Design And Verification Of Autoconfigurable UART Controller

Stepan Harutyunyan
Chair of Microelectronic Circuits and Systems National Polytechnic University of Armenia
Synopsys Armenia Educational Department
Yerevan, Armenia
stepanh@synopsys.com

Taron Kaplanyan
Chair of Microelectronic Circuits and Systems National Polytechnic University of Armenia
Synopsys Armenia Educational Department
Yerevan, Armenia
tkaplan@synopsys.com

Artak Kirakosyan
School of Electrical Engineering and Computer Science University of Ottawa
Ottawa, Canada
akira034@uottawa.ca

Arsen Momjyan
Chair of Microelectronic Circuits and Systems National Polytechnic University of Armenia
Synopsys Armenia Educational Department
Yerevan, Armenia
momjyan@synopsys.com

***Abstract*—UART is one of the most widely used interfaces. It is as single bit TX and RX interface which supports multiple configurations. It supports data length of 5-8 bits, even, odd or missing parity bit and 3 more stop bit counts. As most UART devices have fixed configuration it usually requires manual configuration of both sides to support proper data transfers. This paper presents a novel design of UART interface that will remove that limitation. Proposed UART controller can detect baud rate automatically and adjust to it. Additionally, there is a new command interface which is used to understand remaining config of the device to fully adapt to the connected device. That will allow any device to connect to proposed UART controller and work without any manual configuration. The UART controller RTL has been developed using Verilog and the test environment was developed using SystemVerilog. Simulations have been done using VCS.**

Keywords—UART, Verification, Verilog, System Verilog

I. INTRODUCTION

UART(Universal Asynchronous Receiver Transmitter) is the common and simple interface used in data transfer applications between 2 devices. It has Receive and Transmit lines, which are asynchronous [1]. The data transfer concept in UART is common for all devices, but the packets formats can vary from device to device as well as the speed of the data transfer (baud rate: measured in bit/s) can be changed.

UART is used in different devices. e.g. in PC's COM ports which are backward compatible with UART, bluethoot-UART devices, all devices where configuration process is required, devices with external command interfaces and so on.

We can see the UART is simple serial interface. Most devices have fixed configuration and controlling devices should be configured accordingly. In most microcontrollers or microprocessor chips, UART controller is used thanks to its flexiblility[2]-[4]. It can support all configurations, but user need to gather information about used UART device, and configure it's application correspondingly to be able perform data transfers. To resolve this issue, an enhancement for the UART controller in face of automatically detection of the UART partners parameters is proposed. The only limitation is that, UART partner device should also support this enhancement so that proposed system can detect the parameters. Backward compatibility is also implemented and once parameter detection process fails, UART controller continues to operate with predefined parameters from it's application. There is existing implementation of UART controllers, which can automatically detect baud rate[5], but these systems supports only baud rate detection and not able to automatically detect the packet width and parity bit modes. To even more enhance the concept of flexible UART controller a new command set has been defines. That command set is used to communicate the configuration parameters between UART partners and the Controller addapts to them. This is covered in more details in upcoming chapters.

II. UART OVERVIEW

In all devices the integrated UART controllers have different configuration. Following presenting the parameters of UART packets:

978-1-7281-9713-5/20/$31.00 ©2020 IEEE

- Wide range of baud rate: 300b/s - 1Mb/s
- 5, 6, 7, 8-bit data bits in packet.
- 1, 1.5, 2 – stop bits.
- Even/Odd parity data protection.

From the presented it is seen that the UART can have different packet formats. UART interface is 1-bit serial transmit and 1-bit serial receive signals. The TX(transmit) signal of one device connected to the RX(receive) signal of the second device and vice versa. In Fig. 1 an example of UART data transfer packet is presented.

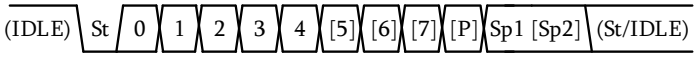

Fig. 1 UART Packet Configurations

When devices, connected through the UART, are powered-on, both lines should be in IDLE state (logical '1'). The communication starts from the Start bit (logical '0'). The link partner should detect the negative edge on the RX line and and monitor. That line should remain low as long as the corresponding baud rate is. 1 bit time in seconds is equal to 1/(Baud Rate). After start bit, transmitter sends data from LSB(least significant bit) to the MSB(most significant bit) and sets as many bits as it is defined in settings. After data transfer completed there is option to insert parity bit. If parity enabled, the transmitter should transmit Parity bit. The value of parity bit calculated as follows

$$P_{even} = d_{n-1} \wedge \ldots \wedge d_2 \wedge d_1 \wedge d_0 \wedge 0 \quad (1)$$

$$P_{odd} = d_{n-1} \wedge \ldots \wedge d_2 \wedge d_1 \wedge d_0 \wedge 1 \quad (2)$$

To calculate the parity bit, the "exclusive or" binary operation is done between all data bits, then, depending on parity type the final result will be xor-ed with logical '0'(even parity) or logical '1'(odd parity). After transmittion of parity bit device is required to send to it's UART partner the Stop bit (logical '1').

III. Proposed UART Enhancement

Designed enhanced for UART controller is able to detect the connected device baud rate, data transfer packet width, parity enable/disable mode, parity calculation method and stop bit count parameters. In original UART there is no master-slave concept, but in this enhancement, it is differentiated the role of auto-configurable UART controller (Master, Host) and connected device (Slave, Device) like in USB [1]. The application of host UART has to decide when to start the auto-configuration process. The Host UART starts initialization of the slave by transmitting reset signal.

As shown in Fig. 2, to send to start transmitting reset signal, Host UART sends logical '0' with duration of 10ms. As the minimum speed of supported Host UART is 300b/s, 10ms duration has been chosen for logic '0'. This will guarantee low state detection for slowest systems. Once, slave received '0' transition on the RX line, it should wait for RX line to become logical '1'. After that the Slave should transmit a packet with data of 0xFF, without parity bit. This is presented in Fig. 3. While slave sends that packet, the host UART calculates baud rate based on bit duration. When data transfer is completed the host UART has already calculated baud rate and notifies to its application If the slave UART is not responding on reset signal within 50ms, host UART retries auto-configuration process, by sending reset signal again. Host UART retries 2 times. If the process fails 3 times, Host UART notifies its application about failure and moves to backward compatibility mode and operating as a usual UART.

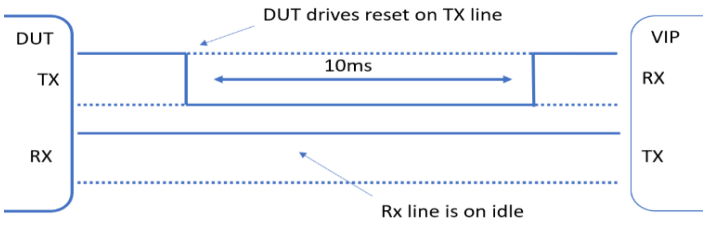

Fig. 2 Reset Sequence

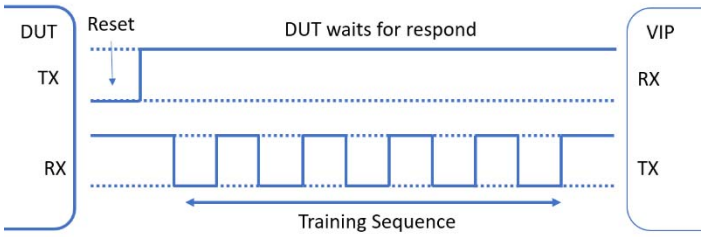

Fig. 3 Training Sequence

Once host UART calculated the baud rate of UART partner, it starts to operate at detected speed. To detect other parameters, new command structures has been developed. In Fig. 4 it is shown the packet width command. Host UART should receive another packet with 8-bit data, without parity. The 1st and 0th bits of data are command ID and for packet width they should have 2'b01 value. The 3rd and 2nd bits define packet width. The other data bits are don't care. Following mentions the values and meaning:

- 2'b00 – 5-bit packet width
- 2'b01 – 6-bit packet width
- 2'b10 – 7-bit packet width
- 2'b11 – 8-bit packet width

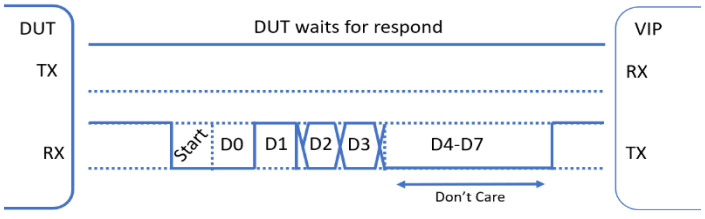

Fig. 4 Packet Width Command

In Fig. 5 the stop bit count configuration is shown. The command ID is 2'b10. Following values applied for the 4th and 3rd bits:

- 2'b00 – 1 stop bit
- 2'b01 – 1.5 stop bit
- 2'b10 – reserved
- 2'b11 – 2 stop bits

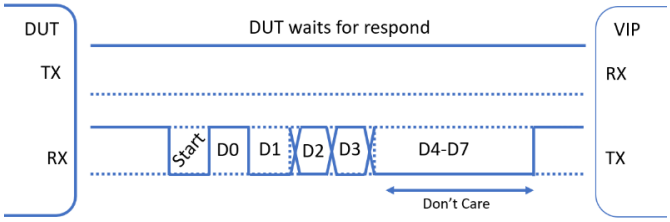

Fig. 5 Stop Bit Count Command

The parity mode configuration is shown in Fig. 6. The command ID is 2'b11. Following values applied for the 4th and 3rd bits:

- 2'b00 – parity disabled
- 2'b01 – parity enabled, even parity
- 2'b10 – parity disabled
- 2'b11 – parity enabled, odd parity

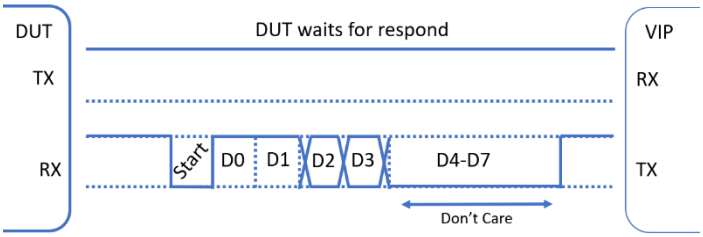

Fig. 6 Parity Mode Command

Once all these parameters detected, Slave can decide to complete the auto-configuration process or change one of the above parameters, beside speed. To end process, it should send command with ID of 2'b00. If some parameters are not sent by the slave UART and auto-configuration process is completed, then those parameters will have their default values from UART Host Application.

IV. UART VIP AND TEST ENVIRONMENT

In order to verify the proposed architecture enhanced UART VIP has been developed. The VIP and Testbench architecture are shown in Fig. 7.

As shown in the figure tesbaench consists only from the VIP and DUT. They are connected with 2 interfaces which are used for both configuration and data transfer. Since the VIP was implemented in SytemVerilog it was seperated into several modules. It is seperated into Received, Driver, generator and Clock Gen module. Short description of those modules:

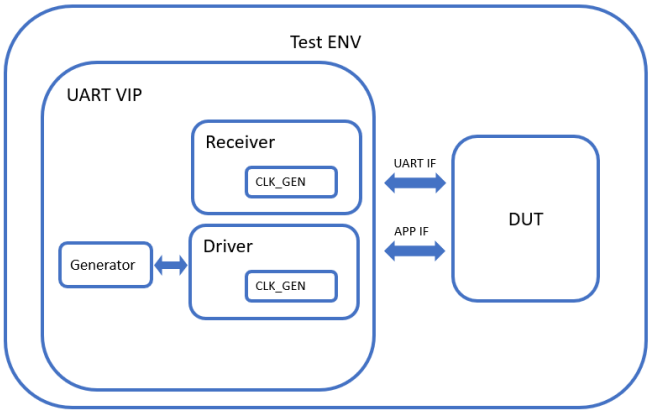

Fig. 7 Test Environment Architecture

Receiver: This module is reposnsible for monitoring the DUTs TX line. It receives data and makes sure that data format is according to the selected format. The Clock Gen module is implemented into Receiver to provide correct UART timings.

Driver: This module is responsible for sending the UART packets from VIP to DUT. The Driver is being configured at the beginng of simulation and sends the packet according to it. Just like Receiver, Driver also has its own Clock Gen to support UART timings.

Generator: This block is responsible for UART packet generation. It generates the packets and passes them to Driver module through mailbox.

CLK_GEN: This module is responsible to keep the correct timings inside Driver and Generator. It generates the required BCLK for proper operation of the UART. This module uses event based mechanism to triger data sampling.

As connected device might not support the proposed command set backward compatability should be supported too. To support this feature the VIP has implemented state mashine which switches to usual UART operating model if the training sequence is not completed correctly or if the RESET is not detected from DUTs side.

V. SIMULATION RESULTS

Simulations have been done using Synopsys VCS [6]. To test the feature simple testcase of device connection has been implemented. When device is connected to the proposed UART controller tries to reset it and determine its baud rate. The simulation waveforms are presented in Fig. 8.

Operating model of the Environment. After the simulation starts the VIP waits for reset sequence. Timeout mechanism has been implemented to avoid infinite loop. Reset will be detected

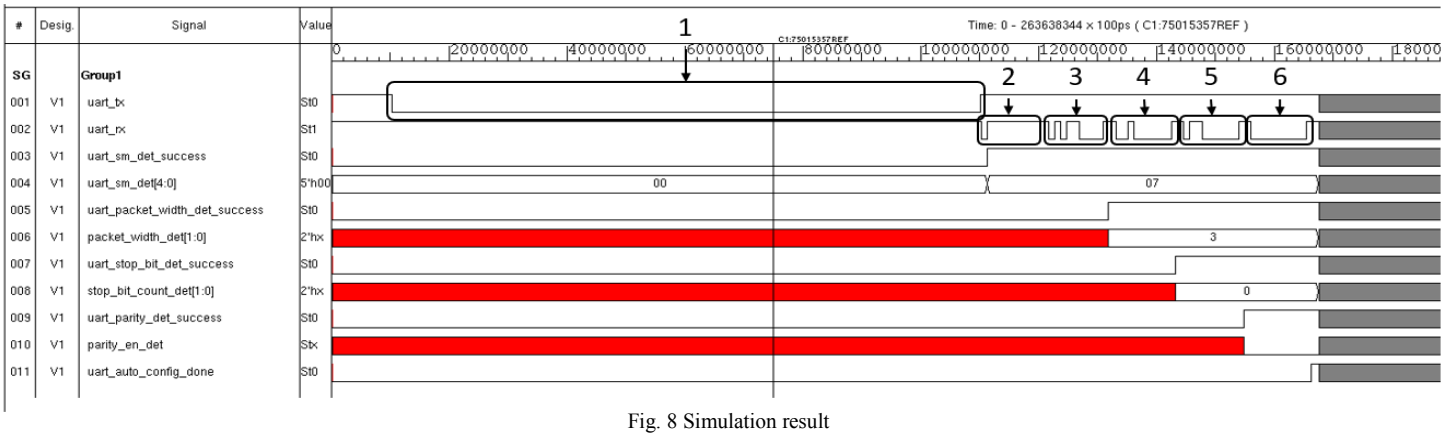

Fig. 8 Simulation result

if the DUT drives logic 0 on the TX line for 10ms. That is shown by "1" in figure.

After that is done DUT restores IDLE state on TX and starts to drive logic high TX line. Now DUT is on wait state. After the reset has been initiated the device (VIP in this case) should start sending training sequence. In Fig. 8 configuration has been done in harsher conditions. Only Start bit is being used to calculate the baud rate. The device will send this data using it's own baud rate. Since the data is exact sequence of 0 and 1 the VIP will be able to detect the length of START bit. This will allow it to understand what baud rate the device is using. Number "2" in Fig 8.

After training sequence VIP sends data length information. The information is coded inside next uart packet. First 2 bits of next packet will corespond to the command, and next 2 will be the packet width of the device. After the packet is received coresponding signals in DUT trigger confirming that packet was received and understood correctly. "3" in Fig. 8.

Next packet holds the information about stop bit count ("4" in Fig. 8). Next comes the configuration packet from which the parity bit information can be extracted ("5" in Fig. 8). At the end the device sends lasts configuration packet of 0s to complete the configuration. Packet with data 0 means that configuration on the DUT side is completed and data transfers can happen according to the Devices configuration.

Fig. 8 shows the successful baud rate generation and configuration. Assertion of specific signals can be seen after each packet. Backward compatibility has also been tested. VIPs configuration has been modified to support only usual UART. And again, connect testcase was initiated. After 3 attempts to send the reset sequence the DUT returned to standard UART operation and starts to send data according to its baud rate.

To make sure that Baud Rate detector on DUT works correctly all the baud rate options has been exercised through VIP. All 19 baud rate configuration have successfully been detected.

VI. CONCLUSION

Most of the UART devices are working on fixed configuration. To fix the configuration limitation an enhanced UART controller architecture has been presented. It allows to automatically detect the baud rate of the connected device and adapt to it. To even further fix configuration problem an instruction set has been introduced. After baud rate detection is completed, those instructions are used to further configure the Controller. Data width, Stop bit count and parity information is received through those commands. To verify the design verification environment has been developed using configurable UART VIP. In case the connected device doesn't support this enhancement, Controller returns to basic UART and continues normal operation. Proposed Controller can be used both as host and as a device.

REFERENCES

[1] Instruments, Texas. "Keystone architecture universal asynchronous receiver/transmitter (uart) user guide.", Texas Instruments User Guide (2010).

[2] Poorani M., Kurunjimalar R. "Design implementation of UART and SPI in single FGPA", IEEE 10th International Conference on Intelligent Systems and Control (ISCO), C. 1-5, 2016.

[3] Wakhle, Garima Bandhawarkar, Aggarwal, Iti., and Gaba, Shweta, "Synthesis and Implementation of UART using VHDL Codes," IEEE International Symposium on Computer, Consumer and Control, pp. 1-3, 2012.

[4] T.P.Blessington, B.B.Murthy, G.V.Ganesh and T.S.R Prasad, "Optimal Implementation of UART-SPI Interface in SOC", Devices, Circuits and Systems (ICDCS), International Conference, pp.673-67, 2012.

[5] AN4908, STM32 USART automatic baud rate detection, STMicroelectronics, 2016

[6] VCS® User Guide,Synopsys Inc. – Dec 2019. -2106p.

[7] Chris Spear, "System Verilog for Verification, A guide to Learning the Testbench Language Features", 3rd ed., pp 295-332, 2012.

[8] Bergeron J. et al. "Verification methodology manual for SystemVerilog" – Springer Science & Business Media, pp. 260-280 2006.

[9] Accellera, Universal Verification Methodology 1.1 User's guide, CA, (2011).

[10] Chu, Xiao Bin, L. U. Tie-Jun, and Y. Zong, "Combination of Assertion and Coverage-Driven Verification Methodology." Microelectronics & Computer 11, 2008.