Design of a 9-bit UART Module Based on Verilog HDL

Nennie Farina Mahat, *Member, IEEE*
Integrated Circuit Development (ICD)
MIMOS Berhad
Technology Park Malaysia
Bukit Jalil, 57000 Kuala Lumpur
Email: farina.mahat@mimos.my

***Abstract*— Universal Asynchronous Receiver Transmitter (UART) is widely used in data communication process especially for its advantages of high reliability, long distance and low cost. In this paper, we present the design of 9-bit UART modules based on Verilog HDL. This design features automatic address identification in the character itself. We have implemented the VLSI design of the module and pass data between the proposed 9-bit UART module with a host CPU. The design consists of receiver module, transmitter module, prescaler module and asynchronous FIFOs. We have explained the functions of each individual sub-modules and how the design works in simulation.**

I. INTRODUCTION

UART is a Universal Asynchronous Receiver Transmitter that performs parallel-to-serial conversion on data character received from the host processor into serial data stream, and serial-to-parallel conversion on serial data bits received from serial device to the host processor. The RS-232 is applied to serial data communication as a standard to be complied with. Besides RS-232, RS-422 and RS-485 standards have been commonly applied into UART chip nowadays. These standards offer more reliable communication over much longer distances compared to RS-232.

A complete data frame format for UART consists of a start bit '0', 5-8 bits data, optional parity bit and stop bit '1'. The stop bit can be in length of 1, 1.5 or 2 bits [1]. Fig. 1 below shows the data frame format of a UART. While in idle state, serial data line will be in logic '1' state. A start bit '0' at the beginning of the data frame will cause a falling edge on the serial data line. This marks the detection of a data character. The idea of start bit and stop bit in UART is to achieve data synchronization [2-3]. An optional parity bit can be in odd parity or even parity. Odd parity means that sum of all bits gives an odd number, while even parity means sum of all bits gives an even number. The serial data frame is shifted out with the least significant bit (LSB) first.

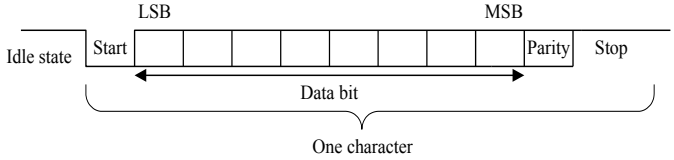

Fig. 1 UART data frame format

II. 9-BIT NETWORK

In a transmission, every Slave devices will search every character transmitted for address byte and try to match with its unique address. This results in a lot of wasted processing time for Slave devices. In a 9-bit network, UART uses the ninth bit of a character to differentiate between an address or a data character [4]. By proposing the ninth bit for address byte indication, Slave devices are able to distinguish an address byte, compare the address and decide whether to accept or discard the following data bytes. This reduces the processing time of the Slave's CPU [5].

The parity bit is set to logic '1' to indicate an address character and set to logic '0' to indicate a data character. Processor will poll for the parity bit and if it is set, the processor will try to match the address with its own. If the address matches, the following data bytes are received. If the address matching failed, the processor will ignore the following data bytes.

The 9-bit UART design proposed in this paper configures the data frame format to have a start bit '0', 8-bit address or data byte, sticky parity '1' or '0' and 1 bit of stop bit '1'. Fig. 2(a) shows an address character with parity '1' and Gig. 2(b) shows a data character with parity '0'. Data communication is in the order that address character to be received or transferred first, followed by one or more data characters.

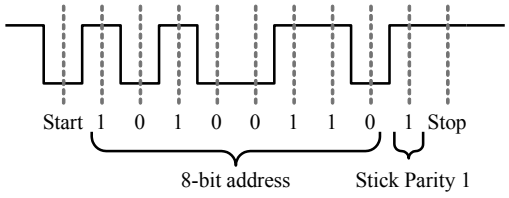

Fig. 2(a) Address character

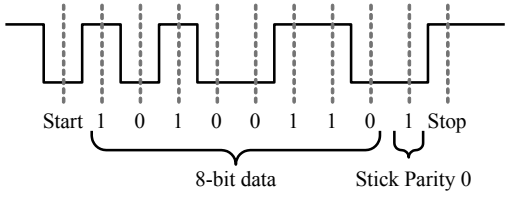

Fig. 2(b) Data character

III. Detailed Description of the Sub-modules

The 9-bit UART design proposed consists of the basic sub-modules of a UART which are the receiver, transmitter and baud rate generator that we called prescaler [6-7]. In addition to that, this design has internal buffers in both receiver and transmitter. Since this design operates in serial clock domain and interfaces with parallel clock domain of the processor, we implement the buffers by using asynchronous FIFOs. The asynchronous FIFO design provides smooth data transfer between two different clock domains. Fig. 3 below illustrates the overall block diagram of the 9-bit UART.

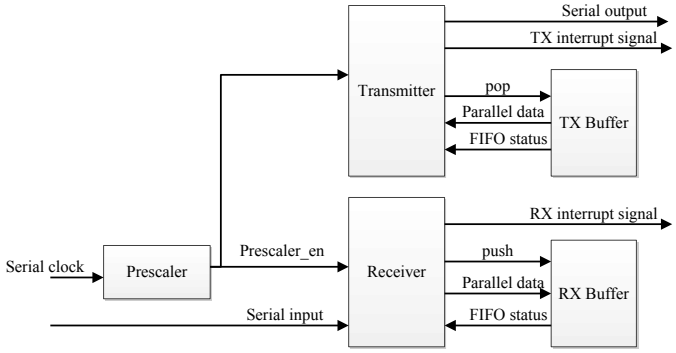

Fig. 3 Overall block diagram of 9-bit UART

A. Receiver

During reception, UART will be in listening mode and always sense for a start bit. A start bit is detected when there is a transition from logic '1' to logic '0' on the serial data line. UART receiver module will then capture the address byte and check the stop bit. If the stop bit is correct, the address captured will be compared with broadcast address and UART own address. UART will determine whether to receive or ignore the incoming data bytes. Receiver module also checks for the integrity of the start bit and stop bit. If a false start bit or stop bit is detected, reception will be terminated and an interrupt will be sent to the processor to read the remaining data, if available, in the RX buffer.

In this paper, we use finite state machine to implement these steps. The receiver finite state machine has five states; RX_ILDE, RX_START, RX_DATA, RX_PARITY and RX_STOP as shown in Fig. 4 below. The state machine changes state at every 16 baud clock cycles, which is equal to 16 prescaler_en pulses.

RX_IDLE State: When UART receiver module is reset, the state machine will be in this state. The state machine will wait for start bit detection on the serial data line. A start bit detection is identified when there is a fall transition on the serial data line from idle state '1' into logic '0'. In order not to detect a false bit, the state machine will only consider startbit_detect signal that has been synchronised to the serial clock domain. A correct start bit detection will cause the state machine to go into RX_START.

RX_START State: In this state, the state machine will wait for 16 baud clock cycles before going into the next state, RX_DATA.

RX_DATA State: The state machine will sample the character received at the most ideal time, which is at the middle of the bit. Each bit is then being stored into an internal register rx_data to form a complete 8-bit data. When a complete character of 8 bit has been received, the state machine will go into RX_PARITY.

RX_PARITY State: In this state, the state machine samples the parity bit at the middle of the bit and determines whether the character received is an address byte or a data byte,

RX_STOP State: Similar as in the previous states, the state machine will sample the stop bit at the midpoint. In this state, state machine will do stop bit error checking and address comparison. If the is no stop bit detected, that is the sampled bit is logic '0', receiver module interrupts the processor for error detection, skip address comparison and state machine will go into RX_IDLE state. If the state machine detects a correct stop bit, it proceeds with address matching by comparing the received address byte with its own address and broadcast address. A matched address will assert match_flag throughout the whole reception process so that when state machine goes into this state again while receiving data byte with parity bit '0', receiver will save the data byte into internal RX buffer, provided that the sampled stop bit is also correct.

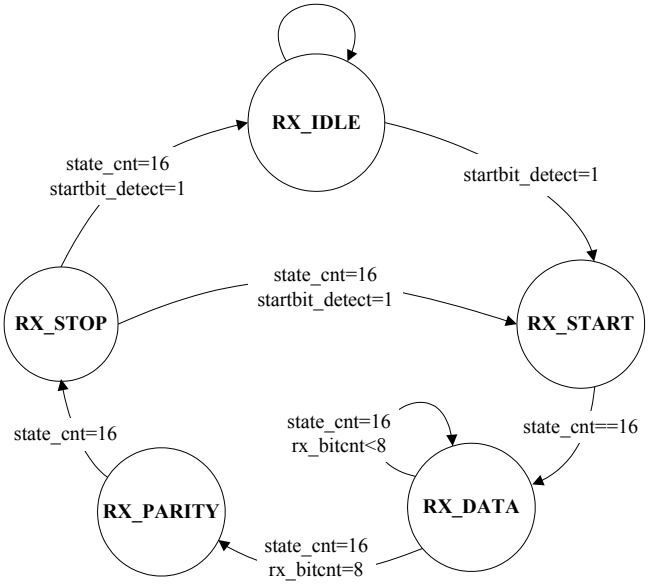

Fig. 4 Receiver module state machine

B. Transmitter

UART will be in transmission mode when TXMODE is enabled in the configuration register of the UART. The processor sets the destination address, data byte(s) to be sent and other transmission settings. Transmission data bytes are saved into an internal TX buffer before being processed for transmission.

Transmitter module converts the address and data received from the processor into serial bits, and adds in start bit of '0', parity bit of '1' for address byte or '0' for data byte, and stop bit of '1'. The address can be a broadcast address or a unique address that belongs to a specific Slave device. Therefore the

following data bytes can be a broadcast message meant for all Slave devices or a specific command for a specific Slave device. When an address byte has been transferred successfully, UART transmit module will continue to send all data byte(s) available in the TX buffer until the buffer is empty.

An internal finite state machine is used to transmit the complete 11-bits character, bit-by-bit. The transition of each state will take place at every 16 baud clock cycles, which is equal to 16 cycles of prescaler_en. There are five states in the transmitter finite state machine; TX_ILDE, TX_START, TX_DATA, TX_PARITY and TX_STOP as shown in Fig. 5 below.

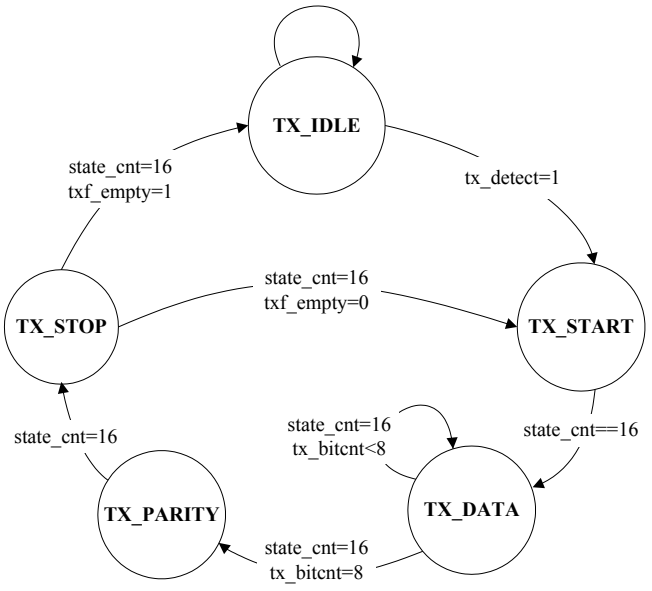

Fig. 5 Transmitter module state machine

TX_IDLE State: When UART is being reset, state machine will be in this state. Transmitter module will wait for tx_detect pulse that indicates data is ready to be transmitted. This enable pulse is coming from the processor and being synchronised to the serial clock.

TX_START State: In this state, transmitter module will send out a start bit '0' on the serial data line. State machine will be in this state for 16 baud clock cycles before going into TX_DATA state.

TX_DATA State: In this state, the state machine loads the internal tx_data register with data to be transmitted. For the first transmission, tx_data register will be loaded with a destination address. The state machine transmits the destination address bit by bit, starting from the least significant bit in tx_data[0] until the most significant bit in tx_data[7].

TX_PARITY State: For an address byte, the state machine transmits out parity bit '1' whereas for a data byte, it will transmit out parity bit '0'.

TX_STOP State: When a complete frame of data has been successfully transmitted out together with the parity bit, the state machine will go into this state to complete the transmission by sending out a stop bit of '1'. The state machine then checks whether there is another data byte in the TX buffer to be sent out. If there is, state machine will go back to TX_START and repeats all the states until it goes back into this state again. If the TX buffer is empty, it means there is no more data byte to be transferred; therefore the state machine will go into TX_IDLE state and wait for another tx_detect pulse.

C. Prescaler Module

This 9-bit UART design can operate at any defined serial clock frequency and at the same time follows the desired baud rate. Here is where the prescaler module comes in hand where it acts as a baud rate generator module that calculates the divider factor. The divider factor will then generate prescaler_en pulse that acts as a baud clock. The prescaler module is used in this design for the transmission and reception to be at the desired baud rate. Fig. 6 below shows the timing diagram for transmission with prescaler ratio of 12. The serial clock is set to 32MHz and the desired baud clock rate is 115200bps. The calculation to get prescaler divide ratio is shown below:

$$Divide\ ratio = \frac{Serial\ clock\ frequency}{16 * Baud\ rate}$$

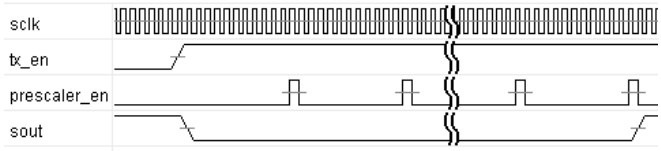

Fig. 6 Timing diagram for transmission with prescaler ratio of 12

D. Transmitter and Receiver Buffers

In serial data transmission, a huge stream of data can come in too fast and UART may not be quick enough to process what it is receiving. This causes data overrun which then results in data loss or other serious errors. This is why we need flow control in the UART where it will halt the flow of data bytes until UART is ready to receive more data.

There are many ways to implement flow control mechanism. One of the older methods used back then is to add bunch of zeros while receiving line is busy. This method is called padding. UART will ignore the zeros while it processes current data. To make it efficient, we need to provide just enough number of zeros and figuring this out is not easy.

Another method is to use RTS (Request to Send) and CTS (Clear to Send) flow control mechanism which is part of RS-232 standard. These two lines allow both sender and receiver to communicate and alert each other on their status. When sender has data to be sent, it will assert RTS to ask receiver whether it is ready to receive data or not. If receiver is free, it will reply by asserting CTS and sender will start transferring data. But if receiver is busy, CTS line will maintain low and sender needs to halt the data and wait until CTS goes high. This method however requires additional wires to the UART design.

Most UART nowadays has internal buffer with the size big enough to hold the data and that is less likely to overrun. The buffer can be set to interrupt the processor when the buffer

reaches certain threshold level. In this paper, we are using this method to do flow control. Asynchronous FIFOs are being sued as an internal TX buffer and RX buffer in transmitter and receiver module respectively. The asynchronous FIFO allows smooth data transmission between two different clock domains and provides reliable empty and full status [8-10].

During reception, receiver module writes the received data bytes into RX buffer using serial clock. When RX buffer is almost full, receiver will interrupt the processor to start reading the data (using parallel clock) before RX buffer starts to overflow. While in transmission, processor writes data into TX buffer using parallel clock. When there is at least one data byte inside TX buffer, transmitter will start reading the data in serial clock, process the data byte and transmit out serial bits. If TX buffer is almost full, an interrupt is sent to processor to halt writing into the buffer.

IV. CONCLUSIONS

We have simulated the design using Synopsys' VCS Simulation tool. Fig. 7 shows the simulation waveform of the receiver module. In the simulation, receiver module is in idle state after being reset. When a start bit is detected, state machine is being activated. Parity bit received is logic '1'; therefore this indicates an address byte. The receiver captures the data address of 8'h01 and save it in rx_data register to be compared with UART's own address and broadcast address.

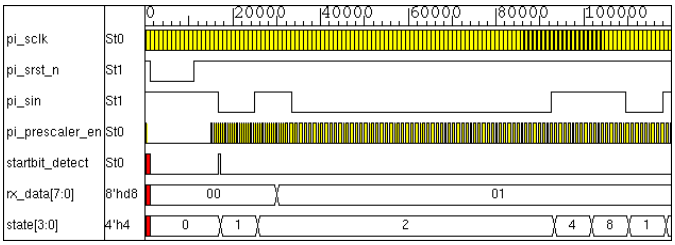

Fig. 7 Simulation waveform of the receiver

Fig. 8 below shows the simulation waveform of the transmitter module. In this simulation, UART is going to transmit out a data byte of 8'hCB. The transmitter module adds in start bit of '0', shifted data byte as '11010011', sets parity bit to '0' and 1 stop bit of '1'. From the waveform, we can see that the output on the serial data line, po_sout is correct.

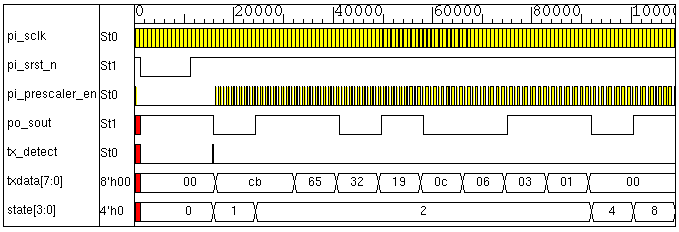

Fig. 8 Simulation waveform of the transmitter

V. CONCLUSIONS

In this paper, a modified UART design is proposed with automatic address indication, which is called 9-bit UART. The 9-bit UART design is implemented using Verilog HDL and simulated to see the functionality of each sub-modules and the result. This design shows that using the ninth bit method gives advantage of saving the UART processing time by comparing the address and decides whether to receive or ignore the incoming data packets. The stop bit error checking mechanism in this design also offers data integrity checking. With all the features mentioned, it adds to the flexibility, stability and reliability to the normal UART design that is widely being used.

REFERENCES

[1] J. Norhuzaimin, and H.H. Maimun, "The design of high speed UART," *Asia Pac. Conf. on Appl. Electromagnetics (APACE 2005),* Johor, Malaysia, Dec. 2005.

[2] C. He, Y. Xia, and L. Wang, "A universal asynchronous receiver transmitter design," *Int'l Conf. on Elect. Comm. and Control (ICECC 2011),* Ningbo, China, Sept. 2011.

[3] Y. Wang, and K. Song, "A new approach to realize UART," *Int'l Conf. on Elect. and Mech. Eng. and IT (EMEIT 2011),* Harbin, Heilongjiang, China, Aug. 2011.

[4] *Using a 9-bit Software UART with Stellaris® Microcontrollers Application Note (AN01280),* Texas Instruments, P.O. Box 655303, Dallas, Texas 75265, USA, Aug. 2010.

[5] *Zilog Z8 Encore! XP® 9-bit UART Implementation Application Note (AN014602-1207),* Zilog Inc., Dec. 2007.

[6] Z. Zhang, and W. Wu, "UART integration in OR1200 based SoC Design," *2nd Int'l Conf. on Comp. Eng. and Tech. (ICCET 2010),* Chengdu, China, Apr. 2010.

[7] Y. Fang, and X. Chen, "Design and simulation of UART serial communication module based on VHDL," *3rd Int'l Workshop on Intel. Sys. and App. (ISA 2011),* Wuhan, China, May 2011.

[8] C.E. Cummings, "Simulation and synthesis techniques for asynchronous FIFO design," *Synopsys User Group (SNUG), San Jose, USA, 2002.*

[9] X. Wang, and J. Nurmi, "A RTL asynchronous FIFO design using modified micropipeline," *The 10th Biennial Baltic Elect. Conf. (BEC 2006),* Tallinn, Estonia, Oct. 2006.

[10] X. Wang, T. Ahonen, and J. Nurmi, "A synthesizable RTL design of asynchronous FIFO," in *Proc. Int'l Sympo. on SoC 2004,* Tampere, Finland, Nov. 2004.