www.ijaceeonline.com ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume 7, Issue 6, June 2022, pp. 1–8

Rules of RTL Conversion on RISC-V Processor Design

Demyana Emil
Teaching Assistant, Electrical Department, Faculty of Engineering,
Fayoum University, Egypt
Email: Dem11@fayoum.edu.eg

Mohammed Hamdy
Assistant professor, Electrical Department, Faculty of Engineering,
Fayoum University, Egypt
Email: Mhm00@fayoum.edu.eg

Gihan Nagib
Associate professor, Electrical Department, Faculty of Engineering,
Fayoum University, Egypt
Email: Gna00@fayoum.edu.eg

Abstract: *Hardware design is an important process in computer architecture, digital electronics and more fields related to computer science. Register transfer level (RTL) synthesis, an important step in digital hardware design, abstracts the model of an implemented design to a digital electronics circuit in registers form. This paper introduces some rules on an implemented design to make it satisfy with the RTL level rules. These rules ease the conversion of this design to RTL circuit. This process is occurring after simulation level directly. Hence, the paper discusses some common issues that some hardware designers may make on simulation level that prevent their design to work on RTL successfully. The work introduces solutions for these issues, in addition to some common simulation errors that can be avoided. If the rules of RTL have been followed in simulation level (initial step), it reduces more time exerted on modification on design after the simulation level for making the design available to work on RTL. It is also rare to find such this topic explained clearly and separately, however, it is so important. This work depends on an implemented open processor. This processor is single-core and works under RISC-V instruction set.*

Keyword: *RTL; Open processors; Hardware design; RISC-V.*

1. INTRODUCTION

There is a big difference between successfully working design in simulation level, and its availability of working as a digital electronics circuit. In case of failing to be converted to hardware electronics circuit, it can't be fabricated. It appears successfully work in simulation but in reality, can't be converted to hardware circuit. More designers start in simulation writing their design but don't make attention to the RTL verifications [1]. It makes them get into redesign the project to adapt RTL rules. This problem is a critical one because the design itself takes more efforts and more time to be successfully work. Not more papers discuss this topic independently. Just few ones dicuss the RTL system such as the work of Johnson, R.E., Mcconnell, C., Lake, J.M. [2].

Cite this paper:

Demyana Emil, Mohammed Hamdy, Gihan Nagib, "Rules of RTL Conversion on RISC-V Processor Design", International Journal of Advances in Computer and Electronics Engineering, Vol. 7, No. 6, pp. 1-8, June 2022.

An open RISC-V processor (Taiga) was chosen to be worked on as a main block [3]. This processor has various bugs and some of them have been corrected depended on our developed algorithms (they will be shown later). This processor is available for modification, is flexible and modular. It has been implemented based on RISC-V instruction set architecture [4]. The design is based on SystemVerilog hardware language [5]. The main simulator tool that has been used for testing is Vivado 2020.2 [6]. As shown in Figure 1, the block diagram of the Taiga processor. Most of implemented blocks for this processor are shown in Figure 1. The blocks that will be talked about is the memory and the register file.

For sure, the coming work isn't the total rules for RTL level implementation and verification. Just this work is deduced from extensive work design and testing on the open-source RISC-V processor. Additionally, it shows some simulation errors, that should be avoided, and they can be avoided.

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume 7, Issue 6, June 2022, pp. 1–8

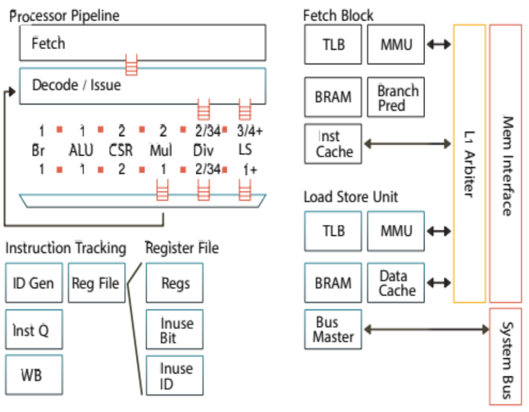


Figure 1 Block diagram of single-core Taiga processor [3]

This work is sectioned as: headline 2 introduces a related work with RTL model. Headline 3 introduces some simulation errors that can be corrected and avoided in the future by the designers. Headline 4 discusses RTL design paradigms. It explains that how to be avoided or corrected as possible depended on set of different developed algorithms. Additionally, it discusses the result of these algorithms after they have been applied on the processor design. It shows the difference among multiple procedures that are mainly used in the digital electronics design. Headline 5 is a conclusion about what has been achieved; the result of applied algorithms. In brief, it shows how they have an effective role on the design, and improved the output result of some blocks. It contains what has been mentioned about simulation errors. In addition, it includes our future work on the processor design and how RTL rules can be used further more for more enhancement.

2. RELATED WORK

This partition explains the historical work about RTL rules of Verilog coding and how it is related with real hardware circuits [7-8]. These cited rules are estimated for Verilog HDL (Hardware Design Language) and can be used for different digital electronics systems like VLSI [9-10]. But for SystemVerilog, the design requires additional rules [11].

Multiple standard HDLs are available for hardware structure design, to get the preferable one, Pong makes a good comparison between SystemVerilog and Verilog [12]. It has been shown from the previous comparison that the SystemVerilog is prefered for more efficient verification. Like that has been used in a peripheral device for management process between two RvCores [13]. SystemVerilog is used for more accurate synthesis for complex design [14]. RTL has Multiple fields that can be implemented like CNN (convolutional neural networks) [15], FINN matrix vector [16] and can be used in Matlab (math works) [17].

RTL coding purpose is for FPGA [18]. The output RTL model is ready to work on the FPGA kits. Like these designs, can be optimized for more enhancement too [19].

Additionally, to avoid getting error such as multiple drivers and so on simulator, some rules are explained for this purpose [20]. RTL designers always use the always procedures. Each hardware language has different forms of always procedures. Verilog language has a fixed always procedure format without additional forms. But SystemVerilog has three types of always procedures like always [21], always_ff [5] and always_comb [22]. For Verilog, to access always body for starting a design, there some assignment rules for included signals like blocking and non-blocking assignments [23].

3. COMMON SIMULATION ERRORS AND CORRECTION

On design, the common problem that may be occurred is “don’t care signals”. This problem expresses that the designed block hasn’t finished its process. So, the output still with no value until the block process is done. It also affects other dependent signals at the same block in case of combining. Additionally, it affects other signals in other blocks in case of block’s output signals dependency.

Example:

If interface1 & interface2, shown in Figure 2, aren’t initiated at time zero, they become “don’t care” signals. In case inteface1 has arrived but interface2 hasn’t arrived yet, the output that depends on interface1 and interface2 may become don’t care.

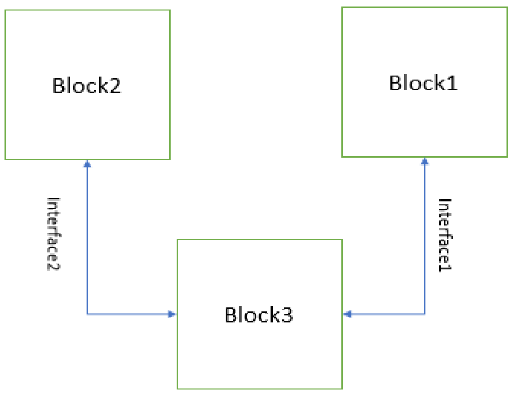


Figure 2 Block diagram of signal dependency

Table I shows different logic operations and the output signal that depends on set of valuable and don’t care signals. If block3’s output depends on interface1 and interface2, the output still “don’t care” in some cases like case1 and case4 in Table I.

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume 7, Issue 6, June 2022, pp. 1–8

TABLE I LOGIC OPERATIONS BETWEEN VALUABLE AND DON'T CARE SIGNALS

	Interface1	Interface2	Logic op-eration	Output (y)
Case1	X 1	1 X	OR	1
Case2	X 0	0 X	OR	X
Case3	X 1	1 X	AND	X
Case4	X 0	0 X	AND	0

To avoid the problem due to "don't care" signals, there are two solutions that can be implemented. They are shown in Table II.

Case	Solution 1	Solution 2
Case2 (in Table I)	Algorithm 1	Initialize interface 1&2 by zero (in block 1 &2)
Case3 (in Table I)	Algorithm 2	Initialize interface 2&1 (in block 2&1) by zero

To solve the above problem in hardware design, Algorithm 1 and Algorithm 2 implement simple temporary solutions. These solutions can be implemented for signals that you don't want to initialize.

TABLE III ALGORITHM OF DON'T CARE SIGNAL SOLUTIONS

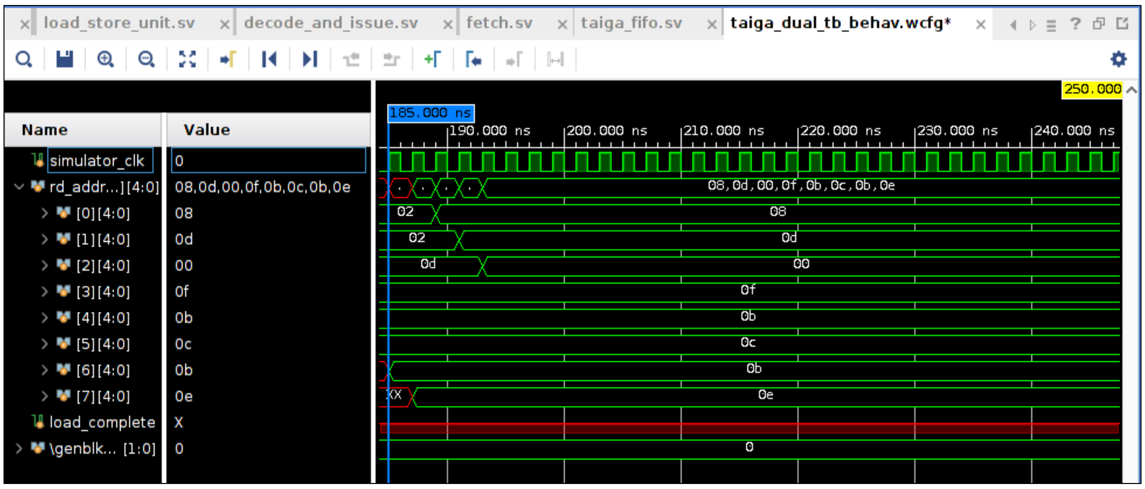
```
Algorithm 1
If (interface1 | interface2)
y =1;
Else
y =0;
```

TABLE IV ALGORITHM 2 OF DON'T CARE SIGNAL SOLUTIONS

```
Algorithm 2
If (interface1 & interface2)
y =1;
Else
y =0;
```

The two solutions can be implemented according to the design. Algorithm 1 and algorithm 2 have been implemented on a RISC-V processor [3]. Figure 3 shows the problem that has been result from "don't care" signals coming as an input signal to a block in the processor design.

The waveforms in Figure 3 express set of data stored at a data structure acting as registers. The locations shown in the Figure 3 entitle that there are set of locations have updated their contents with new data in case the others haven't. This is the problem, the input "don't care" signals have affected this block not to make it available to update some locations at its data structure unit. Location 3 is an example for those which have been affected by this problem.



Name	Value
simulator_clk	0
rd_addr...][4:0]	08,0d,00,0f,0b,0c,0b,0e
[0][4:0]	08
[1][4:0]	0d
[2][4:0]	00
[3][4:0]	0f
[4][4:0]	0b
[5][4:0]	0c
[6][4:0]	0b
[7][4:0]	0e
load_complete	X
\genblk... [1:0]	0

Figure 3 The problem of don't care affecting output signals of a block design

Figure 4 shows the block signals after implementing the previous Algorithm 1. Algorithm 2 is available too. Algorithm 1 has been implemented according to the logic operation processed on the input signals at the block.

Algorithm 2 can be applied if the logic operation on the block input signals is "And" logic.

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume 7, Issue 6, June 2022, pp. 1–8

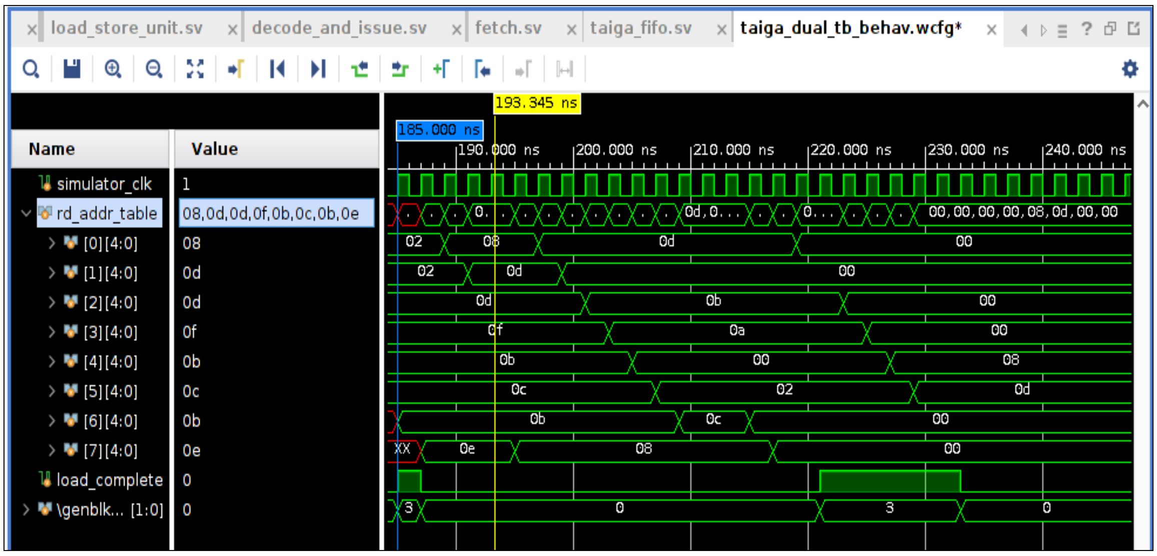


Figure 4 The designed block signals after implementing Algorithm 1

4. RTL RULES

RTL is a required stage in design [24]. As mentioned in *Introduction*, it is for conversion your design to applicable hardware logic circuit.

4.1 Main Procedures

The main important procedures in hardware design are always, always_ff and always_comb. There are also some important statements such as “assign” statement. Each one of the previous processes has a different meaning:

1. **Always:** is a procedure acting as a wrapper of a segment of code in case some special signals change [21].

   Example: always @ (signal) **begin**
   // Code segments;
   End

2. **Always_comb:** is a procedure acting as a wrapper of design code steps. It is used in case you want these code steps permanently executed even in no changing on any signals. It will be triggered automatically once at time zero [5]. It has no limitation to be triggered.

   Example: always_comb **begin**
   // Code steps;
   End

3. **Always_ff:** is a procedure acting as a wrapper of code lines in case of changing at a clock signal. It is triggered on positive edge, negative edge or both [22]. It also can be triggered on another signal acting like the clock signal.

   Example: always_ff @ (posedge clk) **begin**
   // Code lines;
   End

4. **Assign:** is a statement that permanently copies the right-hand side at the left-hand side.

   Example: assign exp_1 = exp_2;

5. **If … statement:** is an if statement like that in C++ language [25].

   Example: if (condition_1 is true) **begin**
   // Code segment;
   End

   Else if (condition_2 is true) **begin**
   // Another code segment;
   End
   Else **begin**
   // code segment;
   End

6. **Generate statement:** it is used if a for loop statement will be used (iterative process as shown in Table V) or for conditional blocks.

TABLE V ALGORITHM 3 OF GENERATE STATEMENT FOR ITERATIVE PROCESS

Algorithm 3 Generate statement for iterative process

```
Int num = 6;
Genvar i;
Generate
For (i= 0 ; i < num; i ++)  begin
If (condition) begin
.
.
.
End
End
Endgenerate
```

4.2 Rules of RTL Level

For RTL implementation, there some rules for hardware design. Below examples show obviously what should be taken on HW development.

- **Rule_1**: not to insert the Assign statement into any type of always procedure.

The big common error is to insert an assign statement in any type of the previous always procedure. If the block has interface of set of signals (Ex: interface_1 shown in the below example (Table VI))

TABLE VI AN EXAMPLE OF ASSIGN STATEMENT INTO AN ALWAYS PROCEDURE

Example_1

```
Generate
Always_ff @ (posedge clk) begin
If (condition_1 is true) begin
Assign interface_1 = value_1;
.
.
.
End
End
Endgenerate
```

Table VI is an example of an unavailable design for RTL level but it works normally in simulation. It should be replaced by another algorithm. Such that is shown in Table VII.

TABLE VII ALGORITHM 4 OF RULE 1

Algorithm 4

```
Logic signal_1;
Assign interface_1 = signal_1;
Generate
Always_ff @ (posedge clk) begin
If (condition_1 is true) begin
Assign interface_1 = value_1;
.
.
.
End
End
Endgenerate
```

From example_1; it is shown that the first rule is not to insert any assign statement in always_ff procedure. Figure 5 shows the problem. Figure 6 shows the waveform after Algorithm 4 is appended.

- **Rule_2**: for any implemented memory, in RTL on Vivado, it is a must for its size not to exceed 1000000 bits.

- **Rule_3**: from example_1, it is noticed that assign statement is usually used when updating an interface on a current block. It is the third rule; when updating some signals in an interface of the block, it is a must to use assign statement.

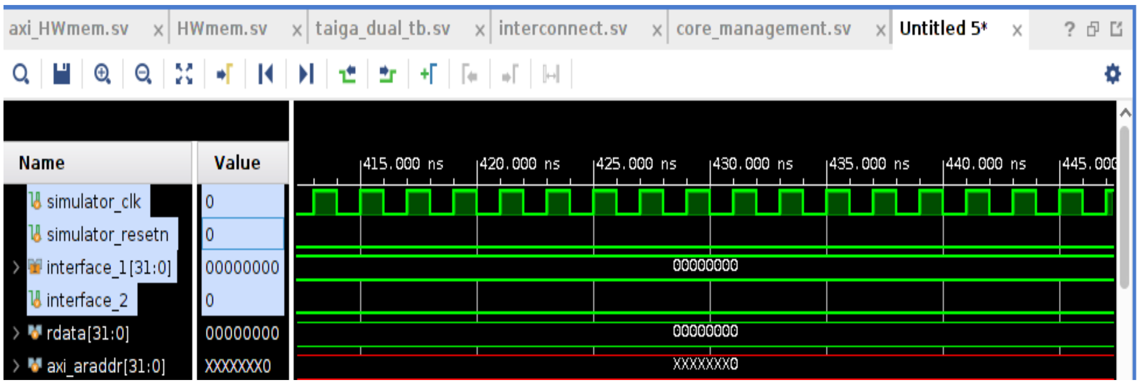


Figure 5 Waveform of signals in case assign statement is appended in always_ff procedure

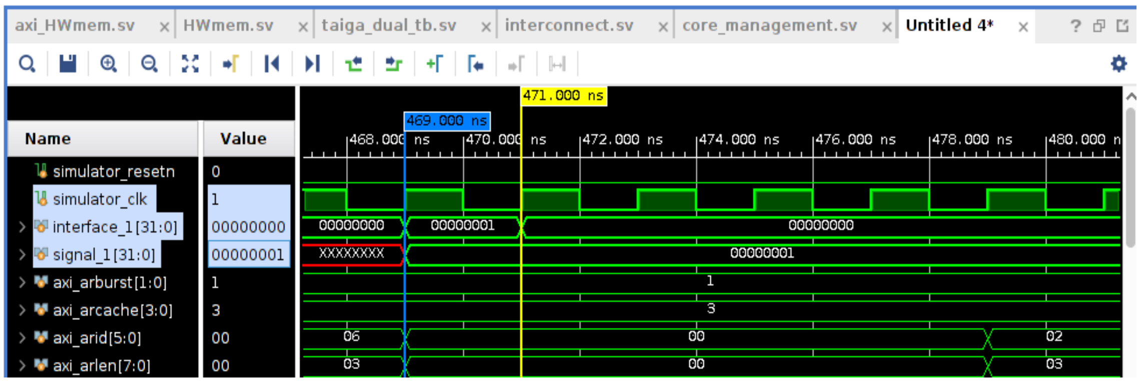


Figure 6 Waveform of signals after algorithm 4 is implemented

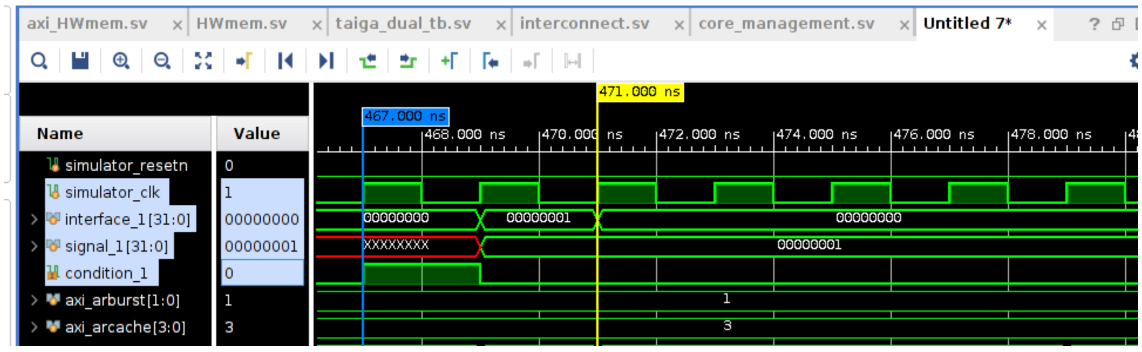


Figure 7 Waveform of signals wrapped in always_ff procedure in case of using if statement

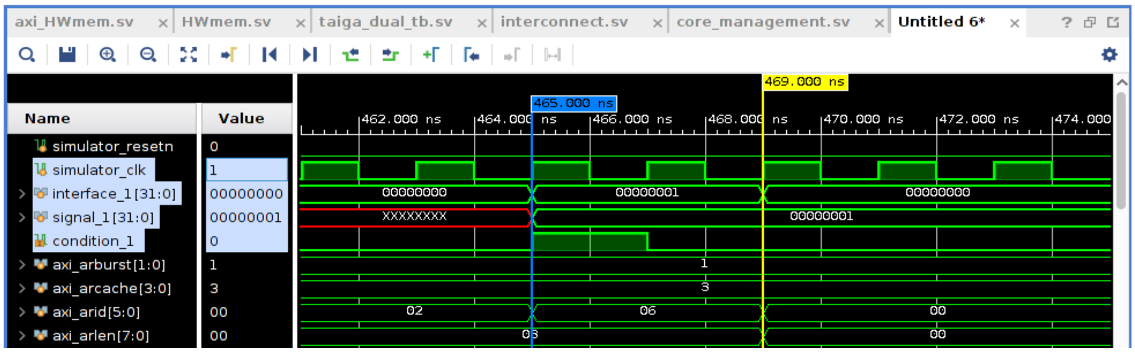


Figure 8 Waveform of signals wrapped in always_comb procedure in case of using if statement

- **Rule_4:** in any always procedure in case of using if statement, if the condition has been true at a clock cycle, then in the next clock cycle, has become false, all signals in the if

statement body would be updated by zero. Table VIII shows this algorithm applied on always_ff.

TABLE VIII IF STATEMENT INSIDE ALWAYS_FF PROCEDURE

```
Algorithm 5
Always_ff @ (posedge clk) begin

If (condition) begin

.

.

.

End

End
```

Rule 4 is interpreted in Figure 7 on the waveforms of an arbitrary block as an application of Algorithm 5.

In case of using always_comb instead of always_ff in Algorithm 5, the waveforms differ a bit (Figure 8). The difference is considered in the clock line.

So, from the two previous figures (Figure 7,8), it is option for the designer to choose the best procedure for his design according to the main behavior. In the two procedures, it has been found that if the condition is true, the copying process is done successfully. Otherwise, (take into consideration Algorithm 4) interface_1 is updated by zero even signal_1 hasn't been changed to zero. Take into your consideration that this copying process doesn't occur continuously nor automatically. It occurs only in case of true condition otherwise interface_1 doesn't keep its value taken from signal_1 at a time while true condition.

- **Rule_5**: if you need to define a memory block, it should be finite in size.

- **Rule_6**: when the memory is loaded by a program, it should start being loaded the initial address of the memory. Also, it should be loaded by a binary program file only.

5. CONCLUSION AND FUTURE WORK

The paper has shown the difference among set of procedures and special statements in simulation. Additionally, simulation level is a base level in digital hardware design to check the working design correctly. The RTL process is a second important level in hardware design following the simulation level. The paper has shown that we can apply some RTL rules in simulation first, hence we can avoid recodification on our developed design. Some extensive tests have applied on an open single RvCore and we have applied our developed algorithms and showed our results of these algorithms on the waveforms of arbitrary blocks of the processor. We have shown the performance of these blocks before and after applying the algorithms. These algorithms have corrected more bugs in this processor design. The paper also has discussed some special simulation errors on design and how they have been corrected by multiple methods. It has shown multiple solutions according to the designer target. It has been shown that RTL synthesis process has different rules from simulation rules. Some rules, result from extensive testing, have been presented and explained in details. SystemVerilog hardware language has been used to implement the previous algorithms. Vivado tool is the main tool that has been used.

In the future, we will invest more time in improving the design performance according to what have been addressed in RTL rules. Additionally, how to implement a main memory based on RTL rules and reduce the total number of clock cycles consumed for it. How to build a simulation memory. How to make the design work on the FPGA or Zedboard kits. Hence, we can compare our result with standard results. We will use standard benchmarks to test and verify the total design.

REFERENCES

[1] Yadav, P. Jindal and D. Basappa, (2020), "Study and Analysis of RTL Verification Tool" *IEEE Students Conference on Engineering & Systems (SCES)*, pp. 1-6, doi: 10.1109/SCES50439.2020.9236747.

[2] Johnson, R.E., Mcconnell, C., Lake, J.M. (1992). "The RTL System: A Framework for Code Optimization" In: Giegerich, R., Graham, S.L. (eds) Code Generation — Concepts, Tools, Techniques. *Workshops in Computing. Springer, London*. https://doi.org/10.1007/978-1-4471-3501-2_14

[3] Eric Matthews and Lesley Shannon. (2017), "TAIGA: A Configurable RISC-V Soft-Processor Framework for Heterogeneous Computing Systems Research" *SEMANTIC SCHOLAR*, Vol III

[4] Andrew Waterman, Yunsup Lee, David A. Pattson, Krste Asanovic, [May, 2014], "The RISC_V Instruction Set Manual" Volume 1, Version 2.0

[5] Vaibbhav Taraate, (2020) "SystemVerilog for Hardware Description RTL Design and Verification". *Springer* https://doi.org/10.1007/978-981-15-4405-7

[6] Vivado Design Suite User Guide, http://www.xilinx.com/warranty.htm

[7] S. Gayathri and T. C. Taranath, (2017), "RTL synthesis of case study using design compiler" International Conference on Electrical, Electronics, Communication, Computer, and

Optimization Techniques (ICEECCOT), pp. 1-7, doi: 10.1109/ICEECCOT.2017.8284603.

[8] Taraate, V. (2019), "RTL Design Guidelines", In: Advanced HDL Synthesis and SOC Prototyping. Springer, Singapore. https://doi.org/10.1007/978-981-10-8776-9_3

[9] (2007), "RTL Coding Guidelines", In: Digital VLSI Systems Design. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-5829-5_5

[10] RTL DESIGN GUIDELINES, retreived date: [2022], online available at: http://www.asic.co.in/DesignGuidlinesRTLcoding.htm

[11] "SystemVerilog RTL Tutorial", retreived date: [2017], online at:https://www.doulos.com/knowhow/systemverilog/systemverilog-tutorials/systemverilog-rtl-tutorial/

[12] Pong P. Chu, (May 2018), "SystemVerilog vs Verilog in RTL Design", pp. 1-4

[13] Demyana Emil, Mohammed Hamdy, and Jihan Nagib, (2022), "Dual-RvCore32IMA: Implementation of a Peripheral device to Manage Operations of Two RvCores" 4th International Conference on Intelligen Computing, Information and Control System, Springer, pp. 4-10

[14] Stuart Sutherland, Don Mills, (2013), "Synthesizing SystemVerilog

[15] Busting the Myth that SystemVerilog is only for Verification", SNUG Silicon Valley, pp. 4-20

[16] Syed Asad Alam, David Gregg, Giulio Gambardella, Thomas Preusser, Michaela Blott, (April 2022), "On the RTL Implementation of FINN Matrix Vector Compute Unit", arXiv:2201.11409v2Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, (May 2017), "Image net classification with deep convolutional neural networks," Commun. ACM, vol. 60, no. 6, pp. 84–90.

[17] "RTL Design Rule Parameters", retrieved date: [2019], online at: https://www.mathworks.com/help/hdlcoder/ug/rtl-design-rules.html

[18] Shipitko, Oleg & Grigoryev, Anton. (2018). Gaussian filtering for FPGA based image processing with High-Level Synthesis tools, pp. 15-200

[19] Johnson, R.E., Mcconnell, C., Lake, J.M. (1992), "The RTL System: A Framework for Code Optimization". In: Giegerich, R., Graham, S.L. (eds) Code Generation — Concepts, Tools, Techniques. Workshops in Computing. Springer, London. https://doi.org/10.1007/978-1-4471-3501-2_14

[20] Y. Zhang, H. Ren and B. Khailany, (2020), "Opportunities for RTL and Gate Level Simulation using GPUs (Invited Talk)", IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 1-5.

[21] Wilson Snyder, (2021) "Verilator"

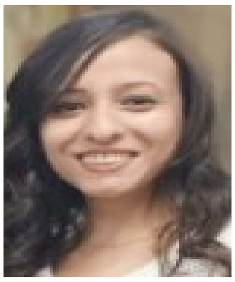
[22] Stuart Sutherland Simon Davidmann Peter Flake, "SystemVerilog For Design Second Edition". *Springer Science+Business Media*, LLC

[23] "Programmable Logic/Verilog RTL Coding Guidelines", retreived date: [2017], online at: https://en.wikibooks.org/wiki/Programmable_Logic/Verilog_RTL_Coding_Guidelines

[24] Michael Keating and Pierre Bricaud, (2001), "RTL Coding Guidelines", National Chiao Tung University, pp. 12-32

[25] Adam Drozdek, "Data Structures and Algorithms in C++" www.cengage.com/highered

Authors Biography



Demyana Emil, is teaching assistant at Electrical Department, Electronics and Communication Engineering major in Faculty of Engineering, Fayoum University, Egypt. She has completed B.Sc. degree in Electronics and Communication Engineering specialized in Embedded System at Fayoum University. Her research interests are computer architecture, synchronized data, hardware design and software development.

Mohammed Hamdy, is assistant professor at Electrical Department, Electronics and Communication Engineering major in Faculty of Engineering, Fayoum University, Egypt. He has completed B.Sc. degree in Electronics and Communication Engineering department at Fayoum University. He has completed M.Sc. degree in Electronics and Communication Engineering at Cairo University. His research interests are hardware design, interconnect network, MultiCore processor and controllers. He has completed PhD in Mechatronics at Japanese University, Egypt. His research interest is image processing.

Gihan Nagib, is associate professor at Electrical Engineering Department in Faculty of Engineering, Fayoum University, Egypt. She has completed B.Sc. degree in Computer and system Engineering, Ain Shams University. She has completed Diplome D'etudes Approfondies en Automatique, productique, Theorie des systemes (DEA) from INPG University, LAG, Grenoble FRANCE. Earned her PhD from INPG University, Grenoble, France in 1994. She has over 26 years' experience in teaching, research and consulting. Her research interests are Intelligence Computation, computer networks and Computer Architecture. She published many Research papers in various international journals and conferences.

Cite this paper:

Demyana Emil, Mohammed Hamdy, Gihan Nagib, "Rules of RTL Conversion on RISC-V Processor Design", International Journal of Advances in Computer and Electronics Engineering, Vol. 7, No. 6, pp. 1-8, June 2022.