International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

Design and Verification of APB Compliant Quad Channel UART

Lakshmisagar H.S[1], Sumathi M.S[2]
[1,2]*Assistant Professors, BMS Institute of Technology, Bangalore, Karnataka, India.*

Abstract: ***Embedded systems have drastically grown in importance and complexity in recent years. Many systems are now designed using Field Programmable Gate Array (FPGA) due to its size, flexibility, and resources. The main aim of this project is to design and verify APB-compliant Quad channel UART (Universal Asynchronous Receive Transmit) suitable for use in embedded systems and System on Chip (SoC). The data transmission in each UART can be performed at different baudrate to communicate with peripheral device by configuring divisor register. The QUAD-UART design proposed in this implementation has a single Host Interface, this was not a feasible solution. Hence, what was attempted was to introduce an arbitrary (Radom) skew while looping back from Transmitter to Receiver.This skewed loop-back still verifies the "Asynchronous" behaviour of the UART. In this design each UART is selected randomly by feeding a seed value.***

The design is implemented using codes in Verilog Hardware Descriptive Language, simulated using the Mentor Graphics QuestaSim 6.4b engine. The translation and mapping process is done in Synthesis using Synopsis design Compiler C-2009.06-SP2 (for Linux).

Keywords: **APB:** ***Advanced Peripheral Bus.***
AMBA: ***Advanced Microcontroller Bus Architecture***

I. INTRODUCTION

UART (Universal Asynchronous receiver Transmitter) is a popular method of serial asynchronous communication. To the processor, The UART appears as an 8bit read write parallel port that performs serial to parallel conversions for the processor, and vice versa for the peripheral. The UART (Universal Asynchronous receiver Transmitter) core provides serial communication capabilities, which allow communication with modem or other external devices, like another computer using a serial cable and RS232 protocol. Specifically, it provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices.

Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms. The transmit and receive paths are buffer with internal transmit and receive buffer registers. The UART usually does not directly generate or receive the external signals used between different items of equipment. Typically, separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels. External signals may be of many different forms. Examples of standards for voltage signaling are RS-232 It is useful to communicate between microcontrollers and also with PCs. Many chips provide UART functionality in silicon, and low-cost chips exist to convert logic level signals (such as TTL voltages) to RS-232 level signals (for example, Maxim'sMAX232).

II. DESIGN AND IMPLEMENTATION

Fig. 1. Depicts the structure of APB Compliant Quad Channel UART (16550 UART Core).

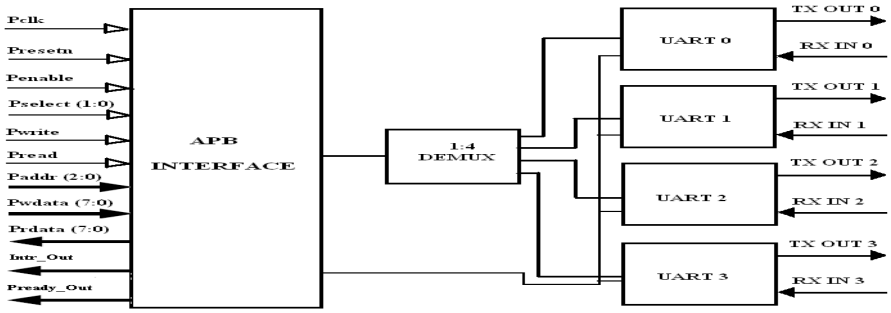


Fig. 1. APB-Quad UART (16550) Architecture

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

The peripheral devices which are attached to the UART 0, UART 1, UART 2, and UART 3 will request for access at the same time but only one UART will be granted for the request. At the APB side the Pselect line is 2-bit (1:0), which will select one of the UART can communicate with the particular peripheral device such as PP 0, PP 1, PP 2, PP 3. The interrupt controller is instantiated with all four UARTs is to service the interrupt of a particular UART. Each UART can be used to transmit the data at different baud rates by configuring divisor latch register, which can communicate with the peripheral device at different speed.

A. APB (Advanced Peripheral Bus)

The APB bus is part of the Advanced Microcontroller Bus Architecture (AMBA) hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity. The APB bus is used to interface to any peripheral device which are low bandwidth and do not require the high performance of a pipelined bus interface. The APB slave interface acts as a bridge between the APB bus and the peripheral device to which the bus is connected. It receives the APB bus signals and converts them to a form in which is understood by the connected peripheral device.

B. Application of the APB Interface

Most common applications of the APB interface are to read and write registers of the connected device. The Peripheral devices connected to the APB bus could be UART, Timer, Keypad, etc.

C. Features

1) Compliant with AMBA [Rev 2.0] for easy Integration with SOC implementations
2) Supports APB bus for a wide range of frequencies (approximately 100 MHz)
3) Programmable Address and data widths
4) Easy integration to any SOC implementation

D. APB Signals

APB Interface performs the interface functions between UART and APB bus. Allows easy connection of the core to existing APB systems.
PCLK (System clock): Internal state machines and the baudrate generator all operate at this frequency.
PRESET_N (Master reset): When this signal is low, it initializes all registers and control logic.
PSELECT: The UART module is selected if select is equal to one.
PENABLE: The UART module is enabled if enable is equal to one.

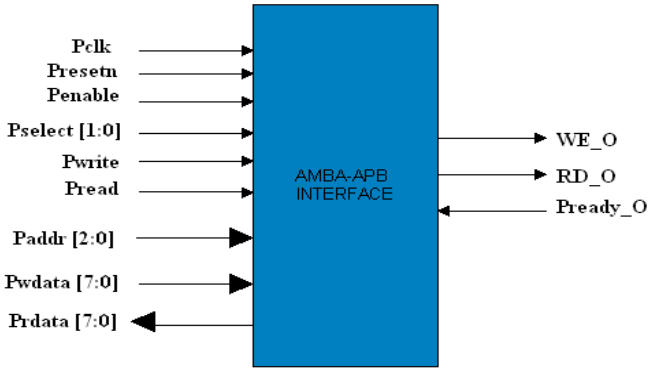


Fig. 2. Input or output signals of APB interface

E. Operating States

The state machine operates through the following states:
1) *Idle :* This is the default state of the APB.
2) *Setup :* When a transfer is required the bus moves into the SETUP state, where the appropriate select signal, PSELx, is asserted. The bus only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

3) Access : The enable signal, PENABLE, is asserted in the ACCESS state. The address, write, select, and write data signals must remain stable during the transition from the SETUP to ACCESS state. Exit from the ACCESS state is controlled by the PREADY signal from the slave:

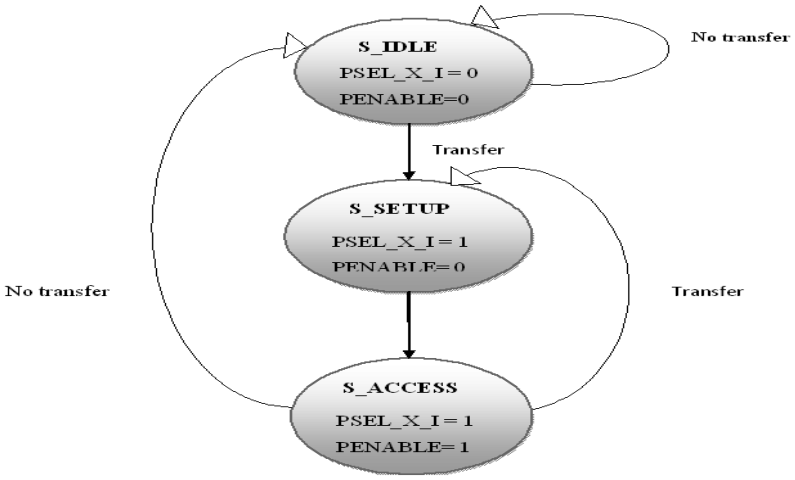


Fig. 3. Operating States of APB interface

4) If PREADY is held HIGH by the slave then the peripheral bus remains in the ACCESS state.

5) If PREADY is driven LOW by the slave then the ACCESS state is exited and the bus returns to the IDLE state if no more transfers are required. Alternatively, the bus moves directly to the SETUP state if another transfer follows.

F. Timing Diagrams

The following diagrams illustrate the timing relationships between various signals during different operations. The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the SETUP cycle. After the following clock edge the enable signal PENABLE is asserted, and

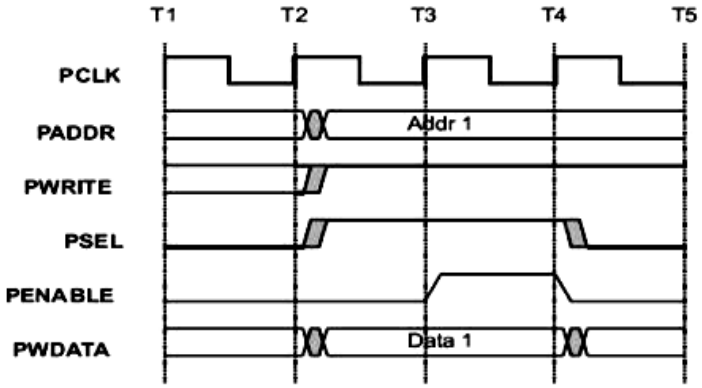


Fig. 4. Write transfer

This indicates that the ENABLE cycle is taking place. The address, data and control signals all remain valid throughout the ENABLE cycle. The transfer completes at the end of this cycle.

The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the SETUP cycle. After the following clock edge the enable signal PENABLE is asserted, and this indicates that the ENABLE cycle is taking place. The address, data and control signals all remain valid throughout the ENABLE cycle. The transfer completes at the end of this cycle.

The enable signal, PENABLE , will be deasserted at the end of the transfer. The select signal will also go LOW, unless the transfer is to be immediately followed by another transfer to the same peripheral. In order to reduce power consumption the address signal and the write signal will not change after a transfer until the next access occurs.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

The protocol only requires a clean transition on the enable signal. It is possible that in the case of back to back transfers the select and write signals may glitch. The timing of the address, write, select and strobe signals are all the same as for the write transfer. In the case of a read, the slave

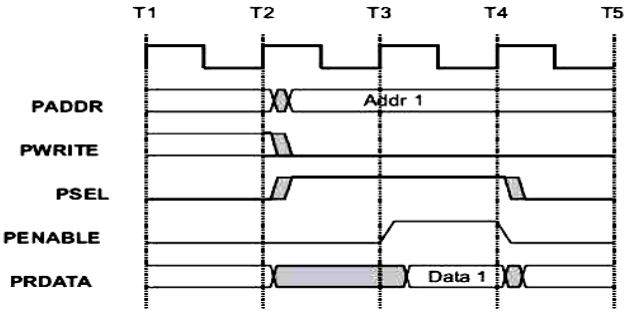


Fig. 5. Read transfer

Must provide the data during the Enable cycle. The data is sampled on the rising edge of clock at the end of the ENABLE cycle.

G. The 16550 UART Transmitter

To send data with the UART, the processor simply writes data to the transmitter address as if it was a memory space. The transmitter will take care of the entire transmission process. The transmitter performs parallel-to-serial conversions, and sends data on the serial line. The processor pushes data (5 to 8 bits wide) into the transmit FIFO, which is 128 bytes deep. When the transmit FIFO is full, no new data can be pushed into it. The transmitter pops the data off the FIFO, and shifts it out at the baud rate. The Divisor Registers determine the baud rate. Using the Line Control Register, the user can configure the number of data bits (5, 6, 7, or 8) per frame, as well as the number of stop bits (1, 1.5, or 2) to be sent at the end of a frame. The transmitter also has parity generation circuits that is capable of creating even, odd, stick even, or sticks odd parity. When parity generation is disabled, no parity will be sent after the data bits. There is one interrupt associated with the transmitter - it is a third-level priority interrupt that occurs when the transmit FIFO is empty. Logically, the transmitter follows the steps below:

1) If there is data available in the FIFO, load a byte into the shift register.
2) Send a start bit on the serial line, indicating the beginning of a frame.
3) Shift out data bits from the shift register to the serial line.
4) If parity is enabled, send Parity Bit after all data bits are sent.
5) Send stop bit(s) on the serial line, indicating the end of a frame.

The transmitter logic presented in the steps above is implemented in the core using a finite state machine.

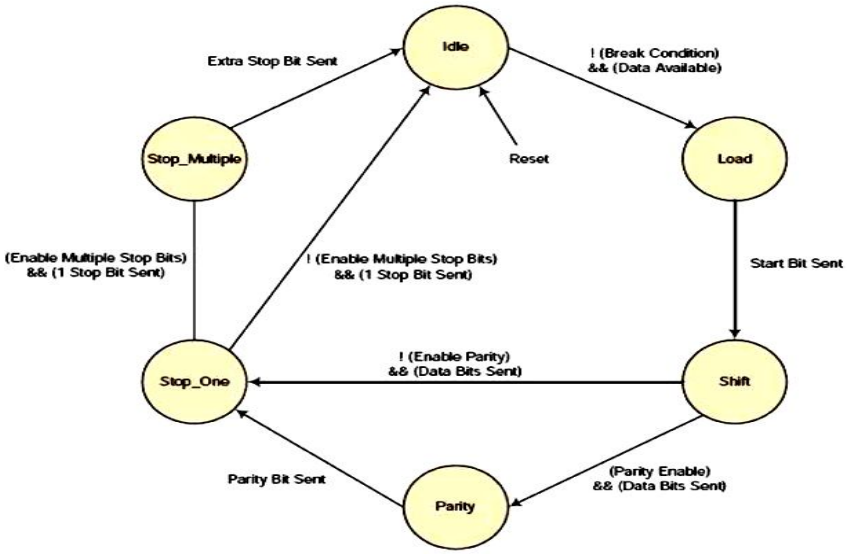


Fig. 6. Transmitter state diagram

H. The 16550 UART Receiver

To receive data with the UART, the processor simply reads data from the receiver address as if it was a memory space. The receiver is responsible for capturing data from the serial line and validating data integrity. The receiver performs serial-to-parallel conversions, and pushes data into the receive FIFO at the baud rate. The processor can pop data from the 128-byte deep receive FIFO at the system clock frequency. The supplied RCLK should be 16times faster than the baud rate. The receiver monitors the serial line. When a valid start bit is detected, the receiver begins to shift data bits from the serial line, and saves the received data in the receive FIFO.

Using the Line Control Register, the user can configure the number of data bits (5, 6, 7, or 8) per frame, as well as the type of parity to expect on the serial line. The receiver has a parity generation circuit that is capable of creating even, odd, stick even, or stick odd parity as data is shifted in from the serial line. The calculated parity is then checked with the received parity to determine data integrity. When parity generation is disabled, no parity will be expected after the data bits.

Besides parity errors, the receiver is also capable of detecting frame errors, overrun errors, and break errors. Frame errors occur when the receiver is expecting a stop bit but received a '0' on the serial line. Overrun errors occur when the receive FIFO is full and the newly received data is destroyed because it cannot be saved in the FIFO. Break errors occur when the serial line is '0' for more than a full frame. Break errors, frame errors, and parity errors are associated with the particular pieces of data in the FIFO that contains the errors. In other words, these errors are revealed to the processor only when the data with error is at the top of the receive FIFO. LSR [7] indicates if there are any errors in the entire receive FIFO. In addition to error detection capabilities, the receiver also feature false start bit detection and self-recovery from frame error. The serial line may be subject to noise, so it is essential that the receiver be able to differentiate noise from a valid start bit. In order for the receiver to recognize a '0' on the serial line as a valid start bit, this '0' value must remain for at least half a baud cycle from the falling edge of the serial input. If the start bit does not remain stable in that period, it is disregarded and the receiver will return to its idle state. The asynchronous nature of a serial line means it is possible that the receiver is out of synch from the transmitter, resulting in a frame error. When a frame error occurs, the receiver will try to resynchronize itself to the transmitter. To achieve this self-recovery feature, the receiver assumes that the invalid stop bit is actually the start bit of the next frame, and will proceed to shift in data bits from the serial line. There are three interrupts associated with the receiver. The highest-level priority interrupt occurs when the received data has an error. A second-level priority interrupt occurs when the receive FIFO has reached its trigger level. Another second-level priority interrupt is the timeout interrupt, which occurs when data is sitting idle in the receive FIFO for over a certain period of time.

To perform all these functionalities, the receiver follows the steps below:

1) Hunt for a valid start bit, which indicates the beginning of a frame.
2) Shift in data bits from the serial line to the shift register.
3) If parity is enabled, compare received parity bit with the expected value.
4) Check for a valid stop bit on the serial line, which indicates the end of a frame.

The receiver interfaces to three clock domains: system clock, receive reference clock, and baud rate clock. In order to correctly handle data across these clock domains, the receiver logic presented in the steps above is implemented in the core using two finite state machines (FSM): main FSM and shift FSM. The main FSM is always active. It is responsible for validating the start and stop bits, and enabling the receive shift FSM. The receiver has a circuit that determines when to sample the data on the serial line, and this circuit is also enabled by the main FSM. After a valid start bit, the main FSM depends on the shift FSM to provide newly received data in the shift register. When the shift FSM is done, the main FSM can then determine if there is any frame, line break, or overrun errors. Finally, the received data and associated error flags are stored. The states are listed below.

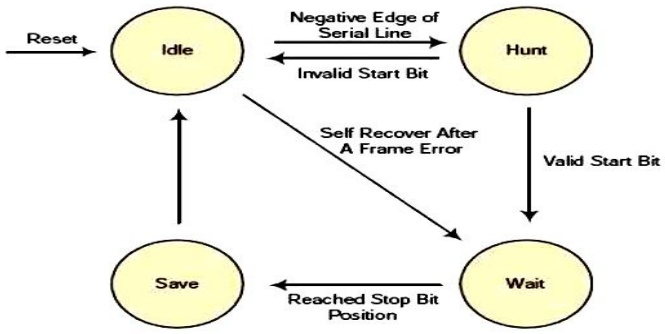


Fig. 7. Receiver main finite state machine diagram

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

When enabled, the shift FSM is responsible for shifting bits into the UART at the baud rate. The number of bits to shift in depends on the character length as indicated in the LCR. After the appropriate number of data bits have been transferred into the receive shift register, this FSM checks the received parity bit to determine whether there is a parity error.

I. Baud Generator

The baud generator is capable of creating a clock by dividing the system clock by any divisor from 2 to 216-1. The divisor is the unsigned value stored in the Divisor Register. The output frequency of the baud generator is 16 times the baud rate. This means, the Divisor Register should hold a value equal to the system clock divided by baud rate and further divided by 16. This output clock can be used as the receive reference clock by the receiving UART.

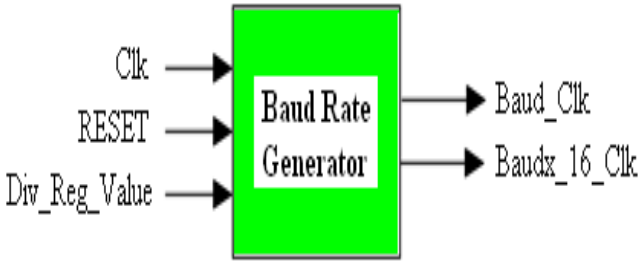


Fig.8. I/O ports of Baud rate generator

J. Flow Charts

The APB interface can be explained using the flow chart as given in the below fig. 9. APB Interface performs the interface functions between UART and APB bus. It operates through three states such as IDLE, SETUP and ACCESS. When a transfer is required the bus moves into the SETUP state from IDLE state, Where the SELECT signal is high. When PENABLE signal is high then the bus moves to the ACCESS state from SETUP state. If SELECT signal and PREADY signal from slave is active low then the ACCESS state is exited and the bus returns to the IDLE state if no more transfers are required.

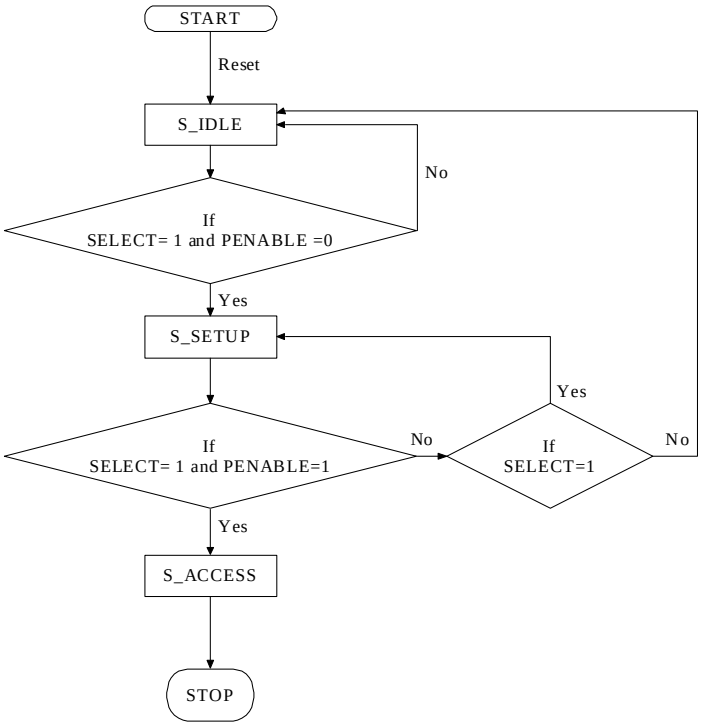


Fig. 9. Flow chart of APB interface

The Baud rate generator can be explained using the flow chart as given in the below figure.

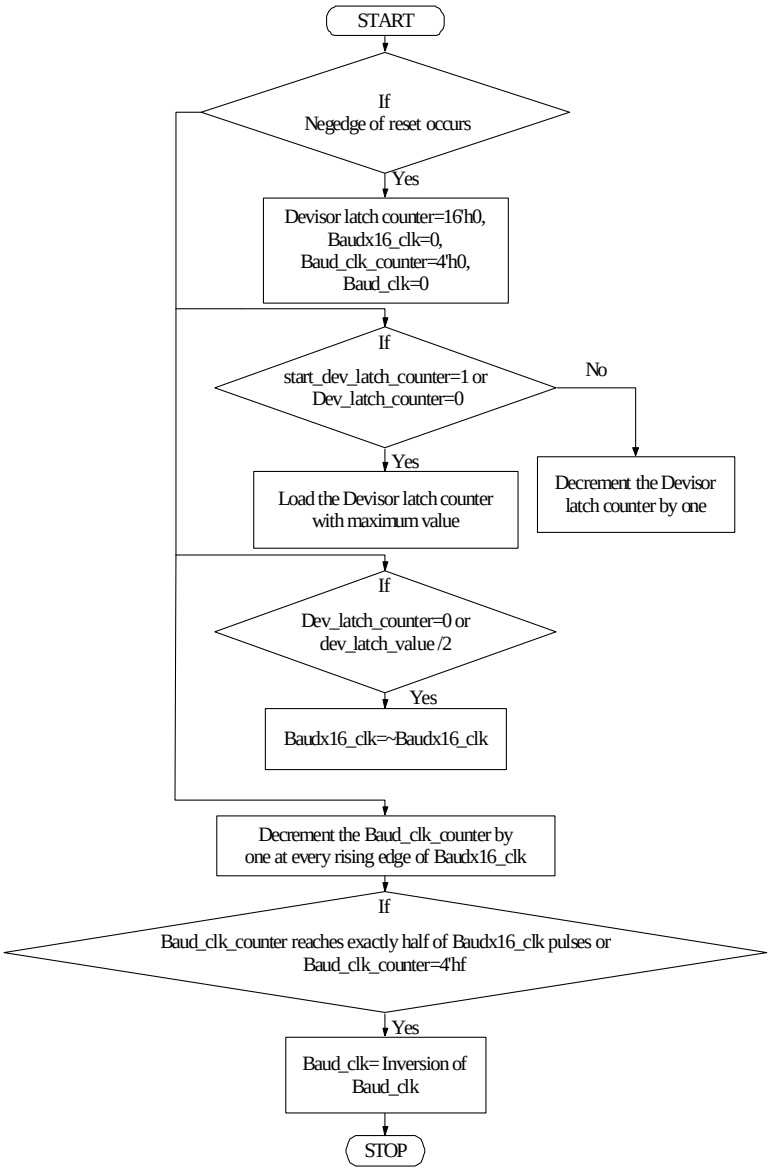


Fig. 10. Flow chart of Baudrate generator

As shown in the above fig. 10, When a start devisor latch counter is active high then the devisor latch counter is loaded with the maximum value otherwise decrement the devisor latch counter. The devisor latch counter decrements at every rising edge of Pclk. When the devisor latch counter reaches zero or half of the devisor latch value then the inversion of Baudx16 clock occurs. The Baud clock can be generated by decrementing the baud clock counter at every rising edge of Baudx16 clock . When Baud clock counter reaches the middle of Baudx16 clock or last pulse of Baudx16 clock then inversion of Baud clock occurs.

The synchronous FIFO can be explained using the flow chart as given in the below figure 4.15. Every memory in which the data word that is written in first also comes out first when the memory is read is a first-in first-out memory. WRITE CLOCK and READ CLOCK are free running. The writing of new data into the FIFO is initialized by a low level on the WRITE ENABLE line. The data are written into the FIFO with the next rising edge of WRITE CLOCK. The READ ENABLE line controls the reading out of data from the FIFO synchronous with READ CLOCK.

The writing and reading of data into or from FIFO occurs at every rising edge of CLOCK (WRITE or READ). The overrun error occurs, When the FIFO counter reaches its trigger level. The underrun error occurs before writing the data into the FIFO or after reading out of all the data from the FIFO.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

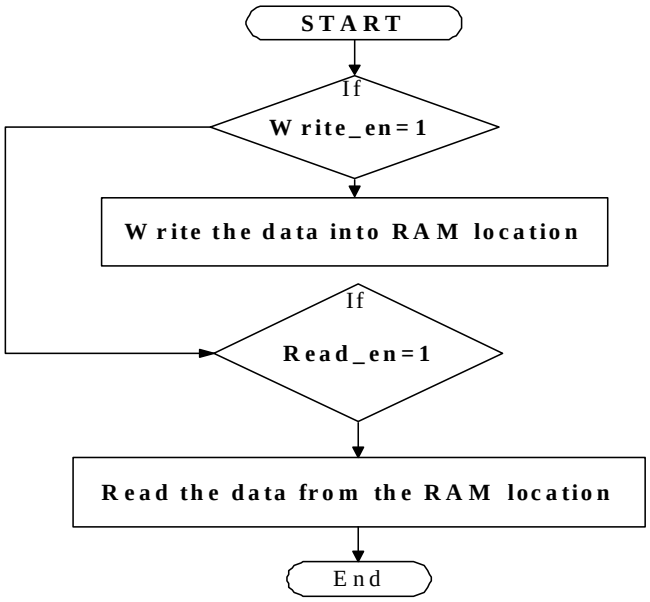
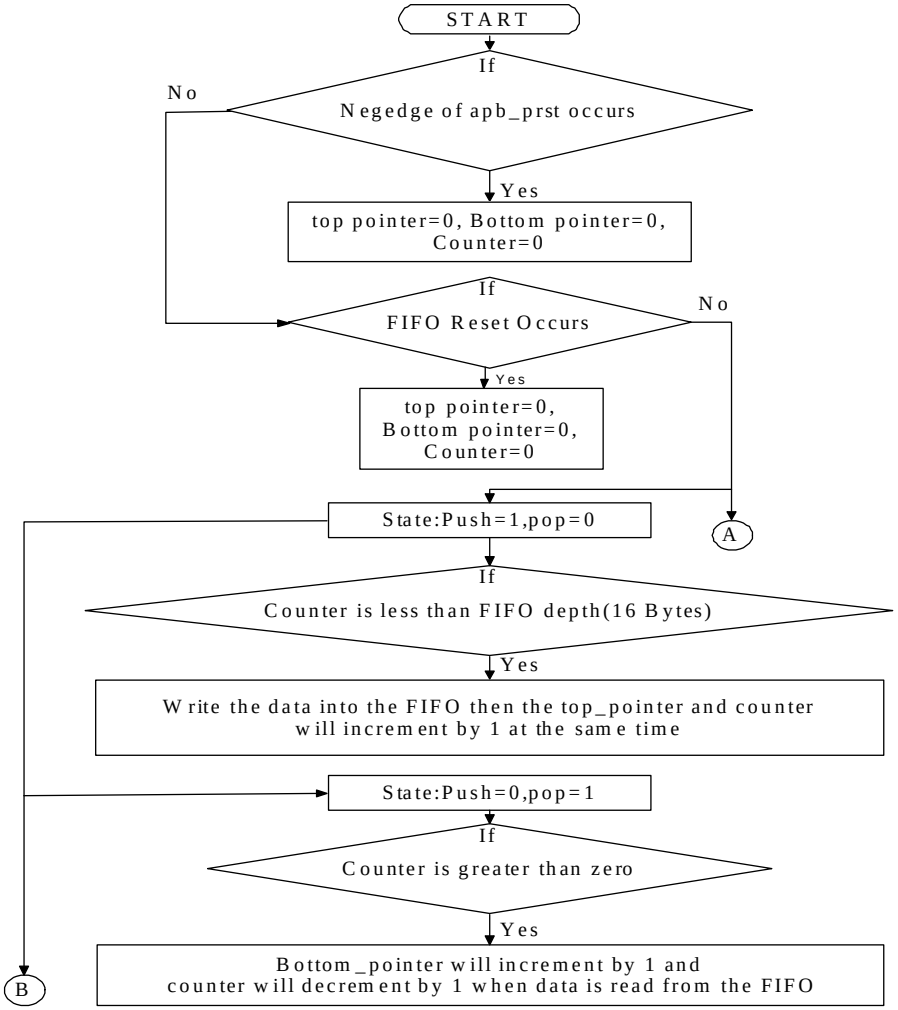


Fig. 11. Flow chart of dual port RAM with synchronous read



ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

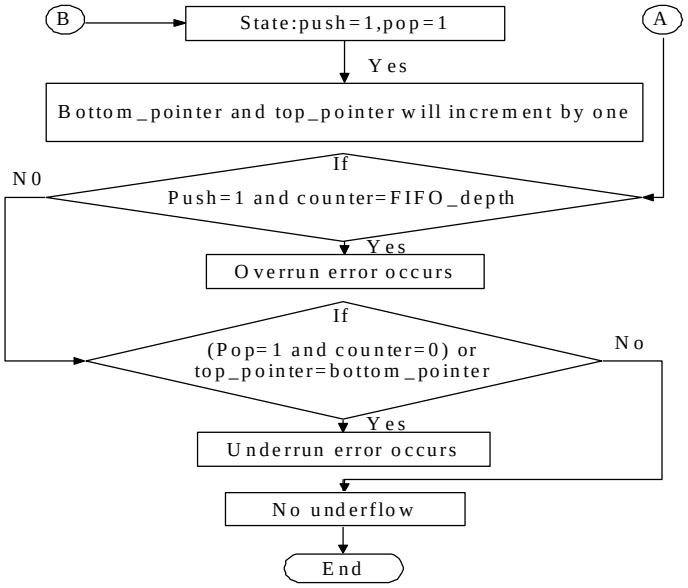


Fig. 12. Flow Chart of transmitter FIFO

III. RESULTS

APB interface operates through three states first one is idle state during this state select (apb_psel_x_i) and penable (apb_penable_i) is active low. Idle state will go to the next state called setup state ,when select line is active high which selects the UART. Setup state will go to the next state called access state when penable is active high . In access state slave (apb_pready_o) is ready to access data for read and write operations. When write signal is active high then write operation can be performed otherwise read operation can be performed.

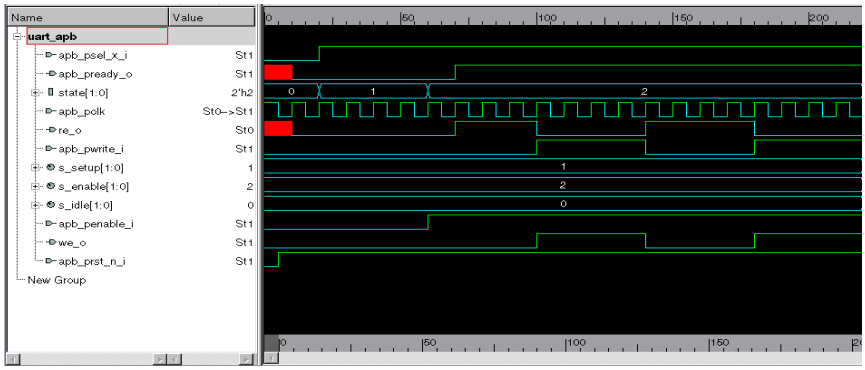


Fig. 13. Simulation result of APB Interface using VCS (Synopsys).

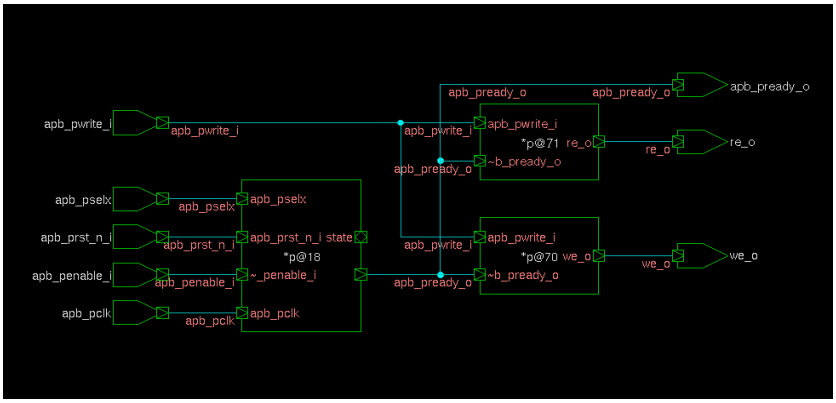


Fig. 14. Showing Schematic view of APB Interaface (I / O ports of APB interface).

The simulation waveform shows two counters first one is devisor latch counter used for the generation of Baudx16 clock with respect to Pclk (system clock) . The second one is Baud clock counter used for the generation of Baud clock with respect Baudx16 clock. The devisor register is of 16 bit to hold the devisor value (Hexadecimal number). When start devisor counter pulse occurs it loads the devisor latch counter with devisor value minus one. The devisor value can be calculated by using the following equation.

Divisor value = System clock / Baud rate × 16

Baudx16 clock = System clock / Divisor value

Baud clock = Baudx16 clock / 16

For a system clock of 100 MHz and Baud rate = 115200 bps.

Divisor value = 100 MHz / (115200 bps × 16)

= 36h (54 in decimal)

The Baudx16_clock is sixteen times faster than Baud clock.

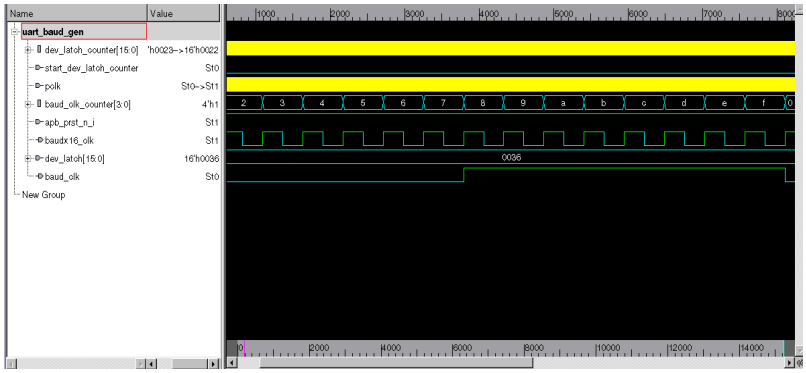


Fig. 15. Simulation result of Baud rate generator using VCS (Synopsys).

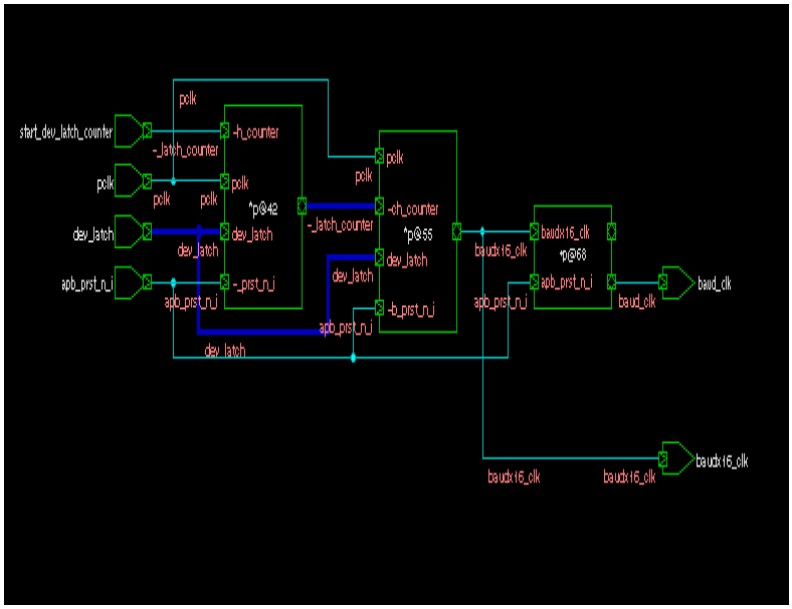


Figure 5.8: Shows Schematic view of Baud rate generator.

The simulation waveform shows synchronous FIFO design (a FIFO where writes to, and reads from the FIFO buffer are conducted in the same clock domain), one implementation counts the number of writes to, and reads from the FIFO buffer to increment (on FIFO write but no read), decrement (on FIFO read but no write) or hold (no writes and reads, or simultaneous write and read operation) the current fill value of the FIFO buffer. The FIFO is full when the FIFO counter reaches a predetermined full value (Threshold level or Trigger level) and the FIFO is empty when the FIFO counter is zero.

At the beginning the top pointer (4 bit) is incremented by Writing the data (00001010) into the FIFO by generating PUSH pulse. The bottom pointer (4 bit) is decremented by reading the data (00001010) into the FIFO by generating POP pulse. The underrun

error occurs when FIFO is empty (all the data has been poped off from FIFO). Similarly the overrun error occurs when FIFO is full (trigger level is reached).

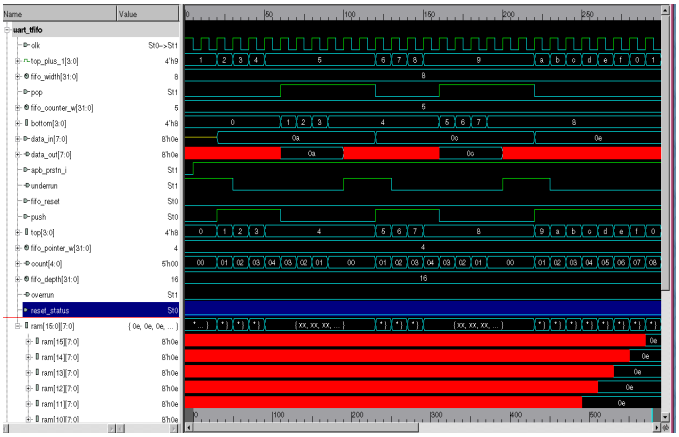


Fig. 16. Simulation result shows Sixteen byte FIFO with write and read operation using VCS (Synopsis).

The below simulation waveform shows writing the data into the FIFO and reading from at every rising edge of PCLK (system clock). First four bytes of data (0a) is loaded and poped off from the FIFO after that again four bytes of data (0c) is loaded and poped off from the FIFO.

Fig. 17. Simulation result shows RAM with different data loaded at different instant of time.

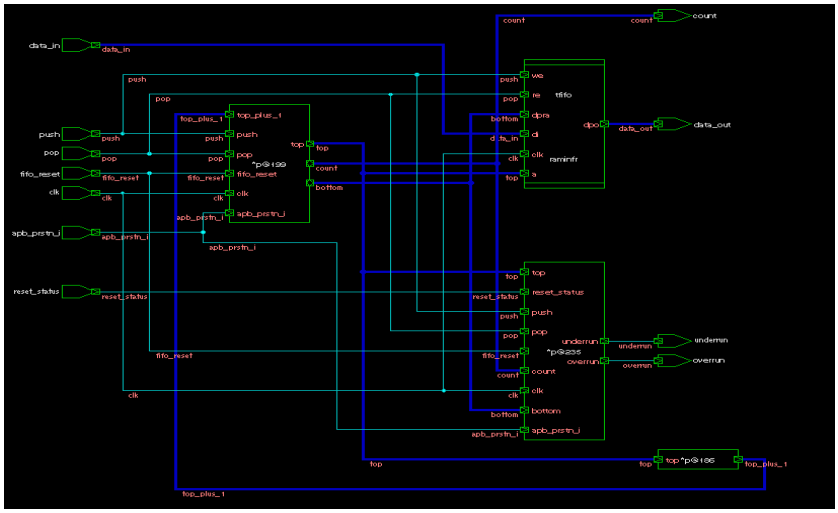


Fig. 18. Shows Schematic view of Synchronous FIFO.

When there is no data into the transmitter FIFO then THRE (Transmitter holding register empty interrupt occurs. The fifth and sixth bit of LSR will set. The reset condition for this interrupt is writing the data into the transmitter FIFO by generating pop pulse as shown in simulation waveform.

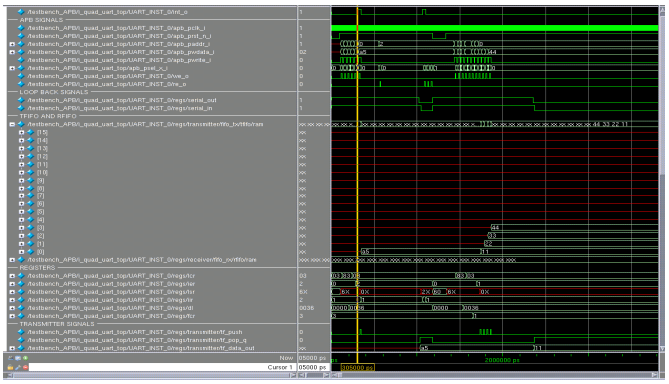


Fig. 19. Simulation result of THRE interrupt using Questasim 6.4b.

The receiver data available interrupt occurs when receiver FIFO reaches its trigger level as set in FCR (FIFO control register). Generates Receiver Line Status interrupt. The reset condition for this interrupt is reading the receiver FIFO by generating pop pulse is as shown in the simulation waveform.

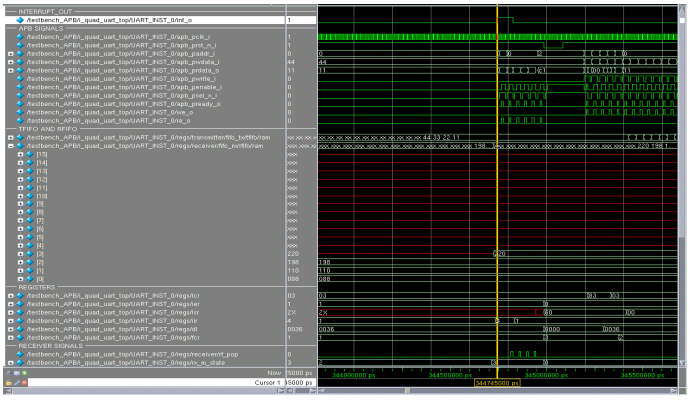


Fig. 20. Simulation result of (RDA) Receiver Data Available Interrupt using Questasim 6.4b.

A character time out (CTO) interrupt will occur when time counter reaches 00h (no character
has been input to the FIFO or read from it for the last 4 Char times. Observe that serial line is high for four character time. The reset condition for this interrupt is read the data from the receiver FIFFO.
One character time = 16 x (start + data bits 5, 6, 7 or 8 + stop bit 1, 1.5 or 2)

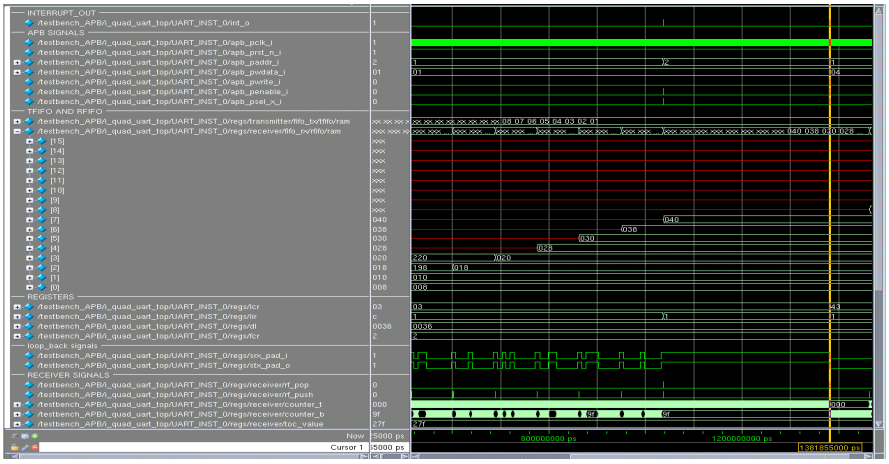


Fig. 21. Simulation result of Character Time Out Interrupt using Questasim 6.4b.

Framing error will occur when the stop bit is start bit (check at the middle of receive counter). A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit) or when the break counter reaches zero (9fh to 00h). Generates Receiver Line Status interrupt. So,Framing error will occur before break error. The reset condition for break error interrupt is reading the LSR (Line status register) by making write signal active low is as shown in the simulation waveform.

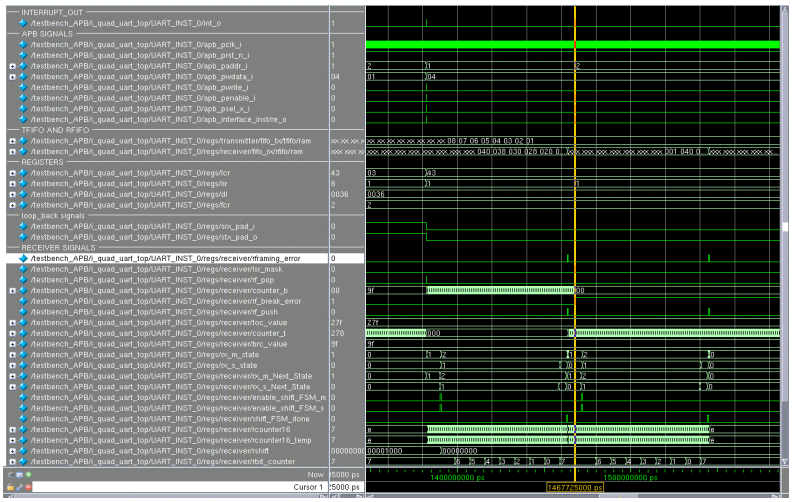


Fig. 22. Simulation result of framing error and break error (Receiver line status interrupt) using Questasim 6.4b.

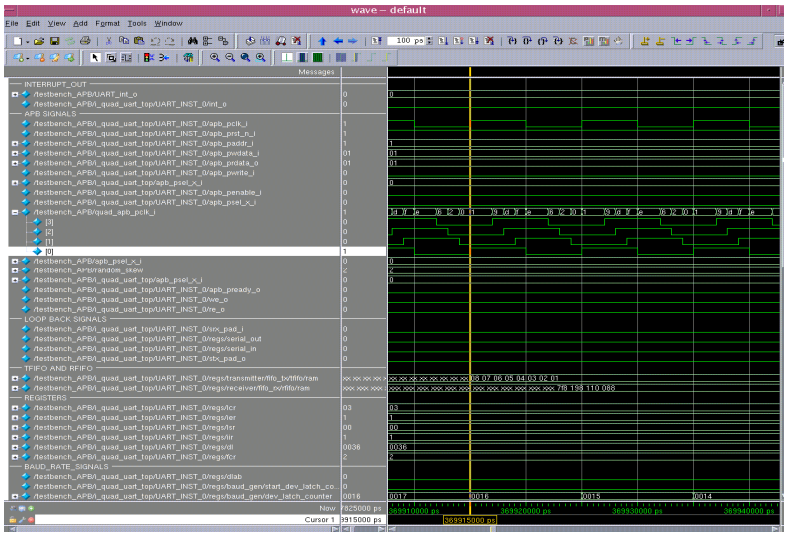


Fig. 22. Quad UART clocks with random skew.

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, “Title of paper if known,” unpublished.
[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev, in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: UniversityScience,1989.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com