A High Speed Reconfigurable USART IP Core with Support for Multi-Drop Networks

ALI H. EL-MOUSA, NASSER ANSSARI, ASHRAF AL-SUYYAGH, HAMZAH AL-ZUBI
Computer Engineering Department
University of Jordan
Faculty of Engineering & Technology, Amman 11942
Jordan
elmousa@ju.edu.jo, n.anssari@ju.edu.jo, a.suyyagh@ju.edu.jo, hamzah_alzubi@yahoo.com

Abstract: - Field Programmable Gate Arrays (FPGA) are increasingly becoming the mainstay of embedded systems due to their flexibility, speed, ease of use and reusability. At the same time, networking and data communications between the different parts of an embedded system and between different embedded systems, is becoming a necessity due to large complex projects. This paper presents the design, implementation, and testing results of a flexible and user reconfigurable Universal Synchronous Asynchronous Receive Transmit (USART) IP core suitable for use in embedded systems and Systems on Chip (SoC). The design scheme employed, allows the USART to be used in various modes of operation such as standalone and 9-bit addressable mode for multi-drop network of serial devices. It also supports high speed data rates of up to 3 Mb/s. The design utilizes Hardware Description Language (HDL) to describe the operation, ease implementation and allow cross platform utilization. The paper shows through a comprehensive testing methodology that the proposed design functions properly while consuming minimum resources from the target FPGA.

Key-Words: - embedded systems, addressable USART, IP core, FPGA, multi-drop networks, HDL

1 Introduction

Interest in embedded systems has grown drastically in recent years. The global market for embedded systems is expected to increase from $92.0 billion in 2008 to an estimated $112.5 billion by the end of 2013, a compound annual growth rate (CAGR) of 4.1%. Embedded hardware was worth $89.8 billion in 2008 and is expected to grow at a CAGR of 4.1% to reach $109.6 billion in 2013. Embedded software generated $2.2 billion in 2008. This should increase to $2.9 billion in 2013, for a CAGR of 5.6%. [1]. At the same time, embedded systems are increasing in complexity and more frequently they are also networked. As designs become more complex, embedded systems based on FPGA are required to interact with software running on stock commercial processors [2]. This interaction more often than not makes use of a serial communications transmission link. Design techniques based on hardware-software co-design are generally implemented on platforms that utilize FPGAs as accelerators together with embedded CPU cores for control and operational procedure definition [3]. Frequently these platforms also require high speed serial data communication blocks.

USARTs have been around for years and they have become established for easy and simple serial transmission. However, most of these take the form of hardwired specialized ICs which make them unattractive and unsuitable for use in recent embedded systems; especially those utilizing FPGA technology, since they cannot be incorporated within the HDL design. Also, most are designed with limited features and capabilities; for example: limited speed and no capability for use in multi-drop networks. Attempts at the design of an HDL-based USART have been reported in the literature. Many are just HDL implementation of the well known industry standard 16550 UART without additional features [4-6].

This paper presents the design, implementation, testing and verification of a high speed user configurable USART suitable to be used efficiently on platforms that utilize FPGAs. The architectural design allows serial communications in multi-drop networks using 9-bit operational mode using master-slave operation. It also features configurable high speed transmission rates and transmission error detection and recovery. User configuration is

accomplished through specialized internal registers. The design is suitable to be used for inter-chip, inter-processor, and inter-system communications among others. The design implements both modes of operation synchronous and asynchronous.

The rest of the paper is organized as follows: Section 2 presents a general description of USART theory of operation and describes the modes of operation. Section 3 provides a detailed description of the specifications of the developed USART. Section 4 discusses the design methodology followed and provides a description and operational features of the various blocks used in the design. Section 5 is concerned with the testing and verification procedures followed. Section 6 is for discussions and conclusions.

2 USART Theory of Operation

USARTs operate as parallel to serial converters at the transmitter's side where data is transmitted as individual bits in a sequential fashion whereas at the receiver's side, USARTs assemble the bits into complete data and thus act as serial to parallel converters.

USARTs mandate preliminary configuration of data format, transfer rates and other options specific to the design model. The user can send individual or a block of data, whose size is determined by the design model, to the transmitter section of the USART. Data is stored in the internal buffer and framed according to the transfer mode and user defined options. Finally, the frame is sent to its destination one bit at a time. On the receiver side, the data frame bits are received and sampled. The extracted data from the received frame resides in the internal receiver buffer waiting to be read by the user. The receiver monitors the reception for possible errors and informs the recipient of their existence should the proper control bit(s) be set. Most USARTs offer a status monitoring mechanism via a dedicated status register(s) through which some of the internal operation aspects can be viewed.

2.1 Modes of Operation

There are two major modes of operation in USARTs: synchronous and asynchronous modes. The latter prevails in most applications.

2.1.1 Synchronous Mode

In synchronous mode, both the transmitter and receiver are synchronized to the same clock signal which is usually generated by the transmitter and sent along with the data on a separate link to the receiver. The receiver in turn utilizes the received clock to extract the timing sequence and to determine the beginning and end of the received bit interval. Therefore, the receiver knows when to read the bit's value and when the next bit in the sequence begins. However, when the line is idle (i.e. no data is exchanged), the transmitter should send a fill character in order not to lose synchronization.

A synchronous communication session begins by sending one or more synchronizing frames to indicate the beginning of transmission then the sequence of data frames follow (a parity bit is appended to the end of the data frame if single error detection is required), transmission is terminated by sending a stop frame after which the line returns to the idle state.

A point in favor of synchronous communication is that it is data efficient especially for long messages since data bits are only sent thus preserving bandwidth in contrast to the asynchronous mode discussed shortly. Also, the required hardware is simpler. Nevertheless, despite the efficient data transmission rate, the overall bandwidth requirements are high due to the clock's bandwidth demands. Moreover, synchronous transmission is more costly since it requires extra wiring, limiting its usage to short distances. Furthermore, synchronous communication is susceptible to clock skew which produces timing discrepancy and reduces the reliability of data transfer though such discrepancy can sometimes be accommodated [7].

2.1.2 Asynchronous Mode

In asynchronous mode of communication, the transmitter and receiver are preconfigured to the required timing parameters in advance and special bits are appended to the data frame for synchronization purposes. Thus timing is embedded in the frame and the need to send a timing signal to the receiver is eliminated.

In asynchronous mode, an idle line remains at a predetermined level. The frame consists of a *start bit* which differs in polarity to that of the line's idle state, followed by the data bits and a parity bit - if single error detection is used - and ends with at least one *stop bit* which has the same polarity as that of an idle line. A stop bit might be followed by another frame – back to back transmission - or an idle state of transmission. Both the transmitter and receiver are preconfigured to run at the same fixed clock rate which is an exact multiple of the required baud rate. Once the receiver recognizes the transition from the

idle state polarity to the opposing polarity, it waits a half bit interval duration and verifies the presence of a start bit, if start bit arrival is confirmed, the receiver reads the values of the bits every full bit-width interval until the reception of a stop bit is confirmed denoting the end of the frame. The receiver resynchronizes its clock repeatedly at the start of every frame thus tolerating any slight discrepancy in frequency between the transmitter and receiver.

Asynchronous communication is advantageous when data is sent sporadically and when it is costly to spare a discrete link for timing purposes. On the other hand, asynchronous communication is bandwidth inefficient when it comes to sending large blocks of data. It is mostly used in medium and long distances [8].

2.2 Common Communication Errors Encountered in USARTs

The most common types of error encountered in USARTs are parity, framing and overrun errors. A parity error indicates that noise affected the data during transmission; such errors occur frequently in hostile environments especially when cables are improperly shielded. A framing error indicates that the start and stop bits are not in their proper places which is mainly due to different baud rate configurations at the transmitter and receiver sides. Finally, the overrun error indicates that data have been lost in the receiver side because the internal reception buffer is full [9].

3 Specifications of Developed USART

The specifications included in the design were chosen to meet modern serial communication demands of high performance and reliability, taking compatibility with legacy devices into consideration.

Performance oriented features include an interrupt driven approach and universal eight bit addressability which make the USART ideal for industrial and control applications. Eight-level buffering allows for data block transfers that is beneficial considering the high speeds the USART can handle in data transfer which can reach 3 MHz.

Reliability oriented specifications include the programmable odd/even parity bit with the ability to detect parity, framing and overrun errors.

To adapt to modern practices, the USART offers eight bit mode synchronous or asynchronous communication, variable stop bit options and full duplex mode. However to retain compatibility, it also offers five to seven bit transfer and half duplex mode of communication. Table 1 lists the detailed specifications of the proposed USART.

4 Design Methodology of the USART System

The methodology adopted in carrying out the USART system design was based on systems and software engineering approaches. It used a variation of both the waterfall and incremental approaches suited to the design environment and constraints. The design steps that the system went through are [10]:

1. System Requirements Definition. The requirements definition phase specified the functionality as well as the essential and desirable system properties. This involved the process of understanding and defining what services were required from the system and identifying the constraints on system operation and development.

2. System/Subsystem Design. This phase was concerned with how the system functionality was to be provided by the components of the system and where the system specification was converted into an executable system specification.

3. Subsystems Implementation and Module Testing. The subsystems identified during subsystem design were implemented and mapped into hardware code using the Verilog Hardware Descriptive Language HDL. In this critical stage, individual modules were extensively tested for correct functional operation. Each component, once implemented, was tested independently without the other system components and the assessment of the functional behavior was concluded from the simulation output.

4. System Integration. During system integration, the independently developed subsystems were merged together to build the overall USART system in an incremental approach.

5. System Testing. The overall integrated system was subjected to an extensive set of tests to assure correct functionality, reliability and performance. The tests were aimed to test the behavior of the system as a whole in addition to the interfacing between the subsystems.

Verilog HDL language was used to develop and simulate the USART system and subsystems under Xilinx ISE 8.2i environment [11]. USART modules were designed and verified separately, and then they were integrated together. Fig. 1 shows the major parts of the USART system:

Table 1: Functional Specifications of the USART

Specification	Justification
Support for the following transmission modes (Programmable): • Asynchronous (Full/Half duplex modes) • Synchronous (Full/Half duplex modes)	Full duplex mode is employed in modern systems while half duplex mode is retained for compatibility with legacy systems.
Supports a wide range of transmission/reception rates (from 50 Hz to 3MHz)	High frequencies are essential for high speed communication. Lower speeds are needed to communicate with older USARTs. Moreover, lower speeds can be used to minimize cross talk if similar links are adjacent.
Eight-level Transmitter/Receiver Buffer	To account for the high speeds of communication that the USART can reach, blocks of data can be received and buffered until read by the user. Also, this allows the user to issue the transmission of a block of eight-frame size in a single operation. This will also reduce the load on the module that controls the USART operation in the system.
Parity Supported (Programmable – Enable/Disable parity and Odd/Even parity).	Single error detection techniques might prove beneficial in noisy operation environments.
Variable data lengths supported (Programmable - five to eight bits)	Byte communication is the modern norm. Five to seven bits data length is to retain compatibility with legacy systems.
Variable stop bits supported (Asynchronous mode) (Programmable – One or two stop bits)	This is to comply with the RS232 standard where two stop bits mode is used to accommodate slightly different clocks in the transmitter and receiver sides when USARTs from different vendors are connected together.
Error Detection of the following errors: • Parity Error • Overrun Error • Framing Error (Asynchronous mode)	Parity error detection provides a measure of the reliability of communication. Framing error detection indicates the necessity of reconfiguring the internal clocking sources at both ends correctly. Finally, overrun error informs that data has been lost and the need to frequently read the received data.
Interrupt Support (Programmable – with ability of Global Masking)	Most modern systems are interrupt-driven for the reason that interrupt techniques save processing clock cycles in comparison with polling techniques and are essential in real time applications.
Supports Addressability (8-bit Universal – Addresses up to 256 devices) while sending 9-bits.	Widely used in industrial and control applications in multi-drop networks where a master USART can communicate with a certain other slave USART(s).

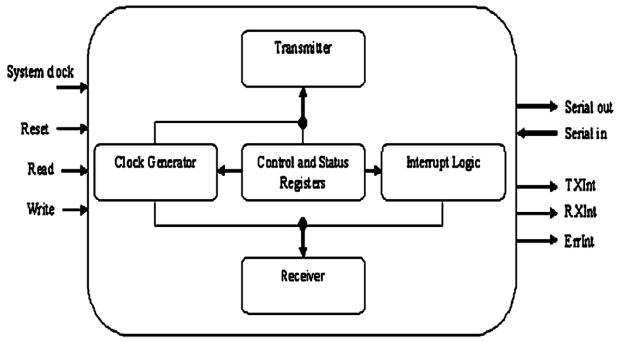

Fig. 1: The USART module

The input/output signals involved are:
System clock: The oscillator clock
Reset: Master reset of the system.
Serial out: Transmitted data
Serial in: Received data
TXInt: Transmitter interrupt
RXInt: Receiver Interrupt
ErrInt: Error Interrupt

The following sections show the design details of the different subsystems used in the USART.

4.1 Control and Status Registers

Table 2 describes the functionality of all registers used in the design including the justification for their use.

Table 2: Control and Status Registers

Reg. Name	Reg. Address	Type*	Width (bits)	Functionality	Justification
Control register 1	0	W	8	Used to select mode of operation, frame length, parity and stop bits options, interrupts enabling and 9th bit mode selection	To Accommodate the multiple options and modes supported in the system
Control register 2	1	W	6		
LS Divisor	2	W	8	Used to determine the clock needed for sampling and the clock needed for transmission (baud rate clock)	Using 16-bit divisor gives wide range of baud rates to select among. This allows the system to be able to support wider range of requirements.
MS Divisor	3	W	8		
TX Buffer	4	W	8	Seven-location buffer used to store the frames wanted to be transmitted until the transmit shift register is empty (ready).	To allow the user to send a block of seven frames each time the buffer is empty.
RX Buffer	5	R	10	Seven-location buffer used to store the received frames until the user reads them.	Decreases frame losses, thus increases system reliability.
Status register	6	R	7	Describes the status of the entire system. It indicates the emptiness of the transmitting buffer, fullness of the receiving buffer, parity error existence, framing error existence and overrun error occurrence.	In many cases, it is not possible for the interrupted system to respond directly for the interrupt signal. Thus, the status register is used such that it can be checked whenever it is possible.
Address register	7	W	8	Stores the address of the USART chip. This address is written by the user (not fixed for a certain chip).	Used when 9th bit mode is selected. The TX must send the address of the destination before transmitting data.

*W indicates writable while R indicates readable register

4.2 Clock Generator

The USART system contains a programmable clock generator. Inputs to this module are the system clock and the values of the two divisor registers. Fig. 2 shows a block diagram of the clock generator.

This module is designed to generate a square clock irrespective of the divisor value (odd or even divisor). In synchronous mode of communication, this clock is transmitted along with the data. Also, it is used to generate the baud-rate clock, through a division by 16.

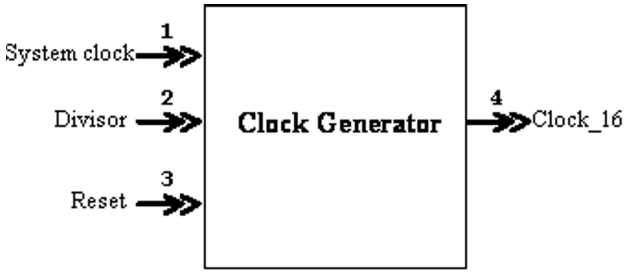

Fig. 2: The clock generator

Divisor: 16-bit value initialized by the user.
Clock_16: 16*baud-rate clock.

The output frequency of the clock generator is determined by (1):

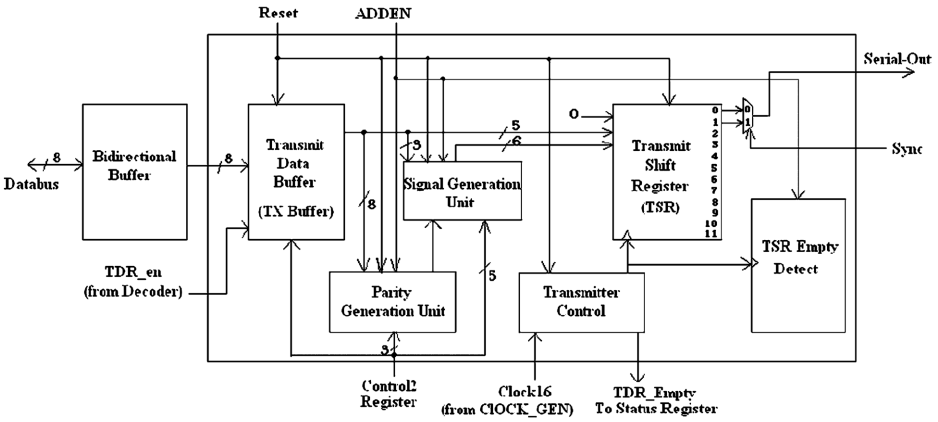

Fig. 3: The transmitter module

$$Clock\_16 = 16 * \text{Calculated Baud Rate} = \frac{fosc}{X} \qquad (1)$$

Where X is a 16-bit integer divisor loaded into the two divisor registers to obtain the desired baud rate. Setting the divisor registers to the value of $(1)_{16}$ yields the maximum clock frequency. Setting the divisor to zero yields the minimum clock frequency (divisor is 2^{16} in this case).

4.3 Transmitter Module

Fig. 3 shows the functional block diagram of the transmitter module.

It consists of the following sub modules:

A- Transmitter buffer
B- Parity Generation Logic
C- Bypass logic
D- Shift logic
E- Transmit shift register (TSR)
F- TSR empty detection logic

A- The Transmitter Buffer

The transmitting buffer is the memory where data to be sent are stored waiting for the transmit shift register (TSR) to be empty. It becomes an essential component when the inter arrival time of transmitted frames becomes very small. Moreover, when the system is accessed using a processor/DSP that operates at a higher frequency than the transmission baud clock, this buffer will reduce the number of times the processor is interrupted to request for new data. The signal TDR_empty is generated to indicate that the buffer has at least one empty slot. Fig. 4 shows the input/output signals associated with the transmitting buffer while Fig. 5 shows its data flow diagram.

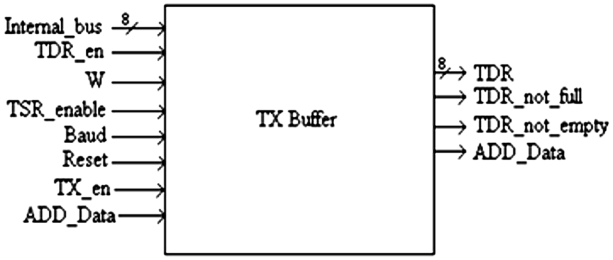

Fig. 4: The I/O signals of the TX buffer

B- The Parity Generation Circuit

Parity generation logic circuit reads the frame to be sent and produces the appropriate parity bit according to the predetermined word length and parity type in the second control register. The maximum frame length possible is 12 bits (1 start bit + 8 data bits + parity bit + 2 stop bits). Fig. 6 shows the I/O signals associated with the parity generator while Fig. 7 shows its data flow diagram.

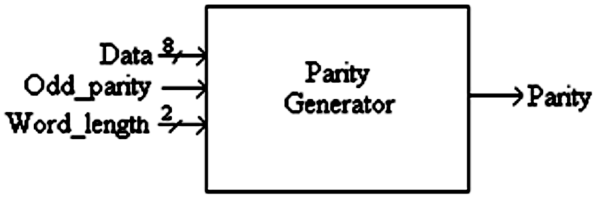

Fig. 6: I/O signals for the parity generator

C- The Bypass Logic Circuit

The bypass logic circuit is used to insert the most significant 6 bits of the data frame in the transmit shift register depending on the control options selected by the user in the appropriate control registers (word length, parity enabled/disabled and number of stop bits).

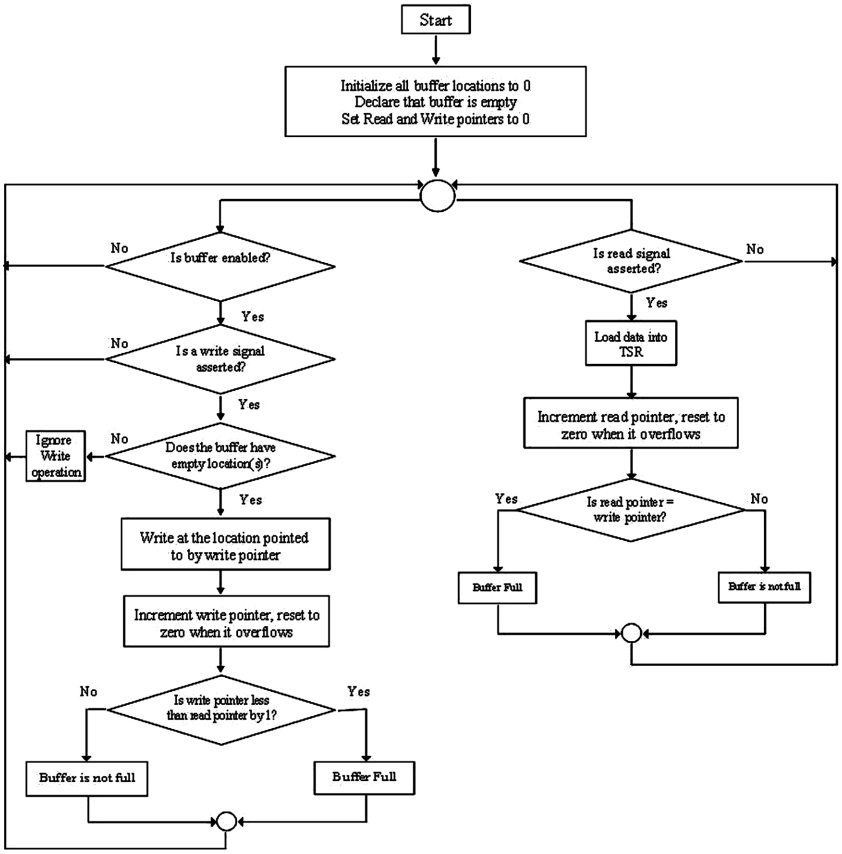

Fig. 5: Transmitting buffer dataflow diagram

If 9th bit mode is used, address/data bit is inserted into the frame instead of the parity bit irrespective of whether parity is enabled or disabled. The least significant 5 bits of the frame in addition to the start bit in asynchronous communication mode are passed directly from the transmitting buffer to the TSR register. Fig. 8 shows the dataflow diagram associated with the bypass logic circuit and Fig. 9 the I/O signals for the same circuit.

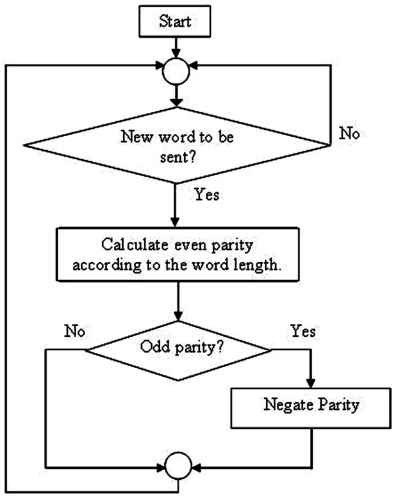

Fig. 7: The parity generation flow diagram

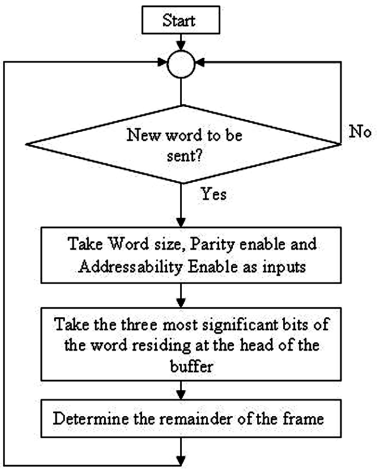

Fig. 8: Dataflow for the bypass circuit

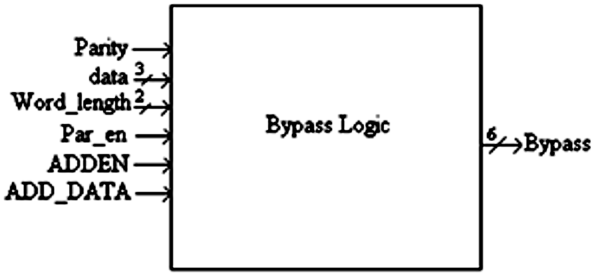

Fig. 9: I/O Signals of the bypass Circuit

D- The Shift logic circuit

The Shift Logic Module is inherently a control module which produces the main control signal of the transmitter module: *TSR_Enable*. This signal is used to indicate that both the TSR is empty and that data exists in the buffer for transmission. This signal is vital for the proper operation of the transmitter. It is used to accomplish the following:

1. Indicate when to load the Transmit Shift Register with data.
2. Advance the read pointer of the transmitter buffer to point to the next data to be transmitted.
3. Reload the 'detect empty TSR logic' internal counter to indicate the next frame size.

This sequence of operation allows the transmission of back-to-back frames. Also, it is further used to derive the baud rate from the *clock_16* clock produced by the Clock Generator module by dividing it by 16. Fig. 10 shows the dataflow diagram associated with the Shift circuit and Fig. 11 shows its I/O signals.

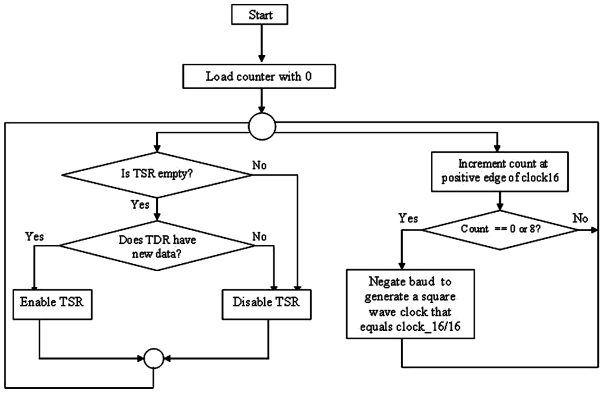

Fig. 10: Dataflow diagram of the shift circuit

Fig. 11: The I/O signals for the Shift Circuit

E- Transmit shift register (TSR)

The Transmit Shift Register or TSR is the parallel to serial converter which is responsible for sending the data in a serial fashion. It is the core of the transmitter communication circuit. Upon initialization, the TSR is loaded with ones. Therefore, the communication line is in the mark state and the TSR is declared empty. Furthermore, the TSR is clocked at the desired baud rate; this means that sixteen cycles of the *clock_16* are required for each bit shifted out the serial output. The baud rate is generated by the shift logic module. The frame contains the data bits and parity (if enabled) in synchronous mode. A Start bit and one or two Stop bits are appended in their appropriate positions into the frame in asynchronous mode. The frame can take a maximum value of 12 bits (1 start bit, 8 data bits, 1 parity bit and 2 stop bits in asynchronous mode) and a minimum of five (Only five data bits in synchronous mode). When the frame size is less than twelve, a string of one's fills the empty positions in the TSR to allow the line to go to the mark state if only one frame is sent. Fig. 12 shows the dataflow diagram associated with the TSR, while Fig. 13 shows the I/O signals.

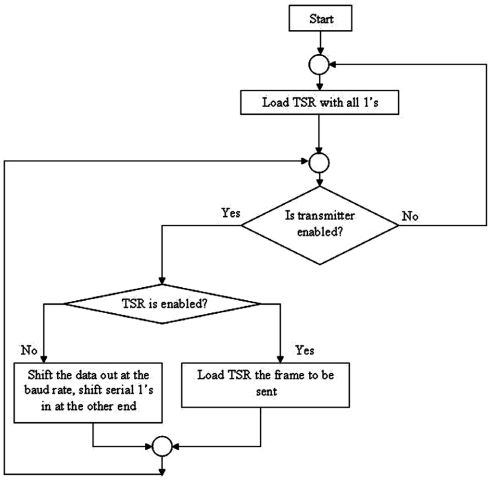

Fig. 12 The dataflow diagram for the TSR

Fig. 13: The I/O signals associated with the TSR

F- TSR empty detection logic

This module is responsible for indicating when the TSR is empty (upon initialization or after frame transmission). The TSR state is passed to the Shift Logic module to enable loading the TSR if data exists in the buffer. The TSR Empty Detect Logic utilizes an internal 32* 4-bit ROM (look-up table) which holds all the possible frame sizes for all modes of operation. The ROM is accessed by a 5-bit address composed of the combination of the following control bits in sequence: Synchronous mode, word length, parity enable and number of stop bits. If addressability mode is selected, the above control bits sequence will be reduced to: Synchronous mode, number of stop bits. This is due to the design choice that addressability is applicable

only in 8-bit word length mode and to the notion that the parity bit is replaced by the ADD-Data bit. The frame size is retrieved in a look-up table manner. Fig. 14 shows the dataflow diagram associated with the TSR empty circuit, while Fig. 15 shows the I/O signals.

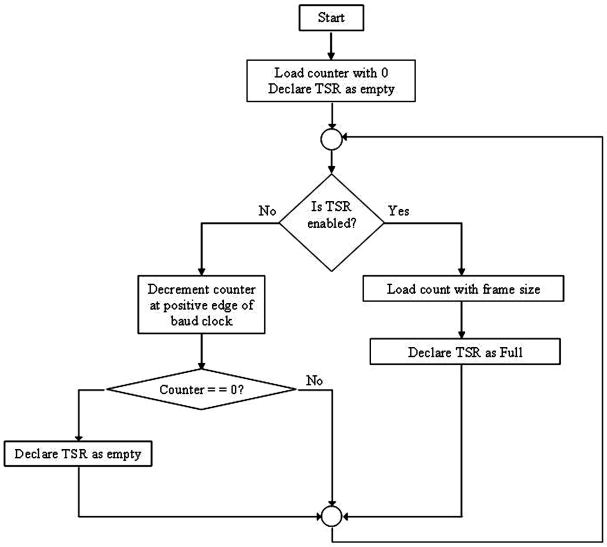

Fig. 14: The TSR empty dataflow diagram

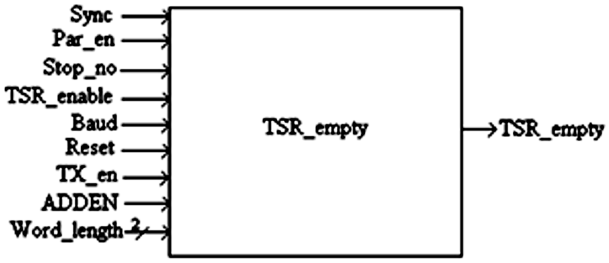

Fig. 15: The I/O signals of TSR empty circuit

4.4 Receiver Module

The receiver subsystem operates much in the same way for both the synchronous and asynchronous modes of communication except for the start and end of the reception detection mechanism.

In the asynchronous mode, the receiver waits for a transition from a mark to space (Logic "1" to Logic "0") after an idle line state or an expected stop bit to initiate the data reception logic provided that the transition is not caused by a noise notch. This is ensured by sampling each of the received bits at three different times and then using a majority detection circuit. The end of asynchronous reception is detected at the frame level by waiting for a stop bit at the end of the frame.

However, in the synchronous mode, instead of waiting for logic transition, the receiver waits for a synchronizing character which if received after an idle state line, a start of reception is detected. In the same way, an end of reception is signaled if a certain stop character is received.

When data reception is detected at the serial_in input, the internal receiver logic is enabled and the received data bits are serially shifted into the Receiver Shift Register (RSR). Meanwhile, the parity is calculated per each received bit for the received *word* and finally compared to the received parity value. Parity and framing errors are evaluated and stored along with the data in the receiver buffer. The buffer can hold up to seven received words, if data are received while the buffer is full, the data is dropped and an overrun error is indicated.

If 9-bit address detection mode is enabled, the previous scenarios for synchronous and asynchronous transmission modes still hold but with slight modifications; one of which is that transmission is fixed at eight bits and that the ADD-Data bit is substituted for parity. The address of the receiving node must be received with ADD-Data bit is set to "1" in order for the frame to be considered an address frame. The address frame is handled as any other frame sent using asynchronous mode in terms of having a start and stop bits.

In the synchronous addressable mode of operation, a synchronizing character with ADD-Data bit value set to zero must be initially received, followed by a frame containing the address of the receiving node but with ADD-Data bit value set to one, followed by the data frames with ADD-Data bit reset again. Fig. 16 shows the functional block diagram of the receiver module, which consists of the following sub-modules:

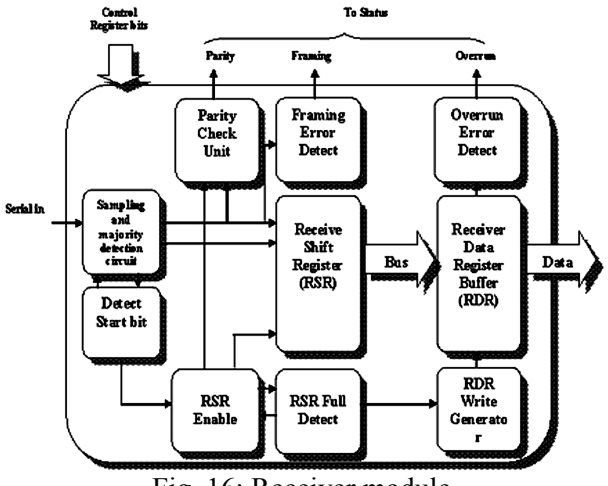

Fig. 16: Receiver module

A- Sampling and Majority Detection Logic
B- Detect Start Bit Logic
C- Receiver Shift Register Enable Logic
D- Receiver Shift Register (RSR)
E- Receiver Shift Register Full Detection Logic

F- Parity Error Detection Logic
G- Framing Error and Stop Bit Detection Logic
H- Receiver Buffer Write with Synchronous and Stop Character Detect Logic
I- Receiver Buffer with Overrun Detection Logic

A - Sampling and Majority Detection Logic

This sub-module detects the beginning of a possible new data reception indicated by a transition from a Mark (idle) state to a Space. Once genuine start of reception is detected, each received bit is sampled three times before its value is determined. This ensures correct decoding of data and minimizes noise effects. This sampling process is illustrated in Fig. 17.

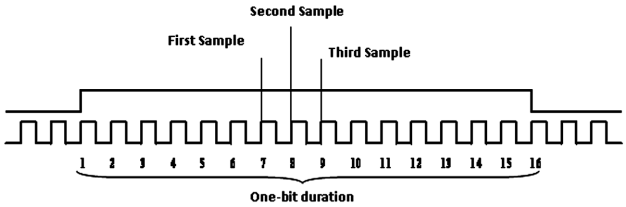

Fig. 17: Sampling operation.

This sub-module extracts the baud rate clock used in the receiver section of the USART, either from the locally generated system clock in asynchronous mode or from the received clock input in synchronous mode. Fig. 18 shows the transmit shift register data flow diagram.

B - Detect Start Bit Logic

This sub-module signals the detection of the first 0 bit on an idle line. This indicates either the start bit in asynchronous mode, or the start of a possible synchronizing character in synchronous mode. Fig. 19 shows the Detect Start Bit Logic data flow diagram.

C - Receiver Shift Register Enable Logic

This sub-module enables the Receiver Shift Register (RSR) to accept received data bits. This occurs when a start bit is detected in asynchronous transmission mode. In synchronous mode, the RSR is always enabled, but received bits are ignored until a valid sync character is received. Fig. 20 shows RSR Enable Logic data flow diagram

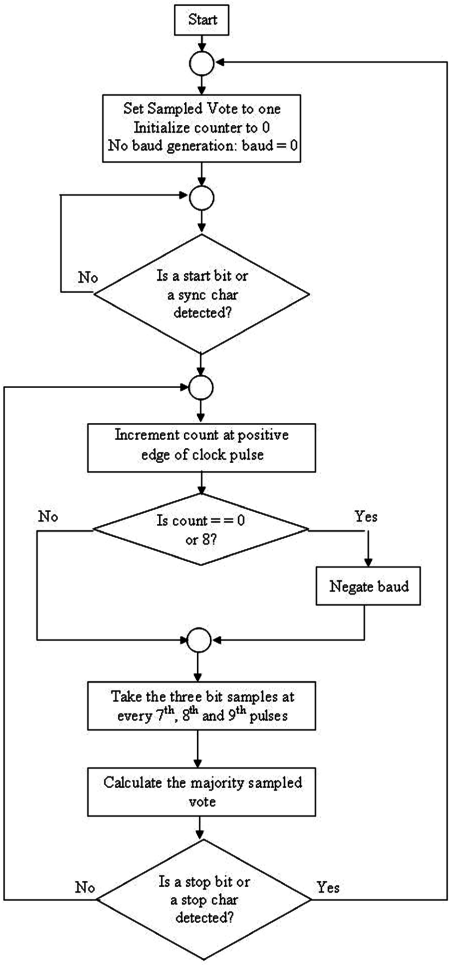

Fig. 18: Sampling and majority detection dataflow diagram.

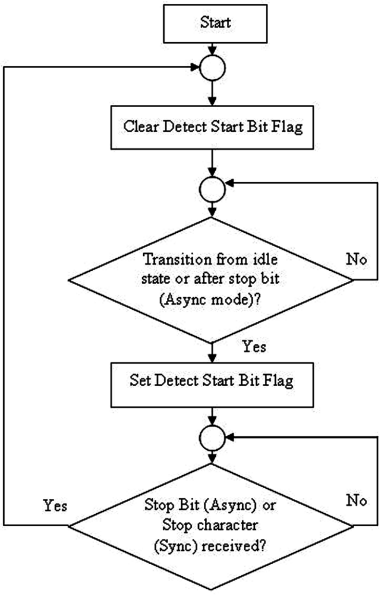

Fig. 19: Detect start bit dataflow diagram.

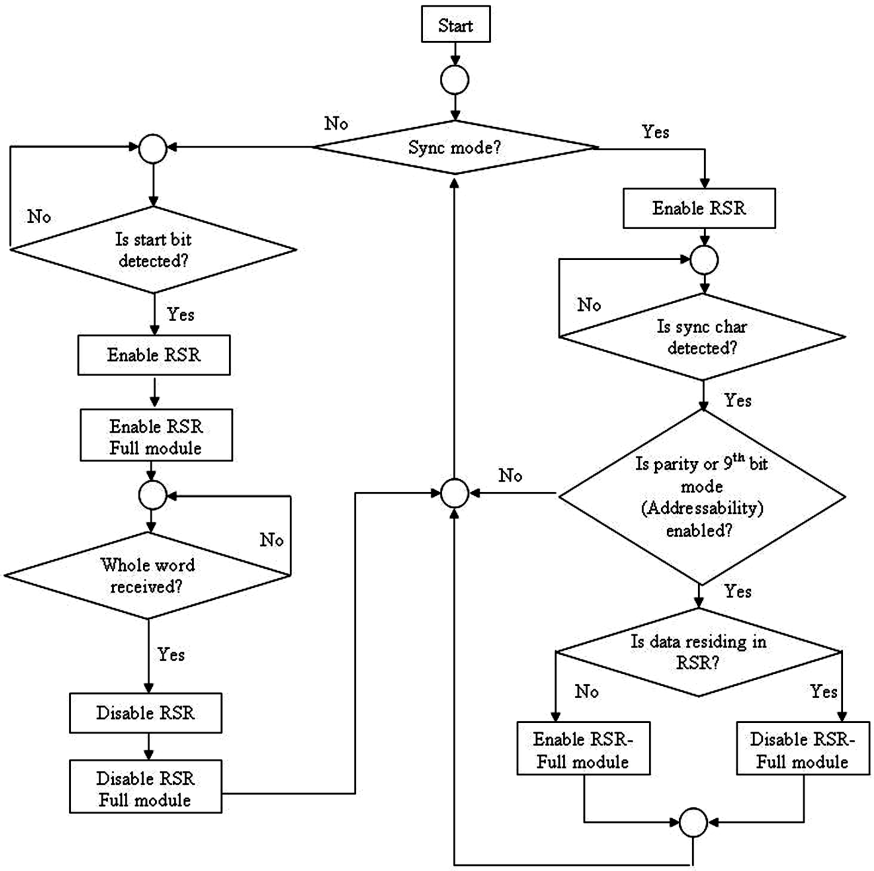

Fig. 20: RSR enable data flow diagram.

D - Receiver Shift Register (RSR)

The Receiver Shift Register (RSR) is the serial-to-parallel converter which is responsible for receiving serially transmitted data. It is the core of the receiver communication circuit. RSR is a Serial In – Parallel Out 8-bit shift register which is clocked at the baud rate generated by the Sampling and Majority Detection Logic. Fig. 21 shows the RSR data flow diagram.

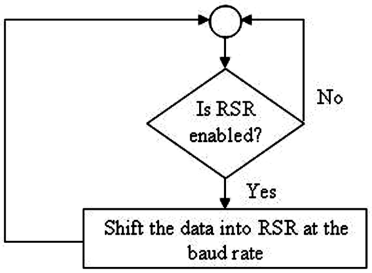

Fig. 21: RSR dataflow diagram

E - Receiver Shift Register Full Detection Logic

This sub-module indicates the reception of a complete frame by the RSR. It is a countdown counter operating at the baud rate and initialized to the predetermined frame size. Fig. 22 shows the RSR full detection logic data flow diagram.

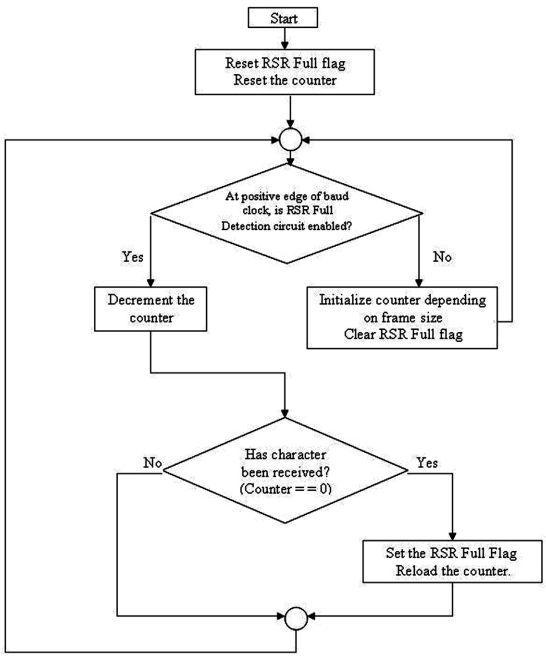

Fig. 22: RSR full detection dataflow diagram

F - Parity Error Detection Logic

This sub-module checks for single-bit errors that occur during transmission. It calculates the expected parity bit for each received frame and compares it with the actual parity bit received. If the extracted

and received parity values do not match, the module flags a parity error to indicate this incidence. This sub-module calculates parity commutatively instead of waiting for receiving the whole frame. This way parity error can be detected as soon as the parity bit arrives. Fig. 23 shows the data flow diagram for this sub-module.

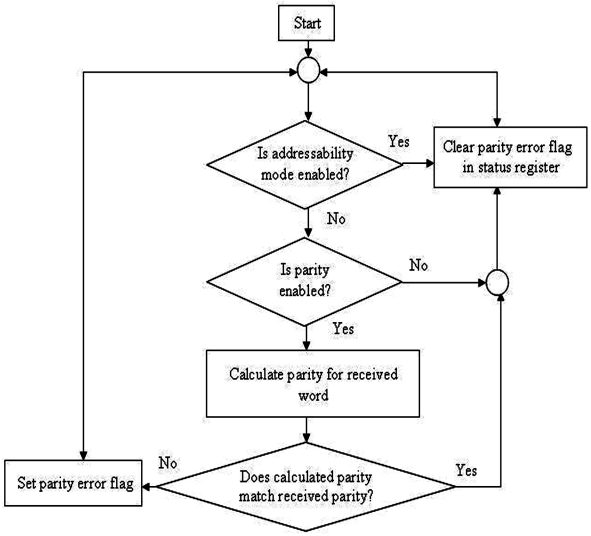

Fig. 23: Parity error detection dataflow diagram

G - Framing Error and Stop Bit Detection Logic
This sub-module is responsible for detecting the end of a received frame (the stop bit) as well as signaling framing errors in asynchronous mode of communication. Fig. 24 shows the data flow diagram for this sub-module.

H - Receiver Buffer Write with Synchronous and Stop Character Detect Logic
This sub-module is responsible for generating an internal signal to write the received frames in RSR, together with their corresponding parity and framing information, into RDR buffer only if certain conditions are met. In synchronous mode of operation, this involves checking for the reception of valid sync and stop characters that delimit a block of consecutive frames. If 9th bit mode is used, all received frames are dropped if the USART is not the targeted node, which is indicated by receiving a valid address frame prior to receiving data. Fig. 25 shows the data flow diagram for this sub-module.

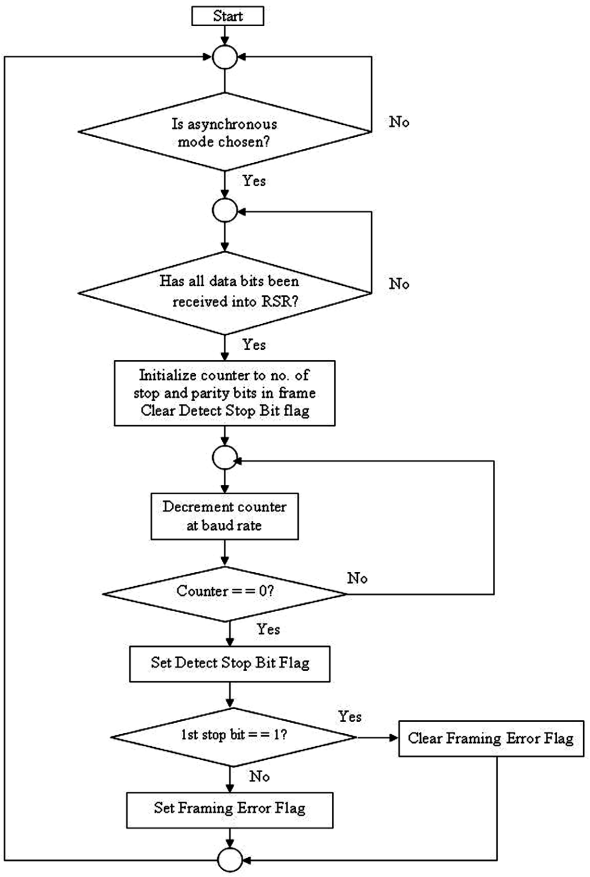

Fig. 24: Framing error and stop bit detection dataflow diagram

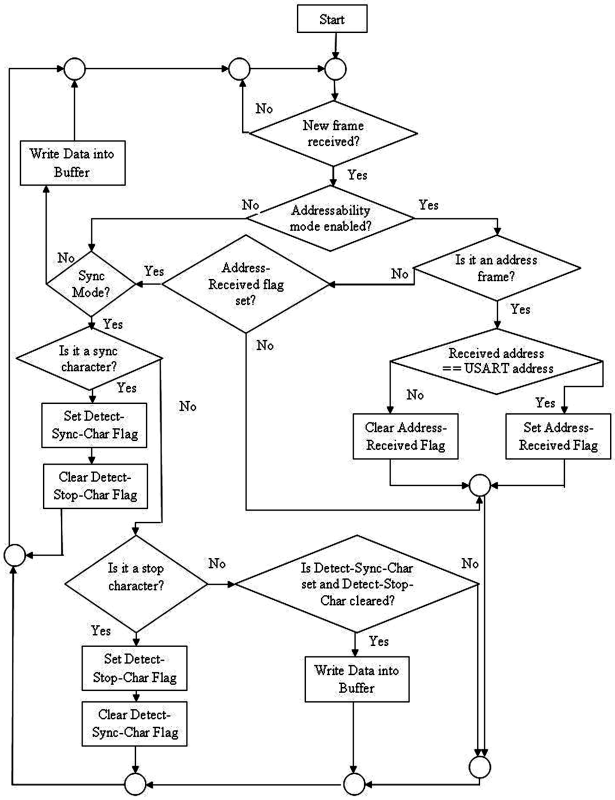

Fig. 25: Receiver buffer write dataflow diagram

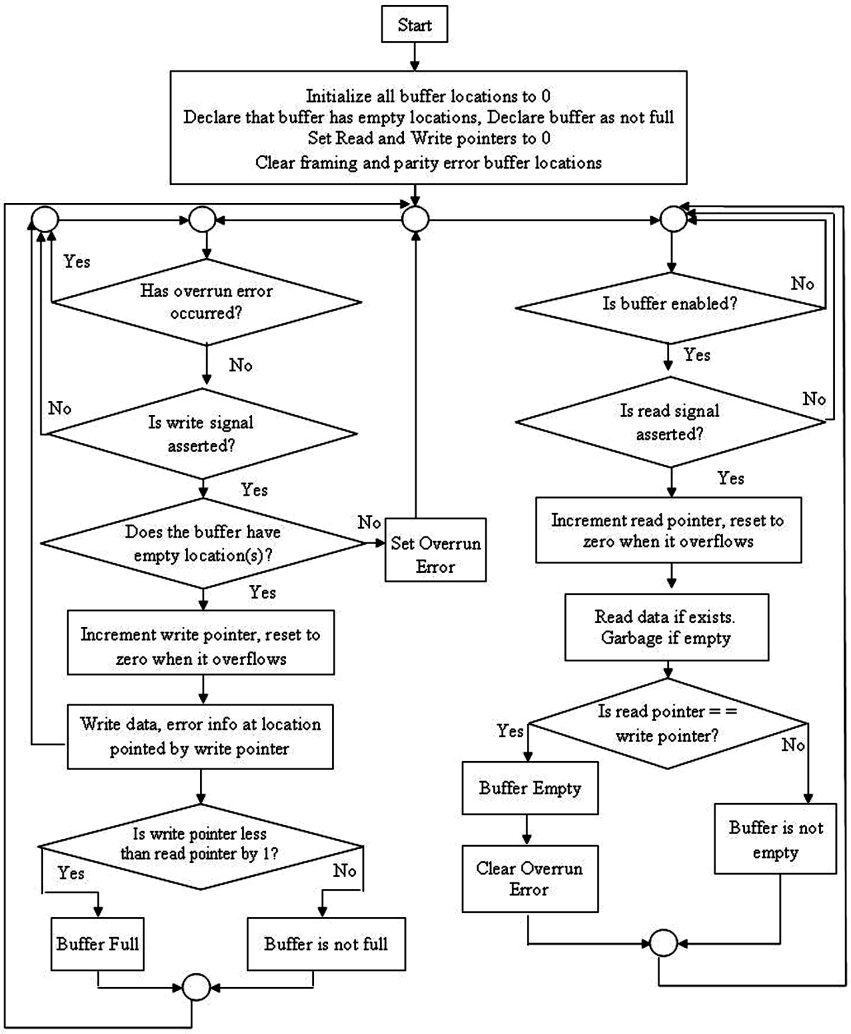

Fig. 26: RDR dataflow diagram

I - Receiver Buffer with Overrun Detection Logic
The receiver's buffer is composed of an array of eight registers which is controlled in a circular fashion. The number of levels is chosen to account for the high speeds the USART can reach in operation and the need to buffer the data at high speeds of operation. For each level, the parity and framing error status is saved and the whole buffering system is monitored for overrun errors. Fig. 26 shows the data flow diagram for this sub-module.

4.5 Interrupt Logic

Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rates. Unlike polling techniques, interrupt processing allows the system to execute other operations while the USART device is in the process of sending or receiving data thus sparing processor cycles.

Interrupt signals are generated when one of the following actions happens:

1. Transmitter buffer becomes empty after being completely full (one empty slot can declare the buffer as empty).
2. New data frame is received and written on the receiving buffer. This means that if the buffer is full, overrun error will be declared and the received frame will get lost.
3. When one or more of the parity, framing or overrun errors occur(s).

All interrupts can be enabled or disabled individually or collectively by masking their corresponding bits in control register 1.

To deliver these interrupt signals to the external system, three different pins are used instead of one pin for all interrupts. This is especially useful when full duplex mode is used since it can be used to

determine the interrupt source by knowing the pin which generates the signal instead of polling the status register.

5. Testing and Verification Procedures

5.1. Introduction

Most systems undergo testing throughout their development and before they are delivered to the customer. In general, the system testing process has two distinct goals:

1. To demonstrate to the developer and the customer that the system meets its requirements.
2. To discover faults or defects in the system where the behavior of the system is incorrect, undesirable or does not conform to its specification [10].

The first goal leads to validation testing, where the system is expected to perform correctly using a given set of test cases that reflect the system's expected use. The second goal leads to defect testing, where the test cases are designed to expose defects. The test cases can be deliberately obscure and need not reflect how the system is normally used. For validation testing, a successful test is one where the system performs correctly. For defect testing, a successful test is one that exposes a defect that causes the system to perform incorrectly.

Tests fall into two general types – black box and white box tests. Black box tests are those that are performed without any internal knowledge of the system. In a black box test, the testing is typically conducted by changing the inputs and observing the system outputs. White box tests are conducted with knowledge of the internal working of the system. Therefore, knowledge of internal operation may influence how the test is constructed [12].

Yunshan and Marshall [13] formalize design verification as a model checking problem, where an implementation is modeled as a finite state machine with a set of properties. Each property typically consists of an assumption and a guarantee, and the verification task is to prove that the guarantee holds for the model under the corresponding assumption. For a design under test, there are two common sources of guarantee properties: assertion, where the design is tested as a white box, and reference model, where the design is tested as a black box. However, in cases where it is impractical to build an abstract reference model that represents all possible implementations, a gray box approach is adopted where most of the reference model is derived from the specification while certain signals from the design under test are treated as witnesses.

Yunshan and Marshall apply this verification technique on a generic UART design using a compositional methodology. The UART is broken into a number of blocks, and each of these blocks is verified on its own before their integration to prove that its guarantee properties hold upon their predefined assumptions. However, unlike verification techniques such as directed simulations where block test-benches are often wasted after block verification, the results obtained during this stage are subsequently utilized in interface verification, where it is proven that the environment satisfies these predefined assumptions, and integration verification, where the verified blocks are abstracted and replaced with simpler models that are in turn tested as a single unit.

Wohlin and Regnell [14] suggest a modeling approach suitable for reliability certification of modular systems. Modularization is pin-pointed as being suitable for reuse, but modules are often not reused if their reliability cannot be guaranteed. Therefore, it is essential to realize that reliability certification is a must when discussing reuse. The reliability must reflect the intended usage of the module, as a module may be viewed as being reliable for one purpose and unreliable for another depending on the intended usage of the module. Therefore, the ability to certify software during testing is based on a user-oriented approach. This requires a model of the anticipated usage of the software and quantification of the expected usage as the software is released. Wohlin and Regnell propose a classification of components to simplify the derivation of such usage profiles, and identify several approaches to determine system reliability based on knowledge about the components.

Yeandel, Thulborn, and Jones [15] implement an on-line testable UART using IFIS (If it Fails It Stops) approach. This approach has several advantages such as eliminating the need to generate and apply test vectors to propagate failures to an observable primary output since any internal fault is automatically propagated to a primary system output. Moreover, since there is only one manifestation of failure using this approach, that is a circuit node does not change its value when expected, fault identification is made easier through having to observe whether a node changes its values, rather than continuously compare actual and expected values. Finally, since the effect of a failure is propagated through all circuit elements causing them all to halt, failure detection can be achieved by merely observing any primary output. However, these benefits are not obtained cost free. Overheads are incurred in terms of both circuit complexity and

slower operation which may be inappropriate for applications where the highest operational speed is required or where device complexity is important. Since this is exactly the case for the system in hand, where both high speed and minimal size were taken into consideration, this approach could not be adopted.

The development of the described USART system was based on an incremental approach. The system specification, design and implementation were broken into a series of increments that were each developed in turn. In this incremental development process, the services to be provided by the system were first identified. A number of delivery increments were then defined, with each increment providing a subset of the system functionality. Once the system increments had been identified, the requirements for the services to be delivered by each increment were defined in detail, and those increments were developed.

This incremental approach to development allowed each increment to be tested as it was developed, with these tests based on the requirements for that increment. This represented a preliminary stage in the testing process; unit testing. As new increments were developed, they were integrated with existing increments so that the system functionality improved with each increment. At this stage, testing was concerned with finding errors that result from unanticipated interactions between components and component interface problems. During these two stages of the testing process, unit testing and integration testing, some of the concepts and methods presented by Yunshan and Marshall mentioned above were utilized. For instance, each component in the system was tested individually, where some were tested as a black box while others were tested as a white box. The verified components were then combined and treated as a single unit in the subsequent tests.

Once the system was completely integrated, it was tested for fulfillment of its requirement specifications. This stage of the testing process; acceptance testing, continued until it was verified that the final system represented an acceptable implementation of its requirements. Finally, as suggested by Wohlin and Regnell above, the reliability of the system was investigated. Not only was the system subjected to a set of tests that reflected the expected mix of services that it should handle, but also tests were designed to expose it to its operating limits to check its behavior and response.

The proposed USART design was implemented and tested using firmware from Xilinx Corporation. For software, the free ISE Webpack version 8.2i was used [16]. As for hardware, different development kits were used throughout the stages of the project. These include: Digilab 2 XL (D2XL) Development Board [17], Digilent Spartan-3 System Board [18], and Spartan-3E Starter Kit Board [19]. However, since the design was entirely implemented using a universal hardware description language, Verilog HDL, it is expected to be directly interoperable with any environment provided by other vendors.

5.2 Testing Process

In general, the project went through several phases during the testing process as illustrated in Fig. 27:

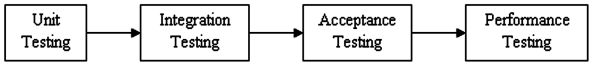

Fig. 27: Phases of the testing process

5.2.1 Unit Testing

A unit test is a test of the functionality of a system module in isolation, and the unit test should be traceable to the detailed design. A unit test consists of a set of test cases used to establish that a subsystem performs a single unit of functionality to some specification. Test cases should be written with the express intent of uncovering undiscovered defects [12].

5.2.1.1 Baud Rate Generator

Actual snapshots were taken for clock16 and baud-clock signals using Agilent DSO5014A oscilloscope. One of these is shown in Fig. 28.

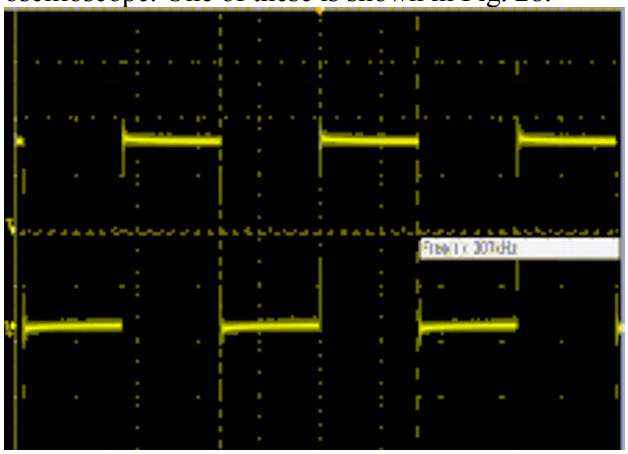

Fig. 28: Clock16 signal for divisor = 00A3H

5.2.1.2 Buffers

Seven-level buffers are used in the transmitter and receiver module to hold data words that are to be sent or that has already been received. Since both buffers fundamentally have the same architecture,

then the test results obtained from one would be equally applicable to its counterpart. Fig. 29 illustrates the operation of the buffer.

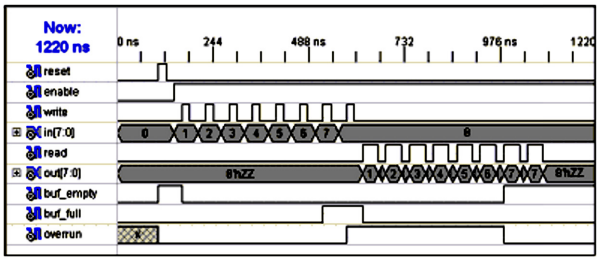

Fig. 29: Simulation of the buffer

At the beginning of the simulation, buf_empty output was high indicating that the buffer was empty. As soon as the first word was written into the buffer at the positive edge of the write signal, buf_empty went low. When another six words were written into the buffer, buf_full signal became high. The next word attempted to be written into the buffer caused overrun signal to go high indicating an overrun condition that caused that word to be lost. Next, seven read pulses were applied to the buffer to investigate its contents. The words previously written into the buffer sequentially appeared on the data bus at the positive edge of each read pulse. The buffer was disconnected from the data bus when the read signal became inactive.

When all the contents of the buffer were read, the overrun signal went low while buf_empty signal went high. The last word written into the buffer before the overrun condition appeared on the data bus again upon the application of an eighth read pulse. This indicates the eighth word applied at the input of the buffer was not written into it since writes are disallowed if the buffer is full.

After confirming from the simulations that the buffer operated as it is supposed to do, the buffer was implemented on the development board, and the test results conformed to the previous simulations for a wide range of frequencies.

5.2.2 Integration Testing

After the units of a system have been constructed and tested, they must then be integrated into larger subcomponents leading eventually to the construction of the entire system. The intermediate subsystems must be tested to make sure that components operate correctly together. The purpose of integration testing is to identify errors in the interactions of subsystems [12].

5.2.2.1 Transmitter Module

After testing the remaining components that comprise the transmitter section of the USART each on its own in a similar manner to the baud rate generator and buffer, these sub-modules were integrated together to test the transmitter module as one unit.

The operation of the transmitter module was simulated for the different modes of operation and options for the word length, parity bit and stop bit(s). One example is shown in Fig. 30 which illustrates the asynchronous mode of operation with word length of five bits, odd parity, one stop bit, and with the addressability feature disabled.

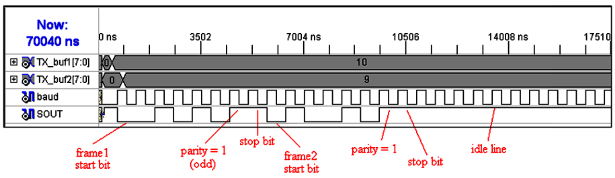

Fig. 30: Asynch. simulation of transmitter module

The simulation illustrates two frames transmitted one after the other. As it is clear from the serial output signal (SOUT), the transmission began with the start bit of the first frame followed by the five data bits that comprise the first word (10) starting with the least significant bit. A parity bit was then transmitted before the stop bit marked the end of the first frame. After that, the second frame was transmitted in a similar manner.

Fig. 31 illustrates the synchronous mode of operation, with word length of eight bits and the addressability feature enabled.

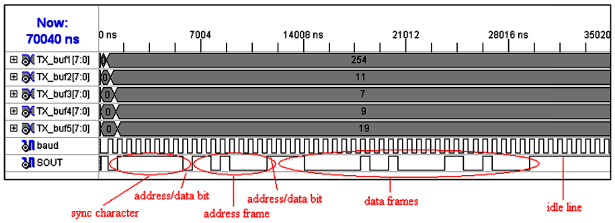

Fig. 31: Synch. simulation of transmitter module

Before transmitting the actual data words, a sync character (254d = FEh) was transmitted to indicate the beginning of a block of data for all possible receivers. To capture the attention of the receiver of interest, its address (11d in the figure) was then transmitted. After that, three data frames were transmitted. Each of them consisted of eight data bits followed by an address/data bit. The clock signal transmitted from the sender to the receiver in the synchronous mode of operation is not shown in the figure for clarity purposes.

Actual snapshots were taken for the serial output of the transmitter module in the different modes of operation and at various baud rates using a digital storage oscilloscope. Fig. 32 illustrates the serial output of the transmitter at a data rate of 1200 bps in

asynchronous mode of operation, with 6 data bits, even parity, one stop bit, and with the addressability feature disabled.

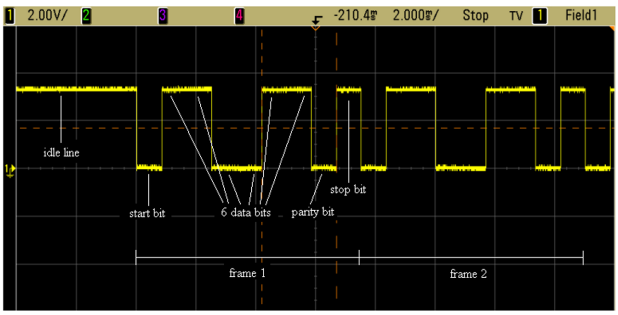

Fig. 32: Tx module in normal asynch. mode

Fig. 33 illustrates the serial output at a data rate of 110 bps in asynchronous mode of operation, with word length of eight bits and the addressability feature enabled.

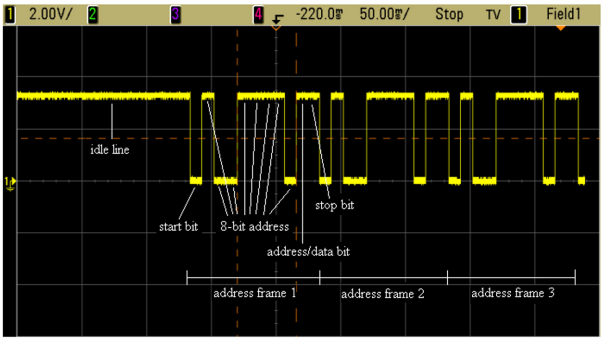

Fig. 33: Asynch. Tx module output with 9-bit addressability feature enabled

The previous results show that the transmitter module operates as expected in the two modes of operation of the USART at various baud rates. Moreover, all word lengths and options for parity and stop bits specified in the requirements of the system are supported.

5.2.3 Acceptance Testing

The primary goal of an acceptance test is to verify that a system meets its requirement specification. To demonstrate that the system meets its requirements, it must be shown that it delivers the specified functionality, performance and dependability, and that it does not fail during normal use. Ideally, the acceptance test plan is developed with the engineering requirements and is traceable to them. Acceptance testing is usually a black-box testing process where the tests are derived from the system specification. The system is treated as a black box whose behavior can only be determined by studying its inputs and the related outputs. Another name for this is functional testing because the tester is only concerned with the functionality and not the implementation of the system [10].

After testing the transmitter module, the next logical step in the testing process would be testing the receiver module. However, since the receiver module is inextricably linked to a corresponding transmitter which must supply the serial data, testing the receiver module was utilized as a test to the whole system at the same time.

5.2.3.1 Testing the USART with the Transmitter Connected to the Receiver

Before connecting the USART to an external system, it was tested at first by connecting the serial output of its transmitter section to the serial input of its receiver section to locate any potential errors within the system itself. Similarly, the clock output of the transmitter section was connected to the clock input of the receiver section in the synchronous mode.

A collection of simulations that covers various features of the system were run using the Xilinx ISE Simulator. Fig. 34 illustrates a simulation of the USART operation in asynchronous mode with eight data bits, no parity, two stop bits, and with the addressability feature disabled.

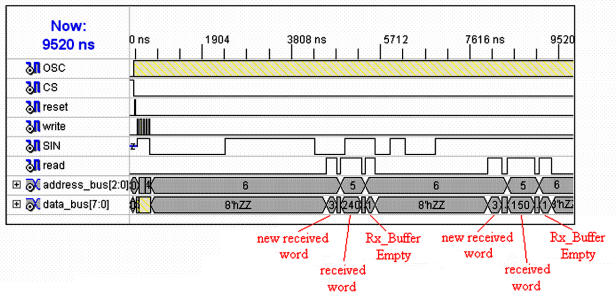

Fig. 34: Asynch. TX & RX Simulation of the USART in normal mode

The data received at the serial input of the USART (SIN) was shifted in one bit at a time starting with the least significant bit. After receiving the whole first frame, the status register (address 6) was read and indicated that a new word was received. This new word (240) appeared on the data bus from the receiver buffer (address 5) while the read input was active. After reading this received word, the status register indicated that the receiver buffer became empty again. Meanwhile, another frame was being received. The received word appeared on the data bus while the read input was active and the status register again indicated the status of the receiver buffer.

Another simulation of the USART operation in the synchronous mode with seven data bits, odd

parity and the addressability feature disabled is shown in Fig. 35. Since it is synchronous mode, a sync character was transmitted at first to indicate the beginning of a data block. This sync character would not be written into the receiver buffer.

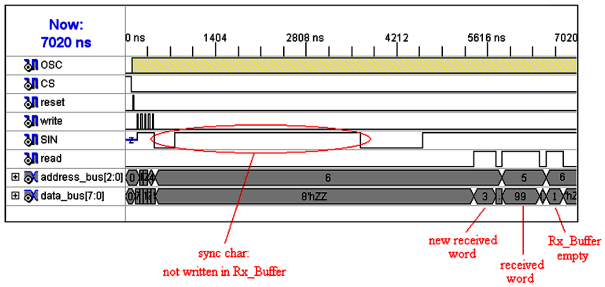

Fig. 35: Tx & Rx synch. simulation of USART

5.2.3.2 Serial Communication between Two directly connected USARTs

Next, the entire system was implemented on two Xilinx development boards. Both were programmed with the same USART design. However, one was used to transmit data, while the other was used to receive the sent data. Special temporary modifications to the internal design were implemented to allow certain internal signals to be observed with the digital storage scope. The PC was used to configure the USARTs with the different communication options by sending the appropriate control words to the respective registers and also to supply the data and to be transmitted serially.

Half-duplex and full-duplex communication sessions were then established between them, with the PC used to supply each system with the data to be transmitted serially and to read the received data. From this test, more indications were obtained that the system complies with its requirements specification. Error conditions reflected the true state of the received data when the two USARTs were deliberately configured with conflicting communications options. Moreover, the USARTs functioned as expected in the 9-bit mode of operation.

Figs. 36-38 illustrate some snapshots of the communication sessions that were established between the two USARTs. Each snapshot indicates the mode of communication options that were used in the session.

5.2.4 Performance Testing

The complex relationships between the components in a system mean that the system is more than simply the sum of its parts. It has properties that are properties of the system as a whole. These emergent properties cannot be attributed to any specific part of the system. Rather, they emerge only once the system components have been integrated. Such properties include reliability, reparability, and usability of the system [10].

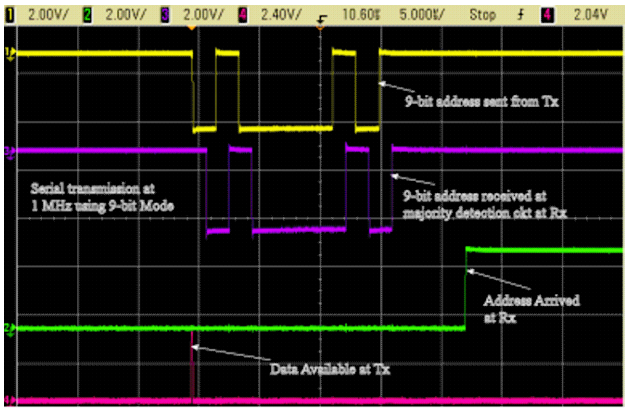

Fig. 36: Asynchronous 9-bit address transmission and detection between two USARTs

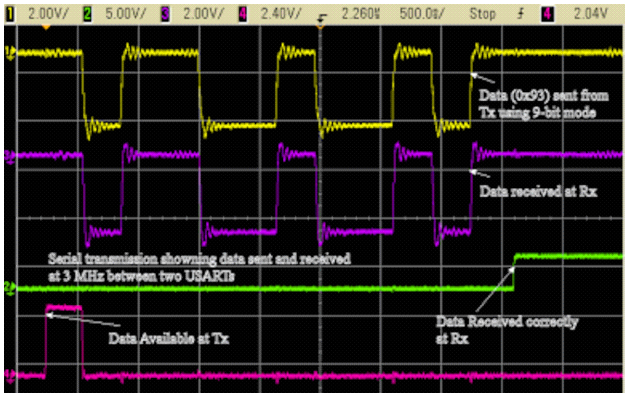

Fig. 37: Asynchronous 9-bit data transmission and reception between two USARTs

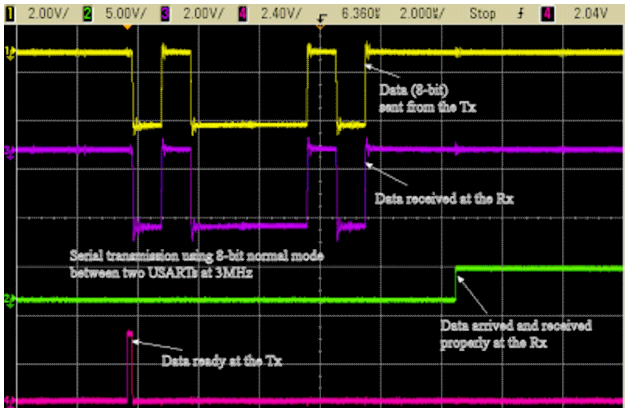

Fig. 38: Asynch. 8-bit normal mode data transmission between two USARTs at 3 MHz

Therefore, once a system has been completely integrated, it is possible to test the system for emergent properties. Performance tests have to be designed to ensure that the system can process its intended load. This usually involves planning a

series of tests where the load is steadily increased until the system performance becomes unacceptable. In performance testing, an effective way to discover defects is to design tests around the limits of the system; that is, stressing the system (hence the name stress testing) by making demands that are outside the design limits of the system until the system fails.

During the test cases described previously, the USART design was subjected to some extreme conditions to explore various aspects of the limits of its operation. For example, the output of the baud rate generator was examined at the highest baud rate possible, and the buffers were tested with increasing read and write speeds. Moreover, the operation of the entire USART was checked while operating at the highest baud rate possible when two systems on separate boards were connected together, as well as when the transmitter section was used to drive the receiver section of the same system.

6 Conclusion

We presented in this paper the detailed design and implementation of a reconfigurable USART IP core suitable for use in FPGA-based systems and systems on chip (SoC). The USART supports both synchronous and asynchronous modes of operation with variable configurable date rates and frame formats. We incorporated support for multi-drop networks of serial devices using 9-bit address generation and detection which increases the utility of such a device in modern networked processors and embedded systems. It is shown through comprehensive testing that the design performs according to its specifications and can operate at high bit rates reaching 3 Mbps. The USART is able to detect and recover from common serial communication errors such as overflow and framing errors. It can also detect false start bits. It features a universal 8-bit bus interface and interrupt driven operation. The design is implemented using HDL so it is platform neutral.

References

[1] Krishnan, Ravi, Embedded Systems: Technologies and Markets, *BCC Research Report*, IFT016C, Published: April 2009.

[2] John A. Stankovic, Insup Lee, Aloysius Mok, Raj Rajkumar, "Opportunities and Obligations for Physical Computing Systems", *IEEE Computer Magazine*, Nov. 2005, pp. 23-31.

[3] Wayne Wolf, "High performance Embedded Computing", Elsevier, 2007, pp.383-387.

[4] Mohd Yamani, Idna Idris, Mashkuri Yaacob and Zaidi Razak, "A VHDL Implementation of UART Design with BIST Capability", *Malaysian Journal of Computer Science*, Vol. 19 (1), 2006, pp. 73-86.

[5] Azita Mofidian, "DesignWare foundation DW_16550: A fine work of UART", Designware Technical bulletin, Technical Forum for Design Automation information, Volume 4 issue 3 Q4/99, http://www.synopsys.com/news/pubs/dwtb/q499/dwtb_art1.html.

[6] Shouqian Yu, Lili Yi, Weihai Chen and Zhaojin Wen, "Implementation of a Multi-channel UART Controller Based on FIFO Technique and FPGA", *Proceedings of 2nd IEEE Conference on Industrial Electronics and Applications ICIEA*, May 2007, pp. 2633 - 2638

[7] Tim Wilmshurst, *Designing Embedded Systems with PIC Microcontrollers – Principles and Applications,* 1st Edition (2007), Elsevier Publication, UK.

[8] Tim Wilmshurst, *An Introduction to the Design of Small-Scale Embedded Systems*, Palgrave, 2001.

[9] Barry Brey, *The Intel Microprocessors – Architecture, Programming and Interfacing*, 7th edition. Pearson Prentice Hall, Upper Saddle River, New Jersey, 2006.

[10] I. Sommerville, *Software Engineering*, 8th Edition. Addison Wesley, 2007.

[11] Xilinx web site, http://www.xilinx.com/ise/logic_design_prod/webpack.htm.

[12] Ford R.M, Coulston C., *Design for Electrical and Computer Engineers*, McGraw Hill, 2005.

[13] Zhu Yunshan and Tom Marshall. "Design Verification Using Formal Techniques". In: *Proceedings of the IEEE 4th International Conference on ASIC,* 2001, pp. 21-28.

[14] C. Wohlin and B. Regnell "Reliability Certification of Software Components", *IEEE Symposium on Software Reuse*, 1998, p.56-64

[15] J. Yeandel, D. Thulborn and S. Jones,"An On-line Testable UART Implemented Using IFIS". In: *Proceedings of the 15th IEEE VLSI Test Symposium*, 1997, pp. 344-349.

[16]http://www.xilinx.com/ise/logic_design_prod/webpack.htm

[17] www.digilentinc.com, *Digilent D2XL System Board Reference Manual,* Revision: June 9, 2004.

[18] www.xilinx.com, *Spartan-3 Starter Kit Board User Guide*, version 1.1: May 13, 2005.

[19] www.xilinx.com, *Spartan-3E Starter Kit Board User Guide*, version 1.0: March 9, 2006.