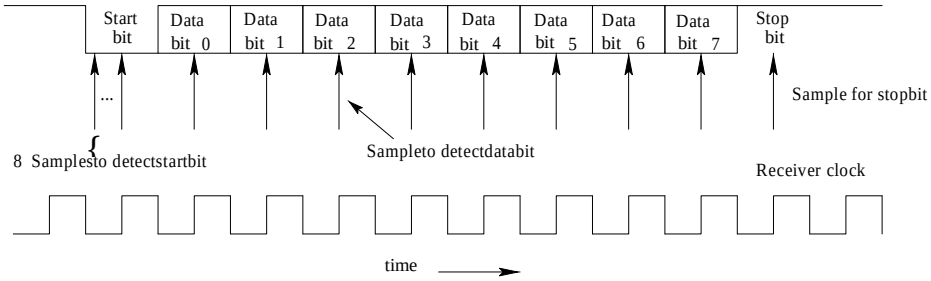

Fig.8. Samples

UART Clock Generator

The UART Clock Generator in Figure 2 is used to generate UART transmitter clock and receiver sample clock based on the following calculation.

- Baud Rate Divisor for transmitter clock = System Clock/baud rate.
- Baud Rate Divisor for Sample clock = (System Clock/baud rate)/16.

The following table shows the baud rate divisor and the minimum number of bits that are required to store the baud rate divisor for sample clock.

Baud Rate	Baud Rate Divisor	# bits
9600	325.5208333	9
19200	162.7604167	8
38400	81.38020833	7
57600	54.25347222	6
115200	27.12673611	5
230400	13.56336806	4
460800	6.781684028	3
921600	3.390842014	2

If, for example, the system clock = 50 MHZ and baud rate is 115200 bps, then the Baud Rate Divisor (BRD) for Sample clock is 27.12673611. If the scaling factor is 2^5, then the fixed-point representation of BDR = $27.12673611 * 2^5 \approx 868$. Therefore, the generated baud rate divisor = $868/2^5 = 27.15625$. The generated baud rate = 50MHA/27.15625)/16 = 115075. Hence, the error is $\frac{115075 - 115200}{115200} * 100 =$ 0.1085%.

Microprogrammed Controller

In digital system design, Control Unit (controller) controls the flow of data through the digital system, and coordinates the activities of the units within the Datapath Unit. Control Unit (CU) receives external instructions (through set of controller inputs) which it converts into a sequence of control signals that the CU applies to the Datapath Unit (DPU) to implement a sequence of register-transfer level (RTL) operations.

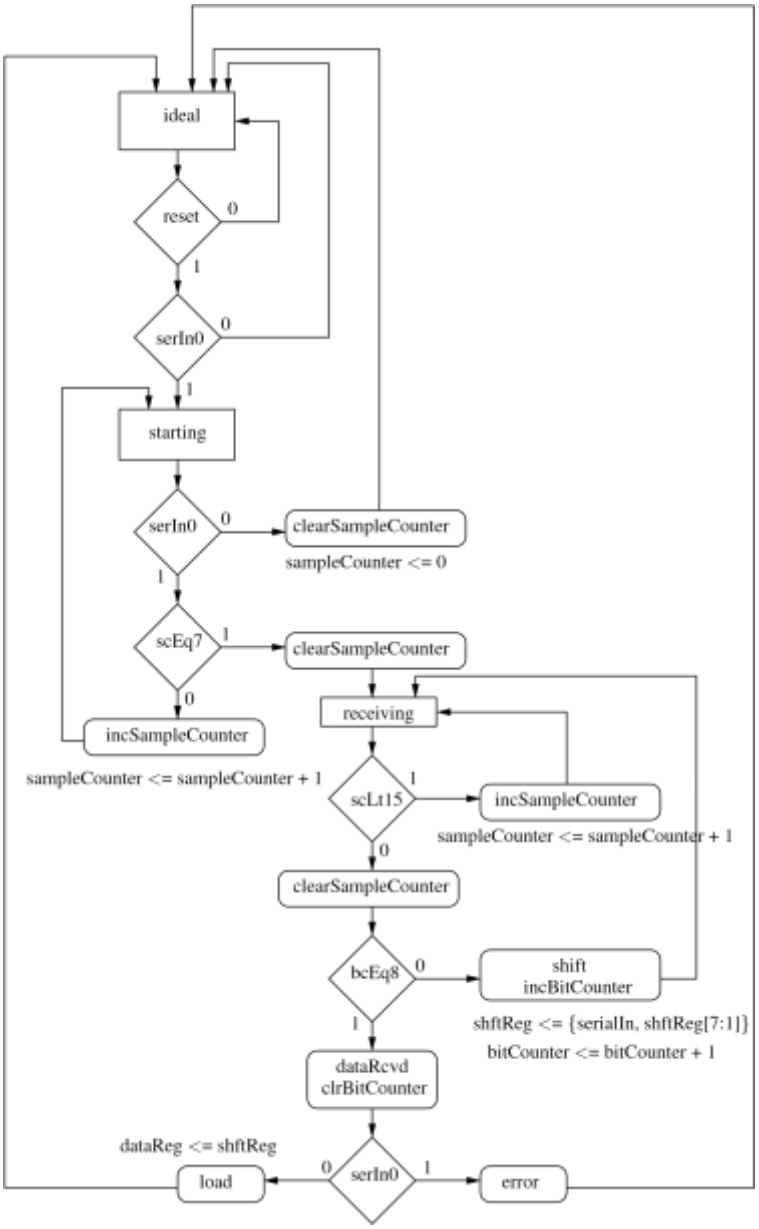

Fig.9. ASM Chart of the Receiver

The algorithm for control unit is usually specified by state machine. Flowchart description [1] may be used to specify the operation of the controller state machine. An algorithmic state machine chart, or ASM chart, is a kind of such flowchart that is used to describe the behavior of a state machine. ASM chart systematically specifies all the controller signals that should be generated at each time during the flow of the controller from the initial state through each of the other controller states.

Definition 1.*An* Algorithmic State Machine Δ *is a directed connected graph,* Δ = h*S,C,D,E,X,Y,Ls,Lc,Ldi[10] where:*

- *S is a finite set of state vertices.*
- *C is a finite set of conditional output vertices.*
- *D is a finite set of decision vertices.*
- *E is a finite set of edges.*
- *X is a finite set of controller inputs. X = {x0,x1,...,xn}*
- *Y is a finite set of control signals. Y = {y0,y1,...,ym}*
- *Ls* :$S \rightarrow 2^Y$ *is a function that labels each state vertex with a set of control signals that are asserted in that state.*
- *Lc*:$C \rightarrow 2^Y$ *is a function that labels each conditional output vertex with a set of control signals that are asserted in that conditional vertex.*
- *Ld*:$D \rightarrow 2^X$ *is a function that labels each decision vertex with a set of control inputs that are to be checked in that decision vertex .*

In the above definition of ASM char, if $C = \varphi$, then Δ defines aa Moore machine. If *Ls* is empty, then Δ defines a mealy machine. Otherwise, Δ defines a Moore/Mealy machine.

An ASM chart is constructed from SM blocks. Each SM block contains exactly one state vertex, together with the decision and conditional vertices associated with that state. An ASM block has one entrance path and one or more exit paths. Each ASM block describes the machine operation during the time that the machine is in one state. When a controller enters the state associated with a given ASM block, the outputs on the output list in the state box asserted.

There are several methods to design a controller, such as hardwired controller and microprogrammed controller [10]. In this paper, we used microprogrammed controller to implement the control unit of UART in FPGA. The main advantage of the microprogrammed controller is its flexibility and simplicity [4, 3].

Microprogrammed controller architecture, Figure 10, consists of a microprogram counter (μPC), Microprogram ROM, and a multiplexer, MUX [4, 8]. Each ROM location stores a *microinstruction*. Each microinstruction consists of three fields; the first filed, *selectInputs*, controls the output of the multiplexer, MUX. It selects which input of controller inputs to be tested. The output of the multiplexer are used to control the order of execution of microoperations (microinstruction). The output of MUX specifies the next value of μPC. As shown in Figure 10, if the output of the MUX is logical 0, then the value of μPC is incremented by 1, μPC = μPC +1. If the output of the MUX is logical 1, then the value of μPC is loaded from the second filed of the microinstruction, *branchAddress*. The last filed of the microinstruction, *controlSignals*, contains the list of control signals to be asserted.

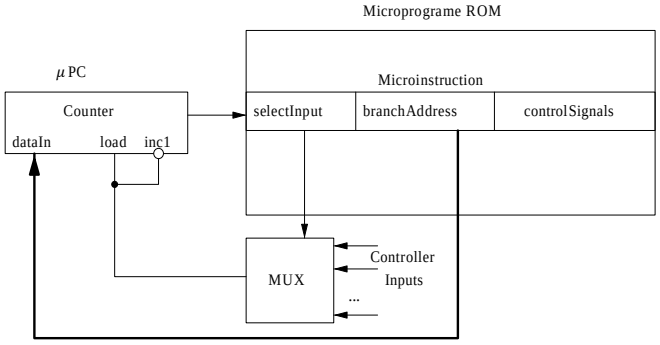

Fig.10. Microprogrammed Controller Architecture

Microprogrammed Implementation

ASM chart description of a control unit is inspected to realize a control unit in microprogrammed controller. A microprogrammed controller based on ASM chart specifies which microinstructions should be executed in each step. Since a microinstruction is done in a clock-by-clockbasis, its timing is similar to a state transition of an ASM chart.

In order to realize a control units of UART using Microprogramming technique, the ASM chart of the control units are used. However, transformations are performed on the ASM chart to facilitate easy and efficient microprogramming [8] using the microprogrammed controller architecture in Figure 10. The ASM chart must describe a Moore machine and only a single decision boxes determining the sequencing between states.

Figure 11 illustrates the modified ASM chart for UART transmitter controller with state assignments. The the state assignments are chosen such that, if the controller input is *false*, the next state should be the current state incremented by 1. The next state when the controller input is *true* will explicitly specified in the microinstruction, in *branchAddress* field. The corresponding microprogrammed implementation is given in Table 1.

Table-1: Content of the Control Memory of the Transmitter Controller

Address	Microinstruction					
	selectInput	branchAddress	controlSignals			
			clear	shift	loadDataReg	loadShiftReg
00	00	00	1	0	0	0
01	01	00	0	0	1	0
10	10	00	0	0	0	1
11	11	11	0	1	0	0

Figure 12 illustrates the modified ASM chart for UART receiver controller with state assignments. The the state assignments are chosen such that, if the controller in-

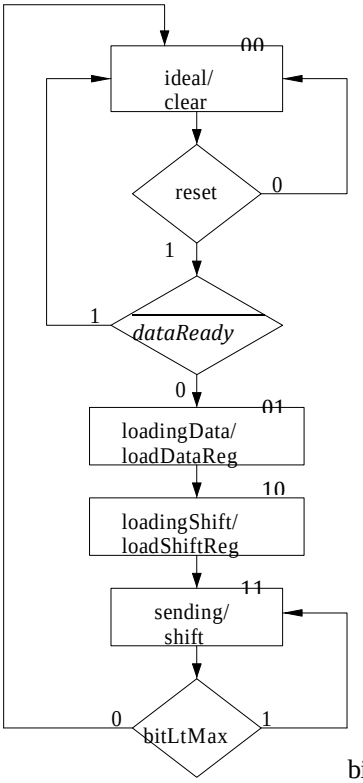

Fig-11: Modified ASM chart for Transmitter Controller put is *true*, the next state should be the current state incremented by 1. The next state when the controller input is *false* will explicitly specified in the microinstruction, in *branchAddress* field. The corresponding microprogrammed implementations is given in Table 2.

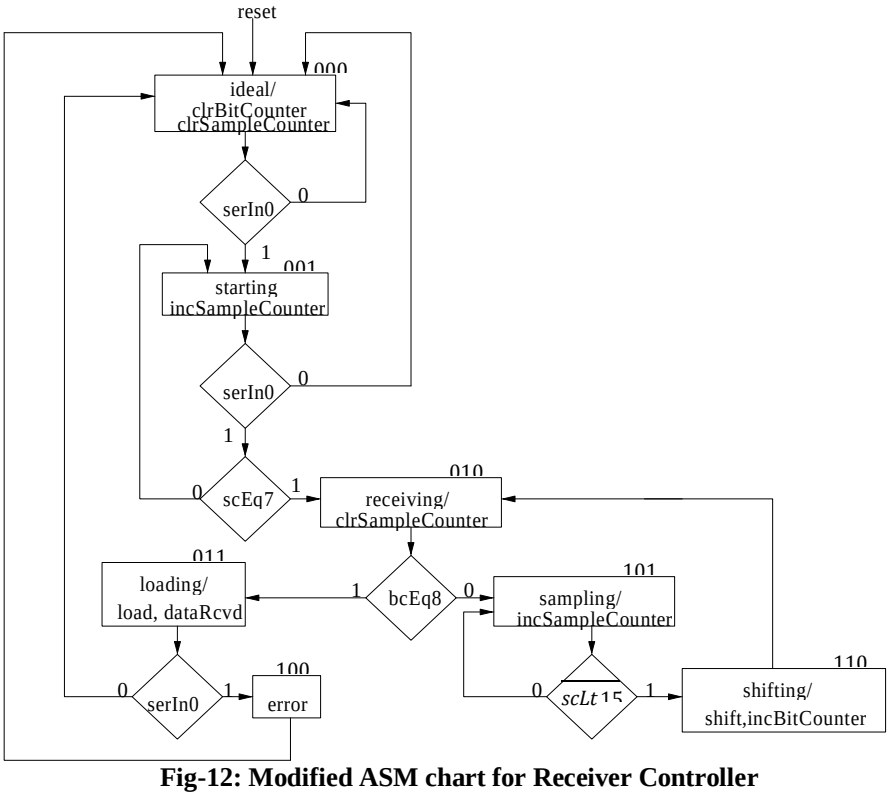

Fig-12: Modified ASM chart for Receiver Controller

If the transmitter controller is implemented by a standard ROM method using Figure 6, the ROM size must be 16 × 6. There are three states, requiring two flip-flops and two next-state equations. There are 2 inputs. Hence, the state table for this state machine will have 2^4 = 16 rows. There will be two next-state equations and 4 outputs, requiring 6 bits in each entry. A comparison of the ROM method with the microprogrammed implementations of the transmitter controller is shown in Table 3. Similarly, a comparison of the ROM method with the microprogrammed implementations of the receiver controller is shown in Table 3.

Table-2: Content of the Control Memory of the Receiver Controller

Address	Microinstruction								
			controlSignals						
	selectInput	branchAddress	clrBit Count	clrSample Counter	incBit Counter	incSample Counter	shift	load	error
000	000	000	1	1	0	0	0	0	0
001	001	001	0	0	0	1	0	0	0
010	010	101	0	1	0	0	0	0	0
011	000	000	0	0	0	0	0	1	0
100	011	000	0	0	0	0	0	0	1
101	100	101	0	0	0	1	0	0	0
110	011	010	0	0	1	0	1	0	0

Table-3: Comparison of Different Implementations of UART Controller

Method	ROM Size of the transmitter Controller	
	Size	No. of bits
ROM method with original SM chart	16×6	96
Microprogrammed implementations	4×8	32
Method	ROM Size of the receiver Controller	
	Size	No. of bits
ROM method with original SM chart	128×10	1280
Microprogrammed implementations	7×13	91

The MUX configuration of the transmitter and receiver controller is illustrated in Figure 13 and Figure 14 respectively.

SYNTHESIS RESULT

The design of UART is coded using Verilog based top-down hierarchical design methodology and realized in Spartan-3E FPGA using Xilinx ISE Webpack 14.7. Table 4 shows the FPGA utilization for the Microprogrammed implementation of the UART controller compared to the Hardwired implementation. Both implementations use almost the same amount of FPGA resources. However, Microprogrammed implementation operates faster: the minimum clock period of the Microprogrammed implementation is 4.582ns (Maximum Frequency is 218.248MHz) and is 5.187ns (Maximum Frequency is 192.773MHz) for Hardwired implementation.

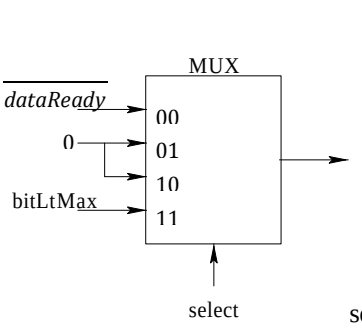

Fig-13: MUX for Transmitter

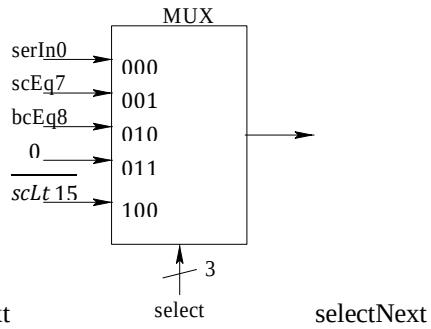

Fig-14: MUX for Receiver

Table-4: FPGA Utilization

Method	FPGA Utilization for Transmitter		
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs
Hardwired Implementation	16	25	20
Microprogrammed Implementation	17	26	22
Method	FPGA Utilization for Receiver		
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs
Hardwired Implementation	20	27	28
Microprogrammed Implementation	21	29	30

EXPERIMENTAL RESULT

To prove the functionality of our design of UART, we implemented our design in Digilent Basys2 Spartan3E FPGA Board [5]. We then connect the FPGA board to a PC that runs a program that transmits and receives large-size (50 MB to 100 MB) of plain text and binary files. The program then compares the sent bytes to the received bytes. All the bytes that are sent are received correctly.

CONCLUSION

In this paper, we have presented FPGA realization of micro programmed implementation of UART controllers. Our design is fully functional and synthesizable and can operate at a maximum clock frequency of 218.248 MHz. The design uses less number of FPGA resources compared to the ROM-based method.

REFERENCES

1. Alexander Barkalov LT; Logic Synthesis for FSM-Based Control Units. Springer Berlin Heidelberg, 2009.
2. Ali L, Sidek R, Aris I, Ali AM, Suparjo BS; Design of a micro-uart for soc application. Computers & Electrical Engineering, 2004; 30(4):257-268.
3. Barkalov AA, Titarenko LA, Efimenko KN; Optimization of circuits of compositional microprogram control units implemented on fpga. Cybernetics and Sys. Anal, 2011; 47(1):166–174.
4. Bomar BW; Implementation of microprogrammed control in fpgas. Industrial Electronics, IEEE Transactions on, 2002; 49(2): 415–422.
5. Digilent. Digilent Basys2 Spartan-3E FPGA Board.
6. HU Likun WQ; Uart-based reliable communication and performance analysis. In Computer Engineering, 2006; 32: 15–21.
7. Norhuzaimin J, Maimun H; The design of high speed uart. In Applied Electromagnetics, APACE 2005. Asia-Pacific Conference on, 2005.
8. Roth Jr. CH, John LK; Digital Systems Design Using VHDL. Thomson-Engineering, 2007.
9. Tomasi W; Advanced electronic communication systems, Third Edition. Prentice-Hall, 1994.
10. Wis´niewski R, Barkalov A, Titarenko L, Halang W; Design of microprogrammed controllers to be implemented in fpgas. Int. J. Appl. Math. Comput. Sci, 2011; 21(2):401–412.
11. Yi-yuan F, Xue-jun C; Design and simulation of uart serial communication module based on vhdl. In Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on, 2011; 1-4.