An Alternative Microprocessor Bus Structure Design on FPGA

Esra Abdalraof Tellisi E.Tellisi@uot.edu.ly

Jumanah Abdulhadi Mansur J.Mansur@uot.edu.ly

Mohamed Muftah Eljhani M.Eljhani@uot.edu.ly

Department of Computer Engineering, College of Engineering, University of Tripoli

الملخص

يعد تنفيذ ناقل ثلاثي (tri-state-based bus) الحالات مفيدًا لأي تطبيقات رقمية تصميماتها كبيرة مع عدد كبير من هياكل التصميم، ولكن في نفس الوقت، يمكن أن يعقّد عملية التسلسل الزمني والاختبار. لا تحتوي رقائق مصفوفة البوابة القابلة للبرمجة (FPGA) على برامج توصيل ثلاثية كافية لتركيب تصاميم رقمية كبيرة. بدلاً من ذلك، يمكن للمصممين استخدام متعدد الإختيار (multiplexer-based buses) في تصميم الأنظمة الرقمية الكبيرة. في هذه الورقة البحثية، تم تصميم وإختبارالوحدات الأساسية لنظام ناقل المعالجات الدقيقة المقترح ومحاكاتها باستخدام لغة وصف أجهزة الكيان المادي Verilog (HDL)، ونقلها إلى الشريحة الإلكترونية Cyclone IV GX FPGA. أثبتت المعالجات الدقيقة التي تحتوي على ناقل ثلاثي القوائم مقارنة بنظام ناقل مع ناقل متعدد الإرسال أنها تستهلك قدرًا أكبر من الطاقة ولديها مرونة أقل في عملية الاختبار والسرعة. حيث يمكن استخدام بنية الناقل متعدد الإرسال المقترحة في هذا البحث للأنظمة المدمجة والأجهزة الإلكترونية المحمولة التي تتطلب سرعة عالية واستهلاكًا منخفضًا للطاقة. في النظام الخاص بتصميم الرقاقة القابلة للبرمجة (SoPC) ، فإن برامج الملكية الفكرية (IP) لها ناقلات محدودة قائمة على ثلاثي القوائم، لذلك يمكن لتطبيقات التصميمات الكبيرة استخدام ناقلات قائمة على متعدد الإرسال. تستخدم تصميمات الدوائر المتكاملة الخاصة بالتطبيق (ASIC) ناقلًا داخليًا قائمًا على متعدد الإرسال لنفس السبب. بالإضافة إلى ذلك ، فإن الناقل الثلاثي لديه العديد من المشاكل أهمها التوقيت واستهلاك الطاقة.

ABSTRACT

A tri-state-based bus implementation is useful for any large design application with a large number of design blocks, but at the same time, it can complicate synchronization and testing. Field programmable gate array (FPGA) chips do not have enough tristate drivers to mount large buses. Alternatively, designers can use bus structures based on multiplexers. In this research paper, the basic modules of the proposed microprocessor bus system are designed, implemented, and simulated using Verilog hardware description language (HDL), and implemented and routed to the FPGA. Microprocessors with a tristate-based bus compared to a bus system with a multiplexer bus have proven to consume more power, and have less timing and test process flexibility. The proposed multiplexed bus architecture can be used for embedded systems and mobile electronic devices that require high speed and low power consumption. In the intellectual property (IP) integration has limited tristate-based buses, so large design applications can use multiplexer-based buses. Application-specific integrated circuit designs use an internal multiplexer-based bus for the same reason. Also, a tristate-based bus has timing and power consumption issues due to the capacitive load of the nodes.

Keywords- FPGA Design; Verilog HDL; Microprocessor; Multiplexed Bus; Tri-state Bus

INTRODUCTION

Early Intel's and other processors were designed by hand, laying out the layers of an integrated circuit (IC) substrate masks using regular drafting techniques. There were little or no electronic design automation (EDA) tools to help the chip developer. This method was so boring that least few people had the patience and skills for such a task. Thankfully, times have changed, and designing custom processors is within reach of many designers of such hobby. There are two predominant HDL languages, Verilog and VHDL. Verilog HDL is adopted in this research paper. [1], [2]. Buses, although the simplest form of interconnect, is a poor choice from a density or power standpoint because the power and space required to drive them at maximum speed grow exponentially with the capacitance of the bus [3]. Early computer buses were literally parallel electrical wires with multiple connections, but modern computer buses can use both parallel and bit serial connections. Buses can also connect two different components at the same time through the usage of the point-to-point or multipoint technique. SoPC bus architectures have a significant effect on system speed and power dissipation. System designers, as well as the research community, have focused on the issue of exploring, evaluating, and designing personal computer (PC) communication architectures to meet the targeted design goals [4]. The replacements to buses are many, and all have been used successfully in various computers, chips, boards, and FPGAs. These replacements are no panacea, just as buses aren’t a cure-all for every interconnection illness. Avoiding the fixed routing and timetable of a standard bus can open up new avenues for design, and restore a bit of glamour and creativity to an otherwise mundane project [5]. The EDA design flow typically follows a path from Verilog/VHDL hardware description language [6], or schematic design entry through synthesis and place and route tools to the programming of the FPGA. The Proposed microprocessor based on a multiplexer bus system is designed, simulated, and compared against the tri-state system bus using the Verilog HDL, and implemented on the Altera Cyclone IV GX FPGA development board [7], [8]. System on-programmable-chip debug has been a apprehension from the beginning of computer era. FPGA has also taken part in this field. For example, work by Jamal et.al [9, 10] proposes better functional changes during on-chip system debug, employing FPGA edge architecture. Present-day works in this field, particularly system debuging, can be found in [11–14]. A number of authors extend the idea to other areas such as machine learning [15,16].

METHODOLOGY

In the design, two methods were used to implement microprocessor system using two different buses. First-way using tri-state bus. The top-level module for tri-state bus and four lower-level modules were used to implement the design, the first module of the lower level for 8_bit register, the second module for 8_bit tri-state bus, the third module for arithmetic logic unit (ALU) and the fourth module for MUX 2 to 1 as shown in figure 1. The second-way using multiplexed-bus. The top-level module for multiplexed-bus and four lower-level modules were used to implement the design, the first module of lower level for 8_bit register, the second module for 8_bit MUX 4 to 1, the third module for ALU and the fourth module for MUX 2 to 1 as shown in figure 2. Each system contains four register that has three inputs clk, ena and x, and one output q. ALU module has two inputs and select lines to control the operations such as, (addition, subtraction, AND, and shift left), and has one output. As shown in table 1. we have 4 to 1 multiplexer that has 4 inputs and two select line that implemented to control the output data, and 2 to 1 multiplexer with two inputs, one select line that implemented to control the output data. The top-level module contains six inputs select, op, move, write, data, enable, clock and has six outputs R0, R1, R2, R3, Cout and out. Registers are connected with tri state, and 4 to 1 multiplexer, then data is loaded into registers, using move and write input signals we can specify the registers that used to enter data, then the data moved from one register to other register. Registers are associated with two 2to1 multiplexers and connected to ALU. The tristate-based bus and multiplexer design is containing of two main modules data path module and control unit module. The data path module is consisting of five sub-modules. The system contains four 8-bit registers, register 0 to register 3, figure 1and 2 displays how these registers are connected using tri-state drivers and multiplexer to implement the bus structure. The data outputs q of each register is connected to tri-state drivers and multiplexer. When selected by their enable signals, the driver places the contents of the consistent register onto the bus wires. If the enable input is set to 1, then the contents of the register will be changed on the next positive edge of the clock. The enable input on each register is registered ena, which positions for enable. The signal that controls the ena input for registers is designated as [3:0] Write, while the signal that controls the associated tri-state driver and multiplexer is called [3:0] Move. These signals are created by the control unit module. In addition to four registers, there is other module block that linked to the bus. The circuit diagram, figures 1, 2 shows how 8-bits of data from an external source that is located on the same bus, using the control input signal that is created by control unit module called Enable. It is important to ensure that only one circuit block tries to place data onto the bus wires at any assumed time. The control unit must guarantee that only one of the tri-state drive enable signals, register 0, register 1, register 2, register 3. are declared at a given time. The control unit also produces the signals [3:0] Write, which determine when data is loaded into each register. In general, the control unit perform a number of functions, such as loading resisters with data and transferring the data stored in one register into another register. The control circuit is synchronized by a clock input, which is the equal clock signal that controls four registers.

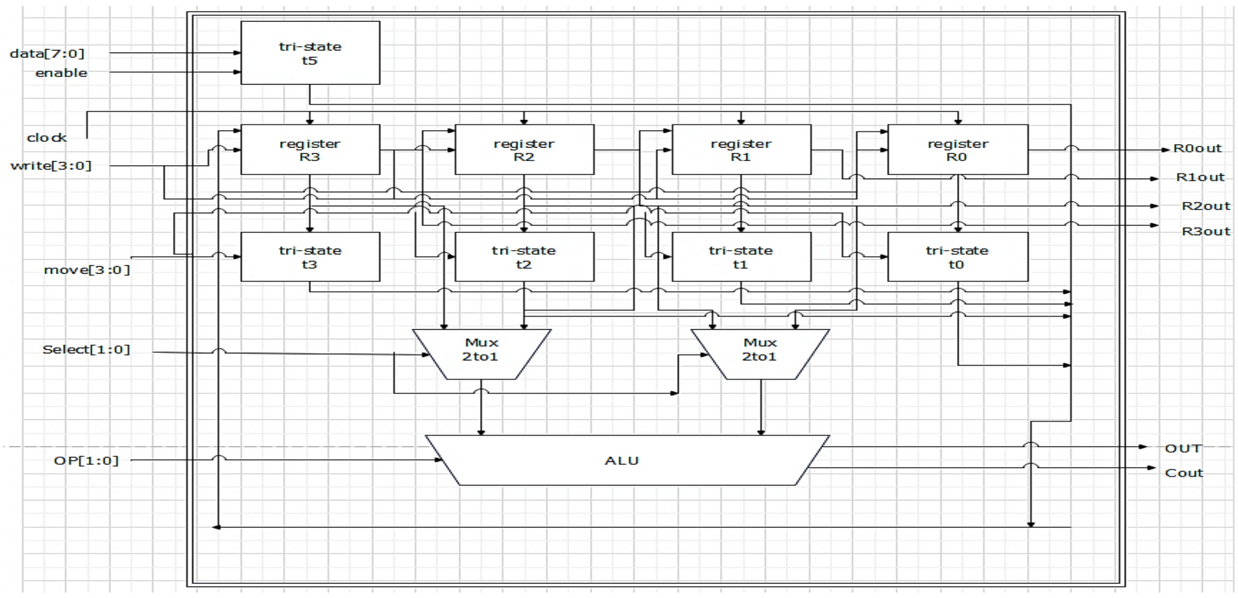


Figure 1. Tri-state top-level schematic.

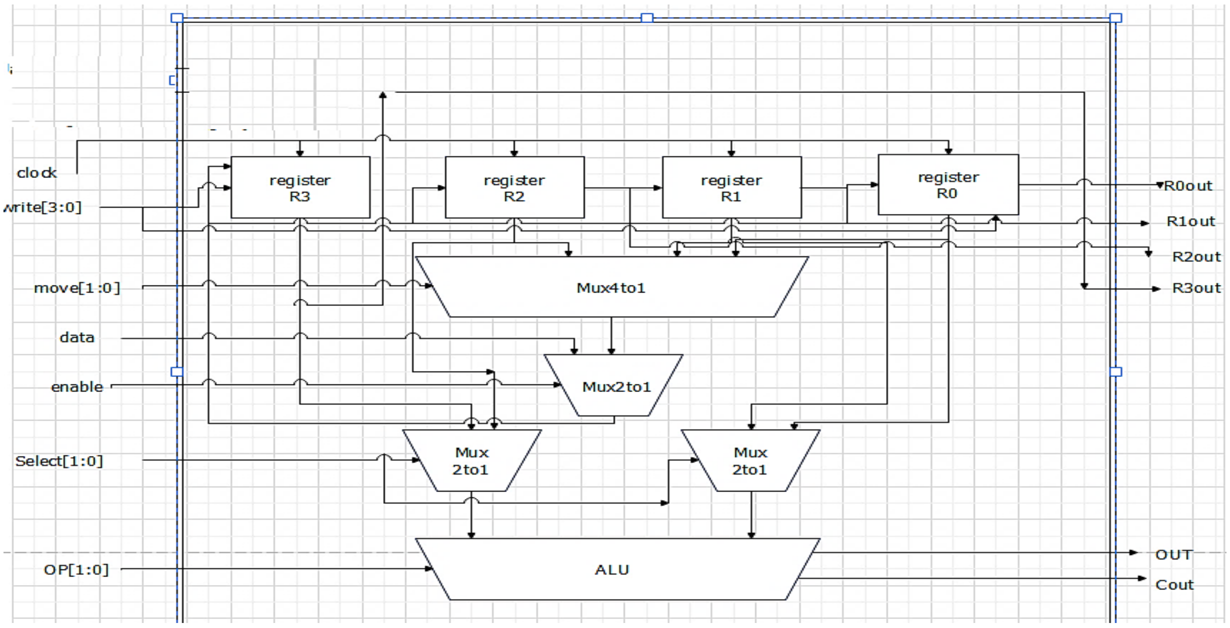


Figure 2. Multiplexed top-level schematic.

SIMULATION AND RESULTS

These results illustrate the work of ALU using four operations and output results shown in figure 3.

TABLE 1. True table of ALU

sel	Instruction	Operation
00	**ADD**	**Out = A+B (Cout is carry)**
01	**SUB**	**Out =A-B**
10	**SHL**	**A<<1**
11	**AND**	**A & B**

*Out = output & ADD = addition & SUB = Subtraction & SHL=shift left.

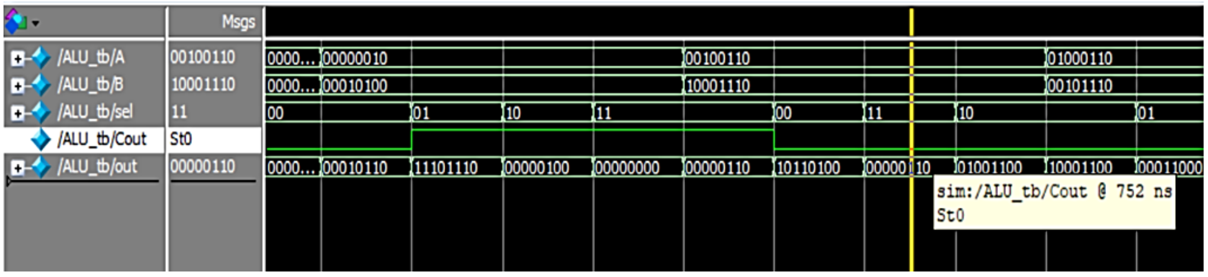


Figure 3. ALU operation.

After designing and simulating the multiplexer bus submodules individually, the system was instantiated, simulated, and validated as a top-level module. The system was then compared to a tristate bus system to evaluate the capabilities, for speed, and power consumption of the proposed multiplexer bus system and the tristate bus system specially designed and implemented for this purpose. In the testbench , the module reads four values of "data" respectively 55, 77, 99, 00, it is stored in registers and transferred via bus to different register using "move". In figure 4 (data is loadded to the registers, and in figure 5 (date is shifted right between registers.

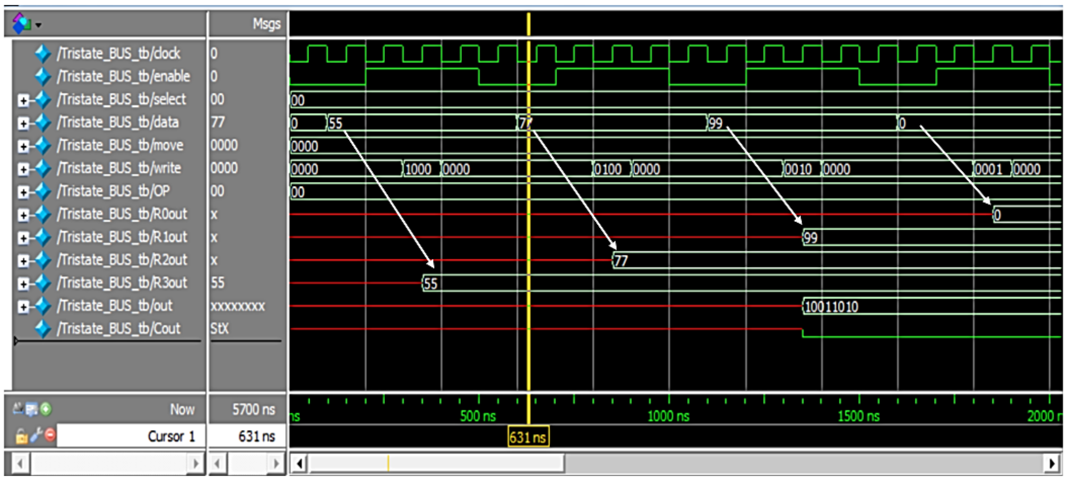


Figure 4. Simulation of the tri-state.

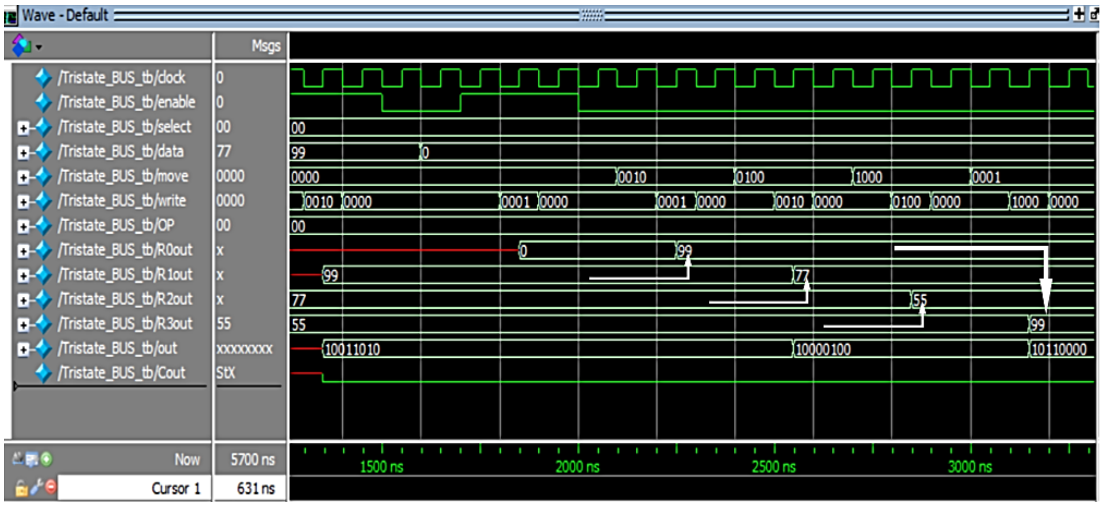


Figure 5. the transfer via bus to different registers.

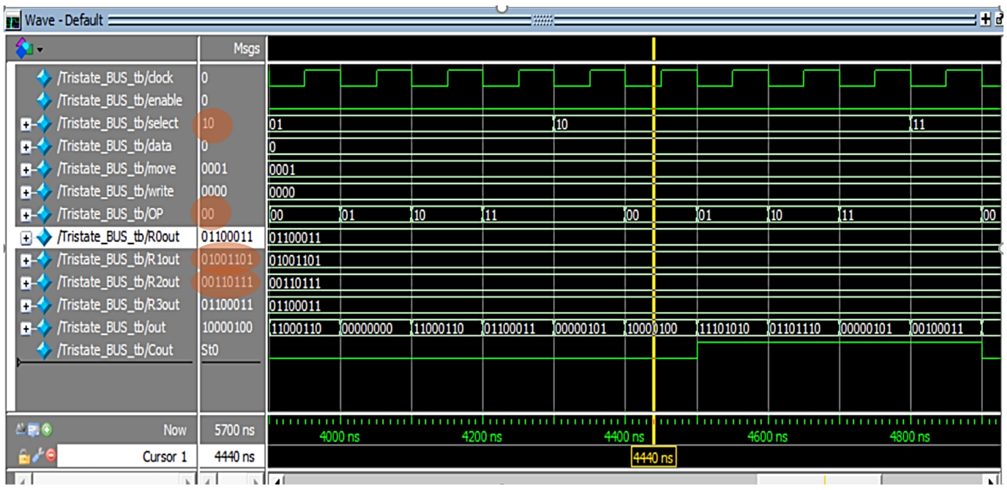
As shown in table 2, after shifting operation of registers, the register that contains the instruction is chosen by "select", and instruction selection is depends on the opcode. Then the instruction is executed, the result of the operation is written into Out, and when the remainder is obtained according to some instruction, it is written into Cout. The simulated waveforms of the tristate bus system register show in figure 6.

When "select"' is equals '10', the registers R1, R2 are used to select the instruction according to the opcode, opcode = 00 , so the instruction is ADD. Since the value of R1 is equal 01001101 and the value of R2 is equal 00110111, the result of the addition process is equal to 10000100, the result is kept in out and in this case there is no remainder, so the value of Cout equals zero.

TABLE 2. **Select resisters and ALU operation.**

Select	Resister	OP	Instruction	Operation
00	R3, R2	00	ADD	Out = A+B (Cout is carry)
00	R3, R2	01	SUB	Out =A-B
00	R3, R2	10	SHL	A<<1
00	R3, R2	11	AND	A & B
01	R3, R0	00	ADD	Out = A+B (Cout is carry)
01	R3, R0	01	SUB	Out =A-B
01	R3, R0	10	SHL	A<<1
01	R3, R0	11	AND	A & B
10	R2, R1	00	ADD	Out = A+B (Cout is carry)
10	R2, R1	01	SUB	Out =A-B
10	R2, R1	10	SHL	A<<1
10	R2, R1	11	AND	A & B
11	R2, R0	00	ADD	Out = A+B (Cout is carry)
11	R2, R0	01	SUB	Out =A-B
11	R2, R0	10	SHL	A<<1
11	R2, R0	11	AND	A & B

* OP =opcode & Out = output & ADD = addition & SUB = Subtraction & SHL=shift left.


Wave - Default
Msgs
/Tristate_BUS_tb/clock 0
/Tristate_BUS_tb/enable 0
/Tristate_BUS_tb/select 10
/Tristate_BUS_tb/data 0
/Tristate_BUS_tb/move 0001
/Tristate_BUS_tb/write 0000
/Tristate_BUS_tb/OP 00
/Tristate_BUS_tb/R0out 01100011
/Tristate_BUS_tb/R1out 01001101
/Tristate_BUS_tb/R2out 00110111
/Tristate_BUS_tb/R3out 01100011
/Tristate_BUS_tb/out 10000100
/Tristate_BUS_tb/Cout St0
01 10 11
0
0001
0000
00 01 10 11 00 01 10 11 00
01100011
01001101
00110111
01100011
11000110 00000000 11000110 01100011 00000101 10000100 11101010 01101110 00000101 00100011
Now 5700 ns
Cursor 1 4440 ns
4000 ns 4200 ns 4400 ns 4600 ns 4800 ns
4440 ns

Figuer 6. ALU “ADD” operation.

The resulting simulation waveform of post synthesis models as shown in figures 7,8,9 demonstrates that both systems use the same dataset, except that the first module uses the tristate bus and the second module uses the multiplexer bus to interconnect the datapath registers. Both systems show that they work identical to each others.

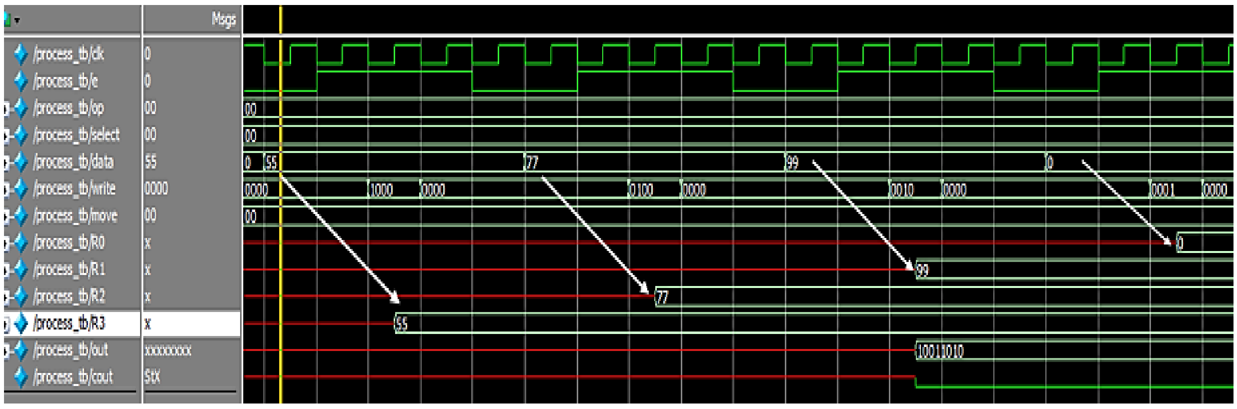


Figure 7. Simulation of the mutiplexser.

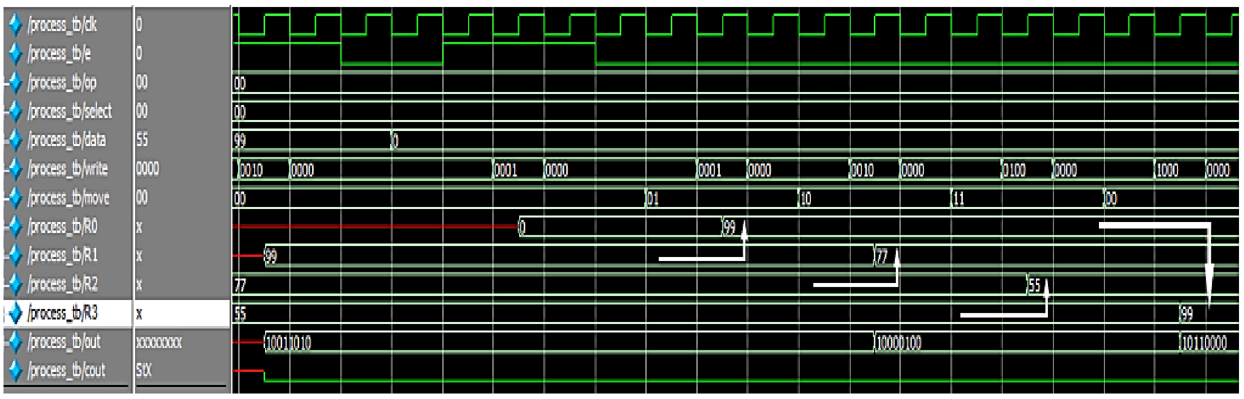
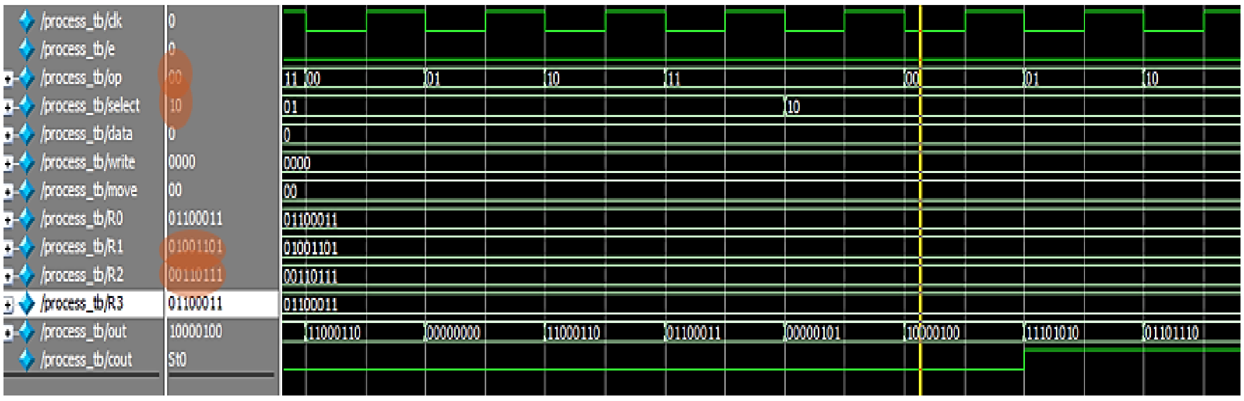


Figure 8. the transfer via bus to different registers.



Figuer 9. ALU “ADD” operation.

REGISTER TRANSFER LEVEL

The proposed multiplexer bus module requires less FPGA chip resources to implement the bus system because it has fewer register transfer levels than the tristate bus module. Figure 10 shows the register transfer level (RTL) of the tristate bus module, and figure 11 shows the RTL of the multiplexer bus module.

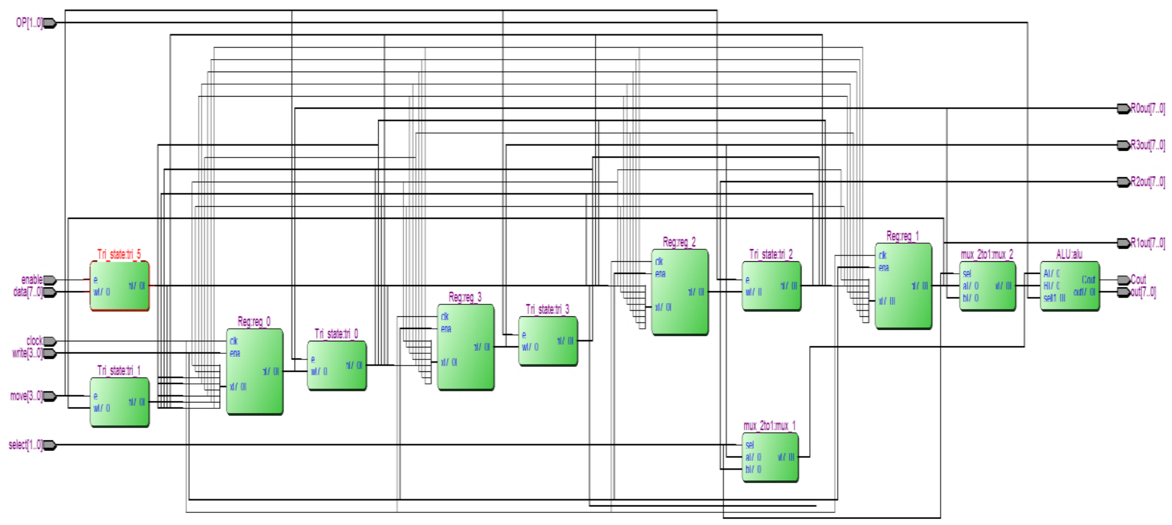


Figure 10. Register transfer level schematic.

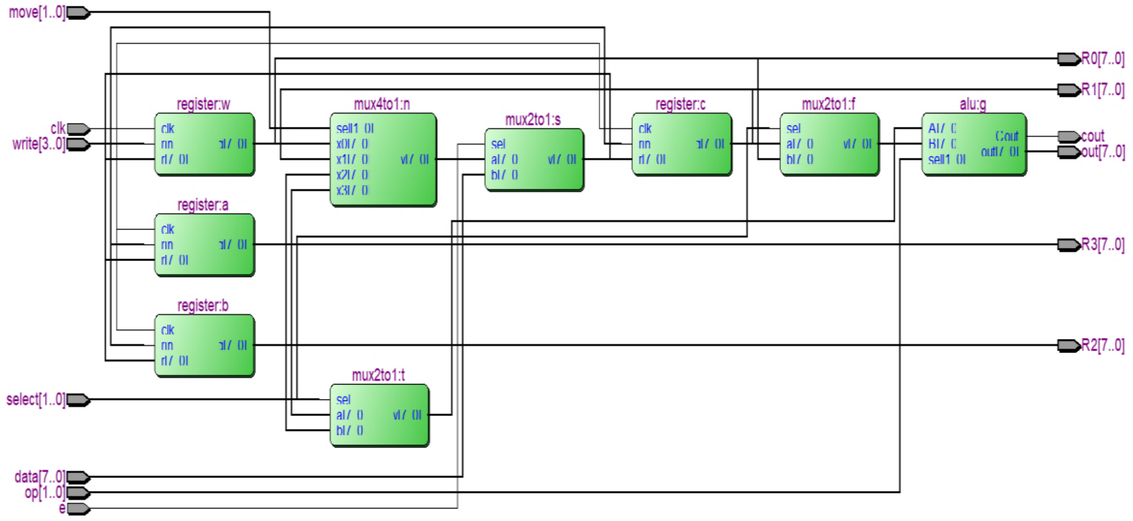
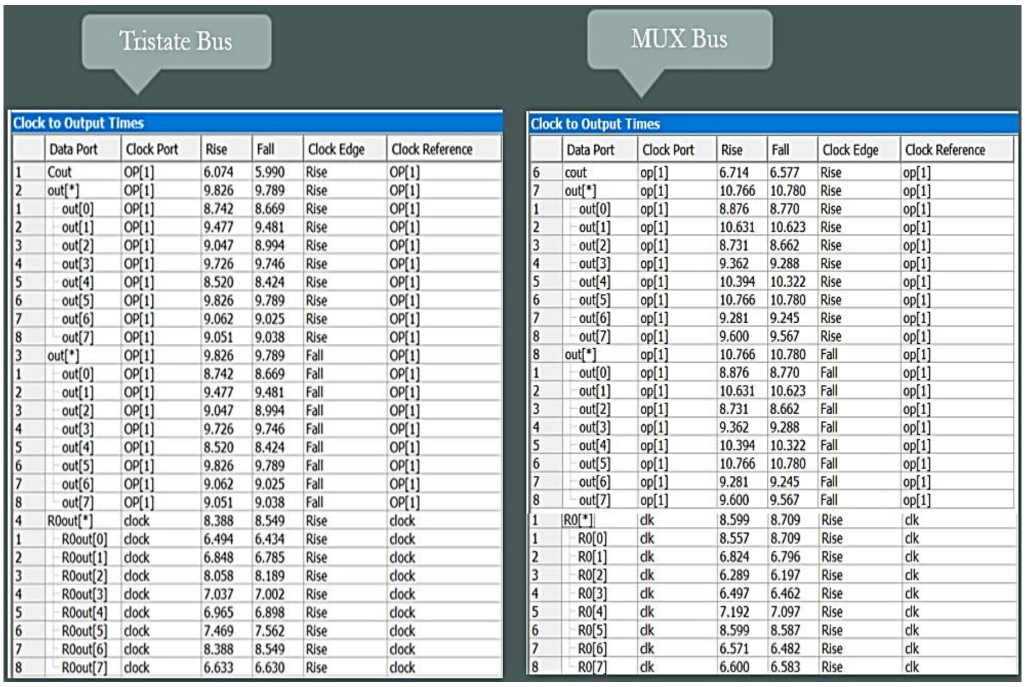


Figure 11. Register transfer level schematic.

CLOCK TO OUTPUT TIMES

Clock to Output Times is a timing analyzer used by Time Quest applications under Intel-Altera Quartus II software tools. Time tests of both modules in figure 12 shows that both modules have approximatly same time delay. In figure 13 shows that the multiplexer bus module has a lower power consomption than the tristate bus module.



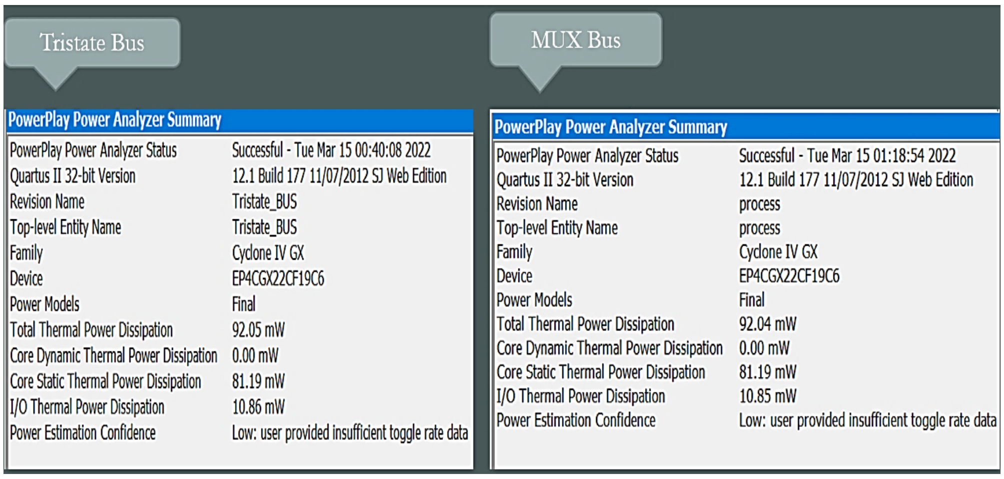
Tristate Bus – Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	Cout	OP[1]	6.074	5.990	Rise	OP[1]
2	out[*]	OP[1]	9.826	9.789	Rise	OP[1]
1	out[0]	OP[1]	8.742	8.669	Rise	OP[1]
2	out[1]	OP[1]	9.477	9.481	Rise	OP[1]
3	out[2]	OP[1]	9.047	8.994	Rise	OP[1]
4	out[3]	OP[1]	9.726	9.746	Rise	OP[1]
5	out[4]	OP[1]	8.520	8.424	Rise	OP[1]
6	out[5]	OP[1]	9.826	9.789	Rise	OP[1]
7	out[6]	OP[1]	9.062	9.025	Rise	OP[1]
8	out[7]	OP[1]	9.051	9.038	Rise	OP[1]
3	out[*]	OP[1]	9.826	9.789	Fall	OP[1]
1	out[0]	OP[1]	8.742	8.669	Fall	OP[1]
2	out[1]	OP[1]	9.477	9.481	Fall	OP[1]
3	out[2]	OP[1]	9.047	8.994	Fall	OP[1]
4	out[3]	OP[1]	9.726	9.746	Fall	OP[1]
5	out[4]	OP[1]	8.520	8.424	Fall	OP[1]
6	out[5]	OP[1]	9.826	9.789	Fall	OP[1]
7	out[6]	OP[1]	9.062	9.025	Fall	OP[1]
8	out[7]	OP[1]	9.051	9.038	Fall	OP[1]
4	R0out[*]	clock	8.388	8.549	Rise	clock
1	R0out[0]	clock	6.494	6.434	Rise	clock
2	R0out[1]	clock	6.848	6.785	Rise	clock
3	R0out[2]	clock	8.058	8.189	Rise	clock
4	R0out[3]	clock	7.037	7.002	Rise	clock
5	R0out[4]	clock	6.965	6.898	Rise	clock
6	R0out[5]	clock	7.469	7.562	Rise	clock
7	R0out[6]	clock	8.388	8.549	Rise	clock
8	R0out[7]	clock	6.633	6.630	Rise	clock

MUX Bus – Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
6	cout	op[1]	6.714	6.577	Rise	op[1]
7	out[*]	op[1]	10.766	10.780	Rise	op[1]
1	out[0]	op[1]	8.876	8.770	Rise	op[1]
2	out[1]	op[1]	10.631	10.623	Rise	op[1]
3	out[2]	op[1]	8.731	8.662	Rise	op[1]
4	out[3]	op[1]	9.362	9.288	Rise	op[1]
5	out[4]	op[1]	10.394	10.322	Rise	op[1]
6	out[5]	op[1]	10.766	10.780	Rise	op[1]
7	out[6]	op[1]	9.281	9.245	Rise	op[1]
8	out[7]	op[1]	9.600	9.567	Rise	op[1]
8	out[*]	op[1]	10.766	10.780	Fall	op[1]
1	out[0]	op[1]	8.876	8.770	Fall	op[1]
2	out[1]	op[1]	10.631	10.623	Fall	op[1]
3	out[2]	op[1]	8.731	8.662	Fall	op[1]
4	out[3]	op[1]	9.362	9.288	Fall	op[1]
5	out[4]	op[1]	10.394	10.322	Fall	op[1]
6	out[5]	op[1]	10.766	10.780	Fall	op[1]
7	out[6]	op[1]	9.281	9.245	Fall	op[1]
8	out[7]	op[1]	9.600	9.567	Fall	op[1]
1	R0[*]	clk	8.599	8.709	Rise	clk
1	R0[0]	clk	8.557	8.709	Rise	clk
2	R0[1]	clk	6.824	6.796	Rise	clk
3	R0[2]	clk	6.289	6.197	Rise	clk
4	R0[3]	clk	6.497	6.462	Rise	clk
5	R0[4]	clk	7.192	7.097	Rise	clk
6	R0[5]	clk	8.599	8.587	Rise	clk
7	R0[6]	clk	6.571	6.482	Rise	clk
8	R0[7]	clk	6.600	6.583	Rise	clk

Figure 12. Different between tristate and MUX in terms of time delay.



Tristate Bus – PowerPlay Power Analyzer Summary

PowerPlay Power Analyzer Status	Successful - Tue Mar 15 00:40:08 2022
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Tristate_BUS
Top-level Entity Name	Tristate_BUS
Family	Cyclone IV GX
Device	EP4CGX22CF19C6
Power Models	Final
Total Thermal Power Dissipation	92.05 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	81.19 mW
I/O Thermal Power Dissipation	10.86 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

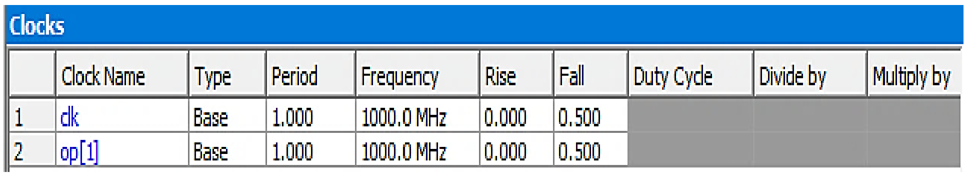
MUX Bus – PowerPlay Power Analyzer Summary

PowerPlay Power Analyzer Status	Successful - Tue Mar 15 01:18:54 2022
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	process
Top-level Entity Name	process
Family	Cyclone IV GX
Device	EP4CGX22CF19C6
Power Models	Final
Total Thermal Power Dissipation	92.04 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	81.19 mW
I/O Thermal Power Dissipation	10.85 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Figure 13. Different between tristate and MUX in terms of thermal power.

SYSTEM FREQUENCY

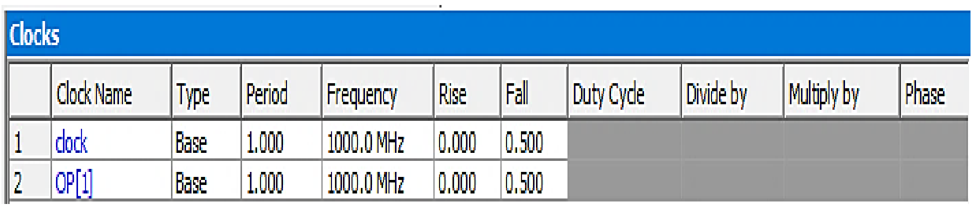
Both systems are working on 1000 MHz as shown in figure 14,15.



Clocks

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by
1	clk	Base	1.000	1000.0 MHz	0.000	0.500			
2	op[1]	Base	1.000	1000.0 MHz	0.000	0.500			

Figure 14. MUX system frequency.



Clocks

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase
1	clock	Base	1.000	1000.0 MHz	0.000	0.500				
2	OP[1]	Base	1.000	1000.0 MHz	0.000	0.500				

Figure 15. Tri-state system frequency.

CONCLUSION

The key contribution of this research work is to design, simulate, and implement an efficient microprocessor system based on multiplexer-based bus structure, that can be used in FPGA designs with limited tristate bus resources. The waveforms and results obtained from the simulation and test processes show that the proposed microprocessor structure uses less hardware resources than the tristate-based bus structure, with almost same time delay and less thermal power consumption. In the future work, the investigation can be extended to include implementations of multiprocessors system that can mix two kind of buses to interconnect between internal registers instead of only tri-state bus. After verifying the design sub-modules individually, the submodules are instantiated, simulated and verified, the the system was implemented and tested using Cyclone EP1C6Q240C8 FPGA evaluation platform that designed specially to test the functionality of the system in hardware.

REFERENCES

[1]. Li Jingpeng, “An optimized design of MCU including predication,” Microelectronics and computer, vol.23, pp.25-27, 2006.
[2]. Tian Hongli, Yan Huiqiang, Geng Hengshan, Liu Su, “Design an implementation of 8-bit micro-controller,” Computer Engineering and Applications, Vol.46, pp.60-63, 2010.
[3]. Johnson and Graham, “High Speed Digital Design: a Handbook of Black Magic,” Prentice Hall, 1993.
[4]. Nikil Dutt, Kaustav Banerjee, Luca Benini, Kanishka Lahiri, Sudeep Pasricha, "Tutorial 5: SoC Communication Architectures: Technology, Current Practice, Research, and Trends",

vlsid, pp.8, 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07), 2007.
[5]. Altera Corporation, “Comparing IP Integration Approaches for FPGA Implementation”.
[6]. The IEEE Standard Hardware Description Language based on the Verilog Hardware Description Language (IEEE Std 1364-2001).
[7]. Micheal D. Ciletti, “Advanced Digital Design with the Verilog HDL “Prentice Hall,2004.
[8]. William Stallings, “Computer Organization and Architecture, Designing for Performance”, Prentice Hall, 2001.
[9]. A.-S. Jamal, J. Goeders, and S. J. E. Wilton, “An FPGA overlay architecture supporting rapid implementation of functional changes during on-chip debug,” in 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018.
[10]. A.-S. Jamal, “An FPGA overlay architecture supporting software-like compile times during on-chip debug of high-level synthesis designs,” Ph.D. dissertation, University of British Columbia, 2018.
[11]. P. Mishra and F.Farahmandi, Post-Silicon Validation and Debug. Cham, Switzerland: Springer, 2019.
[12]. H. Oh, T. Han, I. Choi, and S. Kang, “An on-chip error detection method to reduce the post-silicon debug time,” IEEE Transactions on Computers, vol. 66, no.1, pp . 38_44, Jan 2017.
[13]. H. Oh, I. Choi, and S. Kang, “DRAM-based error detection method to reduce the post-silicon debug time for multiple identical cores,” IEEE Transactions on Computers, vol. 66, no. 9, pp. 1504–1517, Sep. 2017.
[14]. Y. Cao, H. Palombo, S. Ray, and H. Zheng, “Enhancing observability for post-silicon debug with on-chip communication monitors,” in 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2018, pp. 602–607.
[15]. D. Holanda Noronha, R. Zhao, J. Goeders, W. Luk, and S. J. E. Wilton, “On-chip fpga debug instrumentation for machine learning applications,” in Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2019, pp. 110–115.
[16]. K. Rahmani and P. Mishra, “Feature-based signal selection for post-silicon debug using machine learning,” IEEE Transactions on Emerging Topics in Computing, pp. 1–1, 2017.