Reconfigurable Arbitrary Waveform Generator

Swati Mohite

***Abstract*--The waveform generators being used traditionally are basically ASIC based. The features of ASIC based embedded system are not easy to alter. In other words, ASIC based embedded systems are not reconfigurable. The waveform generator under discussion here is developed using FPGA. FPGAs have an unique characteristic namely reconfigurability. Also this waveform generator is interfaced with serial port of personal computer using UART (Universal Asynchronous Receiver Transmitter) protocol. To establish the serial link between PC and FPGA, UART is implemented in VHDL from FPGA side and from PC end Visual Basic is used as design language. Using MSComm component of VB, serial port of PC is programmed. As a result of this serial link established, the netlist running inside FPGA can be altered in runtime. This is again an added advantage of reconfigurable waveform generator.**

***Key words*: Reconfigurable, Runtime updation, FPGA, UART, Waveform Generator**

I. Introduction

TRADITIONALLY used embedded systems are mostly 'ASIC' (Application Specific IC) based. Modifying any of the currently existing system's characteristics is extremely complicated. Rather building a new system with modified characteristics would be cost effective. The use of FPGA in place of ASIC, gives us facility to reconfigure the system without much efforts.

Also, developing a new system with actual components, before even testing it's characteristics, can be risky at times. In such situations, use of FPGA is of great help. We can emulate the new system in FPGA and test it's characteristics before we actually build them.

In the system described in this paper, we have interfaced PC's serial port with FPGA system. The system developed can be represented in Fig.1

The major sections of this paper are serial port communication, algorithms for UART implementation and waveform generation, simulation results of the system developed.

Swati Mohite is a Post Graduate Student at the Department of Electronics Engineering at Fr. C. Rodrigues College of Engineering, Mumbai India (e-mail: mohiteswati@gmail.com).

II. Serial Communication

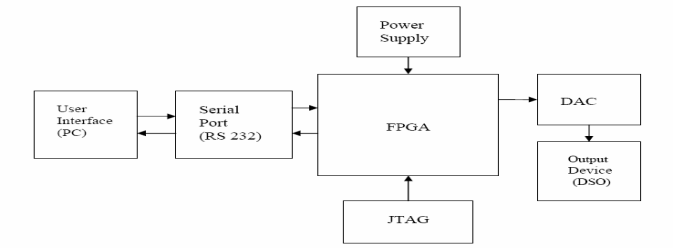

Fig.1 Functional Block Diagram

Although parallel communication is faster than serial communication, serial communication is cost effective over parallel communication. Hence for longer distances, serial communication is preferred.

Fig. 2 shows an example of synchronous serial communication and Fig. 3 shows an example of asynchronous serial communication.

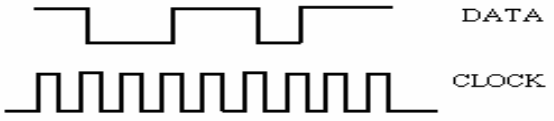

Fig.2 Separate Clock and Data

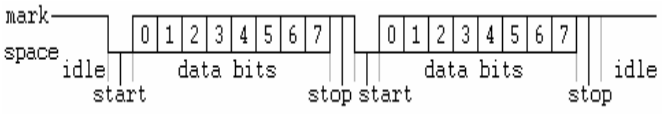

Fig. 3 Asynchronous Transmission

A. Universal Transmitter (UART)

UART is a piece of computer hardware that translates data between parallel and serial interfaces. Used for serial data telecommunication, an UART converts bytes of data to and from asynchronous start-stop bit streams represented as binary electrical impulses.[1] UARTs are commonly used in conjunction with other communication standards such as EIA RS-232. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.

Basic implemented components of UART are shown in Fig. 4.

The designed UART consists of the following blocks

1) Transmitter section
2) Receiver section
3) Baud Rate Generator

1) Transmitter Section: The transmitter section is responsible for transmission of serial data. Fig. 5 shows the flowchart for transmitter section. This section has the following components:

a) Transmitter Buffer: It holds the 8 bit data to be shifted out of the UART. When 'load' is pulsed, the 8 bit data available at 'data in' bus is loaded into the buffer. This data is then given to the shift register and parity generator for further processing.
b) Transmitter Shift Register: When data is loaded into the buffer, the start, stop and parity bits are appended to the 8 bit data stream, and data is serially shifted out of the shift register. The output signal 'BUSY' remains high until all the bits are shifted out of the register.
c) Parity Generator: The parity generator takes the data loaded into the buffer and generates a parity bit by simply EXORing all bits of the data stream. The parity bit obtained is EVEN. This block is only active when 'parity_en' is high or '1'.

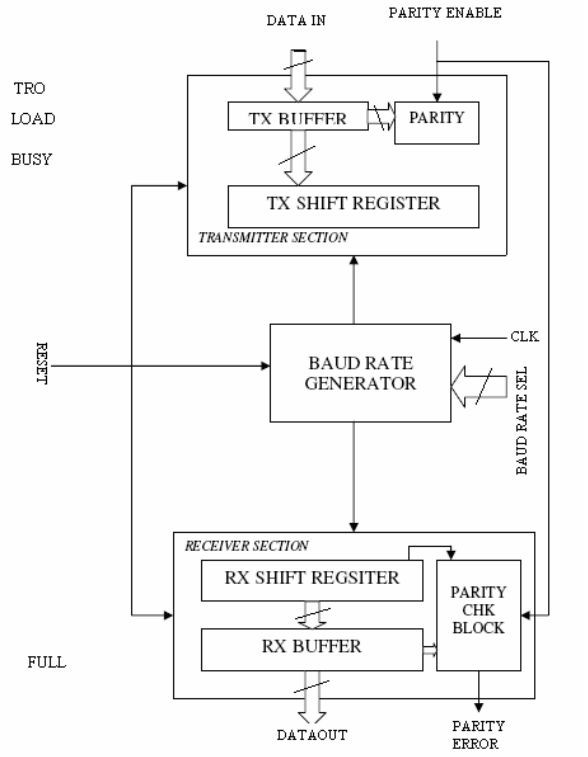

Fig. 4 Implemented UART Block Diagram

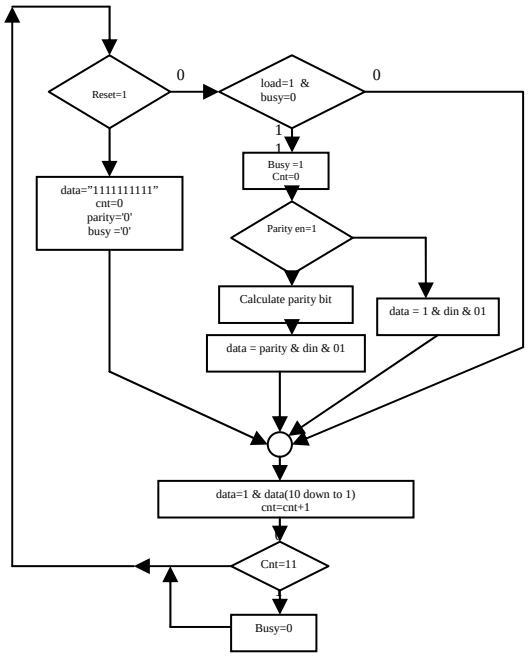

Fig. 5Transmitter Section Flowchart

2) Receiver Section: The Receiver Section converts the serial data stream back into a parallel data stream. Fig. 6 shows the state diagram explaining receiver working. This section has the following components.

a) Receiver Shift Register: When this component detects the start bit, it begins shifting data into the shift register. It does so until the required number of bits is shifted into the shift register.
b) Receiver Buffer: The buffer holds the data until the next data is not completely received. After the next data is received the buffer is cleared and new data is transferred to the buffer.
c) Parity Checker: The parity generator is enabled only when 'PARITY_ENABLE' is high. The parity generator EXORs the data from the buffer, and generates a parity bit. It then compares this bit with that transmitted by the transmitter. If the two do not match parity error occurs and 'PARITY ERROR' signal is made high.

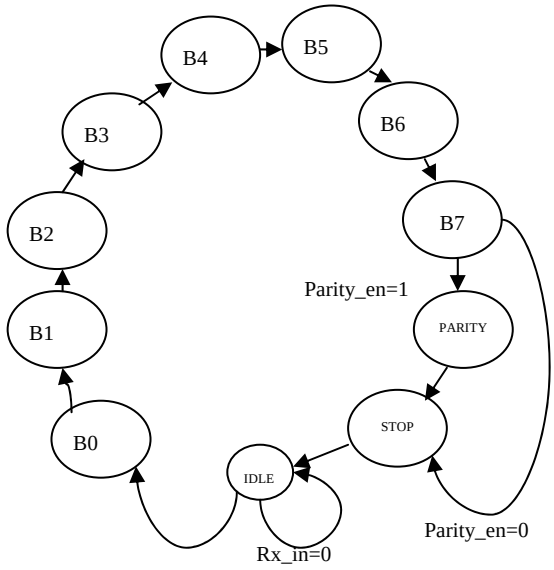

Fig. 6 Receiver State Diagram

3) Baud Rate Generator: The Baud Rate Generator generates the internal clock signal by simply dividing the external input clock frequency by the required amount to obtain the required baud rate. The generator consists of a counter which performs the dividing operation. The amount by which the counter divides the input clock depends on the 'CNT_LIMIT' signal. This bus has a width of sixteen bits, hence the maximum dividing factor is $(2^{16}-1) = 65535$. The minimum count allowed is 2. Fig. 7 shows the flowchart of baud rate generator.

III. WAVEFORM GENERATOR

The waveform generator developed here is an arbitrary waveform generator. The waveforms of the traditional function generator are typically square wave, sine wave, sawtooth wave. The waveform generator developed here is capable of generating any random waveform like staircase waveform. Also, this idea can be extended to any other random waveform as this waveform generator is reconfigurable. To interface this digital output to real world, a digital to analog converter (DAC) is used. The DAC used is 12-bit DAC (AD7541). The digital output of FPGA is given to DAC and analog output of DAC is seen on digital storage oscilloscope (DSO). For the sake of illustration of this idea, three different waveforms are produced.

1) Sawtooth waveform
2) Straircase Waveform
3) Triangular Waveform

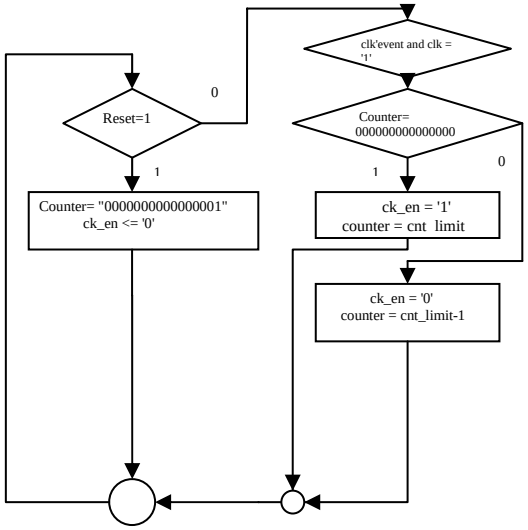

Fig. 7 Flow Chart of Baud Rate Generator

A. Sawtooth Waveform Generator

To generate the sawtooth waveform, a 12-bit counter is used. At every rising edge of clock, counter increases by 1. This continues till the counter reaches to its highest value (cnt > = "111111111111"). After counter reaches to its highest value, counter is reset. This continues in the loop.

Flowchart of sawtooth waveform generation is shown in Fig. 8.

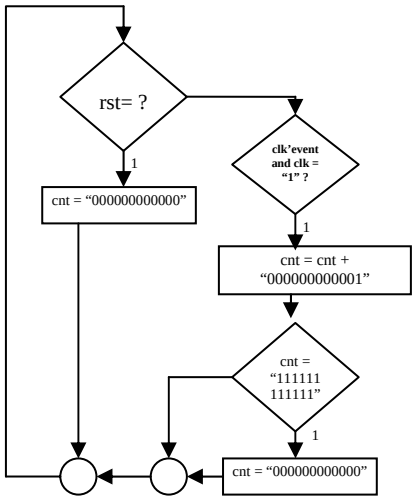

Fig. 8 Sawtooth Waveform Flowchart

B. Staircase Waveform Generator

To generate staircase waveform, a 12-bit counter is used along with an integer variable. At every rising edge of clock, integer variable increases by 1. When integer variable (i) reaches 1000 (which defines width of staircase), counter (cnt) increases by "000001001110" (which defines height of staircase) and integer variable is rest to 0. Once counter reaches its maximum value, it is reset to 0.

Fig. 9 shows the flowchart for staircase waveform generator.

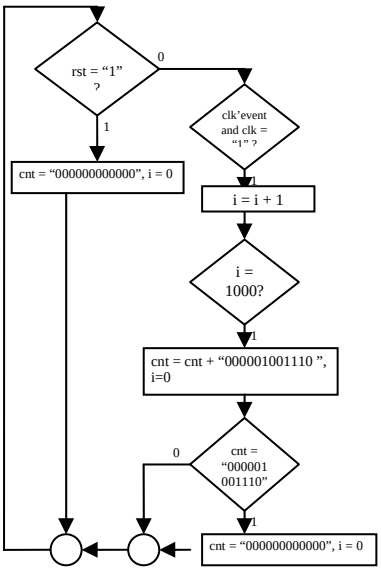

Fig. 9 Staircase Waveform Flowchart

C. Triangular Waveform Generator

To generate triangular waveform, a 12-bit counter is used. This counter moves up and down depending on flag value. If flag is set counter counts in the upward direction; otherwise it counts in downward direction. Working in detail is explained in Fig.10.

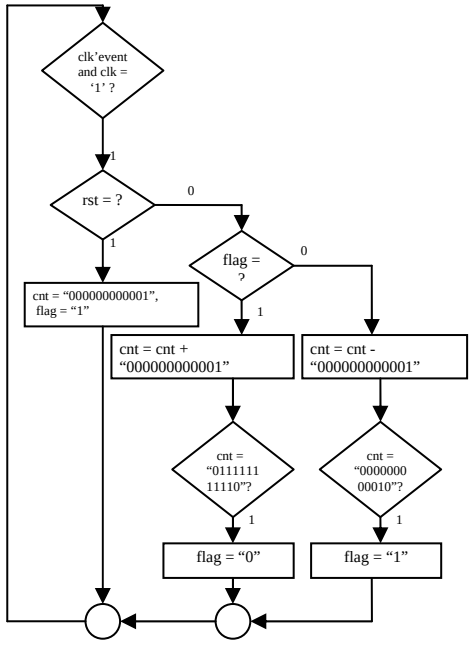

Fig. 10 Flowchart of Triangular Waveform

IV. RESULTS

All the algorithms, explained above are implemented in VHDL (VHSIC – Hardware Description Language). The simulation results of these netlists are shown below. In Fig. 13 we can see the entire system simulation.

We can see in the following simulation result, as per the received code words, actions are taken. For example, when the received codeword is "10010000" (rx_in) then the triangular waveform (waveform) is generated. By default, the waveform generated is sawtooth waveform. Also, by default, the baud generated is half the clock frequency.

V. FRONT END USING VISUAL BASIC

Visual Basic is an integrated Development Environment in which applications can be developed, run, tested and debugged. The project uses Visual Basic to develop the front end of project for the end-user. Visual Basic provides the following advantages:

1) Strong programming language
2) MSCOMM control object for programming and interfacing the serial port.

A. MSComm Properties

Visual-Basic uses the MSComm control to establish RS-232 communications between devices. This feature is only available for Visual Basic's Professional and Enterprise editions.

The MSComm control handles the communications by configuring ports, transferring data, use of handshaking signals and identifying the control. A brief reference on the control is provided in table I.

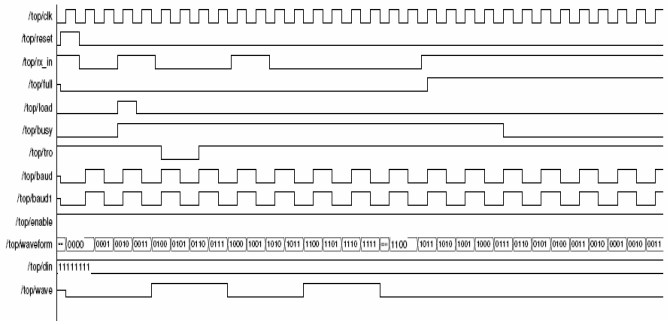

Fig. 11 Simulation Results of Arbitrary Waveform Generator

TABLE I. MSCOMM CONTROL PROPERTIES

MSComm Control Property	Description	Values Assigned	Interpretation of the value
CommPort	Sets or gets the number of the serial port	1	Commport no. 1 is in use.
Settings	Sets or gets the baud rate, parity, data bits and stop bits as a comma-separated string	Refer to Table II	
InputMode	Determines whether data will be read as a string or in binary form	0	Input mode is text
HandShaking	Sets or gets the handshaking protocol to use	0	None

The CommPort property of MSComm comes into play

when a user needs to select from a number of available serial ports for use in communications. The property takes integer number, representing the port number, as its input. Besides setting port for communication, it can also let the user know which port is in use. In order to use this property optimally, the user has to know how many ports are available in his computer and their corresponding numbers. Otherwise, the user can also write a VB code for detecting available ports in the PC [4].

The Settings property determines the Baud rate, the parity as well as the number of data bits and stop bits. These parameters use comma to separate between their respective values. Table II summarizes the possible values for these settings, where the items in bold are the default values [4].

From table II above, the values for parity parameter are as follows:

"E" stands for even parity, "M" for mark parity, "N" for no parity, "O" for odd parity and "S" for space parity. The 1.5 and 2 stop bits allow extra time for the receiver to completely receive the data, which may be useful for receiver with slow processing capability [4].

Table II. MSComm Settings property value options

MSComm Settings Parameter	**Possible Values**
Baud Rate (Parameter 1)	110, 300, 600, 1200, 2400, **9600**, 14400, 19200, 28800
Parity (Parameter 2)	E, M, **N**, O, S
Data bits (Parameter 3)	3, 4, 5, 6, 7, **8**
Stop bits (Parameter 4)	**1**, 1.5, 2

The steps to be followed for the implementation of MSComm are as follows:

1) Set the MSComm properties.
2) Open the port.
3) Do respective operation (read/write)
4) Close the port.

After setting all the parameter values as shown in table, we can use the MSComm object to communicate using serial port. Fig. 12 shows the snapshot of front end used for this experimental setup.

Front end provides facility to open and close the port.Also, baud rate can be varied using drop down menu. Different waveforms can be selected along with variable frequency square wave.

VI. CONCLUSION

This technical paper demonstrates that a net-list downloaded in a FPGA can be updated in run time. This saves the need of halting the execution and updating the net-list.

Also to develop a new system with new characteristics, use of FPGA is a great help as a system with unknown characteristics can be difficult and sometimes dangerous to handle with.

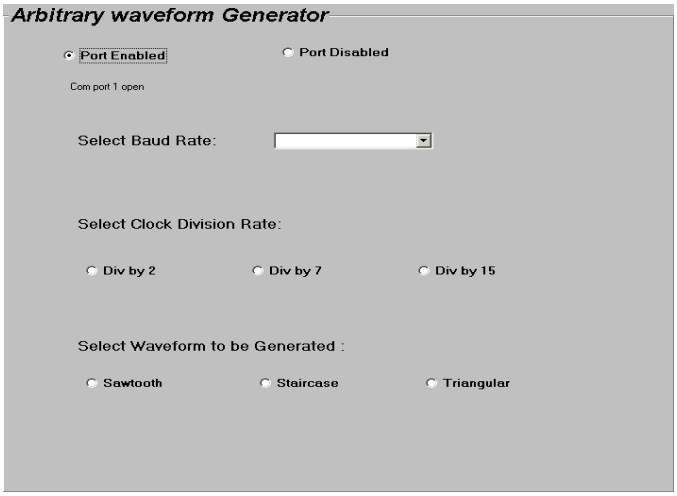

Fig. 12 Front End using Visual Basic

The idea of this project can be implemented to more real time applications like Power Electronics. Also, most of the embedded applications can be replaced by FPGA. This would be advantageous as FPGAs are reconfigurable.

VII. REFERENCES

[1] Jan Axelson, "Serial Port Complete Programming and circuits for RS-232 and RS-485 links and networks" Penram International Publishing.
[2] Douglas L. Perry, "VHDL Programming By Example- 4th edition", Tata McGraw-Hill Publications.
[3] Voleni A. Pedroni, "Circuit Design with VHDL", Eastern Economy Edition
[4] Office 97/Visual Basic: Programmer's Guide 2004. from http://msdn.microsoft.com/library/default.asp?url=/library/enus/office97/html/output/F1/D6/S5B1EB.asp
[5] Russo, Mark F. & Echols, Martin M. 1997, Automating Science and Engineering Laboratories with VISUAL BASIC, John Wiley & Sons, Inc., New York
[6] Charles H. Roth, Jr., "Digital Systems Design Using VHDL", PWS publishing Company.

VIII. BIOGRAPHY

Swati Mohite graduated from Datta Meghe College of Engineering, University of Mumbai in the year 2002 with a Bachelors in Electronic Engineering. She joined Fr. Cr. Rodrigues College of Engineering, Mumbai as a graduate student, pursuing Master of Engineering in Electronics Engineering in the year 2005. Her employment experience includes being a Lecturer in the Department of Information Technology at St. Francis Institute of Technology, and at the Department of Electronics Engineering at Datta Meghe College of Enginnering, Mumbai. Her special fields of interest include embedded systems and signal processing.