Design and Verification of APB Compliant Quad Channel UART with arbitrary random skew

1Laxmisagar H.S, 2Vinutha B,
1Assistant Professor, 2Assistant Professor
1Department of ECE,
1BMS 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) with random skew suitable for use in embedded systems and System on Chip (SoC). The data transmission in each UART can be performed at different baud rate 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" behavior 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).

***Index Terms* - APB, AMBA, Random skew, Baud rate**

I. INTRODUCTION

UART (Universal Asynchronous receiver Transmitter) is a popular method of serial asynchronous communication [1]. 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's MAX232).

II. DESIGN AND IMPLEMENTATION

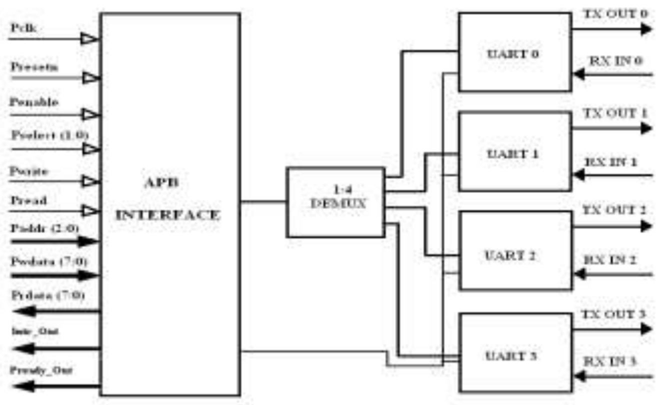


Fig. 1. APB Compliant Quad Channel UART (16550) Architecture.

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

2.1 APB (Advanced Peripheral Bus)

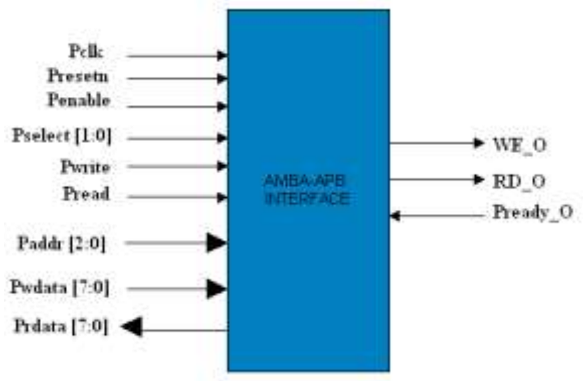


Fig. 2. Input or output signals of APB interface

2.2 Operating States

The state machine operates through the following states:

IDLE: This is the default state of the APB.

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.

ACCESS: The enable signal, PENABLE, is asserted in the ACCESS state. The address, write, select, and 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:

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

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.

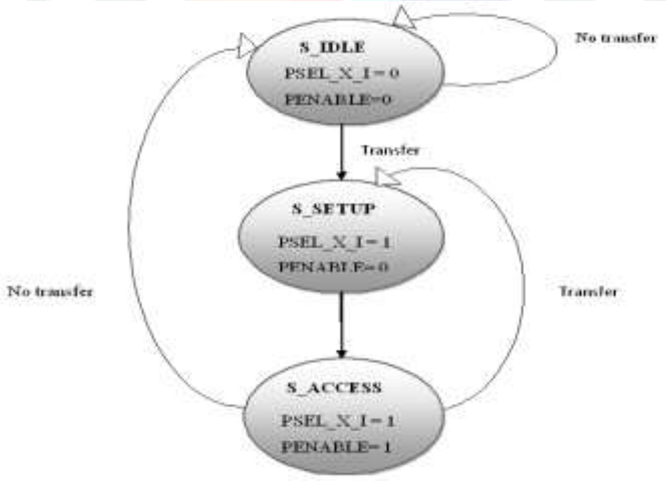


Fig. 3. Operating States of APB interface

2.3 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. Logically, the transmitter follows the steps below:

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

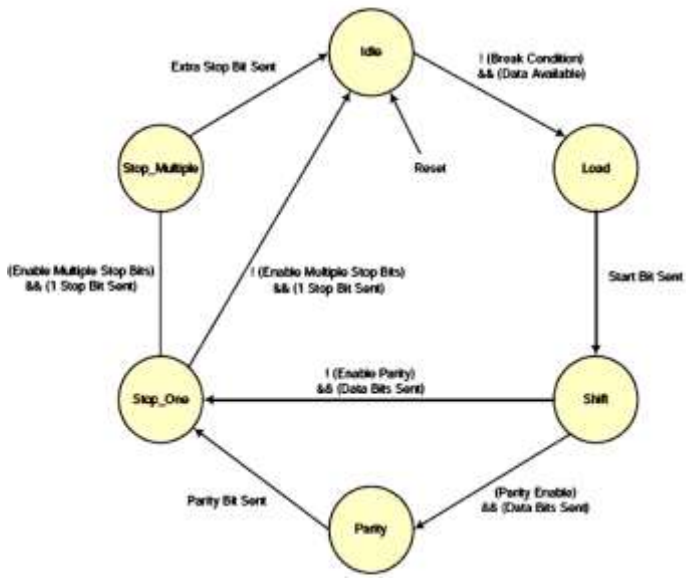

Idle
Extra Stop Bit Sent
! (Break Condition) && (Data Available)
Stop_Multiple
Reset
Load
(Enable Multiple Stop Bits) && (1 Stop Bit Sent)
! (Enable Multiple Stop Bits) && (1 Stop Bit Sent)
Start Bit Sent
! (Enable Parity) && (Data Bits Sent)
Stop_One
Shift
Parity Bit Sent
(Parity Enable) && (Data Bits Sent)
Parity

Fig. 4. Transmitter state diagram

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

- Hunt for a valid start bit, which indicates the beginning of a frame.
- Shift in data bits from the serial line to the shift register.
- If parity is enabled, compare received parity bit with the expected value.
- 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.

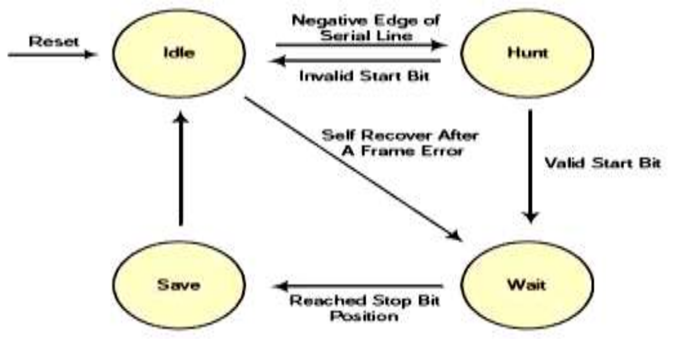

Reset
Idle
Negative Edge of Serial Line
Invalid Start Bit
Hunt
Self Recover After A Frame Error
Valid Start Bit
Save
Reached Stop Bit Position
Wait

Fig. 5. Receiver main finite state machine diagram

2.5. Baud Rate 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.

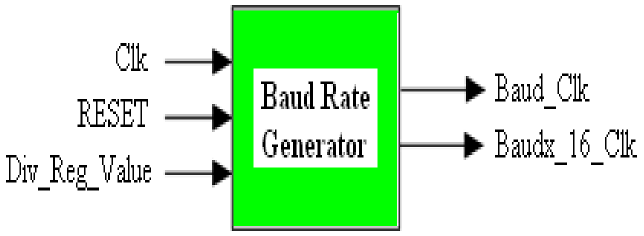


Fig. 6. I/O ports of Baud rate generator

III. RESULTS AND DISCUSSION

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.

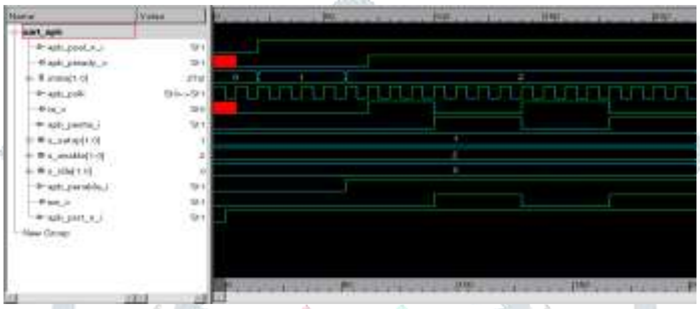


Fig. 7. Simulation result of APB Interface using VCS (Synopsys).

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.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.

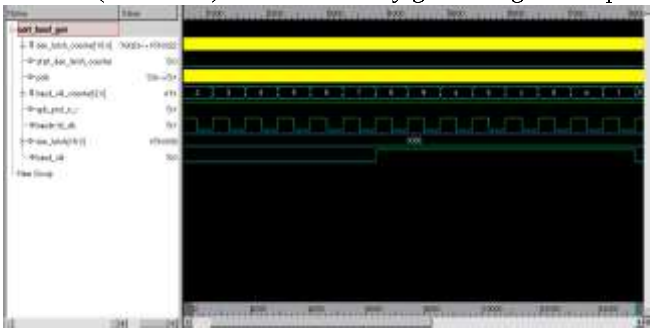


Fig. 8. Simulation result of Baud rate generator using VCS (Synopsys).

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 popped off from FIFO). Similarly the overrun error occurs when FIFO is full (trigger level is reached).

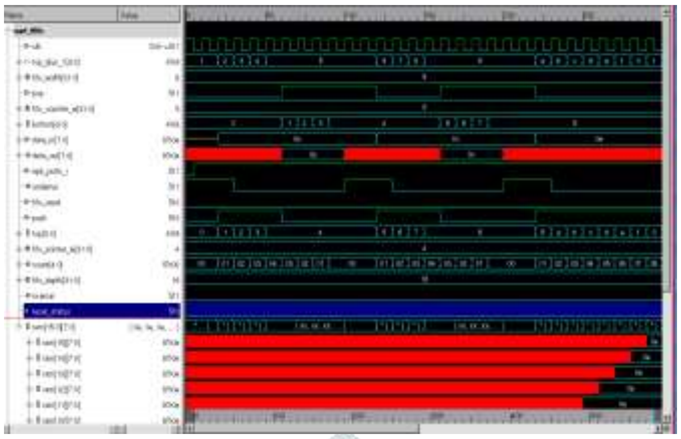


Fig. 9. Simulation result shows Sixteen byte FIFO with write and read operation using VCS (Synopsis).

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 popped off from the FIFO after that again four bytes of data (0c) is loaded and popped off from the FIFO.

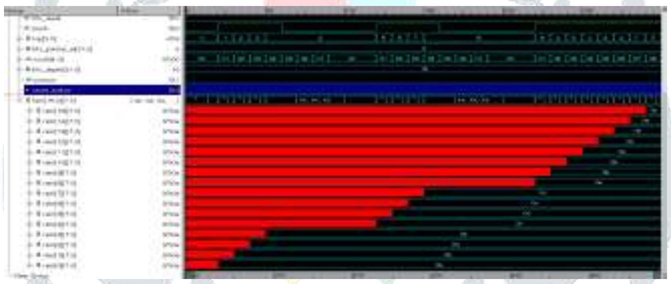


Fig. 10. Simulation result shows RAM with different data loaded at different instant of time.

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.

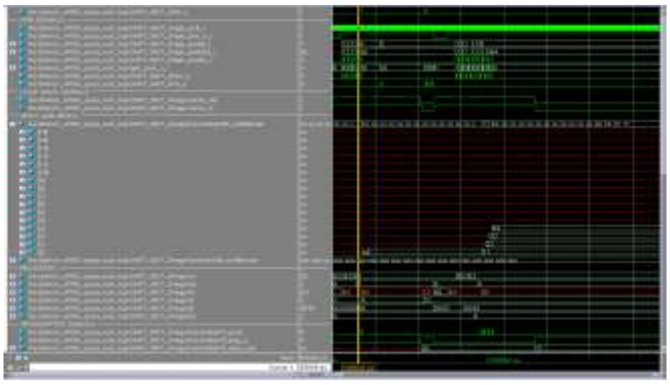


Fig. 11. Simulation result of THRE interrupt using Questasim 6.4b.

The receiver data available interrupt occurs when receiver FIFO reaches its trigger level as set in FCR (FIFO control register) and, it 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.

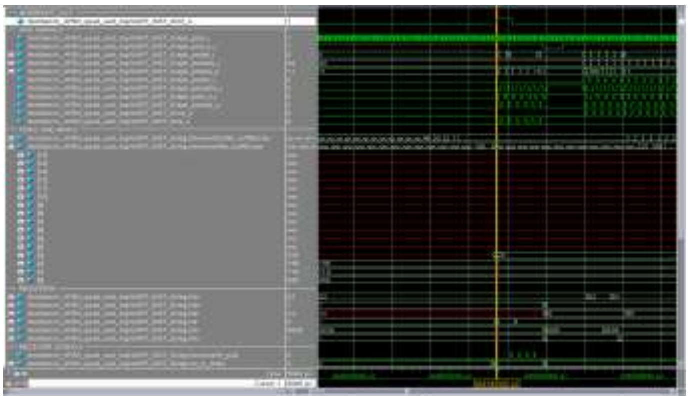


Fig. 12. Simulation result of (RDA) Receiver Data Available Interrupt using Questasim 6.4b.

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)

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), it Generates Receiver Line Status interrupt.

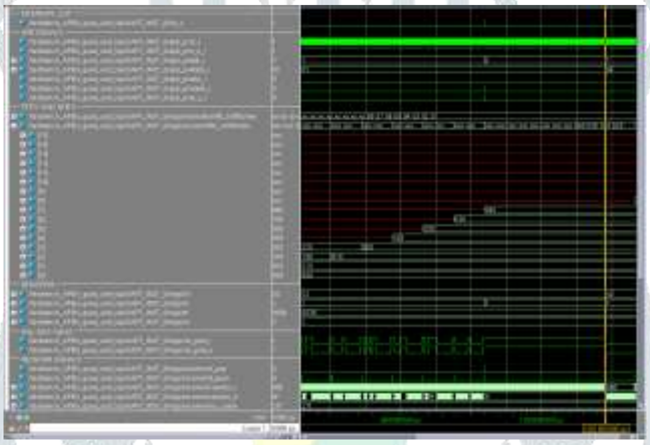


Fig. 13. Simulation result of Character Time Out Interrupt using Questasim 6.4b.

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.

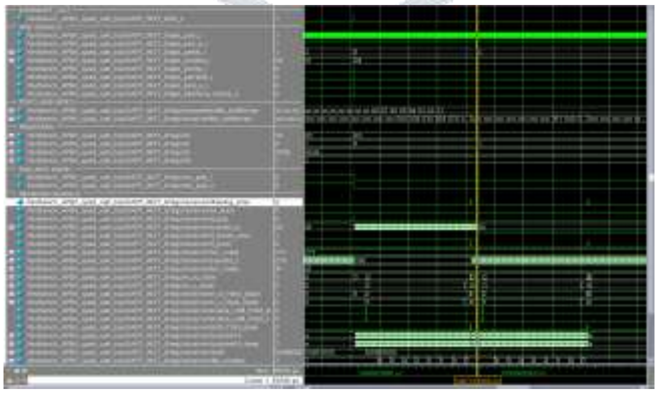


Fig. 14. Simulation result of framing error and break error (Receiver line status interrupt) using Questasim 6.4b.

IV. RANDOM SKEW GENERATION WITH SIMULATION RESULT

For a UART to work in a truly "Asynchronous" fashion, the Transmitter and the Receiver path need not have a predefined "phase" relationship, However, they still need to be operating at same Baud Rates, but need not be phase-aligned. This feature of the UART will have to be verified. One way of inducing this phase offset is by generating clocks for different instances that are randomly skewed. Although, this is one known way of inducing phase-offset between two UARTS, the implementation presented a few challenges, in that, a separate host

BFM instance would be needed to configure the device for the specified Baud rate. Since, 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.

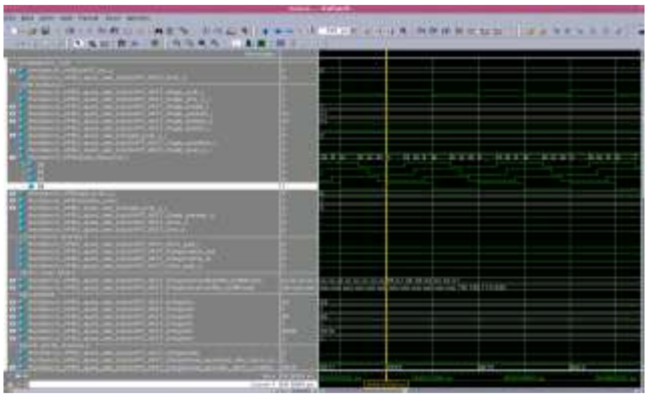
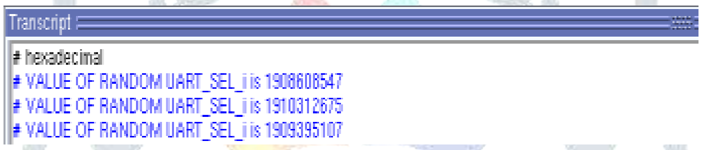


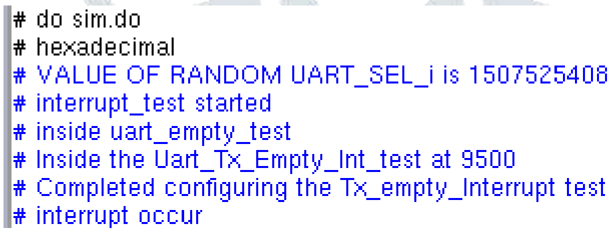
Fig. 15. Quad UART clocks with random skew.

The seed parameter controls the numbers that $random returns such that different seeds generate different random streams. The seed parameter may be a reg., an integer, or a time variable. The seed value should be assigned to this variable prior to calling $urandom. Each UART can be selected randomly by passing a seed value, which selects the corresponding random number (first two bit of LSB will be assigned to UART_SEL) accordingly selects the UART.



```
Transcript
# hexadecimal
# VALUE OF RANDOM UART_SEL_i is 1908608547
# VALUE OF RANDOM UART_SEL_i is 1910312675
# VALUE OF RANDOM UART_SEL_i is 1909395107
```

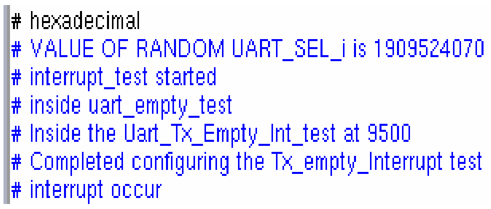
If UART_SEL is equal to "00" then first UART will be selected. The seed value for the corresponding random number 1507525408 is 12.



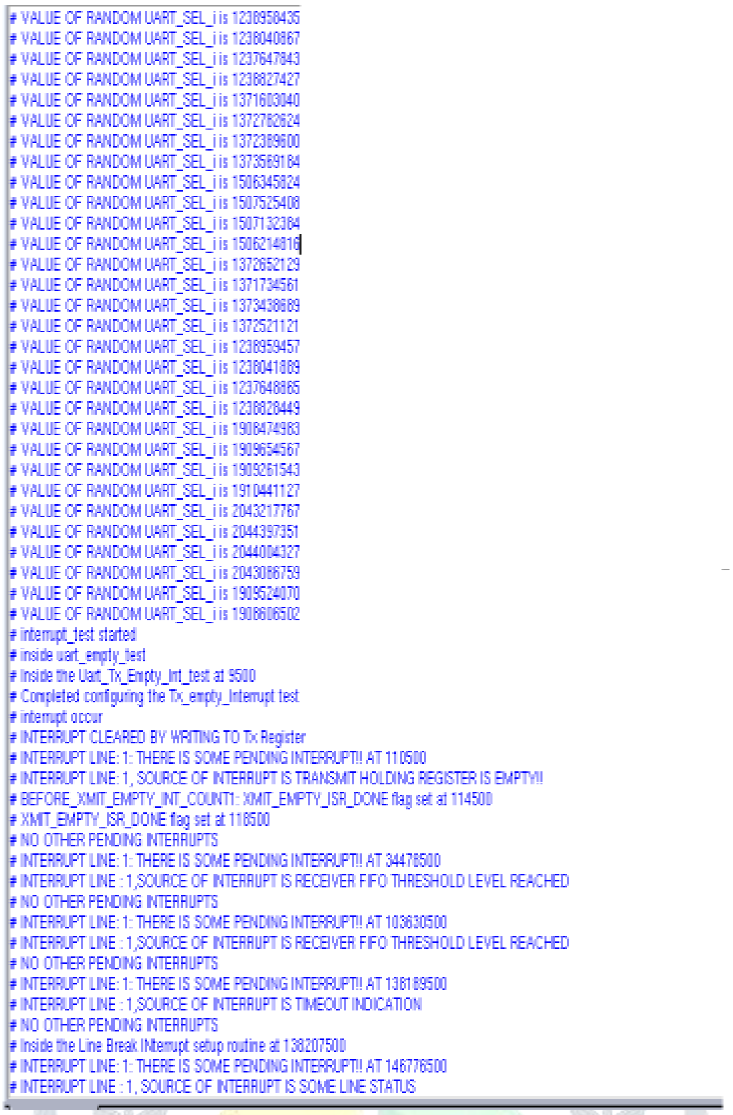
```
# do sim.do
# hexadecimal
# VALUE OF RANDOM UART_SEL_i is 1507525408
# interrupt_test started
# inside uart_empty_test
# Inside the Uart_Tx_Empty_Int_test at 9500
# Completed configuring the Tx_empty_Interrupt test
# interrupt occur
```

If UART_SEL is equal to "01" then second UART will be selected. The seed value for the corresponding random number 1372652129 is 15.

If UART_SEL is equal to "10" then third UART will be selected. The seed value for the corresponding random number 1909524070 is 31.



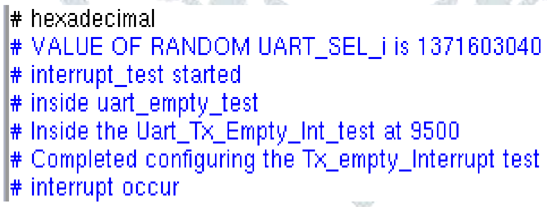
```
# hexadecimal
# VALUE OF RANDOM UART_SEL_i is 1909524070
# interrupt_test started
# inside uart_empty_test
# Inside the Uart_Tx_Empty_Int_test at 9500
# Completed configuring the Tx_empty_Interrupt test
# interrupt occur
```



```
# VALUE OF RANDOM UART_SEL_i is 1238958435
# VALUE OF RANDOM UART_SEL_i is 1238040867
# VALUE OF RANDOM UART_SEL_i is 1237647843
# VALUE OF RANDOM UART_SEL_i is 1238827427
# VALUE OF RANDOM UART_SEL_i is 1371603040
# VALUE OF RANDOM UART_SEL_i is 1372782624
# VALUE OF RANDOM UART_SEL_i is 1372389600
# VALUE OF RANDOM UART_SEL_i is 1373569184
# VALUE OF RANDOM UART_SEL_i is 1506345824
# VALUE OF RANDOM UART_SEL_i is 1507525408
# VALUE OF RANDOM UART_SEL_i is 1507132384
# VALUE OF RANDOM UART_SEL_i is 1506214816
# VALUE OF RANDOM UART_SEL_i is 1372652129
# VALUE OF RANDOM UART_SEL_i is 1371734561
# VALUE OF RANDOM UART_SEL_i is 1373438689
# VALUE OF RANDOM UART_SEL_i is 1372521121
# VALUE OF RANDOM UART_SEL_i is 1238959457
# VALUE OF RANDOM UART_SEL_i is 1238041889
# VALUE OF RANDOM UART_SEL_i is 1237648865
# VALUE OF RANDOM UART_SEL_i is 1238828449
# VALUE OF RANDOM UART_SEL_i is 1908474983
# VALUE OF RANDOM UART_SEL_i is 1909654567
# VALUE OF RANDOM UART_SEL_i is 1909261543
# VALUE OF RANDOM UART_SEL_i is 1910441127
# VALUE OF RANDOM UART_SEL_i is 2043217767
# VALUE OF RANDOM UART_SEL_i is 2044397351
# VALUE OF RANDOM UART_SEL_i is 2044004327
# VALUE OF RANDOM UART_SEL_i is 2043086759
# VALUE OF RANDOM UART_SEL_i is 1909524070
# VALUE OF RANDOM UART_SEL_i is 1908606502
# interrupt_test started
# inside uart_empty_test
# Inside the Uart_Tx_Empty_Int_test at 9500
# Completed configuring the Tx_empty_Interrupt test
# interrupt occur
# INTERRUPT CLEARED BY WRITING TO Tx Register
# INTERRUPT LINE: 1: THERE IS SOME PENDING INTERRUPT!! AT 110500
# INTERRUPT LINE: 1, SOURCE OF INTERRUPT IS TRANSMIT HOLDING REGISTER IS EMPTY!!
# BEFORE_XMIT_EMPTY_INT_COUNT1: XMIT_EMPTY_ISR_DONE flag set at 114500
# XMIT_EMPTY_ISR_DONE flag set at 118500
# NO OTHER PENDING INTERRUPTS
# INTERRUPT LINE: 1: THERE IS SOME PENDING INTERRUPT!! AT 34478500
# INTERRUPT LINE : 1,SOURCE OF INTERRUPT IS RECEIVER FIFO THRESHOLD LEVEL REACHED
# NO OTHER PENDING INTERRUPTS
# INTERRUPT LINE: 1: THERE IS SOME PENDING INTERRUPT!! AT 103630500
# INTERRUPT LINE : 1,SOURCE OF INTERRUPT IS RECEIVER FIFO THRESHOLD LEVEL REACHED
# NO OTHER PENDING INTERRUPTS
# INTERRUPT LINE: 1: THERE IS SOME PENDING INTERRUPT!! AT 138189500
# INTERRUPT LINE : 1,SOURCE OF INTERRUPT IS TIMEOUT INDICATION
# NO OTHER PENDING INTERRUPTS
# Inside the Line Break INterrupt setup routine at 138207500
# INTERRUPT LINE: 1: THERE IS SOME PENDING INTERRUPT!! AT 146776500
# INTERRUPT LINE : 1, SOURCE OF INTERRUPT IS SOME LINE STATUS
```

Fig. 16. Showing simulation report of random numbers using Questasim 6.4b.

If UART_SEL is equal to "11" then fourth UART will be selected. The seed value for the corresponding random number 1371603040 is 7.



```
# hexadecimal
# VALUE OF RANDOM UART_SEL_i is 1371603040
# interrupt_test started
# inside uart_empty_test
# Inside the Uart_Tx_Empty_Int_test at 9500
# Completed configuring the Tx_empty_Interrupt test
# interrupt occur
```

V. CONCLUSION

For a UART to work in a truly "Asynchronous" fashion, the Transmitter and the Receiver path need not have a predefined "phase" relationship, However, they still need to be operating at same Baud Rates, but need not be phase-aligned. This feature of the UART will have to be verified. One way of inducing this phase offset is by generating clocks for different instances that are randomly skewed. Although, this is one known way of inducing phase-offset between two UARTS, the implementation presented a few challenges, in that, a separate host BFM instance would be needed to configure the device for the specified Baud rate. Since, 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" behavior of the UART.

REFERENCES

[1] E. P. Superior, "Universidad Carlos III de Madrid."

[2] J. Teubner and L. Woods, "Data Processing on FPGAs," *Synth. Lect. Data Manag.*, vol. 5, no. 2, pp. 1–118, 2013.

[3] F. Y. C. Xue-jun, "Design and Simulation of UART Serial Communication Module Based on VHDL," vol. 1, 2011.

[4] H. H. Maimun, "The Design of High Speed UART," pp. 306–310, 2005.

[5] G. B. Wakhle, "Synthesis and Implementation of UART using VHDL Codes," pp. 3–5, 2012.