FPGA-BASED INTERFACING FOR 8-BIT AND 32-BIT ELECTRONIC DEVICES

Prateek Khurana[1], Rajat Arora[2], Monika Nagaria[3], Megha Sharma[4], Rajendra Bahadur Singh[5]

[1]School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh – 201308, India
[2]School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh – 201308, India
[3]School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh – 201308, India
[4]School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh – 201308, India
[5]School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh – 201308, India

Abstract

Most of the earlier electronic devices used buses with lesser number of bits but with the advancement in technology, devices having buses with larger number of bits are available in the market. Thus, there is an urging need to interface both these old and new technologies. In this paper, an interfacing unit has been proposed to interface the devices using buses with lower number of bits and higher number of bits.

In the present work, two types of circuits are used for interfacing these devices; those have been termed as UPSIZER and DOWNSIZER circuit. Upsizer concatenates the bits generated by a lower bit device and sends it to a higher bit device. On the other hand, Downsizer divides the bits generated by higher bit devices and send it to lower bit devices.

Keywords—*FPGA; Verilog; Xilinx; Interfacing Unit; Microprocessors; VLSI*

----------***----------

1. INTRODUCTION

A microprocessor is one of the central part of a modern personal computer or, in fact, any advance computer device. The microprocessor has undergone a tremendous amount of development more than any other component of the modern computer. It has experienced some great moments in the history since 1960s due to its significant importance in modern computing. Intel was a forerunner in early microprocessor technology, releasing its first 8-bit microprocessor, the 8008, in 1972. However, Intel's 8080 microprocessor was the major attraction of computer developers and engineers in the middle of that decade [1]. The development of 32-bit microprocessor began in late 1970s and they appeared on the mass market in the 1980s [2, 3]. 64-bit chips have been available since 1992 and are now in the mainstream of computer use [4, 5]. However, the advancement of technology with this pace requires a mechanism to support communication of new technology with that of the obsolete one. Thus there is a need of an interfacing unit which can form a link, so that technologies belonging to different generations can communicate easily.

In this paper, an electronic circuit has been proposed which acts as an interfacing unit between an 8-bit device and a 32-bit device. Since this interfacing circuit increases size of bits from input to output, it was termed as "Upsizer" circuit.

Similarly, another interfacing unit has been proposed in which a 32-bit device interacts with an 8-bit device. This interfacing unit downsizes the bit size from input to output and hence a term "Downsizer" can be coined for this circuit.

A thorough description of the architecture of proposed circuit is given in section II. Upsizer leads to the concatenation of bits/bytes of same id/group/address and then transfers in the forward direction. So Upsizer contains input buses of lower number of bits and output buses of higher number of bits. Upsizer can be used to concatenate the bits generated by a lower bit device and sends it to higher bit device. Downsizer divides the bits generated by higher bit devices and sends it to lower bit devices. So a downsizer contains input buses of higher number of bits and output buses of lower number of bits. The complete working of these circuits is explained in section III. ModelSim PE student edition10.3 software has been used for writing the Verilog code for the Upsizer circuit. The waveforms

obtained on ModelSim have been shown in section IV. Xilinx ISE tool has been used for the extensive synthesis of the Verilog code which models this circuit. The results of this synthesis are compiled in section V. The Verilog code was also dumped on Xilinx Spartan®-3 FPGA kit which is explained in section VI. This demonstrates the working of the circuit proposed in this paper. The block diagram of the Upsizer and Downsizer circuit are shown in Fig. 1 and Fig. 2 respectively.

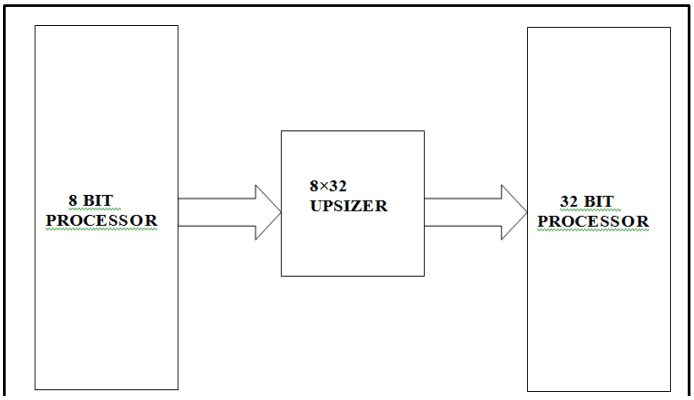

Fig. 1.Block Diagram of Upsizer as an Interfacing Unit

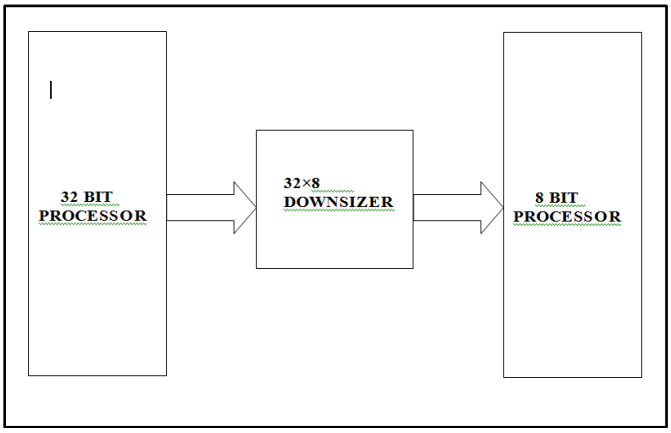

Fig 2 Block Diagram of Downsizer as an Interfacing Unit

2. ARCHITECTURAL DESCRIPTION

To interface the devices using buses containing lower number of bits and devices using buses containing higher number of bits an Upsizer circuit was proposed. In this circuit, concatenation of bits/bytes of same id/group/address was done and transferred forward. So this circuit contained input buses of lower number of bits and output buses of higher number of bits. When an Upsizer circuit was employed, the device having a lower fan-out acted as a master and drove the Upsizer circuit that further lead to concatenation of bits and it transferred higher number of bits to slave having a higher fan-in. Input signals like VALID, DATA [7:0], ID [3:0] etc. were given by the master to the Upsizer circuits. Output signals like VALIDM, IDM [3:0], DATAM [31:0] etc. were used to drive the slave by the Upsizer. The I/O pins of the Upsizer circuit are shown in Fig. 3.

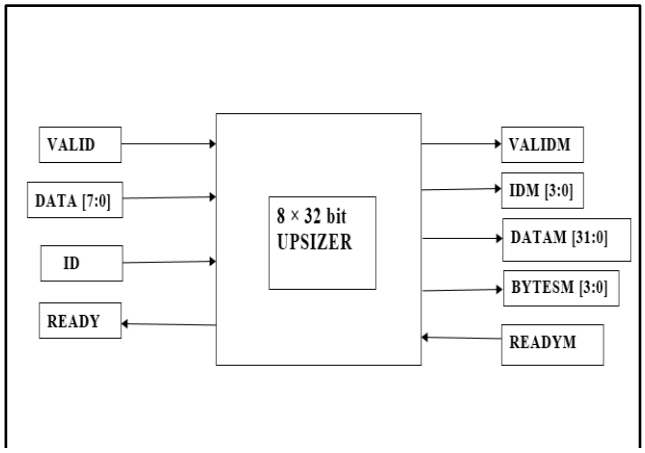

Fig 3 I/O pins of the Upsizer circuit.

In the above figure, except READYM signal, all other signals were given as input to the Upsizer by the master which in this case was a microprocessor or any other device having lower number bits. This input data was concatenated according to same id/group/address and transferred to the slave. VALID input pin indicated the validity of input data, if it was high the input data was considered valid. For its low value, the input was considered invalid/unacceptable. This signal was generated by the master. If it was required to perform the concatenation of data, then data was to be accepted by the Upsizer, which was only possible if VALID signal was high. The input data of 8 bits was sent by the master device along DATA [7:0] bus. It was this data that was concatenated and transferred forward to the slave device. ID [3:0] was used to differentiate the input 8 bit data. It acted like an address of the incoming data. It helped in concatenating the input data because data of same id were concatenated in 32 bits and transferred to slave. Master sent the ID along with the 8-bit data. READYM signal was used in communicating between slave and Upsizer. This signal was generated by the slave. Whenever slave was not ready to accept the data or was busy, this signal went low and it indicated the Upsizer to stop transferring the data and wait for some clock cycles. Data was not transferred to slave in that case even though if it was concatenated and stored by the Upsizer. Upsizer waited for the READYM signal to go high and then resumed the data transfer to slave.

In the Fig. 3, except READY signal, all other signals were given by the Upsizer as output to the slave which in this case was a microprocessor or a device having higher no of bits. The slave device received concatenated data according to same id/group/address and was transferred by the Upsizer as generated by the master. VALIDM signal was generated by the Upsizer only when the input 8-bit data having same id were concatenated and ready to transfer. This signal indicated the slave that input data was valid and allowed acceptance of data

by the slave. The output data of 32 bits was sent by the Upsizer to the slave device along OUTPUT [31:0] bus. 8-bit data from master of same id were concatenated in Upsizer and transferred to slave through this 32 bit bus. IDM [3:0] gave the id of the data which was getting transferred to the slave. BYTESM [4:0] signal was generated by the Upsizer. It was a 4-bit wide data. Each bit corresponded to the 8-bit wide in the 32-bit output data. READY signal was another output signal generated by the Upsizer. When this signal went low it indicated to master that the Upsizer was busy/not ready to accept the data. Meanwhile Upsizer generated some wait cycles and completed the execution of the previous transfers.

Downsizer circuit was same as that of the Upsizer. The input and the output signals of the Downsizer model had the same functions as that of the Upsizer model. The only difference was that Downsizer divided the bits generated by higher bit devices and sent it to lower bit devices. Therefore a Downsizer was used to convert large number of bits into small number of bits, so that the data used by higher bit devices may be used by the devices with lower bits. A Downsizer had input buses of higher number of bits and output buses of lower number of bits. When a Downsizer circuit was employed, the device having higher fan-out acted as a master and drove the downsizer circuit. This lead to division of bits and Downsizer transferred lower number of bits to slave having lower fan-in.

Input signals like VALID, DATA [31:0], ID [3:0] etc. were given by the master to the Downsizer circuit. Output signals like VALIDM, IDM [3:0], DATAM [7:0] etc. were used to drive the slave device by the Downsizer circuit. The I/O pins of the Downsizer circuit are shown in Fig. 4.

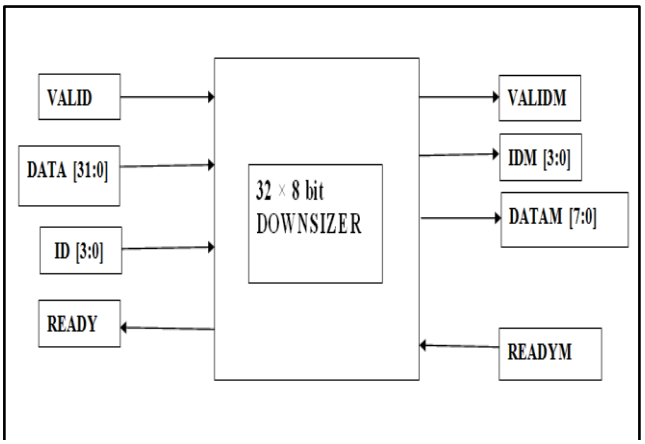

Fig 4 I/O pins of the Downsizer circuit.

3. WORKING OF THE INTERFACING CIRCUITS

Upsizer handled the routing of data only and did not modify the transfer type or response information. Input data of same id were grouped together, so maximum 4 bytes data of same id were concatenated if they arrived consecutively. If there was any change in id after 1/2/3 bytes of data, then rest of the upper bytes were zero padded and a total 4 bytes were forwarded. Thus ID [3:0] played a major role in determining the address of the incoming data and created groups of 32-bit data accordingly, which were then transferred to the slave device. The other input signals, including the 8-bit input data bus DATA [7:0] were generated by the master device for the Upsizer circuit.

If input data, generated by the master device for the first four clocks is 11001100,00001111,11110000,10101010 and ID remained same as 1001, then all these four bytes of data were concatenated by the Upsizer circuit. The 32-bit output data transferred to the slave device in 5th clock cycle in this case was 10101010111100000000111111001100. This example has been shown in Fig. 5.

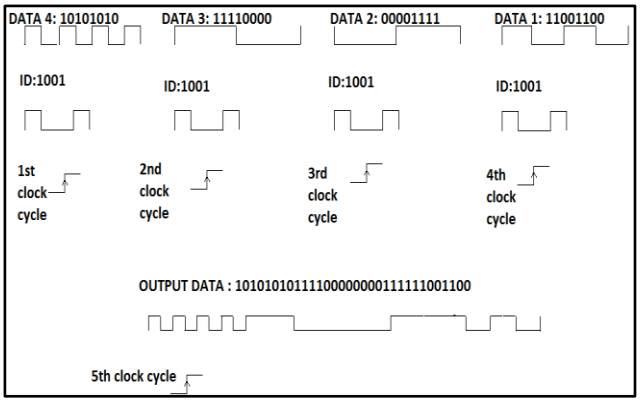

Fig 5 Working of Upsize circuit without zero padding condition

Now for another instance, if input data, generated by the master device for the first two clocks is10101010, 11110000 and ID is 1001, and for the next clock cycle ID got changed to 1010. Then zero padding would be done and the two input data would be sent by the Upsizer to the slave device as 32-bit output data: 00000000000000000111100001010101O. This example has been shown in Fig. 6.

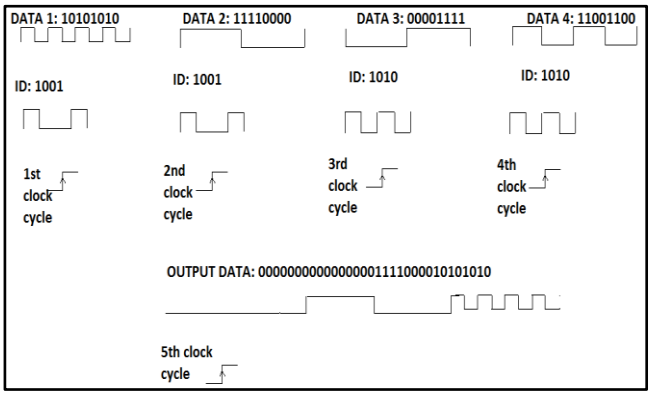

Fig.6 Working of Upsizer circuit with zero padding condition

The Downsizer worked in the similar fashion as Upsizer circuit. The Downsizer divided the 32-bit data into four different bytes of data and transferred it to the slave device. For the Downsizer circuit, the master device had a higher fan-out i.e. 32-bit data. The slave device had a lower fan-in i.e. 8-bit data. In this no zero padding was required, because whatever the 32-bit data was coming it got divided into 4 bytes and these 4 bytes were transferred to the slave device.

For instance, if the input 32-bit data generated by the master device was 11001100000011111111000010101010, then this input data got divided into 4 bytes as 10101010, 11110000, 00001111, 11001100 with lower byte in the starting and higher byte in the last of the data transfer. This example has been shown in Fig. 7.

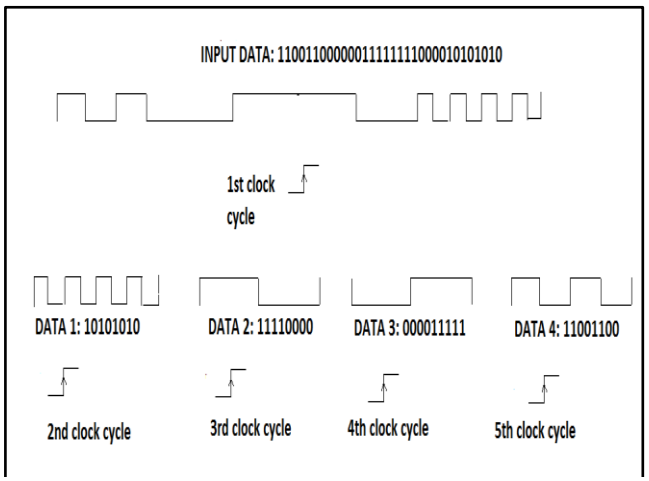

Fig 7 Working of Downsizer circuit

4. MODELSIM ALTERA SIMULATION

The Upsizer circuit whose logical description is given in the above section has been realized with a hardware description language (HDL) i.e. Verilog. Verilog language was invented by a company called Gateway Design Automation in the early 1980s. Most of the modern electronic systems can be modeled using this language at register transfer level of abstraction [6, 7 and 8]. The simulation software that has been used to simulate the Verilog code of the circuit is MODELSIM PE 10.3. This tool is basically a simulation and verification tool for Verilog, VHDL, System C and mixed level designs. This simulator is best suited for both ASIC and FPGA designs as it has an excellent platform support that makes it easy to adopt in majority of design and project flows. It provides high capacity, performance and debugging capabilities that are essential to simulate large blocks of code. The graphical user interface (GUI) provided is also intelligently engineered that makes the graphical interpretation of the design flexible and easily understandable [9].

To design a circuit with this software, a certain step by step process is followed otherwise it becomes quite a tedious task. Firstly, a working design library needs to be created, and then all the design files required for the simulation are compiled. The next step is to load all the design files and then multiple simulations are performed. The I/O waveforms are studied on the waveform viewer, which is an essential part of this software. The waveforms are also useful in studying and debugging the results. The basic simulation flow of any HDL code on this software is shown in Fig 8.

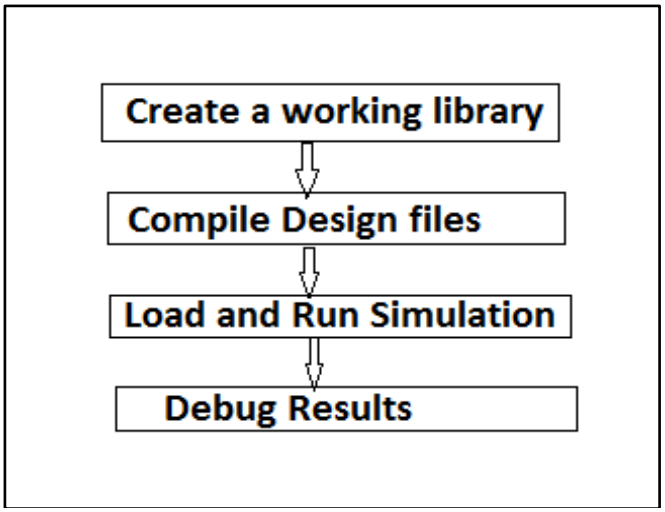

Fig 8 Basic Simulation Flow

A Verilog code was written for the Upsizer circuit according to the basic simulation flow shown in Fig. 8. This Verilog code was successfully compiled with no errors. A test bench was written for this Verilog code that helped to force all possible inputs at regular time intervals. Both the Verilog code for the Upsizer circuit and its test bench were simulated and I/O waveforms were studied in the waveform viewer. These waveforms when extensively studied, verified the working of the Upsizer circuit according to the logic discussed in the previous section.

Consider for instance, if the input data was 00001111 and ID was 0001 for four clock cycles, then the expected output data is 00001111000011110000111100001111. In this case, the input data from the master device was concatenated since ID did not change its value for four clock cycles. Thus Upsizer performed its functions as it was desired. The simulation result for this case has been shown in Fig. 9.

In another example, the input data was 10101010 for four clock cycles and ID remained as 0000. As it was expected, the Upsizer circuit concatenated the input data from the master four times and the 32-bit output data sent to the slave device was 10101010101010101010101010101010. This has been shown in Fig. 10.

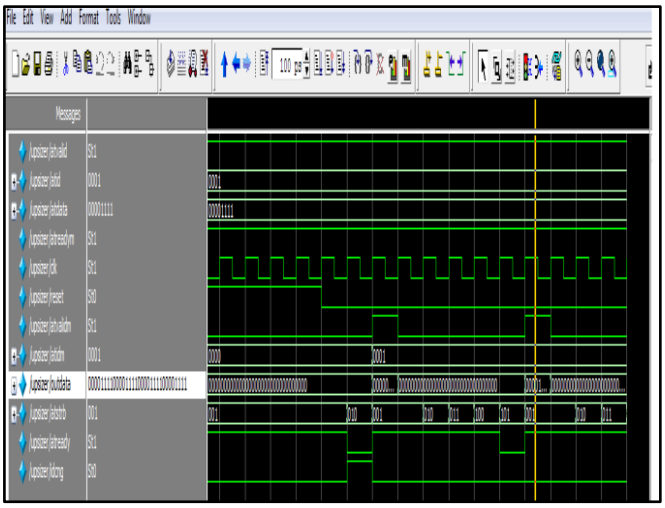

Fig 9 Simulation result with input data =00001111 and id = 0001 for four clocks and output = 00001111000011110000111100001111

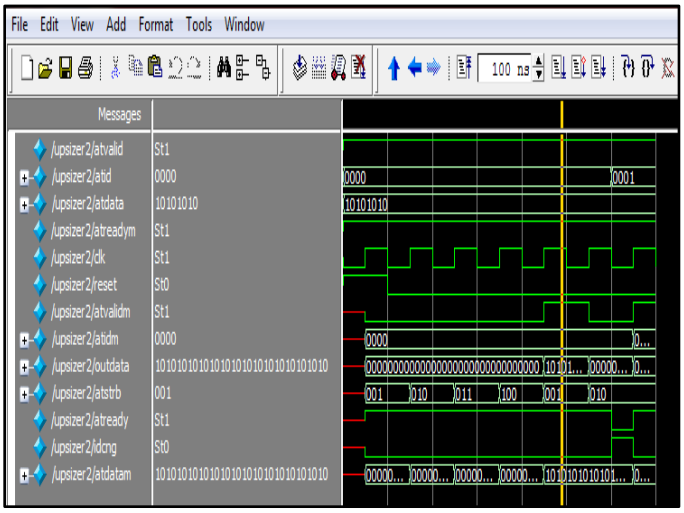

Fig 10 Simulation result with input data =10101010 and id = 0000 for four clocks and output = 10101010101010101010101010101010

Now for the case when ID was 0001 for the first clock cycle, and it was changed for the next clock cycle with input data remaining constant as 10101010, zero padding was done. This verifies the logic of Upsizer circuit. Since ID changed its value just after the first clock cycle, the higher 24 bits of 32-bit data were zero padded. The 32-bit output data obtained in this case was 00000000000000000000000010101010. This simulation result has been shown in Fig. 11.

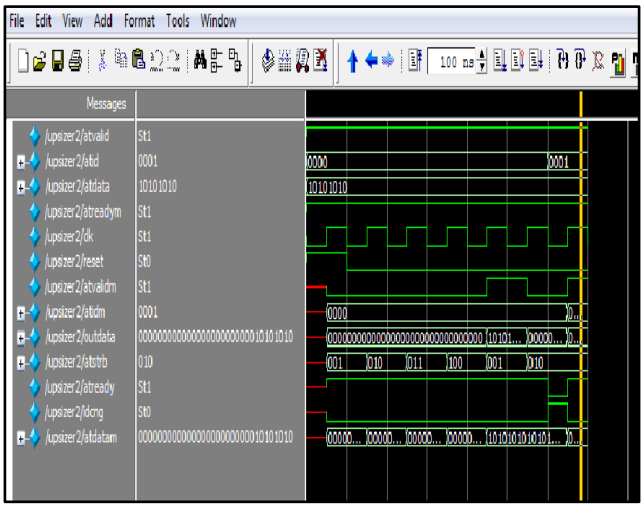

Fig 11 Simulation result with input data = 10101010 and id = 0001 for first clock and id got changed,so the output = 00000000000000000000000010101010.

In another case, input 8-bit data was 10101010 and ID was 0010 for first clock cycle. For the second clock cycle, the input data was 11110000 and ID was still 0010. The ID got changed after second clock cycle. Considering the function of Upsizer circuit, the higher 16 bits of the 32-bit output data were zero padded. However, lower 16 bits consisted of the concatenated input data. The simulation results has been shown in Fig. 12.

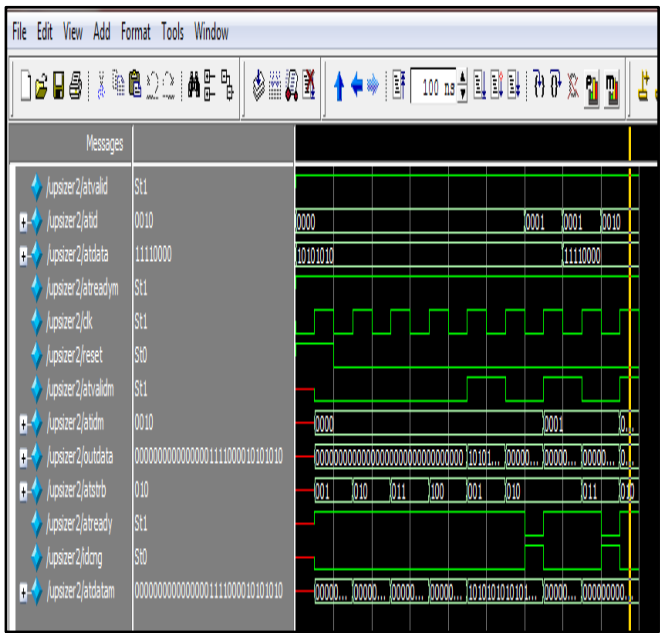

Fig 12 Simulation result with input data = 10101010 and id =0010 for first clock and input data = 11110000 and id = 0010 for second clock, then id got changed, therefore output = 00000000000000001111000010101010

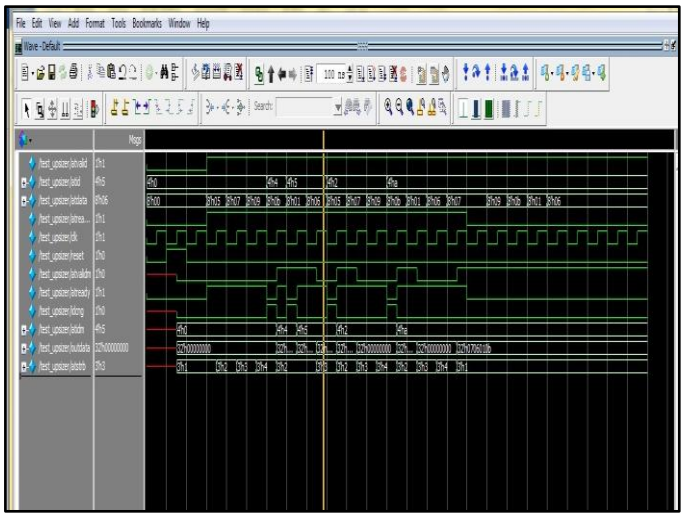

Fig 13 A Verilog Testbench for the Upsizer Circuit Model

In all of the above cases, the input had to be forced manually to achieve the output waveforms. Therefore it was not possible to simulate all possible input combinations. In Verilog, it is possible to create a Testbench for a module. Testbench is just like a dummy template which basically declares inputs to the circuit as registers and outputs from circuit as wires, then instantiates the circuit for all these specified inputs. This testbench can be used to automatically force all the input combinations. Testbench techniques and virtual tester development are heavily utilized in the presentation of circuit simulations. To show the simulation of the Upsizer circuit for all input combinations, a Testbench was written in Verilog and was simulated on the ModelSim software. All the possible input data combinations as well as IDs were considered. The output waveforms so produced were studied. All possible output waveforms were achieved. The simulation of the Testbench has been shown in Fig. 13.

5. XILINX SYNTHESIS

Synthesis is the most important step in implementing a Verilog design in the real world environment. Simulation only relies on language features that are not synthesizable, but in this step, the Verilog design is brought near to the actual hardware. It basically transforms high level Verilog/ VHDL modules, which don't have any real physical hardware, which can be wired up to perform the desired logic, into low level logical constructs. These construct are modeled in the form of look up tables or ASIC and FPGA hardware components. Xilinx ISE (Integrated Software Tool) is the tool that has been used to perform the synthesis of the Upsizer circuit. Xilinx is the inventor of this tool. This tool mainly focuses on the synthesis and analysis of HDL designs, giving an exact idea about the utilization of various components of the hardware used in the design [10]. It has the capability to perform timing analysis, examine the register transfer level (RTL) diagrams, check syntax errors, simulate the design for different set of input conditions and thus configure the target device.

The Verilog code for the Upsizer circuit was also written in Xilinx ISE tool. The test bench of this Verilog code was also formed. The SPARTAN 3E version of the field programmable gate array (FPGA) family was selected for synthesis of this Verilog code. It was found that the Verilog code was successfully synthesized with no errors or warnings. The device utilization summary showing the logic utilization was generated as shown in the Fig. 14.

upsizer Project Status (04/13/2014 - 17:27:03)			
Project File:	upsizer_2.xise	**Parser Errors:**	No Errors
Module Name:	upsizer	**Implementation State:**	Programming File Generated
Target Device:	xa3s200a-4ftg256	**• Errors:**	No Errors
Product Version:	ISE 14.4	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	41	3,584	1%	
Number of 4 input LUTs	87	3,584	2%	
Number of occupied Slices	53	1,792	2%	
Number of Slices containing only related logic	53	53	100%	
Number of Slices containing unrelated logic	0	53	0%	
Total Number of 4 input LUTs	87	3,584	2%	
Number of bonded IOBs	90	195	46%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.47			

Fig 14 Design Summary generated by the Xilinx ISE tool

As seen from the Fig. 14, the numbers of slice flip flops used were 41 out of the total available 3541. The number of 4 input LUTs (look up tables) used were 47 out of 3584 total LUTs available. The logic distribution in the detailed map report showed that the number of slices containing only related logic were fully utilized i.e. 53 out of 53. Related logic is defined as logic that shares connectivity - e.g. two LUTs are "related" if they share common inputs. When assembling slices, Map gives priority to combine logic that is related.

Doing so results in the best timing performance. The numbers of slices containing unrelated logic were 0 out of 53. Unrelated logic shares no connectivity. Map will only begin packingunrelated logic into a slice once 99% of the slices are occupied throughrelated logic packing.The percentage utilization of the number of IOBs used was 46% i.e. 90 out of the total 195 available IOBs were used.

The timing detail of the synthesis report is shown in Fig. 15. The synthesis report gives the actual value of delay which includes both gate delay and net delay of particular LUTs and the total delay. The total memory usage for the synthesis was 189080 kilobytes. The total REAL time to Xst (Xilinx Synthesis Technology) completion was 6.00 seconds and the total CPU time to Xst completion was 6.97 seconds.

```
Timing constraint: Default path analysis
  Total number of paths / destination ports: 9 / 2
--------------------------------------------------------------------------------
Delay:               10.012ns (Levels of Logic = 5)
  Source:            atid<3> (PAD)
  Destination:       idcng (PAD)

  Data Path: atid<3> to idcng
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             3   0.849   0.674  atid_3_IBUF (atid_3_IBUF)
     LUT4:I0->O            1   0.648   0.452  idcng_and000054 (idcng_and000054)
     LUT4:I2->O            4   0.648   0.590  idcng_and000055 (idcng_and000055)
     LUT4:I3->O           13   0.648   0.983  idcng_and000083 (idcng_OBUF)
     OBUF:I->O                 4.520          idcng_OBUF (idcng)
    ----------------------------------------
    Total                     10.012ns (7.313ns logic, 2.699ns route)
                                       (73.0% logic, 27.0% route)

==========================================================================


Total REAL time to Xst completion: 6.00 secs
Total CPU time to Xst completion: 6.97 secs

-->

Total memory usage is 189080 kilobytes
```

Fig 15 Timing Details of the Synthesis Report

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : upsizer.ngr
Top Level Output File Name         : upsizer
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : No

Design Statistics
# IOs                              : 90

Cell Usage :
# BELS                             : 88
#      LUT2                        : 52
#      LUT2_L                      : 2
#      LUT3                        : 5
#      LUT3_L                      : 1
#      LUT4                        : 23
#      LUT4_D                      : 4
#      VCC                         : 1
# FlipFlops/Latches                : 41
#      FDRE                        : 40
#      FDRSE                       : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 89
#      IBUF                        : 15
#      OBUF                        : 74
```

Fig.17 Final Report of the HDL synthesis

The RTL synthesis of the Verilog code was also carried out. The schematic diagram of RTL synthesis was generated, that is shown in Fig. 16. The input and output pins and the wires declared in the Verilog Model were clearly shown in the schematic diagram.

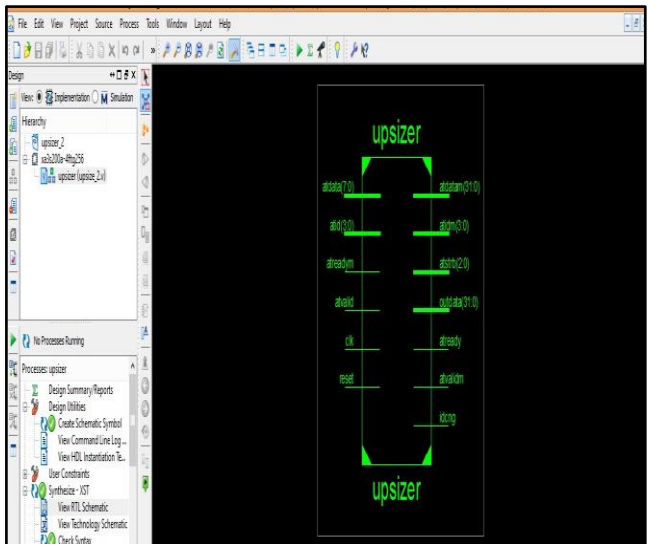

Fig 16 Schematic Diagram of RTL Synthesis

The final report of the HDL synthesis was generated which showed the design statistics and cell usage. The cell usage depicted that the number of flip flops used were 41 and there were 89 IO buffers. Out of these IO buffers, 15 were input buffers and 74 were output buffers. The cell usage showed that number of BELS utilized were 88 and only 1 clock buffer was used. This detailed final report has been shown in Fig. 17.

6. FPGA IMPLEMENTATION

After the synthesis of the Verilog code on Xilinx ISE tool, the next step was FPGA (Field Programmable Gate Array) implementation. A large number of circuits have been implemented on FPGA for verifying the Verilog/VHDL design proposed for these circuits. [11, 12 and 13]. The code synthesized on Xilinx ISE tool was verified using a SPARTAN 3E FPGA kit manufactured by Texas Instruments. The FPGA kit which was used has been shown in Fig. 18. The Verilog code for the interfacing circuit was dumped on this FPGA kit.

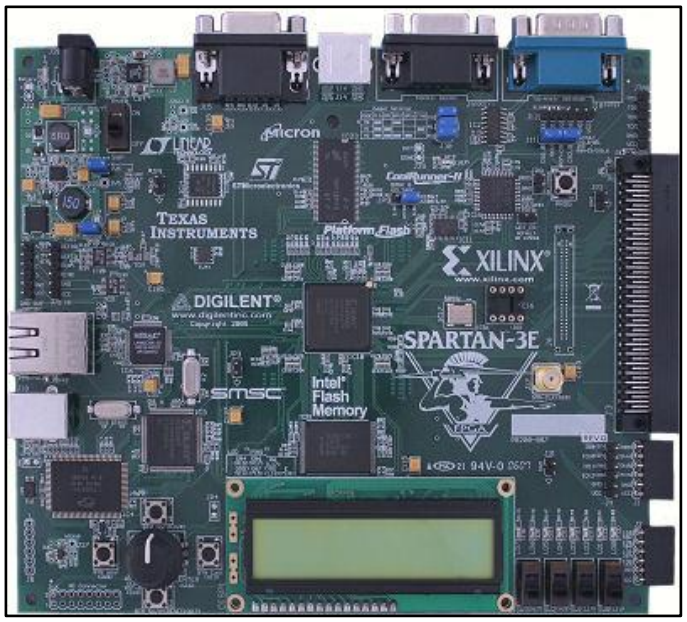

Fig 18 SPARTAN-3E FPGA kit used for Implementation

This input switches and output LEDs were linked with the input and output variables of the code respectively to verify the Verilog Design. Thus the programming file that was generated was dumped on the FPGA kit successfully to verify the working of Interfacing Circuit. For the working of the Upsizer Circuit the 8-bit input data was given using the input switches on the FPGA board. The output LEDs verified the result that explained the working of Interfacing Circuit.

7. CONCLUSIONS

In this paper, a Verilog code for the interfacing circuit that has been proposed, was written and simulated on ModelSim. The results obtained from ModelSim simulation waveforms were verified. A comprehensive synthesis of the Verilog model was performed on the Xilinx ISE tool to get an idea of the requirement of the resources on the FPGA kit. Finally the Verilog code was dumped on the SPARTAN-3 FPGA kit which further proved the proper functioning of the interfacing circuit.
The output has been checked and verified under various test cases and hence this UPSIZER circuit can act as an interfacing unit between an 8 bit and a 32 bit interface. The proposed interfacing circuit may be used in burst transfers in which bulk of data are transferred from master to slave having different bit processors.

ACKNOWLEDGMENTS

The authors wish to thank Mr. Sharad and Mr. Rajendra Bahadur Singh for their immense support and help whenever required and for providing access to laboratory for testing purposes.

REFERENCES

[1] P.J. Horne, "Implementation Of An Intel 8080 Microprocessor Developments System Using Existing Minicomputers," IEEE transactions on nuclear science, Vol.NS-24, No.3, June 1977

[2] R. Sherbume, M. Katevenis, D. Patterson, and C. Sequin, "A 32-bit microprocessor with a large register file," IEEE J. Solid-State Circuits, Vol. SC-19, no. 5, Oct. 1984.

[3] Joan M. Pendleton, Shing I. Kong, Emil W. Brown, Frank Dunlap, Christopher Marino, David M. Ungar, David A. Patterson, and David A. Hodges, " Design A 32-bit Microprocessor for Smalltalk," IEEE Journal Of Solid-State Circuits,Vol. SC-21, No. 5, October 1986.

[4] Hunt, Doug. "Advanced performance features of the 64-bit PA-8000." InCompcon'95.'Technologies for the Information Superhighway', Digest of Papers. pp. 123-128. IEEE, 1995.

[5] Niveditha Domse, Kris Kumar, and K. N. Balasubramanya Murthy, "64 bit Computer Architectures for Space Applications – A study,"World Academy of Science, Engineering and Technoloy,Vol:3 2009-03-28

[6] Girish Kumar B, Prabhu V, Siva Prasad T, Ruban Thomas " Design and Implementation of FPGA Based Efficient Data Transmission Using Verilog," International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 11,ISSN 2250-2459**,** November 2013

[7] Ulrich Golze, Peter Blinzer, Elma rCochlovius, Michael Schafers,Klaus-Peter Wachsmann, "VLSI Chip Design with the Hardware Description Language VERILOG: An Introduction Based on a Large RISC Processor Design,"VLSI Chip Design with the Hardware Description Language VERILOG: An Introduction Based on a Large RISC Processor Design 1st, Springer-Verlag New York, Inc. Secaucus, NJ, USA 1996, ISBN 3540600329.

[8] Zaher S. Andraus, Karem A. Sakallah, "VLSI Automatic abstraction and verification of verilog models,"DAC '04 Proceedings of the 41st annual Design Automation Conference, Pages 218-223, 2004-06-07,ISBN: 1-58113-828-8

[9] P.V.Sasanka, Y.V.RamanaRao, A.L.Siridhara, "A Verilog Model of Universal Scalable Binary Sequence Detector," International Journal of Scientific and Research Publications, Volume 3, Issue 4, ISSN 2250-3153, April 2011.

[10] Kaushik Chandra Deva Sarma, Amlan Deep Borah, Lalan Kumar Mishra, "Design and Synthesis of 32 BIT ALU Using Xilinx ISE V9.1i," International Journal of Engineering Research & Technology (IJERT), Vol. 2 Issue 5, ISSN: 2278-0181, May – 2013.

[11] Mangesh V. Benodkar and Prasad K. Bhasme, "A Review Paper on Design and Simulation of Universal Asynchronous Receiver Transmitter on Field Programmable Gate Array Using VHDL,"International Journal of Advance Research in Computer Science and Management Studies, Volume 2, Issue 1, January 2014.

[12] Amit kumar Singh, S.K. Dubey,M.G. Bhatia," Design and Simulation of FPGA based Digital System for Peak Detection and Counting, "International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11,ISSN: 2277 128X November 2013

[13] Li Shang, Alireza S. Kaviani, Kusuma Bathala, "Dynamic power consumption in Virtex™-II FPGA family," FPGA '02 Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays, Pages 157-164, 2002-02-24.

[14] Ciletti, Michael D. Advanced digital design with the Verilog HDL. Vol. 2. Prentice Hall, 2003**.**

[15] Palnitkar, Samir. Verilog HDL: a guide to digital design and synthesis. Vol. 1. Prentice Hall Professional, 2003.