Arid Zone Journal of Engineering, Technology and Environment. August, 2013; Vol. 9: 51-67 Copyright© Faculty of Engineering, University of Maiduguri, Nigeria. Print ISSN: 1596-2490, Electronic ISSN: 2545-5818 www.azojete.com.ng

DESIGN AND IMPLEMENTATION OF EMBEDDED HARDWARE ACCELERATOR FOR DIAGNOSING HDL-CODE IN ASSERTION-BASED VERIFICATION ENVIRONMENT

Ngene C. U.

(Department of Computer Engineering University of Maiduguri, Maiduguri, Nigeria) e-mail address: umerahlove@yahoo.co.uk

Abstract

The use of assertions for monitoring the designer's intention in hardware description language (HDL) model is gaining popularity as it helps the designer to observe internal errors at the output ports of the device under verification. During verification assertions are synthesised and the generated data are represented in a tabular forms. The amount of data generated can be enormous depending on the size of the code and the number of modules that constitute the code. Furthermore, to manually inspect these data and diagnose the module with functional violation is a time consuming process which negatively affects the overall product development time. To locate the module with functional violation within acceptable diagnostic time, the data processing and analysis procedure must be accelerated. In this paper a multi-array processor (hardware accelerator) was designed and implemented in Virtex6 field programmable gate array (FPGA) and it can be integrated into verification environment. The design was captured in very high speed integrated circuit HDL (VHDL). The design was synthesised with Xilinx design suite ISE 13.1 and simulated with Xilinx ISIM. The multi-array processor (MAP) executes three logical operations (AND, OR, XOR) and a one's compaction operation on array of data in parallel. An improvement in processing and analysis time was recorded as compared to the manual procedure after the multi-array processor was integrated into the verification environment. It was also found that the multi-array processor which was developed as an Intellectual Property (IP) core can also be used in applications where output responses and golden model that are represented in the form of matrices can be compared for searching, recognition and decision-making.

Keywords: Hardware accelerator, HDL-code, design, simulation, multi-array processor

1. Introduction

In recent years, the cost of software has become the dominant factor that determines the effectiveness of the creation of electronic devices in the electronic industry. Practically all electronic designs (from super computers, PCs to mobile handheld devices) in the industry are HDL-based (Ashenden, 2001; Begeron, 2003). HDL based design has established itself as the modern approach to design of digital systems, with VHDL and Verilog HDL being the two dominant HDLs. The benefits of HDL is that designs can be reused thus speeding up time to market. The continuous increase in the complexity of systems on chip led to the fact that traditional approaches to the verification can no longer be used effectively because of the limitations associated with the reduction in observability and controllability of the internal lines of the system. In general, assertion-based verification is used as part of the traditional methodology of simulation -directed and random simulation, formal and semi-formal method and emulation. An *assertion* is a statement about a design's intended behaviour or property, which must be verified (Foster, 2004). Its popularity is partly because it increases the controllability and observability which in turn improves the diagnosability (diagnostic resolution) of the code. Furthermore, assertions reduce debug time, improve integration through correct usage checking, improve verification efficiency and improve communication through documentation. Assertion synthesis is a technology that allows you to automatically

generate high-approval to verify the key constraints and design specifications, attributes, functional coverage, and identify problems in the test bench.

When assertions are synthesised for a very large code, the data generated become very large to the extent that it becomes very difficult to analyse and process them, thus defeating the purposes for which they were used in the first place. This situation increases the time required to locate the module with functional violation, which in turn negatively affect the product development time. It therefore, becomes necessary to reduce the time required to diagnose functional violation in a module by designing a built-in hardware accelerator (MAP) to concurrently process all the synthesised assertions.

This paper seeks to use the new models and methods for testing and diagnosis of HDL model system on chip (SoC) developed by Ngene (2011), to implement a built-in testing infrastructure that aids in the reduction of verification time and improves the quality of SoC models using hardware accelerators at the stages of system design. A MAP with a minimal instruction set architecture was designed and implemented in Virtex 6 FPGA. The processor was practically verified by using appropriate test benches and synthesised before it was committed to silicon. The specific objectives include the determination of requirement specification in terms of the instruction set architecture; determination a priori of the HDL and EDA tool to use; partitioning of the design in line with the chosen HDL; programming the chosen FPGA using appropriate tools.

2. Materials and method 2.1 Theoretical Background

Usually, computer engineers create a model of a design written in hardware description languages (Verilog or VHDL) and test benches which includes a copy of the model or device under verification. The test vectors are read every clock cycle and the output response vectors from the device are compared with a reference model (Seward, 2003; Bergeron, 2003). This is black-box testing approach. This approach does not allow for direct observation and validation. This is a major setback of black-box testing in that the device under verification may exhibit improper internal behaviour, but still have a proper output response at a specific or observed point in time. In this situation a design error exists, but it will definitely be missed because some problem prevents the error from being propagated to an output port and as a result cannot be directly observed on the output ports (Foster, 2004). If other set of test vectors is applied or previous test vectors are run for few clock cycles longer the internal error might be observable. Test benches have evolved over the years to what is known as self-checking test benches, which allows for direct observation and validation. Test benches have become complex verification environments that are regularly built with a hardware verification language (for example Questasim from Mentor Graphics) that combines automatic vector generation, output response validation, and coverage analysis (Foster, 2004; Andrew, 2003). In order to increase the observability of internal points of a device under verification, assertions or monitors are placed close to these critical points so as to catch any violation that might occur. An assertion is an extra line of code that is added to the HDL model to catch any violation of a design's property (intended behaviour). An assertion

statement does not contribute in any way to the functionality of the design; its sole purpose is in ensuring consistency between the designer's intention and what is created. The use of assertions in codes is good example of white-box testing approach as it allows us access to the internal nodes within the design and thus results in an increase in observable behaviour during testing. Emerging hardware verification languages include various forms of assertion library templates. Furthermore, HDLs include constructs that support assertion specification. For example, VHDL [IEEE 1076-1993] (IEEE, 1994) includes a keyword **assert**, which can be used in behavioural modelling. The VHDL assertion syntax is shown in Figure 1.

[label]: assert boolean_expression
[report expression]
[severity expression];
Example
check: process is
begin
assert not (s = '1' and r = '1')
report "Incorrect use of S R flip_flop: s and r both '1";
wait on s, r;
end process check;

Figure 1: VHDL Assertion syntax

An error is reported when the *Boolean_expression* evaluates to FALSE. The assertion's optional report clause specifies a message string that will be included in error messages generated by the assertion (Ashenden, 2001; IEEE, 1994). In the absence of a report clause for a given assertion, the string "Assertion violation" is the default value for the message string.

2.2 Assertion-based transaction model

A model for representing HDL code was developed to aid the diagnosis of functional violation that may be present in a VHDL code (Figure 2). An assertion based transaction graph was used. Assertions embedded in the vertices further increase the diagnosability of the HDL-code.

Figure 2 was further transformed into its equivalent matrix data structure (Table 1) to further simplify the analysis of the code during verification. In order to implement this model and process the matrix after verification three methods were developed by Ngene (2011). These include method of logical analysis of columns, method of logical analysis of rows and matrix method. The vertices (S) in Figure 2 represent the state of the various variables when test patterns are applied and the edges (B) represent the program module with their associated assertions. The state of each program module depends on the states of program module/modules preceding it. For example the module B₁₁ depends on the state S₅ which is the cumulative results of processing modules B₁, B₂ and B₇. The state S₉ is the state at which the final result or results of the program is/are obtained and there are only two modules (B₁₃ and B14) that are finally processed to the yield this state. The number of test sets required to test this program is derived from the Boolean equations Equation 1 of the program modules

and represented in Equation 2. Shown in Equation 1 is sum-of-product representation of Boolean expression, where the product terms of the blocks or modules is logical ANDing and the symbol V is logical ORing operations. Shown in Equation 2 are six (6) products representing six test sets (T_1 to T_6) depicted in Table 1, which is an alternative representation of Figure 1.



Figure 2: Example of Assertion-based transaction graph for an arbitrary code with fourteen modules

Test						Со	de N	1odu	les					
Sets	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈	B ₉	B ₁₀	B ₁₁	B ₁₂	B ₁₃	B ₁₄
T ₁	1		1						1				1	
T ₂	1			1						1				1
T ₃	1	1			1						1		1	
T ₄		1				1				1				1
T ₅		1					1				1		1	
T ₆								1				1		1

Table 1 Code modules activation matrix

$$B = (B_1 B_3 B_9 \vee (B_2 B_7 \vee B_1 B_5) B_{11}) B_{13} \vee ((B_1 B_4 \vee B_2 B_6) B_{10} \vee B_2 B_8 B_{12}) B_{14}$$

$$= B_1 B_3 B_9 B_{13} \vee B_2 B_7 B_{11} B_{13} \vee B_1 B_5 B_{11} B_{13} \vee B_1 B_4 B_{10} B_{14} \vee B_2 B_6 B_{10} B_{14} \vee B_2 B_8 B_{12} B_{14}.$$
(1)

$$T = S_0 S_1 S_3 S_7 S_9 \vee S_0 S_1 S_4 S_8 S_9 \vee S_0 S_1 S_5 S_7 S_9 \vee S_0 S_2 S_4 S_8 S_9 \vee S_0 S_2 S_5 S_7 S_9 \vee S_0 S_2 S_6 S_8 S_9.$$
(2)

The row is the relationship between test segment (T) and a subset of activated segment of program blocks $T_i \approx (B_{i1}, B_{i2},..., B_{ij}, ..., B_{in})$. The column of the table forms the relationship between the program modules and test segments $B_j \approx (T_{1j}, T_{2j},..., T_{ij}, ..., T_{pj})$ that activate it. Otherwise, the column is the assertion vector identifying functional violation within the module. The presence of 1 in a row corresponding to a given code modules means that the test activates the block and a single test can activate more than one program modules. For example T_1 activates B_1 , B_3 , B_9 and B_{13} . The functional violation matrix shown in Table 2 helps to determine which of the blocks has functional violation.

Test Sets						C	ode n	nodule	es						Output Response
	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈	B ₉	B ₁₀	B ₁₁	B ₁₂	B ₁₃	B ₁₄	
T ₁	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0
T ₂	1	0	0	1	0	0	0	0	0	1	0	0	0	1	1
Τ ₃	1	1	0	0	1	0	0	0	0	0	1	0	1	0	0
T ₄	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
T₅	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0
T ₆	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1

Table 2 Functional Violation Matrix

This is achieved by applying all the test sets to the program and observing the output response. The column test sets and Code modules serve as fault table and by comparing the output response vector with each of the columns the module with functional violation can be determined. In Table 2 the output response vector corresponds to column B_{14} . A single module with functional violation is located in this case. There are cases where more than one experiment is required to generate corresponding output responses. This comparison should be carried out for each output response with each column of the code modules. This means that B_{14} has functional violation and further debugging can be concentrated in this module for logical errors rather than blindly searching for the fault in all the modules.

2.3 Method of matrix analysis

Further to the software transaction graph shown in Figure 1, a method for diagnosing functional failures in software modules uses the triad of matrices (B, A and L) of the same dimension. In order to illustrate the method of matrix analysis, Table 2 is restructured into Table 3. Only eight code modules from Table 2 have been considered for simplicity. Also considered are eight assertions (A) corresponding to each code module (B). Here matrices

form: B – block activation on test segments during simulation; A – activity of assertions, corresponding to blocks on test segments during simulation; L – faulty blocks, obtained as result of XOR-operation on the two matrices above is illustrated by Equation 3. XOR-operation is defined by Equation 4. The presence of functional violation in a code can be determined by carrying out coordinate-wise analysis of the matrices B and A. If the resulting matrix coordinates are all zero, then there is no violation (Table 3). Otherwise a functional violation exists in the code. And the next task is to determine the module in which functional violation occurs – diagnosis.

$$\mathbf{M} = \mathbf{B} \oplus \mathbf{A} \oplus \mathbf{L} = 0, \ \mathbf{L} = \mathbf{B} \oplus \mathbf{A} \leftarrow \mathbf{L}_{ij} = \mathbf{B}_{ij} \oplus \mathbf{A}_{ij} \leftarrow \{\mathbf{B}_{ij}, \mathbf{A}_{ij}, \mathbf{L}_{ij}\} = \{0, 1\}$$
(3)

$$\mathbf{B} = [\mathbf{B}_{ij}], \ \mathbf{A} = [\mathbf{A}_{ij}], \ \mathbf{L} = [\mathbf{L}_{ij}], \ \mathbf{i} = \overline{\mathbf{1}, \mathbf{n}}; \ \mathbf{j} = \overline{\mathbf{1}, \mathbf{m}}; \ \mathbf{\oplus} = a\overline{b} \vee \overline{a}b.$$
(4)

										-									-									
Bijj	B1	B ₂	B ₃	B4	B ₅	B ₆	B7	B8		Aij	A ₁	A2	A ₃	A4	A5	A ₆	A7	A ₈		Lijj	L1	L2	L ₃	L4	L ₅	L ₆	L7	L ₈
T ₁	1	0	1	0	0	0	0	0		T_1	1	0	1	0	0	0	0	0		T ₁	0	0	0	0	0	0	0	0
T_2	1	0	0	1	0	0	0	0		T_2	1	0	0	1	0	0	0	0		T_2	0	0	0	0	0	0	0	0
T_3	1	1	0	0	0	0	0	0		T_3	1	1	0	0	0	0	0	0	=	T ₃	0	0	0	0	0	0	0	0
T_4	0	1	0	0	0	1	0	0	\oplus	T_4	0	1	0	0	0	1	0	0		T 4	0	0	0	0	0	0	0	0
T_5	0	1	0	0	0	0	1	0	-	T_5	0	1	0	0	0	0	1	0		T ₅	0	0	0	0	0	0	0	0
T ₆	0	0	0	0	0	0	0	1		T ₆	0	0	0	0	0	0	0	1		T ₆	0	0	0	0	0	0	0	0

 Table 3 Triad Matrices without Functional Violation

In Table 4, it has been shown that the module contains functional violations by examining the L-matrix. In order to unambiguously detect the modules with functional violation, a logical OR operation of individual columns is carried out as shown in the last row of the L-matrix of Table 4. The software modules B_1 and B_3 contain functional violation because of the presence of ones in columns L_1 and L_3 at the last row (i.e. $L = \{B_1, B_3\}$.

Table 4 Triad Matrices with Functional Violation

Bijj	B1	B ₂	B ₃	B4	B ₅	B ₆	B7	B8		Aiii	A ₁	A_2	A ₃	A_4	A_5	A_6	A ₇	A ₈		Lijj	L ₁	L ₂	L ₃	L4	L ₅	L ₆	L7	L ₈
T_1	1	0	1	0	0	0	0	0		T ₁	1	0	1	0	0	0	0	0		T ₁	0	0	0	0	0	0	0	0
T_2	0	0	0	1	0	0	0	0		T ₂	1	0	0	1	0	0	0	0		T ₂	1	0	0	0	0	0	0	0
T_3	0	1	1	0	0	0	0	0	\oplus	T ₂	1	1	0	0	0	0	0	0	=	T ₃	1	0	1	0	0	0	0	0
T_4	1	1	0	0	0	1	0	0		T ₄	0	1	0	0	0	1	0	0		T₄	1	0	0	0	0	0	0	0
T_5	0	1	1	0	0	0	1	0		T ₅	0	1	0	0	0	0	1	0		T ₅	0	0	1	0	0	0	0	0
T_6	0	0	0	0	0	0	0	1		Te	0	0	0	0	0	0	0	1		T ₆	0	0	0	0	0	0	0	0
									-	VAi	1	1	1	1	0	1	1	1		VLi	1	0	1	0	0	0	0	0

2.4 Design methodology

A modular and hierarchical approach was adopted for this design, and the model was captured using VHDL (Chang, 1995; Chang, 1996). The corresponding VHDL model of each module was written in a top-down approach. The device, which was implemented as an IP-core can be used to process not only the synthesised assertions but is also suitable for comparing golden (reference) values and output response values that are represented in the form of matrices. This is appropriate for pattern recognition applications and other applications for searching, recognition and decision making.

2.5 Choice of methods:

The design was based on a tabular data structure with rows containing the test vectors that activate a subset of the code module (columns) and corresponding output response vector (Figure 2 and Table 1). The matrix method discussed in the previous section was used to accomplish the design because it provides the right relationship between the test vectors, code modules, assertions and functional violations and can achieve the same results as the other two methods discussed in Ngene (2011). With the achievement of the desired diagnostic resolution using the proposed methods, logic synthesis scheme and functional coverage analysis and test generation were used to create software and hardware infrastructure for the diagnosis of HDL- code.

2.6 Choice of HDL and EDA tool

As earlier mentioned there are two popular HDLs available (VHDL and Verilog) to use for this design, but VHDL was used in view of its strength in paper documentation. The EDA tool used for the VHDL code was Xilinx ISE 13 design suite web edition. The design was partitioned into 2 – the top module and the execution unit. The execution unit in turn consists of 3 modules – AND, XOR, OR and SLC (Shift-left with 1s compaction). The SLC is used in the circuit for selecting the optimal solution. The solution with more number of 1s in the bit position is a less optimal solution than the solution with lower number of 1s. The worst solution is one that has 1s in all bit positions. An example of SLC operation is shown in Figure 3. The register is used to compact 1s and fill the remaining part of the register with 0s . In view of the simple nature of the processor, which has only 4 instructions, the control unit was incorporated into the register file in the top entity module of the design. In this case a hardwired control system was implemented.



Figure 3: SLC example: (a) Uncompacted. (b) Compacted

2.7 Instruction set architecture

The MAP is a processor that is purely hardware-based; none of the registers is software addressable, consequently it has no program counter or any special purpose registers. It is a 16-bit processor with a simple instruction set. The choice of 16-bit processor was not accidental; it was based on the computational resources available for use at the time of design. The positive side is that a parameterised design approach was used, whereby the number of bits can be changed only in the VHDL package. Increasing the bits should be considered when enough computation resources (high speed multiprocessor computers etc.)

that could speed up the synthesis to a few days depending on the size of the code are available. There are 32 registers (Figure 4) used for holding `two 16x16 matrices that are loaded at the same time from their respective RAMs and intermediate results. All the four instructions are shown in Table 5 alongside their functions and operation codes (opcode).



Figure 4: MAP registers

Table 5MAP Logical Instruction

Instruction	Name	Function	Opcode
XOR	Modular 2 addition	R0-15 ← R0-15 ⊕ R16-32	00
AND	Logical multiplication	R0-15 ← R0-15 ∧ R16-32	01
OR	Logical addition	R0-15 ← R0-15 ∨ R16-32	10
SLC	Shift-left bit crowding (Compaction of 1s)	R0-15 ← SLC R0-15	11

2.8 Data transfers

Loading and transfer of appropriate data to and from the memory is triggered by a special signal gen_rd_wr. Data transfer is not implemented as a specific instruction; rather it is executed once the appropriate value of gen_rd_wr signal is asserted. The gen_rd_wr codes with their corresponding function are shown in Table 6. This signal has 4 states. Two of the states are used for memory operations while the rest are used for data transfers between the register file unit and the Execution unit.

Transfer	Function	Opcode	Gen_rd_wr
From RAM	R0-15 ← (RAM1)	any	001
To Exec_unit	Exec ← (R0-15)	any	010
From Exec_unit	R0-15 ← (Exec)	any	011
To RAM	RAM4 ← (R0-15)	any	100

Table 6 Data Transfer Function and their Codes

2.9 Design Units

The MAP was divided into two major units: the register file and the execution unit. The execution unit is further reduced into sub modules xorandor and SLC. The modules that realises the logical operations XOR, OR and AND were combined in the xorandor sub unit, while that of SLC was implemented in a separate module as depicted in Figure 5. The memory elements present in the FPGA were used to implement the various RAMs. In order to improve the speed of loading and storing data, a separate RAM was implemented for each of the logical operations. The VHDL code for the MAP was not included in this paper because of the size (over one thousand lines of code including test bench).



Figure 5: The structure of the Multi-array processor

2.10 VHDL coding considerations

The VHDL model for the MAP consists of entity declarations and a mixture of synthesisable behavioural and structural descriptions. As mentioned earlier the code was divided into two major units: register file and the execution units. The execution unit where the actual

computation takes place was further divided into sub units; one of the sub units does the XOR, AND and OR computation and the second sub unit executes the SLC operations. The sub units were instantiated as components in the execution unit and the execution unit was further instantiated as a component in the top module entity –register file. However, before the units were written the types used in the external and internal interfaces in a package called map_type_pkg were defined. In order to ensure future enhancements of the code, the code was parameterised to allow for the register and the bus widths to be increased only in one place (i.e. in the MAP type's package).

4. Results and discussion

The results of the actual design of the multi-array processor are first presented by examining the synthesis and the simulation results. Then, the overall improvement brought about by the use of the device in an assertion-based integrated verification environment for HDL models of SoCs was highlighted.

4.1 Synthesis

The processor was synthesised with Xilinx ISE 13.1 design suite and simulated using Xilinx ISIM. Test benches were used to verify the functionality of the design, which was implemented in Virtex 6 FPGA by using Xilinx configuration tools. The device utilisation summary for the targeted Virtex 6 and the top module interfaces are shown in Table 7 and Figure 6.

	multiarray_processor Project	t Status (06/20/2011 - 20:26:55)
Project File:	multi_array_processor.xise	Parser Errors:	No Errors
Module Name:	multiarray_processor	Implementation State:	Synthesized
Target Device:	xc6vlx75t-3ff484	•Errors:	No Errors
Product Version:	ISE 13.1	• Warnings:	76 Warnings (0 new)
Design Goal:	Balanced	 Routing Results: 	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Table 7 Device utilisation summary

Device Utilizat	ion Summary (estima	ited values)		[-]
Logic Utilization	Used	Available	Utilization	
Number of Slice Registers	3232	93120		3%
Number of Slice LUTs	7552	46560		16%
Number of fully used LUT-FF pairs	2261	8523		26%
Number of bonded IOBs	140	240		58%
Number of BUFG/BUFGCTRLs	1	32		3%



Figure 6: Interfaces of MAP

It is evident from the summary that some parts of the device were poorly utilised, especially the number of slice registers and BUFG/BUFGCTRLs. The reason for choosing Virtex 6 was because there was a large number of the bonded input/output blocks whose utilisation was above 50%. The input/output lines were large due to the high level of parallelism of the design, which generally improved the speed of operation of the device at the expense of area or large number of primitives used. The components interconnection implementation is shown in Figure 7. The synthesis report showing various components used and their total number is shown in Figure 8 rather than the actual netlist (logic components and their interconnection).



Figure 7: Components interconnection implementation

HDL Synthesis Report

Macro Statistics	
# RAMs	:1
4x1-bit single-port Read Only RAM	:1
# Adders/Subtractors	: 245
31-bit adder	:3
4-bit adder	:2
4-bit subtractor	:16
6-bit subtractor	: 224
# Registers	:1361
1-bit register	: 1157
16-bit register	: 184
31-bit register	:2
32-bit register	:16
4-bit register	:2
# Comparators	:2
31-bit comparator greater	:2
# Multiplexers	: 5911
1-bit 2-to-1 multiplexer	: 5151
1-bit 4-to-1 multiplexer	: 256
16-bit 16-to-1 multiplexer	:2
16-bit 2-to-1 multiplexer	: 278
5-bit 2-to-1 multiplexer	: 224
# Tristates	: 1154
1-bit tristate buffer	: 1154
# Xors	:16
16-bit xor2	:16

Figure 8: Synthesis report summary

4.2 Verification of results

The test benches used to verify the functionality of the design is not shown in this report because of limited space. The golden matrix values were loaded into RAM2 whereas the input matrix values (i.e. input vectors) to be compared with the golden values were loaded into RAM1. After they have been processed by the execution unit the contents of both RAMs can be overwritten by the results. The simulation in Figure 9 shows that the contents of the RAMs were read simultaneously into corresponding registers in the file at the rising edge of the clock when mem_rd signal is high.

1				515 190 pc							429.900	Ins
Name	Value		100 n		(150 ns	200 ns	(250 ns	(300 ms	(350 ns	400 ns	3	1450 ms
lik cik	1	n n n	П	DODE	nnnnn	nonne	mmmm	nnnn	nonno	DDC	DE	nnnnn
Un reset	0		-									
Les ready	0		1			[23] (23)		(TT) (TT)				
Le gen rd wr	1		-	based to be see	and the design of the second s		1	and the factor from	and had	and have	X	2 X 3
Le mem rd	1		Г							-		
Un mem wr	0		-									
Addatafrom_ra	tett	2222	(efff (abce)	00ac × 1111 × 47	If (fbff) abor)	0000 × 1111 × 6	H X fbff X 15b1	F7HF X IbHF X ab	ce X	2	feff
M datafrom ra	ebfe	2222	-0	Bfff Xba54	e9ec 2222 a9	tb (f6e2) f654	ecec / 2222 / a5	tb x f6e2 x d222	and fee2 16	54)		ebfe
► W instr_code[1	σ		-				U				X	0
allwords_ste	0	-										
in mem_rd_wr	σ	-	_		1							
ready_tostor	σ		-					-		_		5
🕨 駴 datato_ram1	ZZZZ						2222					
▶ 💓 datato_ram2	2222						2222					
▶ 🖬 datato_ram3	2222						2222			_		
► M datato_ram4	2222						2222					
▶ 🖬 read_mem_a	000f	2222	0	0000 0001	0002 0003 00	04 0005 0006	0007 0008 00	009 (000a) 000b	(000c) 000d) 00	0e (000f)		2222
▶ 💘 write_mem_c	2222	- Aniol		0	10000000-00000000000000000000000000000	and the second se	7777	A CONTRACTOR OF A CONTRACTOR O	A SERVICE AND A POLICE OF		-	
1 clk_period	10000						10000 ps					
V M reg_1[0:31]	lefff	(0000,0000	.0	[efff)[efff	(efff.) (efff.) (e	efff)([efff)([efff	(efff.)(efff.)	efff. (efff.) [efff	(fefff.)(fefff.)(f	efff)([efff,	abce,00	ac, 1111, F7H, fbH, a .
► ** 101	offf	0000	-	1				efff			-	
			_	*							-	
											429.90	0 ns
				115.190 ms	222200	2120.0					429.90	0 ns
Name	Value		1200 1	115.190 rs	150 ns	200 ns	250 ns	(300 ns	350 ms	1400 ms	429.90	450 ns
Name 11)	Value 1551		100 r	113.190 ms	150 ns	200 ns	250 ns	1300 ns	350 ns	1400 ns	429.90	1450 ns
Name	Value 1551 1711		200 /	[115.130 rs]	150 ns	200 ns	1250 ns	300 ns	1350 ns	400 ns 1501 f7	429.90	450 ns
Name ▶ 11 [11] ▶ 11 [12] ▶ 11 [13]	Value 1551 17ff fbff		1200 r	[115.150 ms]	150 m	200 ms 000 0000 0000	1250 ns	300 ms	350 ns	1900 ms 1501 f7	429.90 F	1450 ns
Name	Value 1551 f7ff fbff abce		200 /	[115.150 ms]	150 ns	200 ns 000 0000 0000 0000	(250 ns	1300 ms	1350 ns	1900 ns 1501 17	429.90 H fb# at	450 ns
Name * * [11] * * [12] * * [12] * * [12] * * * * * * * * * * * * * * * * *	Value 1551 foff abce feff		1200 1	113.130 ns	150 mi	200 ms 000 0000 0000 0000 0000 0000	1250 ns	300 ns	350 m	400 ns 1501 f7	429.90 H 6.H ab	0 ms) 450 ms
Name ▶ 10 [11] ▶ 10 [12] ▶ 10 [13] ▶ 10 [15] ▶ 10 [15]	Value 1551 foff abce feff off	0000	(200 r	[13.130 ms]	150 ms	200 ms 000 0000 0000 0000 0000	1250 ns	300 ms	1350 ms	900 ns 1501 17	429.90 H 65H ab	1450 ns
Name ▶ 111 ▶ 112 ▶ 112 ■	Value 15b1 foff abce feff 0fff ba54	0000	200 /	113.130 rs	(150 ms 0	200 ms 000 0000 0000 0000 0000	1250 ns	300 ns X	350 ns	400 ns 1501 17	429.90 1 16.11 ab	0 ms 450 ms feff
Name ▶ 12 [11] ▶ 12 [12] ▶ 12 [15] ▶ 12 [15] ▶ 12 [15] ▶ 12 [15] ▶ 12 [15]	Value 15b1 foff abce feff ba54 e9ec	0000	200 r	115.190 ms)	(150 ms 0	200 ns 000 0000 0000 0000 0000	1250 ns	300 ns X	350 ns	1501 1501 77	429.90 4 654 at	1450 ns
Name ▶ 102 [11] ▶ 102 [12] ▶ 102 [13] ▶ 102 [14] ▶ 102 [15] ▶ 102 [15] ▶ 102 [15] ▶ 102 [15] ▶ 102 [15]	Value 15b1 foff abce feff 0fff ba54 e9ec 2222	0000	100 r	115. 190 ms)	150 ms	200 ns 000 0000 0000 0000 0000	1250 ms	300 ns X	1350 ns	1900 ms 1501 17	429.90 H 65H at	1450 ns 450 ns feff feff
Name	Value 15b1 f7ff abce feff ba54 e9ec 2222 a3cb	0000	200 r	115.199 ms) 	x	200 ns 000 0000 0000 0000 0000	1250 ms	Stff ba54 eSec 2222 a5	1350 ms	1400 ns 15901 75	429.90	1450 ns 450 ns feff
Name	Value 15b1 f7ff abce feff ba54 e9ec 2222 a3cb f6e2	0000	(200 r 000	115.199 ms 	150 ms o	200 ns 000 0000 0000 0000 0000	1250 ms	300 ns X	1350 ms	1400 ns 15901 75	429.90 H 6/H ab	1450 ns 450 ns feff
Name → 10 [11] → 10 [12] → 10 [13] → 10 [14] → 10 [15] → 10	Value 15b1 f7ff fbff abce feff ba54 e9ec 2222 a3cb f6e2 f6e2 f654	0000	000	115.299 ms 1. 1. 1 00000 0000 000000	150 ms	200 ns 000 0000 0000 0000 0000	1250 ms	300 ns X stff ba54 e9ec 2222 at	(350 ns X X X X X X X X X X X X X X X X X X X	1400 ms 1501 77	429.90 F 654 at	450 ns
Name ▶ 100 [11] ▶ 100 [11] ▶ 100 [11] ▶ 100 [11] ▶ 100 [11] ▶ 100 [10] ▶ 100 [10]	Value 15b1 f7ff fbff abce feff ba54 e9ec 2222 a3cb f6e2 f654 ecec	0000	000 000	115.199 rs X X X X X X X X X X X X X	150 ms	200 ns 000 0000 0000 0000 0000 0000	250 ms	300 ns X stff ba54 e9ec 2222 at	350 ms	1400 ms 1501 17	429.90 F 6.47 at	1450 ns 450 ns feff
Name ▶ 10 [11] ▶ 10 [12] ▶ 10 [12] 10 [12] 10 [12] 10 [12] \hline 10	Value 15b1 f7ff fbff abce feff 0fff ba54 e9ec 2222 a9cb f6e2 f654 ecec 2222	0000	200 r	115.199 rs 	150 ms	200 ns 0000 0000 0000 0000 0000	250 ns	300 ns X stff ba54 e9ec 2222 at	350 ns X X X X X X X X X X X X X X X X X X X	1400 ms 1591 77	429.90 Fr fb/ff ab	1450 ns
Name	Value 15b1 f7ff fbff abce feff ba54 a9ec 2222 a9cb f6e2 f654 ecsc 2222 a9cb	0000	200 r	115.1997ms) 9 9 9 9	150 ms 0 0 0 0 0 0 0 0 0000 0000 0000	200 ns 0000 0000 0000 0000 0000	250 ns	300 ns X	350 ns X X X X X X X X X X X X X X X X X X X	1400 ms 1591 77 X	423.900	1450 ns
Name	Value 15b1 f7ff fbff abce zeff ba54 e9ec 2222 a9cb f6e2 z222 a9cb f6e2 cesc 2222 a9cb	0000	(200 r	115.1997rs) . , , . , , , . , , , . , , , , . , , , , , , , , , , , , , , , , , , ,	150 ms 0	200 ns 0000 0000 0000 0000 0000	250 ms	300 ns X	350 ns X X X X X X X X X X X X X X X X X X X	1400 ns 1501 1501 177 77 77 77 77 77 77 77 77 77 77 77 7	423.90 Y 6.H at	1450 ns
Name	Value 1551 f7ff fbff abce 2222 a9cb f6e2 2222 a9cb f6e2 2222 a9cb f6e2 d222	0000	(100 r 000	115.1997rs) . , , , . , , , . , , , . , , , . , , , , . , , , , , , , , , , , , , , , , , , ,	150 ms 0 X X 0000 0000 0000 0000 0000 0000 0000	200 ns 000 0000 0000 0000 0000 0000	250 ms	300 ns X	350 ms	1400 ns 1591 77 X	423.90 Y 6.Ff at	1450 ns
Name ▶ 12 ▶ 12 ▶ 12 ▶ 12 ▶ 12 ▶ 12 ▶ 12 ▶ 12 ■ 12	Value 15b1 fpff abce feff ba54 e9ec 2222 45cb 56e2 45e5 eeec 2222 a3cb 56e2 4222 a3cb	0000	1200 r	115.1997ms) . , , . , , , . , , , . , , , , . , , , , , , , , , , , , , , , , , , ,	150 ms o X X 0000 0000 0000 0000 0000 0000	200 ns 000 0000 0000 0000 0000 0000 X X X	250 ns	300 ns X	1350 ms X X X X X X X X X X X X X	1900 ns 1591 // // // // // // // // // // // /// ////	423.90 1 6.4 6.4 8 8	1450 ns
Name ▲ ■ <	Value 1551 1551 157ff abce 16ff 2222 48cb 1662 2654 ecec 2222 249cb 1662 2222 249cb 1662 2222 249cb 1662 1662	0000	200 /	115.1997ms 4	150 ms O	200 ns 000 0000 0000 0000 0000 0000 X X X	250 ns	300 ns X stff ba54 e9ec 2222 as	1350 ms	400 ns 1551 77 X 1662 4222 a9	423.90 F fo:ff at fo:ff fo:f2	450 ns
Name ▲ ■ <	Value 15b1 15b1 15b7	0000	200 /	115. 1997 rs 	150 ms	200 ns 0000 0000 0000 0000 0000 0000 0000	250 ns	200 ns X	350 ms	1400 ms 1591 77 X X 1592 80 1562 80 1562 80 1562 80 1562 80 1562 80 1562 80 1562 80 1562 80 1562 80 1562 80 1563 1563 1563 1563 1563 1563 1563 1563	423.90 - - - - - - - - - - - - -	450 ns

Figure 9: Input vectors verification waveforms

After the memory contents have been read they are transferred to the execution unit in parallel and concurrently executed depending on the logical instruction to be executed. The result of XOR operation is depicted in Figure 10 and that of SLC operation is shown in Figure 11. The simulation results for AND and OR operations were similarly carried out and were deliberately omitted in this paper.

3			465.0	190 pc			2	510	515.0	00 ns									
Name	Value	146	0 05	750 Haj	1480 ns		1500 ns	510.	1	520 ns	1540 ns		1560 n	s	-	580 ns		1600	ns
Name 10. au	value				100 113		500113					4		<u> </u>			444		
10 mont	1							10				1					1		
10 reset	0	-								-		-	_					_	F
	1	=	2		<u> </u>	2			_		-			4	-			-	
ug gen_ra_wr	4	-	2		^	3		-	-				4		-			65	
	0		-					-	-		-		-		-			-	_
ue mem_wr	1		_		-				-	6.6									_
	feff	_	-					-	-	теп	-		-		-			-	
datafrom_ram2[15:0]	ebfe	_							_	ebte			_					-	
instr_code[1:0]	0	_	-		-			4.15	<u>.</u>	0			-		-			_	
allwords_stored	1																		
u mem_rd_wr	ū				-										1				_
Le ready_tostore	1																		_
b 🛃 datato_ram1[15:0]	6000	_	_		ZZZZ				600	0) ZZZZ (11	9a) Z	ZZZ	e940)	ZZZZ	333	33_)	ZZZZ	(5eB4) ZZZZ
datato_ram2[15:0]	ZZZZ	_							_	ZZZZ									
datato_ram3[15:0]	ZZZZ								-	ZZZZ									
datato_ram4[15:0]	ZZZZ									ZZZZ									
read_mem_addr[15:0]	ZZZZ									ZZZZ									
Write_mem_addr[15:0]	0000	_			ZZZZ				00	10 X 00	D1	X	0002		00)3		0004	X
🔓 clk_period	10000					_			a	10000 ps		-						6	-
🔻 🌃 xor_dataout[0:15]	[6000	[UU				[6000,	119a,e940	,333	8,5e34	4,0d1d,5d9a,ecec,3	333,5e34	1,0d1d,c7	93,5e34	,0d1d,5	d9a,150	01]		-	_
🕨 🎽 [0]	01100	υψ								011000000	0000000								
-			-					-	515.0	00 ns									
			465.0)90 ns			4	510.	515.0)90 ns	00 ns				1.12				777	
Name	Value	46	465.0 0 ns)90 ns	480 ns	<u></u>	500 ns	510.	515.0)90 ns	00 ns	540 ns	ببين	560 n	s LLLL		580 ns	ىرلىت	600) <mark>ns</mark>
Name Witte_mem_addr[15:0]	Value 0000	46	465.0 0 ns	090 ns	480 ns ZZZZ	1	500 ns	510.	515.0)90 ns	00 ns 520 ns 10 X 00	540 ns	<u></u>	560 n	s X	000	580 ns)3	<u></u>	0004	ons X
Name Mame Market Mark	Value 0000 10000	46	465.0 0 ns	090 ns	480 ns ZZZZ	L	500 ns	510.	515.0)90 ns 000	00 ns 520 ns 0 X 00 10000 ps	540 ns	<u></u>	560 n	s 	00	580 ns)3	<u>.</u>	0004) ns X
Name Name write_mem_addr[15:0] ck_period ataaout[0:15] 	Value 0000 10000 [6000	46 	465.0 0 ns	090 ns	480 ns ZZZZ	[6000,	500 ns	510.	515.0)90 ns 1 000 3,5e3	00 ns 520 ns 100 X 00 10000 ps 4,0d1d,5d9a,ecec,3	540 ns 01 333,5e34	 X i,0d1d,c7	560 n 0002 93,5e34	s 	000 d9a,150	580 ns)3)1]	<u>, 1</u> , .	0004	0 ns X
Name ▶ ∰ write_mem_addr[15:0] ↓ clk_period ▼ ∰ xor_dataout[0:15] ▶ ∰ [0]	Value 0000 10000 [6000 01100		465.0)90 ns	480 ns ZZZZ	[6000,	500 ns 119a,e94(510.	515.0)90 ns 000 3,5e3	00 ns 520 ns 100 X 00 10000 ps 4,001d,5d9a,ecec,3 011000000	540 ns 01 333,5e34 0000000	X 1,0d1d,c7	560 n 0002 93,5e34	s X ,0d1d,50	000 d9a,150	580 ns)3)1]	<u>.</u>	0004) ns X
Name ■ write_mem_addr[15:0] 1: clk_period ▼ xor_dataout[0:15] ■ [0] ■ [1]	Value 0000 10000 [6000 01100 00010	146 1 1 [UU UU	465.0)90 ns	480 ns	[6000,	500 ns 119a,e940	510.	515.0)90 ns 000 3,5e3	00 ns 520 ns 100 X 00 10000 ps 4,0d1d,5d9a,ecec,3 011000000 000100011	540 ns 01 333,5e34 0000000 0011010	1,0d1d,c7	560 n 0002 93,5e34	s X ,0d1d,50	000 d9a,150	580 ns)3)1]	<u>.</u>	0004	0 ns X
Name ■	Value 0000 10000 [6000 01100 00010 11101		465.0 0 ris)90 ns	480 ns	[6000,	500 ns	510.	515.0)90 ns 000 3,5e3	00 ns 520 ns 10000 ps 4,0d1d,5d9a,ecec,3 011000000 0001000111 111010010	540 ns 01 333,5e34 000000 0011010 1000000	1,0d1d,c7	93,5e34	s	000 d9a,150	580 ns 03	<u>'</u>	00004) ns X
Name ▼ write_mem_addr[15:0] ↓ clk_period ▼ % ∞ or_dataout[0:15] ▶ % 0 11 ▶ % [2] %	Value 0000 10000 (6000 01100 00010 11101 00110		465.0 0 ris)90 ns	480 ns 2ZZZ	[6000,	500 ns	,333	515.0)90 ns 000 3,5e3	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 0110000001 0001000110 001100100 00110011	540 ns 01 333,5e34 0000000 0011010 1000000 0110011	1,0d1d,c7	560 n 00D2 93,5e34	s 	000 d9a,150	580 ns 03	<u>'</u>	0004) ns X
Name ■	Value 0000 10000 (6000 01100 00010 11101 00110 01011		465.0 0 rs)90 ns	480 ns	[6000,	500 ns	,333	515.0)90 ns 000 3,5e3	00 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 0110000011 111010010 001100110 011111000	540 ns 01 3333,5e34 0000000 0011010 1000000 0110011 0110100	1,0d1d,c7	560 n 0002 93,5e34	s X _,0d1d,5d	000 d9a,150	580 ns)3)1]	<u>'</u> X''	6004) ns X
Name ■ write_mem_addr[15:0] ↓ clk_period ▼ xor_dataout[0:15] ● cli ↓ cli	Value 0000 10000 (6000 01100 00010 11101 00110 01011 00001		465.0 0 rs)90 ns	480 ns	[6000,	500 ns	,333	515.0)90 ns 000 3,5e3	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,ecec,3 011000000 0001000111 111010010 00110011	540 ns 01 333,5e34 0000000 0011010 0110011 0110100 0011101	1,0d1d,c7	93,5e34	s ,0d1d,5	000 d9a,150	580 ns 03	·· <u>/</u> ···	600) ns X
Name ▶ write_mem_addr[15:0] ↓ clk_period ▼ xor_dataout[0:15] ▶ 0 ↓ 0	Value 0000 10000 (6000 01100 00010 11101 00110 01011 00001 01011)90 ns	1480 ns	[6000,	500 ns	,333	515.0)90 ns 000 3,5e3	00 ns 520 ns 100 X 000 10000 ps 4,0d1d,5d9a,ecec,3 0110000001 0001000111 111010010 00110011	540 ns 01 3333,5e32 0000000 0011010 0100011 0110100 0011101 0011010	\$,0d1d,c7	93,5e34	s ,0d1d,5	000 d9a,150	580 ns)3	<u>.</u>) ns X X
Name ▶ write_mem_addr[15:0] ↓ clk_period ▼ xor_dataout[0:15] ▶ 0 ↓ 0 <t< th=""><th>Value 0000 10000 (6000 01100 01110 01011 00011 01011 11101</th><th></th><th></th><th>)90 ns]</th><th>1480 ns</th><th>[6000,</th><th>119a,e94</th><th>,333</th><th>515.00 ns</th><th>00 ns 520 ns 10 × 0000 ps 4,001d,5d9a,ecec,3 0110000000 0001000110 111010010 00110011</th><th>540 ns 01 333,5e34 0000000 011010 010000 0110011 0110100 011101 101100</th><th>1,, X,0d1d,c7</th><th>560 n 0002 93,5e34</th><th>s ,0d1d,5</th><th>002 09a,150</th><th>580 ns)3</th><th><u> </u></th><th></th><th>) ns X X</th></t<>	Value 0000 10000 (6000 01100 01110 01011 00011 01011 11101)90 ns]	1480 ns	[6000,	119a,e94	,333	515.00 ns	00 ns 520 ns 10 × 0000 ps 4,001d,5d9a,ecec,3 0110000000 0001000110 111010010 00110011	540 ns 01 333,5e34 0000000 011010 010000 0110011 0110100 011101 101100	1,, X,0d1d,c7	560 n 0002 93,5e34	s ,0d1d,5	002 09a,150	580 ns)3	<u> </u>) ns X X
Name write_mem_addr[15:0] clk_period cl	Value 0000 10000 (6000 01100 01110 01011 01011 11101 00110)90 ns]	1480 ns	[6000,	500 ns	,333	515.00	00 ns 520 ns 520 ns 10000 ps 4,001d,5d9a,ecec,3 011000000 0001000110 001100110 0101111000 010111010 010111011	540 ns 01 333,5e34 0000000 011010 1000000 0110011 0110100 0110011 0110011	1,, X,, 3,0d1d,c7	560 n 0002 93,5e34	s ,0d1d,5c	000 d9a,150	580 ns 13 33 01]	<u> </u>) ns X
Name Mame write_mem_addr[15:0] ck_period ck_period (0] cd [0] dd [0] dd [1] dd [2] dd [3] dd [4] dd [5] dd [5] dd [6] dd [7] dd [8] dd [9]	Value 0000 10000 (6000 01100 01110 01011 01011 11101 00110 01011)90 ns		[6000,	500 ns	,333	515.00 000 3,5E3	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,ecc,3 0110000011 010100100 000100110 010110100 010111100 010110110 010110110 010111001 010111100	540 ns 01 333,5e3 0000000 011010 010011 011010 011101 011010 011001 110110	1, , , , , , , , , , , , , , , , , , ,	560 n 0002 93,5e34	s,0d1d,5	000 d9a,150	580 ns	·) ns X
Name Mame Market Market Market Market Market Ma	Value 0000 10000 (6000 01100 01110 01011 01011 01011 11101 00110 01011 00011)90 ns	480 ns	[6000,	500 ns	,333	515.00 ns	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 011000000 0001000110 111010010 00110011	540 ns 01 333,553 0000000 011010 011001 011010 011101 011010 011001 011001 011011	\$,0d1d,c7	560 n 0002 93,5e34	s,0d1d,5	000 d9a,150	580 ns) ns X
Name Mame write_mem_addr[15:0] ck_period v dataout[0:15] data	Value 0000 10000 (6000 01100 01110 01011 01011 11101 00111 00111 01011 11101 01011			90 ns	480 ns	[6000,	500 ns	,333	515.0 ns	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 0110000001 0001000110 001100100 000110011	540 ns 01 333,5e34 0000000 011010 0011011 0110100 0110101 1101100 0111010 0111010 0011101 0010011	1, X 4,0d1d,c7	560 n 0002 93,5e34	s ,0d1d,5	002 (19a, 150	580 ns) ns X
Name Mame clk_period clk_period clk_period clk_period clk_period clk_period clk_period cli clk_period cli cli cli cli cli cli cli cli	Value 0000 10000 01100 00100 11101 00101 01011 11101 00101 01011 11000 01011			90 ns		[6000,	119a,e940		515.00 ns	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 0110000001 0001000110 000100110 0011011100 0000110110 010111100 00100110 010111100 0000110101 010111100 0000110101 0100111100	540 ns p1 333,5e34 000000 001101 001101 001101 001101 1101100 0110111 0110101 0011101 0011101 0011101 0010011 010011	1, 1 1 1 X 1, 0d 1d, c7	560 n 200002	S ,0d1d,5c	002 d9a,150	580 ns			Dins
Name Mame Cik_period Cik_pe	Value 0000 10000 01100 00100 11101 00111 00011 01011 01011 00011 00011)90 ns		[6000,	500 ns		515.00 ns	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 0110000001 0001000110 001001100 0001101100 010111100 01011110010 0100111100 0000110101 0100111100 0000110100 0000110100	540 ns 51 333,5e3 0000000 011010 011001 011010 011010 011001 011000 011010 0011011 0011011 0110100 0011011	1,, X,0d1d,c7	560 n 93,5e34	,0011d,5c	00 d9a,150	580 ns 33 01]	<u>X</u>		
Name ↓ write_mem_addr[15:0] ↓ clk_period	Value 0000 10000 (6000 01100 0110 01010 01011 01011 00110 01011 11000 01011 00001 11000					[6000,	500 ns		515.00 ns	00 ns 520 ns 520 ns 10000 ps 4,0d1d,5d9a,eccc,3 011000000 0001000111 111010010 00110011	540 ns 1 333,5634 000000 011010 010011 101000 011010 0001110 0001110 0001110 0001110 0001110 0001100 0001100 0001100 0001100 0000 000000	1,, X,0d1d,c7	560 n 93,5e34	,0d1d,5c	00 d9a,150	580 ns 33 01]	X		
Name Mame write_mem_addr[15:0] clk_period clk_peri	Value 0000 10000 (6000 01100 0110 01011 00011 01011 01011 00011 11000 01011 00011 00011 01011 00010			990 ns		[6000,	500 ns		515.0 ns	00 ns 520 ns 10000 ps 4,0d1d,5d9a,ecc,3 011000000 0001000111 01110010 000110011	540 ns 333,5=34 000000 011010 011001 011010 011010 011010 011010 011010 011010 011010 011010 011010 011010 0011010 0000001	1,	560 n	,001d,50		580 ns			
Name Mame write_mem_addr[15:0] clt_period clt_peri	Value 0000 10000 (6000 01100 0110 01011 00011 11101 00011 11000 01011 01011 01011 01011 01011			990 ns		[6000,	500 ns	510. ,333 ,333	515.0 ns	00 ns 520 ns 10000 ps 10000 ps 4,0d1d,5d9a,ecc,3 011000000 0001000111 111010010 00110011	540 ns 333,5=34 000000 011010 010001 101000 011001 101000 011001 011010 011010 011010 011010 011010 011010 011010 011010 011010 011010 011010 011010 011010 011010 011000 011001 011000 00000 00000 011000 000000	1,, 4,0d1d,c7		s ,0d1d,5x		580 ns			

Figure 10: XOR operation verification waveforms

	<u> </u>	445 000 eel			510.0	515.	200 ms							
Name	Value	AD 2 000 THE	1480 ms	1500 ms		14.0-1	520 ms	1540 ms	(560 ms	(580	ns .	1600 m		62
Lie seady	1	and the second	in alun					in the second		-	i luu		ulu	
1 gen_rd_wr	100	30	11						100					-
12 men_rd	0	1 10 1	20				-		1. 20247					
Le men wr	1					-								14
> M datafrom ramifis	tett				-		1	feff		-		-		-
► M datafrom ram2015	ebfe	1				-		ebfe		-		-		-
▶ M instr_code[1:0]	11	-		-				11	-	-		-		-
12 allwords_stored	1								2			-		-
1 men rd wr	U	1		-		-				_		_		_
14 ready tostore	1	1		-	-	199						-		-
> M datato ramifi50	22222	1.					2222	mmm	-					-
► M datato ram2115:01	22222				-		7277	mmm						
► M datato ram3/15/01	22222	-					1111							-
► M datato ram4/15:01	11111		mmmmm			111	1 72222 (111	11 22222 (11	10 ZZZZZ (1	1110	22222 (1)	112	mn (i	111
> W read mem addrils	2772					-		7777			Constraint Constraint	1		-
b W write mem addelt	0000		7777			00	00 / 00	01 0	007 V	0003	V (004	V 000	15
10 effe marind	10000	-					· · · · ·	10000.00		4444				-
N M vor dataout@151	10000	[7777777777777 7				777		7777 7777777777777777777777777777777777	*****	,,,,,,,	· ····	,,,,,,,	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	
b M or detaod (0.15)	12000	(1)111111111	*****	11111 111		777	****	****	*****		* ********		77777777777	
b M and dataset (0.15)	Lecar	(111111111111	****	*****		777	****	****	*****	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	* *********		*******	
and_dataoutju:15j	12662	(and and a second se		1000	a	Lill.	long the the H.C.	and have the the	Loo W. W. H.	and a	c,eccentere	ciccic,	mm	666 m
A .E ric osteontinital	TILLE	N.1		11116	C11107	1000	truco, me, me, mou,	Contraction of the second s	recu, me, me, me	, marger				
	_	No.		-	Property lies	100	The local distance in	A REAL PROPERTY AND	A providence of the party of the					100
		465.000 ms		1	510.0	515.	200 ns		Alexandra and			-		400
Name	Value	465.000 ms	1480 ns	1500 ns	510.0	515.	200 ms 200 ms 520 ms	1540 ns	(560 ns	1580	ns ,	600 m		62
Name e dk.period	Value	465.000 ms	480 ns	500 ns	510.0	515.	200 ns 200 ns 520 ns	540 ns	560 ns	1580	ىبىيلىت	600 m	سىن	1 22-
Names Le dk_period	Value 10000	465.000 ms	480 ns	500 ns	510.0	515.	200 ns	540 ns	560 ns	, 580	ns 1	600 m		62
Name k dk_period Marcataout[015] Marcataout[015]	Value 10000 [2222	465.000 mg 4.4.1.4.4.4.4.4 (2222222222222222222222222222	480 ms	500 ms	510.0	515. 515. 1.1. 2221	200 ms 520 ms 520 ms 7777 27777777777777	540 ns 10000 ps 2272,22722222222222222	560 ns	, 580 2222222	15 	600 m		621
Name k dk_period M arc_dataout[0:15] G or_dataout[0:15] M and dataout[0:15]	Value 10000 [2222 [2222 [2222	465.000 mg 	480 ns	500 ns 12272,222 12272,222		515. -15 	200 ms 520 ms 7222,222222222222222222222222222222222	540 ns 1	560 ms	, 580 27.222 22.222 22.222	ns 2,22222222222 2,22222222222 2,22222222	600 m 222222, 222222,		
Name k dk_period Marc_dataout[015] Gr_dataout[015] Gr_dataout[015] Gr_dataout[015] Gr_dataout[015]	Value 10000 [2222 [2222 [2222 [2222	(222222222222 (22222222222222222222222	480 ms	1500 rs 122222,222 122222,222 122222,222 12222,222	510 1111 1111 1111 1111 1111	515. 515 1.1 2221 2222 2222 6000	200 ms 200 ms 520 ms 1272, 22727777777 1272, 2777777777777777777777777777777777	540 ns	1560 ms	, 580 121222 121222 121222 121222	ns 2,2222222222 2,2222222222 2,2222222222	600 m 222222, 222222, 222222,		
Name k dk_period Marc_dataout[015] marc_dataout[015] marc_dataout[015] marc_dataout[015] marc_dataout[015] marc_dataout[015] marc_dataout[015]	Value 10000 [2222 [2222 [2222 [fff= 1111]	465.000 re) 1 1 1 1 1 1 1 1 1 [22222222222 [2222222222	480 ns	1500 ms 12222,222 12222,222 12222,222 12222,222	510 111 1111 1111 1111 1111 1111 1111	515. -15 -2221 2221 2221 7000	200 ms 200 ms 520 ms 1272, 22727777777 1272, 227277777777 1272, 2272777777777777777777777777777777	540 ns 10000 ps 2222,2222222222 2222,2222222222 2222,222222	1560 ns 22222,222222222 22222,222222222 22222,222222	, 580 222222 222222 222222 222222 222222	1,2222222222 2,2222222222 2,2222222222	600 ne 222222, 222222, 222222,		62 1 121 121
Name Le dk_period ► dk_or_dataout[0.15] ► dot_dataout[0.15] ► dataout[0.15] ► dataout[0.15] ► dataout[0.15] ► dataout[0.15] ► dataout[0.15] ► dataout[0.15] ► dataout[0.15]	Value 10000 [2222 [2222 [2222 [fff= 11111 11111	465.000 rs 	1430 ms 17772,727777777777777777777777777777777	500 ms 12272,222 12222,222 12222,222 [fffe	510 1111 1111 1111 1111 1111 1111	515. -15 -15 	200 rs 520 rs 520 rs 7272,222222222222 7222,22222222222 7222,22222222	540 ns 1 10000 ps 2222,2222222222 2222,22222222222 0000,5000,fffe,fffe, 1111111110	560 ns 22222,222222222 22222,222222222 22222,222222	, 580 222222 222222 222222 222222 222222 2222	1,2222222222 2,222222222 2,2222222222	600 re 222222, 222222,		62
Name Le dk_period Le dk_period ■ wor_dataout[0.15] ■ or_dataout[0.15] ■ ist_dataout[0.15] ■ <tr< td=""><td>Value 10000 [2222 [2222 [fffe 11111 11111</td><td></td><td>1430 ms 1777, 22772727777 1777, 22777777777 1777, 227777777777 1777, 227777777777777</td><td>500 ns 22272,222 22272,222 22272,222 [#ffe</td><td>510.0 1111 1111 1111 1111 1111 1111 1111</td><td>515. 15 12 12 12 12 12 12 12 12 12 12 12 12 12</td><td>200 ns 520 ns 520 ns 7222,22222222222 7222,22222222222 7222,22222222</td><td>540 ns 1 10000 ps 2222,2222222222 2222,22222222222 000,500,ffe,ffe, 111111110 11111000000 0000000000000</td><td>560 ns 27272, 272727277 2722, 272727777 27272, 272727777 77272, 272727777 f=00.fffe,4ffe,4ffc</td><td>, 580, 222222, 222222, 222222, ,#fe]</td><td>ns , 2222222222 , 2222222222 , 2222222222</td><td>600 ns 222222, 222222, 222222,</td><td></td><td>62</td></tr<>	Value 10000 [2222 [2222 [fffe 11111 11111		1430 ms 1777, 22772727777 1777, 22777777777 1777, 227777777777 1777, 227777777777777	500 ns 22272,222 22272,222 22272,222 [#ffe	510.0 1111 1111 1111 1111 1111 1111 1111	515. 15 12 12 12 12 12 12 12 12 12 12 12 12 12	200 ns 520 ns 520 ns 7222,22222222222 7222,22222222222 7222,22222222	540 ns 1 10000 ps 2222,2222222222 2222,22222222222 000,500,ffe,ffe, 111111110 11111000000 0000000000000	560 ns 27272, 272727277 2722, 272727777 27272, 272727777 77272, 272727777 f=00.fffe,4ffe,4ffc	, 580, 222222, 222222, 222222, ,#fe]	ns , 2222222222 , 2222222222 , 2222222222	600 ns 222222, 222222, 222222,		62
Name Le dk_period Le dk_period Max or_dataout[0.15] Max or_dataout[0.15] Max or_dataout[0.15]	Value 10000 [2222 [2222 [2222 [fff= 11111 11111 11111 11110	200228	480 ns 1777, 77777777777 1777, 77777777777 1777, 7777777777	1500 ns 22222,222 22222,222 22222,222 [fffic	510.4 7777 7777 7777 7777 7777 7777 7777	515. 11 11 11 11 11 11 11 11 11 11 11 11 11	200 ns 520 ns 520 ns 2222,22222222222 2222,22222222222 2222,222222	540 ns 1	560 ns 17772, 27772727 1772, 17772727 1772, 17772727 1772, 17772727 1500, fffe, 4the, ffc0	, [580] 222222 222222 222222 ,#fe]	ns , 2222222222 , 2222222222 , 2222222222	1500 ns 2722222, 2722222, 2722222,		62 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Nume ic dk_period ▶ # xor_dataout[0:15] > ▶ # or_dataout[0:15] > ▶ # sk_dataout[0:15] > > # sk_dataout[0:15] > > # sk_dataout[0:15] >	Value 10000 [2222 [2222 [fff= 11111 11111 11110 11110		480 ns 1111, 1111, 1111, 111 1111, 1111, 1111, 1111 1111, 11111, 1111, 1111, 1	1500 ns 12222,222 12222,222 12222,222 [fffic	510 111 1111 1111 1111 1111	515. 111 7727 7727 7727 7727	200 ms 520 ms 520 ms 2222,22222222222 2222,22222222222 2222,222222	540 ns 10000 ps 2222,22222222222 2222,22222222222 0000,5000,fffe,fffe, 111111110 11111000000 00000000000 00000000	560 ns 17772, 27772727 1772, 77772727 17727, 77772727 17727, 77772727 1500, HTe, HTe, HSo	, [580, 2222222 2222222 2222222 ,##e]	1,2222222222 2,2222222222 2,2222222222	1500 m 2722222, 2722222, 2722222,		22 1 22 1 23 1 2 2 3 1 1 2 3 1 2 3 1 1 2 3 1 1 2 3 1 1 2 3 1 1 1 1
Name is dk_period is dk_period is xor_dataout[0:15] is or_dataout[0:15] is x_dataout[0:15] is and_dataout[0:15] is x_dataout[0:15] is and_dataout[0:15] is x_dataout[0:15] is x_dataout[0:15] is x_dataout[0:15] is x_dataout[0:16] is x_dataout[0:16] is x_dataout[0:16] is x_dataout[0:16] is x_dataout[0:16] is x_dataout[0:16]	Value 10000 [2222 [2222 [ffe 11111 11111 11110 11110 11111		480 ns 77772,22722722222 72227,22722222222 72227,22722222222	500 ns 22222,222 22222,222 22222,222 [fffe	510.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1	515. 1.1. 7777 7777 7000	200 ms 520 ms 520 ms 520 ms 7272,727777777777777777777777777777777	540 ns 1000 ps 2222,22222222222 2222,22222222222 000,500,fffe,fffe, 111111110 11111000000 0000000000 00000000	560 ns 22222,222222222 22222,222222222 22222,222222	, 580 222222 2222222 2222222 2222222 2222222	ns , , , , , , , , , , , , , , , , , , ,	600 m		
Name ic dk_period ▶ If xor_dataout[0:15] ▶ If or_dataout[0:15] ▶ If and_dataout[0:15] ▶ If and_dataout[0:16] ▶ If and_dataout[0:16] ▶ If and_dataout[0:16] ▶ If and_dataout[0:16]	Value 10000 [2222 [2222 [ffe 11111 11111 11110 11110 11111 11111		480 ns 17772,22772727277 22727,22727272727 22727,227272727	500 ns 22222,222 22222,222 22222,222 [fffe	510.1 12222 12222 12222 1,ffc0	515. 15 1222. 7722. 7722. 7722.	200 ms 520 ms	540 ns 1 10000 ps 2222,22222222222222222222222222222222	560 ns 22222,222222222 22222,22222222 22222,222222	, 580 222222 222222 222222 222222 2222222 2222	ns	600 m 272222, 272222,		
Name ↓ dk_period ▶ sor_dataout[0:15] ▶ or_dataout[0:15] ▶ and_dataout[0:15] ▶ fill	Valor 10000 [2222 [2222 [fff= 11111 11111 11110 11110 11111 11111		480 ns 1777, 2777 2777 2777 2777, 2777 2777 2777	1500 rs 12222,222 12222,222 12222,222 [fffe	510.1 12222 12222 2,#fc0	515. 115. 122. 122. 122. 122. 122.	200 ms 520 ms 520 ms 7272, 22727277777 7272, 227277777777 7272, 2272777777777 7272, 2272777777777 7272, 2272777777777 7272, 2272777777777 7272, 22727777777777 7272, 227277777777777777777777777777777	540 ns 1 10000 ps 2222,22222222222222222222222222222222	1560 ns 12722, 22727272 12722, 277272727 12722, 277272727 17727, 277272727 17700, HTe, HTe, HTe, HTe	, 530 222222 222222 222222 ,#fe]	n, 2000 2000 2000 2, 2000 2000 2000 2, 2000 2000	500 m		
Name ↓ dk_period ▶ sor_dataout[0:15] ■ or_dataout[0:15] ▶ and_dataout[0:15] ▶ it_dataout[0:15] ▶ it_dataout[0:16]	Value 10000 [2222 [2222 [fff= 11111 11110 11110 11110 11111 11111 11111		480 ns 1777, 2777777777777777777777777777777777	1500 rs 12222,222 12222,222 12222,222 12222,222 12222,222	510.4 12222 12222 12222 12222	515. 15 11 7777 7777 7777	200 ms 520 ms 520 ms 522 ns 7222,22222222222 7222,22222222222 7222,2222222222	540 ns 1 10000 ps 2222,22222222222 2222,22222222222 2222,222222	1560 ns 12222, 222222222 12222, 222222222 12222, 222222222 12000 (Hfe, Hfe, Hfe, Hfe)	, [580] 2222222 2222222 ,##e]	************ * , 2222 2227 222 * , 2222 222 222	500 m		
Name ↓ dk_period ▶ soc_dataout[0:15] ▶ or_dataout[0:15] ▶ and_dataout[0:15] ▶ ff 00 ▶ ff 01 ▶ ff 01 ▶ ff 01 ▶ ff 02	Value 10000 [2222 [2222 [fff= 11111 11110 11110 11110 11111 11111 11111 00000 11110		480 ns 1777, 2777777777777777777777777777777777	1500 rs 22222,222 22222,222 22222,222 [fffe	510.4 12222 12222 12222 12222	515 11 7777 7777 7777 7777	200 ms 520 ms 520 ms 522 ms 7222,222222222222 7222,222222222222 7222,2222222222	540 ns 1 10000 ps 2222,22222222222 2222,22222222222 2222,22222222	1560 ns 12722, 227222222 12722, 227222222 12722, 272222222 17222, 272222222 1722, 272222222	1580 122,222 122,122 123,123,123,123 123,123,123 123,123,123 123,123,123 123,123,123 123,123,123,123 123,123,123,123,123,123,123,123,123,123,	ns , 2222222222 , 2222222222 , 2222222222	600 m		
Nume 12 dk_period 13 dx_o_dstaout[0:15] 14 or_dstaout[0:15] 15 16 16 11 17 14 18 12 19 12 10 12 11 12 12 12 13 12 14 12 15 12 16 12 17 12 18 12 19 12 10 12 11 12 12 12 13 12 14 12 15 12 16 12 17 12 18 12 19 12 10 12 11 12 12 12 13 12 14 12 15 12 16 12 17	Value 10000 [2222 [7222 [fffe 11111 11110 11110 11111 11111 11111 11111 11111 11111		480 ms 1777, 27777777777 2777, 27777777777 2777, 27777777777	1500 rs 22222,222 22222,222 22222,222 [Fffe	510.4 72222 72222 7860	515. 115 111 1222 1222 1000	200 ms 520 ms 520 ms 7272,2222222222222 7272,222222222222 7272,2222222222	540 ns 1 10000 ps 2222,22222222222 2222,22222222222 2222,22222222	1560 ns 12722, 222222222 12222, 222222222 12222, 222222222 1222, 2222222222	580 22222 22222 222222 222222 222222 222222	ns , 2122222222 , 2222222222 , 2222222222			
Nume 12 dk_period 2 dt_out[0:15] 2 or_dataout[0:15] 2 dt_dataout[0:15] 2	Value 10000 [2222 [2222 [fff= 11111 11110 11110 11111 11111 00000 11110 11111 11111		H420 ms 17772, 22772727777 22772, 27772777777777 22772, 277727777777777	1500 rs 22272, 222 22272, 222 22222, 222 [Fffee	510.4 72222 72222 7860	515	200 rej 520 rej 520 rej 520 rej 7272,72727777777 7272,72727777777 7272,72777777777 7272,72777777777 7272,7277777777 7272,7277777777 7272,7277777777 7272,7277777777 7272,7277777777 7272,7277777777 7272,7277777777 7272,7277777777 7272,72777777777 7272,72777777777 7272,727777777777	540 ms	560 ns 22222, 222222222 22222, 222222222 22222, 22222222	580 222222 222222 222222 222222 222222 2222	n, 2222222222 2, 2222222222 2, 2222222222			
Nume 1 ≤ dk_period > stataout[0:15] > or_dataout[0:15] > stat_dataout[0:15] > <t< td=""><td>Valor 10000 [2222 [2222 [fff= 11111 11110 11110 11110 11111 11111 11111 11111 11111</td><td></td><td>480 ns 77772,22722722222 72227,22722222222 72222,22222222</td><td>1500 ns 22272,222 22222,222 [fffe</td><td>510.4 12222 12222 ,#CO</td><td>515- </td><td>200 ms 520 ms</td><td>540 ns 10000 ps 2222,22222222222222222222222222222222</td><td>560 ns 22222, 222222222 22222, 222222222 22222, 22222222</td><td>580 2222222 2222222 2222222 2222222 222222</td><td>n,</td><td></td><td></td><td></td></t<>	Valor 10000 [2222 [2222 [fff= 11111 11110 11110 11110 11111 11111 11111 11111 11111		480 ns 77772,22722722222 72227,22722222222 72222,22222222	1500 ns 22272,222 22222,222 [fffe	510.4 12222 12222 ,#CO	515- 	200 ms 520 ms	540 ns 10000 ps 2222,22222222222222222222222222222222	560 ns 22222, 222222222 22222, 222222222 22222, 22222222	580 2222222 2222222 2222222 2222222 222222	n,			
Nonce 1 dt, period	Value 10000 [2222 [2222 [2222 [fffe 11111 11111 11111 11111 11111 11111 11111 11111 11111 11111		480 ns 12222,22222222222 22221,22222222222 22222,22222222	1500 rss	510.1 12222 12222 17222 17600	515. 1.1. 2222 7222 7000	200 ms 520 ms	540 ns 10000 ps 2222,22222222222222222222222222222222	1560 ns 12722, 227272727 12722, 227272727 12722, 2772727272 12722, 27727272727 1500, HTe, HTe, HTe, HTe,	580 2222222 2222222 2222222 2222222 222222	ns , , , , , , , , , , , , , , , , , , ,	600 nc 272222 2722 27222 27222 27222 27222 27222 27222 27222 27222 27222 272 272 272 2722 2722 2722 2722 2722 2722 2722 2722 2722 2722 2722 2722 2722 2722 272 272 272 27		
Name 1: dk_period > disacut[0:15] > disacut[0:16] > disacut[0:1	Value 10000 [2222 [2222 [fff= 11111 11110 11110 11111 11111 11111 11111 11111 11111 11111 11111 11111		480 ns 1777, 2277, 2277, 2777, 2777 2277, 2777, 2777, 2777, 2777 2777, 2	1500 rss	510.1 22222 22222 2000	515 1 2222. 7000	200 ms 520 ms	540 rs 2222,22222222222222222222222222222222	1560 ns 12722, 227272727 2722, 277272727 27272, 277272727 17272, 277272727 17272, 277272727 1720, 17727 1720, 17727 1720, 17727 1720, 17727 1720, 17727 1720, 17727 1720, 17277 1720, 17277 1720, 17277 1720, 17277 1727, 172777 1727, 17277 1727, 17277 17277, 17277 17277, 17277 17277, 17277 17277, 17277 17277, 17277 17277, 17277 17277, 172777 172777, 172777 172777, 172777, 1727777 172777, 1727777, 1727777, 17277777 1727777, 17277777, 172777777777777777777	580 222222 222222 222222 222222 222222 2222	n: , , , , , , , , , , , , , , , , , , ,			
Since ↓ dk_period ▶ acd_dataout[0:15] ▶ acd_dataout[0:15] ▶ acd_dataout[0:15] ▶ dk_costaout[0:15] ▶ dk_costaout[0:16]	Value 10000 [2222 [2222 [fffe 11111 11110 11110 11111 11111 11111 11111 11111 11111 11111 11111 11111		480 ns 1777, 2777777777777777777777777777777777	500 ms	510.1	515. 15 1222 7222 7000	200 ms 520 ms	540 ns 1 10000 ps 2222,22222222222222222222222222222222	1560 ns 12722, 227227272 12722, 277227272 12722, 27722, 277227272 1500. (Hite, Hite, Hite)	580 22 2222 22 2222 22 2222 22 2222 22 222 22 222 22 222 22 22 22	*, 2122277777 , 2222777777 , 22277777777 , 2227777777777	500 nc 222222, 222222, 222222, 222222, 222222, 222222		

Figure: 11 SLC operation verification waveforms

4.3 Improvement analysis

The results of the introduction and integration of the MAP into the verification environment were plotted as speed of data analysis versus the number of program modules for manual and MAP analyses (Figure 12). The fact here is that the same number of engineers analysed the result of the assertion synthesis for all the program modules considered in this paper.



Figure 12: Chart of speed against number of modules

Results show a significant improvement in the analysis time and the overall verification time. With the use of the MAP, about 17% of the time taken to manually analyse the data generated from synthesised assertions when the number of modules are small was achieved. Figure 12 shows that with fewer modules the effectiveness of the processor is marginal and slightly better than manual analysis. With increase in the number modules the superiority of the device is clearly shown on the graph. The difficulty of humans to examine enormous data generated by simulation tools was once again highlighted.

5. Conclusions

This paper was embarked upon in order to reduce the analysis and computation time of synthesised assertions in a software/hardware verification environment. The design was verified and implemented by using simulation, synthesis and configuration tools of Xilinx Inc. USA. Although it appears that a cheaper FPGA could have been used instead of Virtex6 for this design, the available components can be of advantage when new functionality is added to the device. A further optimisation of the code could lead to a reduction in area, which would make it possible to use a cheaper FPGA. The HDL code has been parameterised such that the width of the registers can be increased to be able to handle papers with increased program modules if there are enough computing resources.

It has been shown that the MAP reduces the time required to analyse and process synthesised assertions or any other data presented in the form of matrices that compares a golden model and input response vectors.

References

Andrew, A., O'Fallon, A. and Chen T. 2003. A Rule-Based Software Testing Method forVHDL Models Proceedings. IFIPInternational Conf. of VLSI SoC, Darmstadt, Germany.

Ashenden, PJ. 2001. The Designer's Guide to VHDL, 2nd ed. Morgan Kaufmann Publishers, San Francisco. pp138 – 140.

Bergeron, J. 2003. Writing Test Benches: Functional Verification of HDL Models, 2nd ed., Kluwer Academic Publishers. Boston, USA.

Chang, M. 1995. From VHDL to CPLD — a Synthesizable Journey, One-day Tutorial Workshop, IEEE International ASIC Conference, Austin, Texas.

Chang, M. 1996. Teaching Top-down Design Using VHDL and CPLD. Proceedings of Frontiers in Education Conference, 2:514 - 517.

Foster, HD., Krolnik, AC. and Lacey, DJ. 2004. Assertion-Based Design, 2nd ed. Kluwer Academic Publishers, Boston, USA.

IEEE. 1994. IEEE Standard VHDL Language Reference Manual: ANSI/IEEE Standard 1076–1993. New York.

Kalyaev, IA., Levin, II., Semernikov, EA. 2011. Reconfigurable System Uses Large FPGA Computation Fields. Xcell Journal, 75: 24 – 29.

Ngene, CU. 2011. Embedded Diagnostic Infrastructure for HDL-models of Digital Systems on Chip, Ph.D Dissertation, Kharkov National University of Radioelectronics.

Seward, SR. and Lala, PK. 2003. Fault Injection for Verifying Testability at the VHDL Level. IEEE ITC International Test Conference, pp 131 – 137.

Xilinx. 2011. ISE In-Depth Tutorial. UG695 (v13.1). www.xilinx.com

Xilinx. 2010. XST User Guide for Virtex-6 and Spartan-6 Devices. UG687 (v 12.1) www.xilinx.com