ASIC Thread for Decimal (BCD) Algorithm: A Tutorial on How Create a Thread and to Evaluate ISPMACH4256ZE CPLD

Gelacio Castillo C, Martha P. Jiménez V, Aurora Aparicio C

Abstract— Here it is explained how can be designed by an easy form, and using HDL tool, a thread for implement the algorithm for natural binary format to decimal (BCD) format. In order to achieve that, here is released an explanation of such algorithm in a fast and needed way. In VHDL, structural style will be used for build each one modules for the Arithmetic Unit as well as those modules for Control Unit. The program is the set of instructions. Each instruction is a single operation as a sum, a shift, a comparison and so on. Every those instructions are carried out by a single module in VHDL. The memory to store the program it is implement by array of registers. That array is executed in a sequence by which is driven by a Program Counter (PC). The complete architecture it is explain step by step in order to it can be used as application note or a tutorial, and repeated by teachers, students and hobbyist. The complete processor it is builds in a single CPLD from Lattice Semiconductor. That is the ispMACH LC4256ZE 5TN144C device.

Index Terms— Binary natural to decimal BCD format, tutorial on how design a thread

I. INTRODUCTION

From the point of view of a programmer user, a tread is just one instruction which is used in high level lenguage for do easier the task, nevertheless, from a point of view of a engineer designer a thread is a set of low level instructions and involves heavy work at hardware [1]. So, this work is an architecture for a thread, which deals with implementation of a single algorithm with a few instructions, it can be easily located inside of harvard model with a memory for the program and other one memory for data. The first is an array with a kind of flash registers built here with VHDL, and the last, actually is a temporary register for show in an external display the decimal number. The data is an input 8 bits number in natural binary format introduced by an external dipswitch. Just an external bit is used for enable and reset, nevertheless can be used an input bit for reset and other one for enable.

Manuscript Received on December 2014.

Dr. Gelacio Castillo C, Ingeniería en Sistemas Computacionales, Instituto Politécnico Nacional, Escuela Superior de Cómputo, México D. F. México.

M. en C. Martha P. Jiménez V, Ingeniería en Sistemas Computacionales, Instituto Politécnico Nacional, Escuela Superior de Cómputo, México D. F. México.

M. en C. Aurora Aparicio C, Escuela Superior de Ingeniería Mecánica y Eléctrica, Instituto Politécnico Nacional, México D. F., México.

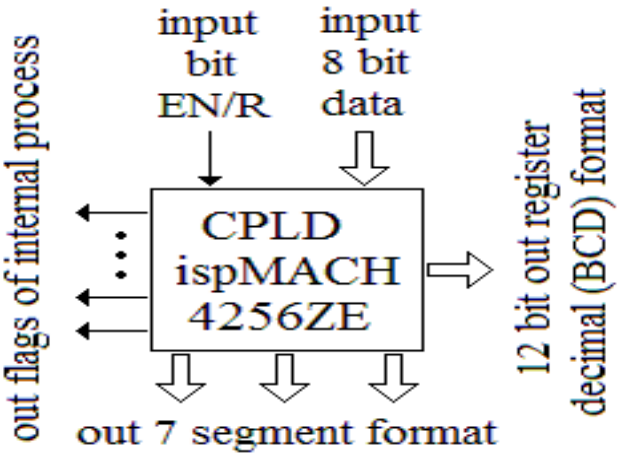

Figure 1: Entity top level block diagram

As it can see, only two inputs are used, the first is an 8 bits vector and the other one is a single bit input, for enable and reset. The main output is a buffered 12 bits register for decimal (BCD) format. Inside of the same device is implemented a BCD to seven segment encoder. Optionally other output can be leaves in order to monitoring inter flag or signal control of the process. All of these elements are shows in the Fig. 1. One guide for develop is given in the Chapter 5 of Sajjan [2]. Documentation available can be download from web site of Lattice Semiconductor [3], [4]. A scheme, made by authors for this work, of the Breakout Board Evaluation Kit, it is shown in the Fig. 2.

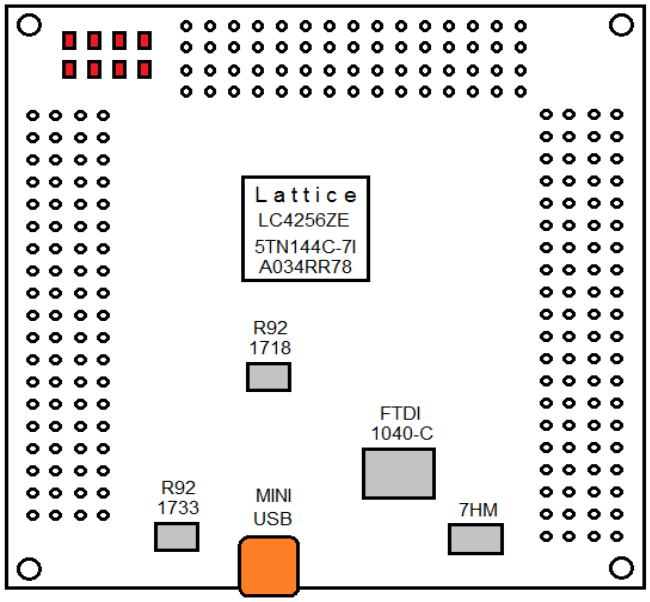

Figure 2: Breakout board development kit

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

II. BINARY NATURAL TO DECIMAL (BCD) ALGORITHM

Any input data and output to a computer system, mainly are in decimal (BCD) format. However, because digital machines processes data from natural binary format, then a methodology for translate binary-decimal (BCD) format and decimal (BCD) to binary format it is needed. This methodology is the popular binary to decimal (BCD) algorithm and reverse. Such algorithm issued by the first time in 1997 by an application note of Xilinx Company [5]. The implementation this algorithm to translate a number in natural binary format to decimal (BCD) format it is released here. In 8 bit, the bigger account reached is 11111111, which is equivalent to 255 in decimal format. So, a 12 bits register is needed to get decimal format, split it in 3 groups of 4 bit (nibble), each nibble for a digit.

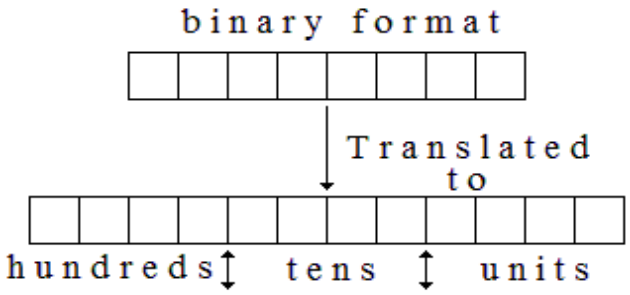

Figure 3: What it must do by algorithm

To start the explanation, the main idea it is to note that after a complete rotation of a binary number it not changes. That idea it is shows in the following Fig. 4. The MSB bit, in the "source register" it is placed in LSB position of the target 8 bits register. This is just the idea.

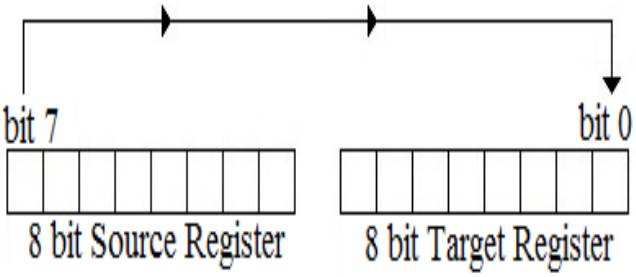

Figure 4: The first idea of algorithm

So, it must do some operations in order put the number inside a register of 12 bit instead a register of 8 bit, and at the same time leave that in decimal (BCD) format. What it must do, it is shows in the Fig. 5.

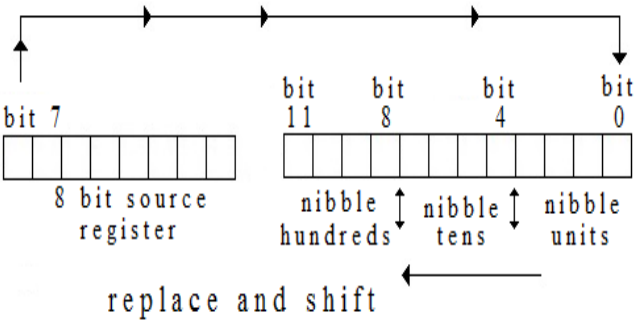

Figure 5: Activity inside registers

Operations what it must do are: **1A**.- Rotation by one bit in each iteration. The MSB in the source register (of 8 bit) replace to the LSB target register (of 12 bit). An iteration start with a rotation. **2A**.- Then, the 12 bit register it must be split in 3 nibbles; Low Nibble (L-N), Intermediate Nibble (I-N), and Higher Nibble (H-N). **3A**.- Now follows comparisons. If L-N > 0100 then add 0011 else do nothing. If I-N > 0100 then add 0011 else do nothing. If H-N > 0100 then add 0011 else do nothing. **4A**.-Ok, now follows join them together, concatenate (with & in VHDL) the three nibbles in order to recovery the 12 bits register. **5A**.- Finally do rotation by one bit on the register of 12 bit. This point is the end of the iteration. However, the number of iterations is the same as the number of bits in the source register.

III. INSTRUCTIONS

According with algorithm there are a little instructions and these are very simple. These shown in the following Table 1.

Table 1: Instructions

mnemonic	code
Init	0000
ReadPort	0001
Sust	0010
CompAdd	0011
shift8	0100
shift12	0101

Init is the instruction by which are initialized the registers of 8 and 12 bit respectively. Furthermore, this is the module by which reset is carried out by external bit EN/R. ReadPort is the instruction by which is read the number in binary format from external dipswitch (input 8 bit data). Sust is the instruction by which the substitution were carried out from MSB bit in the register of 8 bit to the LSB bit in the register of 12 bit. CompAdd is the module by which is carried out (a) split in nibbles, (b) comparison, and (c) add by 0011 if it is necessary in each nibble. This is because they are very simple operations. So, CompAdd is just an instruction. Shift8 is the instruction for shift by one bit, from the right to left, in the register of 8 bit. Shift12 carries out shift by one bit, from the right to the left in the register of 12 bit. The last are component in Arithmetic and Logic Unit (ALU). There is two needed counters; a counter program (PC) and counter iterations (ItC). Actually, the PC is a pointer to the next instruction that read from E2PROM. ItC will have the control on the number of iterations, of course. Some others modules where implemented in order to get the complete design of the Unit Control, along with PC and ItC. Modules in the Unit control are not instruction. Two of these are PC, ItC. ReadCode is a module that read the following operation from E2PROM. Accumulators for 8 and 12 bits registers, which are temporary register. Module for codec to translate from decimal (BCD) format in 12 bit to seven segments display. Complete architecture it is draws in the Figure 6.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

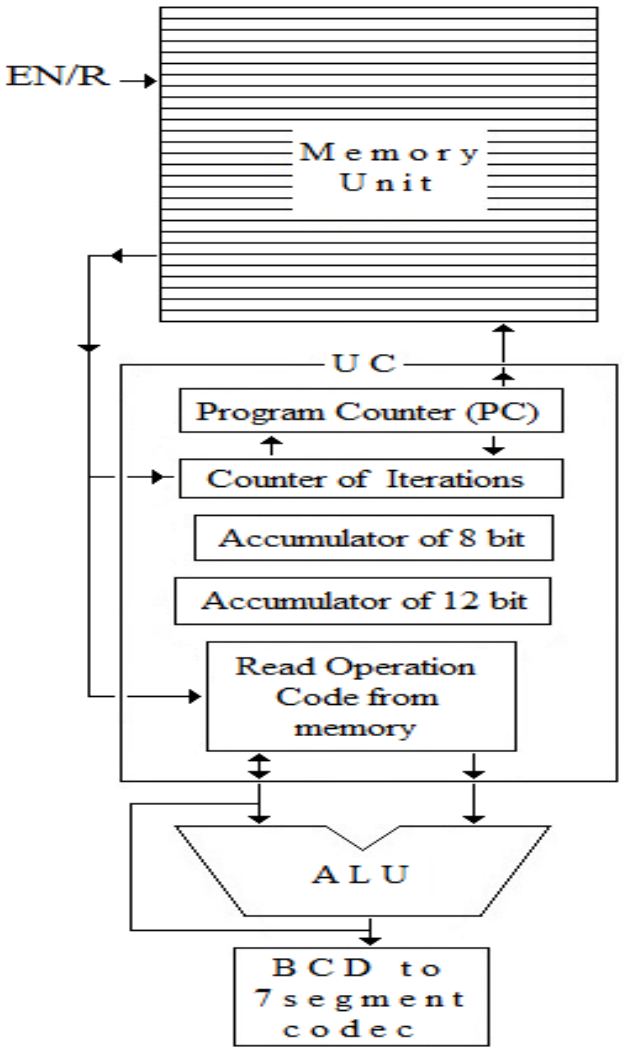

Figure 6: Block diagram architecture

IV. STATES MACHINE

In the following Fig. 7, is drawn the states machine by which was designed in VHDL the algorithm. With EN/R = '0' the process return to Init state, that is; initialize PC then read code of Init instruction, execute instruction Init and initialize accumulators, however ItC is not increased and so PC is not increased. With EN/R = '1' the process go to the next state by increasing PC and ItC, that implies that PC is initialized to "0000" when EN/R = '0' and it is increased when EN/R = '1', after the respective accumulator has been executed, and if ItC has been increased. At the end of each iteration, the ItC is increased and if its account is minor of number of bit in the source register. If ItC it is not increased at the end of an iteration, the process it is stoped. Each instruction late four cycles of clock module of Control Unit take at least three cycle of clock. The registers of PC, ItC and "codop" are all of four bits, however all of these have different rolls. Each module has two flags. One input Flag, and one output Flag. Through these flags, the dialog it is carry out in all the process.

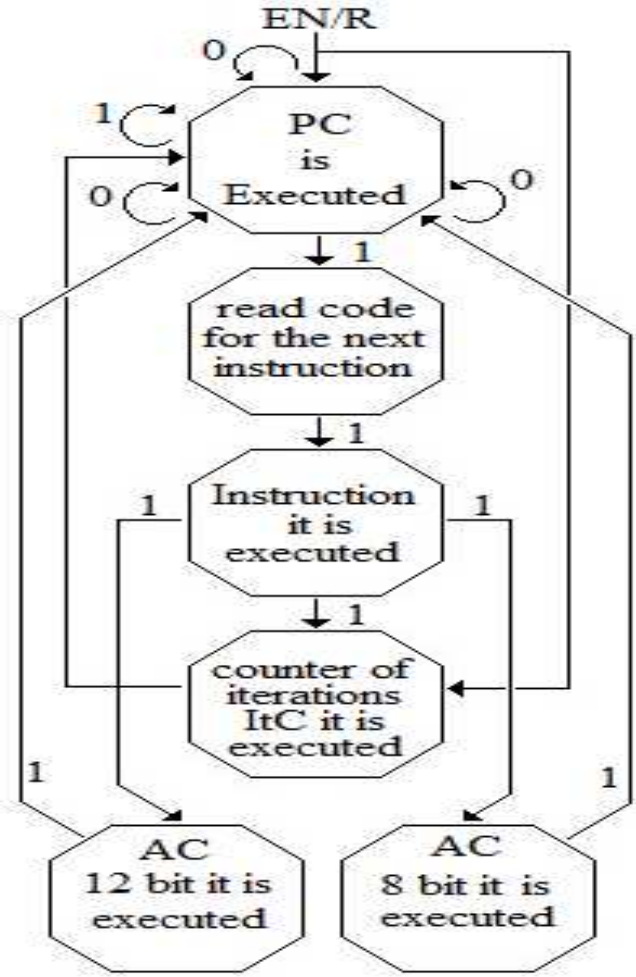

Figure 7: State machine

V. INIT INSTRUCTION

The Fig. 8 shows a block diagram of Init module. Remember this is an instrucction and its code is "0000". Module Init in the Fig. 8 shows two input bit, "clkinit" of course is the clock signal, and "inFlaginit" comes from "read code" module, as well as "codopinit" just when the process it is initialized, and after that the process it is carried out according with diagram of Fig. 7. Initialization to 12 and 8 bits accumulators is carry out by "outAC8init" and "outAC12init", enabled with the flags "outFlag8init" and "outFlag12init" respectively. Fig. 9 is the flow diagram for Init module.

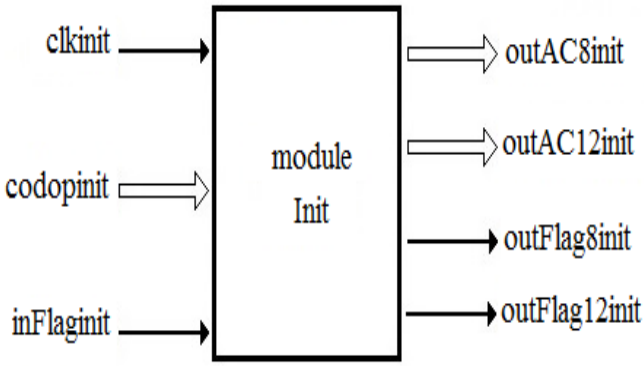

Figure 8: Entity module Init

Now in the Fig. 9 it is going to show details of flow for its proramming on VHDL.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

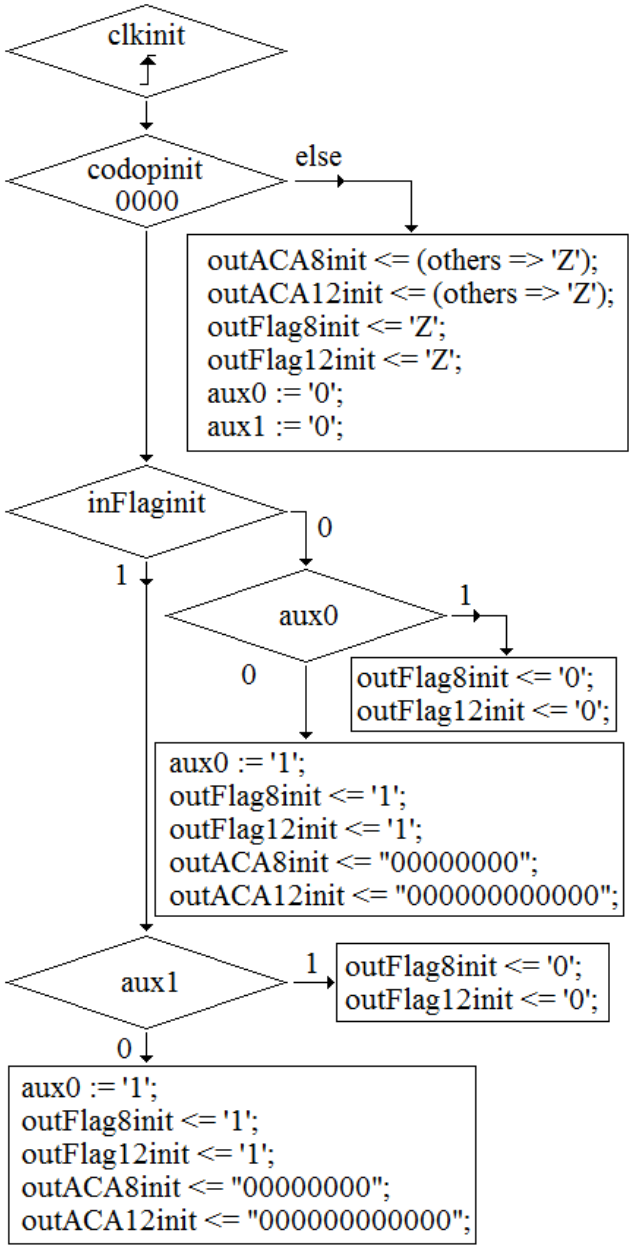

Figure 9: Flow diagram Init instruction

In the Fig. 9, "aux0" and "aux1" are variables inside it module VHDL. Both of these have the same roll, aboid that this part of process it is execute twice. Furthermore, operation of initializing must be executed under two conditions: (a) the first one when algorithm is solicited by the first time, "inFlaginit" can has '0' at the start of process, (b) the second one when EN/R = '1', it must be guarantised accumulators are put in '0's. After each accumulator, 8 bit or 12 bit, has been executed PC it is increased if and only when EN/R = '1'. If EN/R = '0' PC must not be increased. Only in the first step, both accumulators it is execute at the same time. Then, when the process is inside of the cycle of iterations, just one accumulator executed at the time. While outFlagIter = '1', then PC is increased if the other flags are accomplished. The instruction "Init" is executed just one time, at the beginning the process. The instruction "ReadPort", also executed only one time, after of "Init". Both of these instructions are not part of iterations. Table 2 which include circle and rows, shows those instruction part of iterations, "Sust", "CompAdd", "shift8", and "shift12". Remember that each cycle it is execute as the state machine in the Fig. 7.

Table 2: Cycle

mnemonic	code
Init	0000
ReadPort	0001
Sust	0010
CompAdd	0011
shift8	0100
shift12	0101

VI. READ-PORT INSTRUCTION

Fig. 10 shows block entity for the ReadPort Instruction. Input "inPortALp" read external data from dipswitch. Input "inFlagLp" come from "read code from the next instruction".

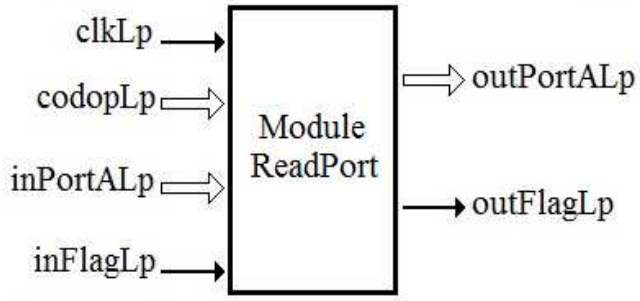

Figure 10: Entity ReadPort

Input "codopLP" also come from "read code from the next instruction". Bus "outPortALp" go to the 8 bits accumulator. The output "outFlagLP" also go to the accumulator. The logic of the code VHDL it is show in the Fig. 11.

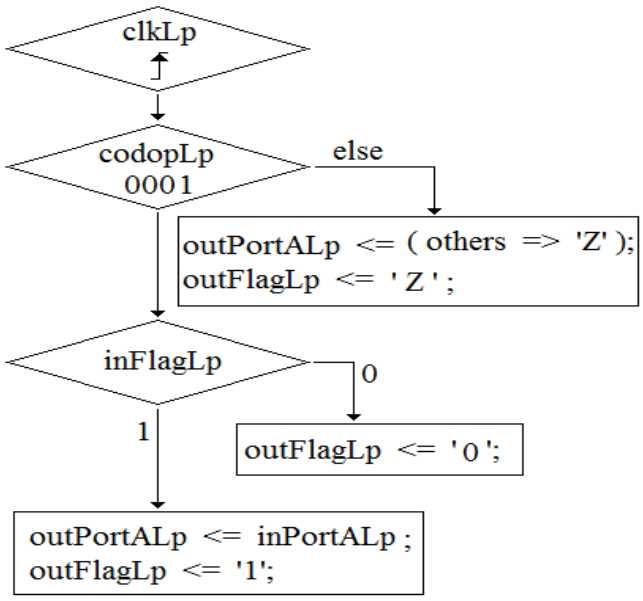

Figure 11: Flow Diagram ReadPort

At the end of of the instruction "ReadPort" the bus "outPortALp" must be put in high impedance 'Z', due to the bus of 8 bit could be used by others instructions. Furthermore, although "codop" it is "0001" the data it is not put on outPortALp until "inFlagLp" = '1'. Furthermore outFlagLp is put in high value, “1” logic, so, accumulator of 8 bit know that a data must be saved, such as indicated in the state machine in the Fig. 7.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

VII. SUST INSTRUCTION

Fig. 12 shows block entity for the “Sust” instruction. As stated earlier, the LSB in 12 bits register it is replace by MSB from 8 bits register. This is the only thing that makes by the module “Sust” instruction.

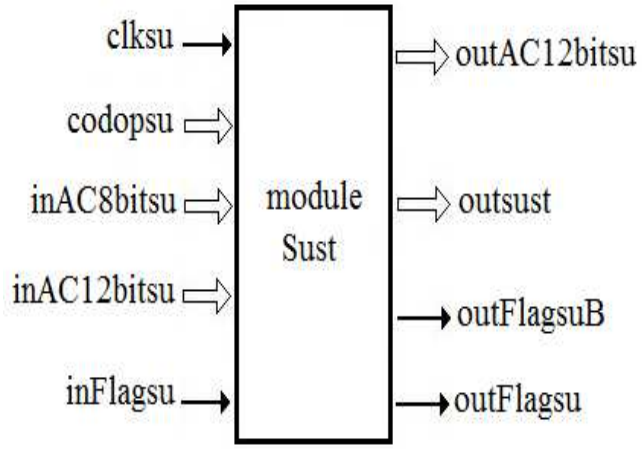

Figure 12: Entity “Sust”

Module in the Fig. 12 has codop because such module is an instruction. Has "inAC8bitsu" which come from accumulator of 8 bit, "inAC12bitsu" that come from accumulator of 12 bit. The last it is due to this module replace LSB in the register of 12 bit by MSB from the register of 8 bit, such it is shows in the Fig. 5. "inFlagsu" come from whether, accumulator of 8 bit or accumulator of 12 bit. The register of 8 bit is not update, however the register of 12 bit it changes. It has a bus "outAC12bitsu" which go to accumulator of 12 bits. Output "outsust" it is not needed, however it is used as RAM or buffer. Flag called "outFlagsu" inform to accumulator of 12 bit that the data it is ready to be stored. Fig. 13 shows flow for the VHDL code. It can see from Fig. 11 and 13, how the logic is the same. The block in the bottom from Figure 13, in the first two lines. The register "outsust" is not needed is the simplest form of RAM storage. It can see from Figure 11 and 13, how the logic is the same. The block in the bottom from Fig. 13, in the first two lines. The register "outsust" is not needed is the simplest form of RAM storage.

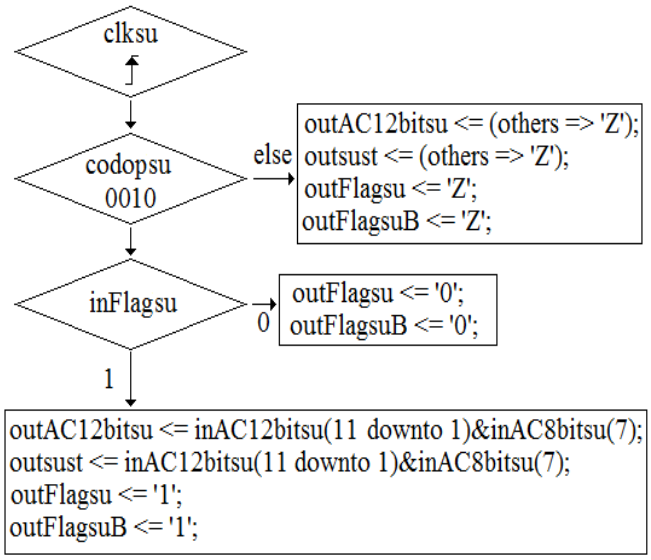

Figure 13: Flow diagram “Sust” module

VIII. COMPADD INSTRUCTION

Any other instruction have similar architecture. However, in order to be clearer, block diagram, for "CompAdd" instruction is in the Fig. 14. Entity is shown, with a data input of 12 bit called "inBuf12ca", on this data it is carried out operation of split register, comparison of nibbles with the binary number "0100", add if it is needed and concatenation.

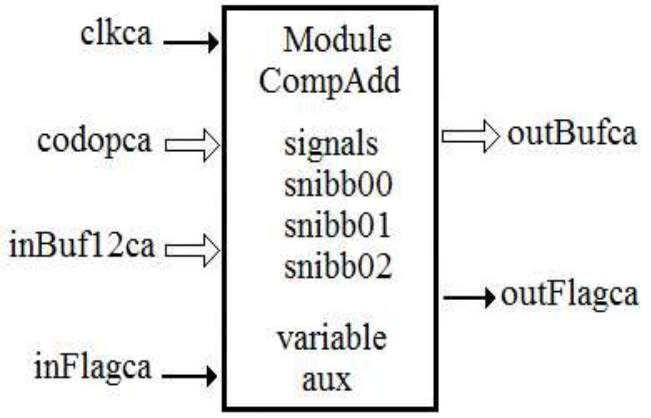

Figure 14: Entity CompAdd

Due to deal with an ASIC processor and operations are very simple, the design of this module include operations previously given; (splits register of 12 bit in nibbles, comparisons, add and concatenation). Its flow diagram given in the Fig. 15. That shows the first part of flow of the logic for do this module. Fig. 16 is the second part of the flow diagram.

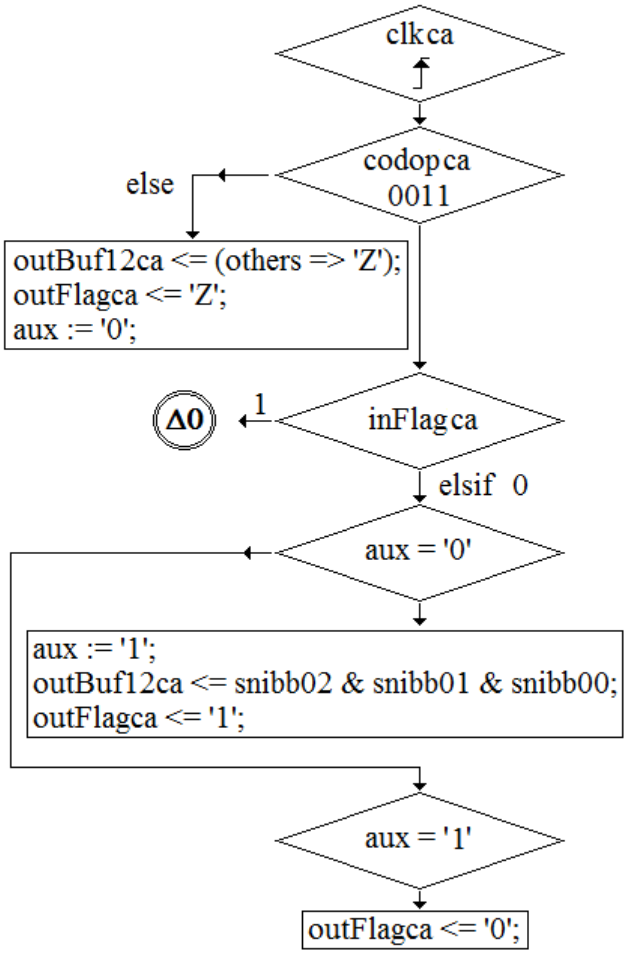

Figure 15: Flow diagram “CompAdd” first part

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

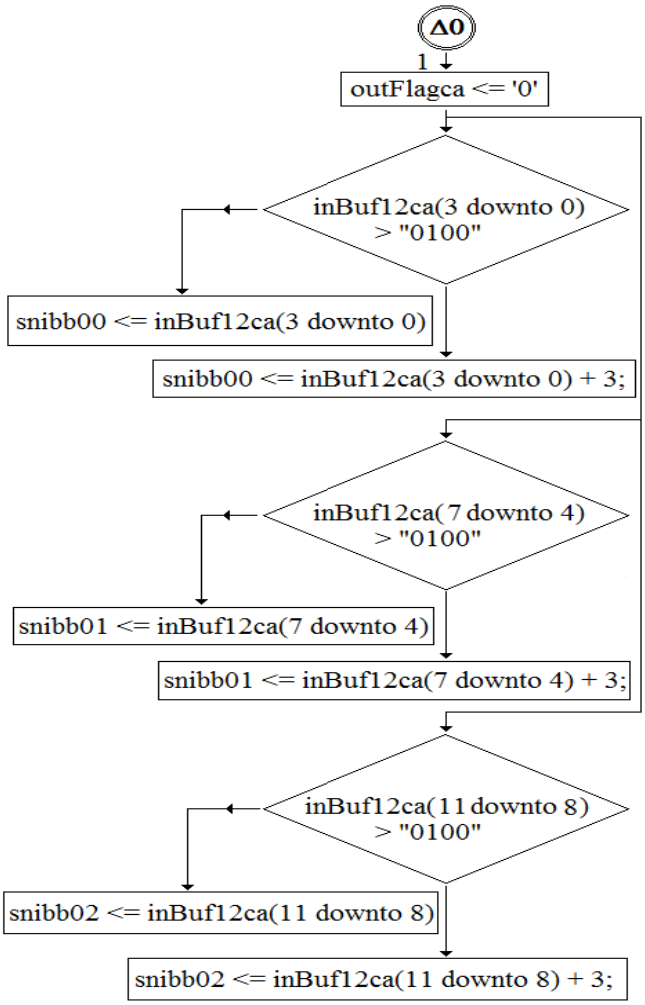

Figure 16: Flow diagram "CompAdd" second part

Note that if codop = "0011" then input to "inFlagac", else nothing it is done and so module it is switch off from output by high impedance 'Z' leaving free the bus. Signals "snibb" defined inside of architecture of VHDL syntax. The variable "aux" it is define inside of process and is revalued in the next clock cycle in order to not to repeat the operation. It must be ensured that at a first stage "inFlagac" = '1' and then '0', in order to can do operations. After that, when '0' concatenation is carried out and sent to the output. In others word, circled Δ0 block must be done in the first step, and then "inFlagac" = '0' block.

IX. SHIFT8BIT INSTRUCTION

Shift8bit instruction deal with shift by one bit in the register of 8 bit from of right to left. Fig. 17 shows block diagram this module.

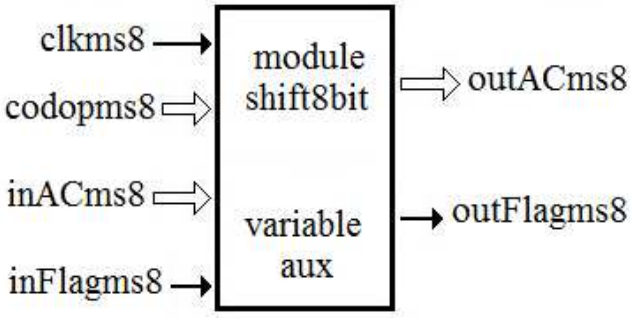

Figure 17: Entity "Shift8bit"

The "inFlagms8" and "codopms8" comes from "ReadCode" module, however "inACms8" comes from 8 bits accumulator (see Fig. 7). At the end, when "outFlagms8" = '1', 8 bit accumulator is updated by the data on the bus "outACms8" shown in Fig. 17. Fig. 18 shows logic flow for shift by one bit. If "codopms8" = "0100" then do it shift, else switch off from output by high impedance in the bus. Shift on accumulator of 12 bit is equal except by size of register. Modules "PC", "read code", ItC, "AC 12 bit", and "AC 8 bit" are modules of Unit Control UC (Fig. 7). However, the logic flow is alike of instruction in the ALU.

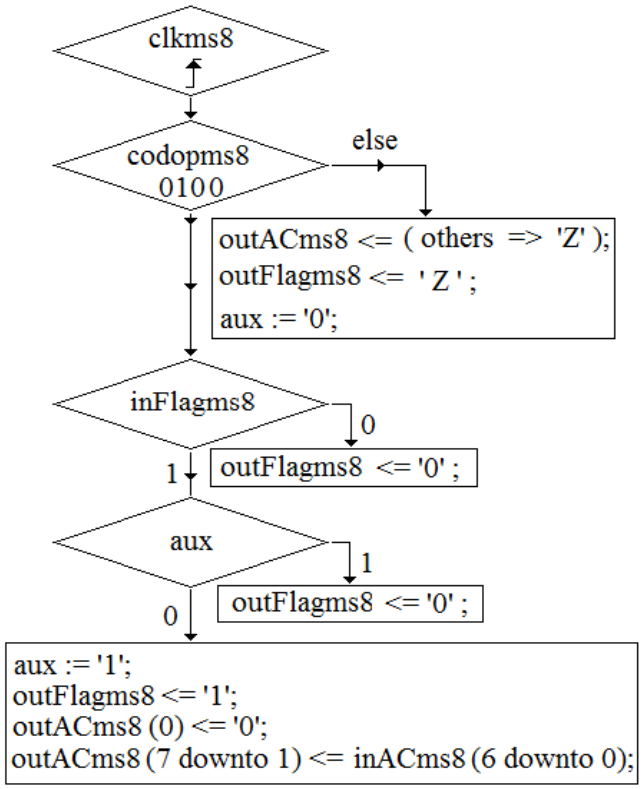

Figure 18: Flow diagram "Shift8bit"

X. ACCUMULATOR OF 8 BIT

Accumulator save result of an operation, then this data is ready for a next operation. The chat on every one modules on all this architecture it is carry out by "inFlag" and "outFlag" of each module whether is module of instruction on ALU, or module on UC.

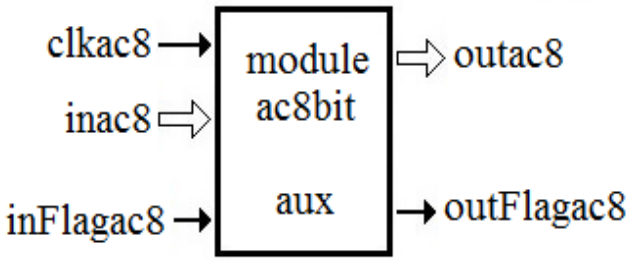

Figure 19: Entity 8 bits accumulator

From entity Fig. 19, the data "inac8" and "inFlagac8" comes from the last 8 bits operation, and enables to this accumulator to save the new data. As has been indicated, variable "aux" allows update the accumulator just one time, although this module could delay for more one clock cycle.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

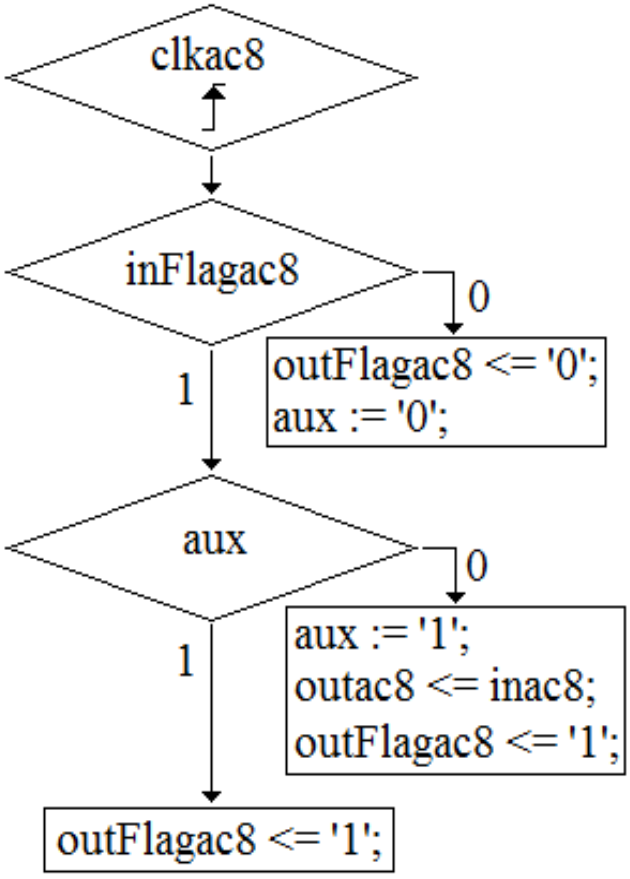

Figure 20: Flow diagram "ac8bit"

XI. Accumulator of 12 Bits

Module for 12 bit accumulator has the same logic of 8 bit accumulator shown in the Fig. 19 and 20, except that is of 12 bit.

XII. Program Counter (PC)

Block diagram of module PC is in the Fig. 21. In this processor, which is actually a thread, PC module is the most complex, although it is quite simple and easy.

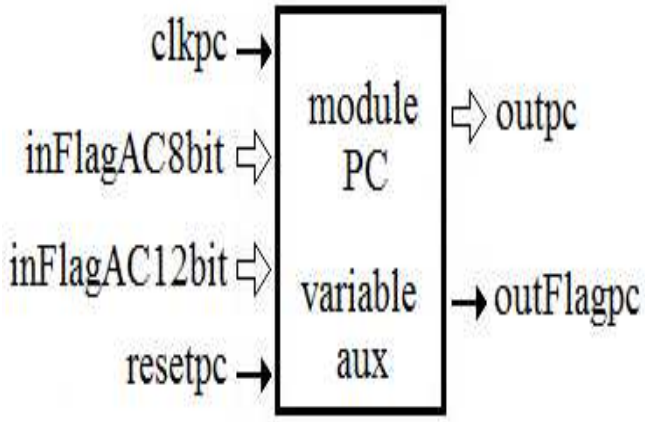

Figure 21: Entity PC

Fig. 22 is a first part of flow diagram PC for build it in VHDL code. As it was shown previously from the Fig. 7, after one operation which is carried out by instruction, accumulator it is executed for save the new data, and then PC it is increased, just until accumulator module have finished, whether it is 8 or 12 bit. PC module knows that information by "inFlagAC8bit" and "inFlagAC12bit" respectively, which comes from accumulators. Actually "outpc" it is the pointer to memory. The "outFlagpc" = '1' means that PC module has been increased. After that, the following module it is "ReadCode" (Fig. 7), which is part of UC. Such module read code from memory, the data which pointed by PC. The second part in the flow of logic it is show by the Fig. 23.

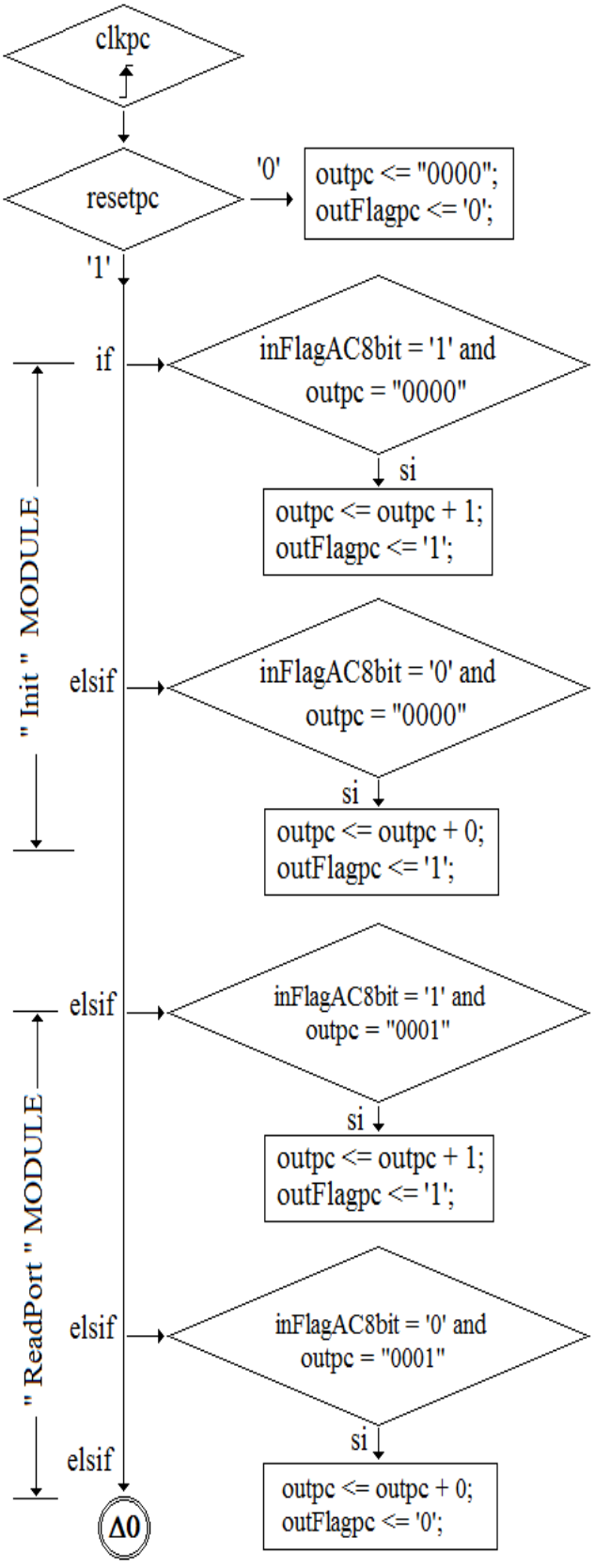

Figure 22: Flow diagram PC first part

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

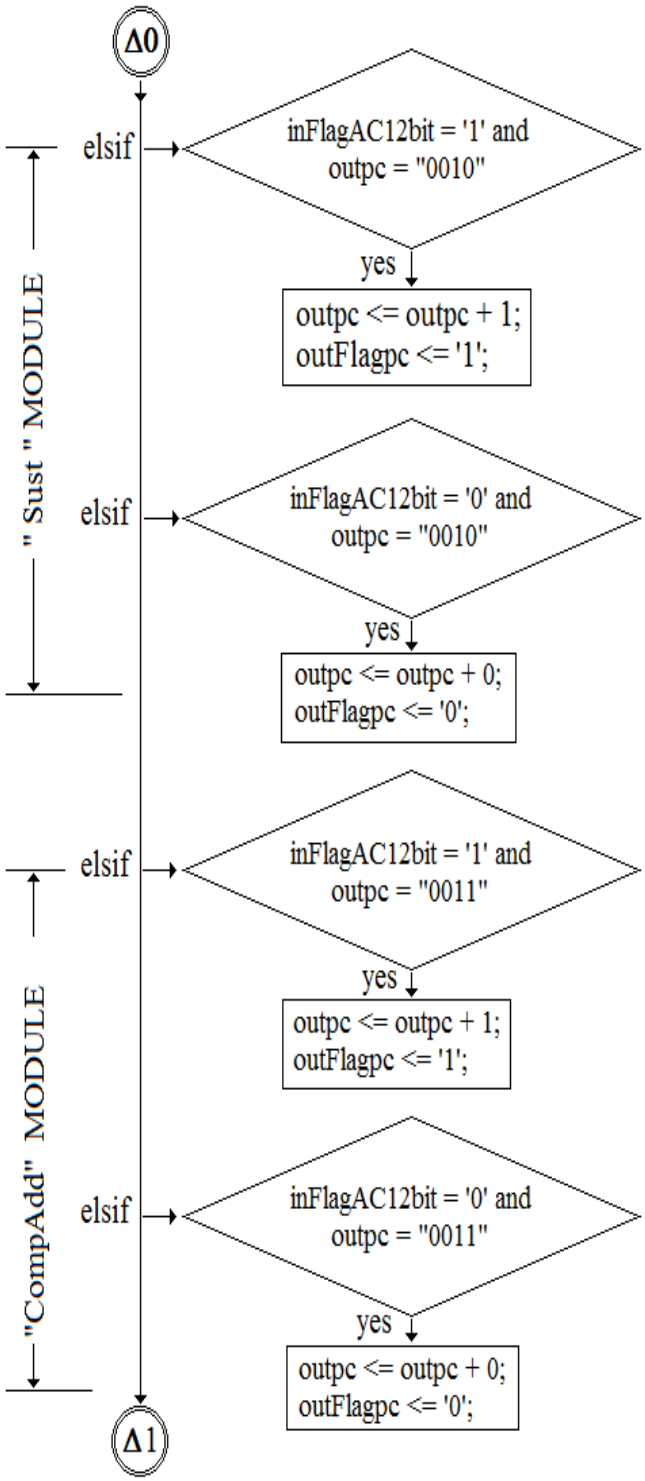

Figure 23: Flow diagram PC second part

Iteration counter "ItC" it is increased until an iteration cycle has finished, after "shift12" and before "Sust" modules, as it is shown in the Table 2. Fig. 24 shows the last part of logic flow of PC module. Fig. 22, 23 and 24 outlines logic flow for PC as part of Unit Control (CU). As it can see, this is the more complex module inside the UC. Remember that PC it is increased until accumulator has finished. That is true if the account of ItC has not achieved his higher value. When ItC has its higher value, “ItC” and then PC stop. PC know such information by input flags "inFlagAC8bit", "inFlagAC12bit", which come from accumulators, as well as flag from ItC module"flagiter", as is seen from state machine in the Fig. 7 and in the last block at the bottom Fig. 24. Note that "outpc" = "0010" it is the code for "Sust" instruction.

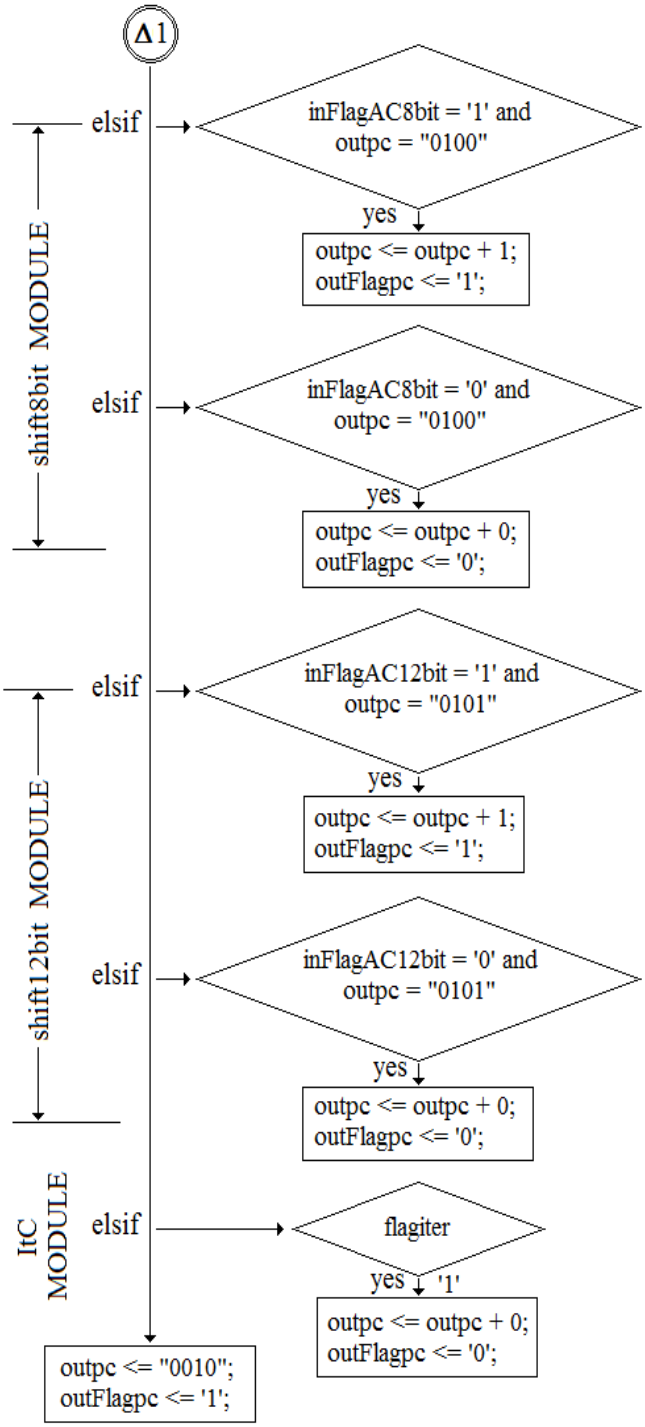

Figure 24: Flow diagram PC third and last part

XIII. READ CODE MODULE

Flag "inFlagInstrom" in the Fig. 25 comes from PC, as well as "inPCrom". Flag "outFlagrom" enables to next instruction and "outcode" is the code of this.

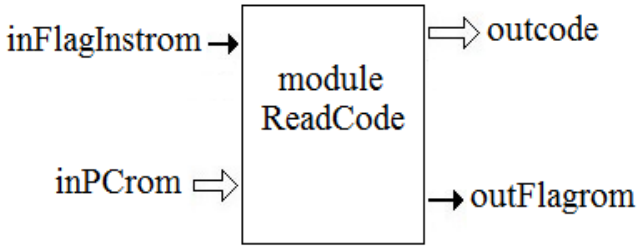

Figure 25: Entity ReadCode module

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

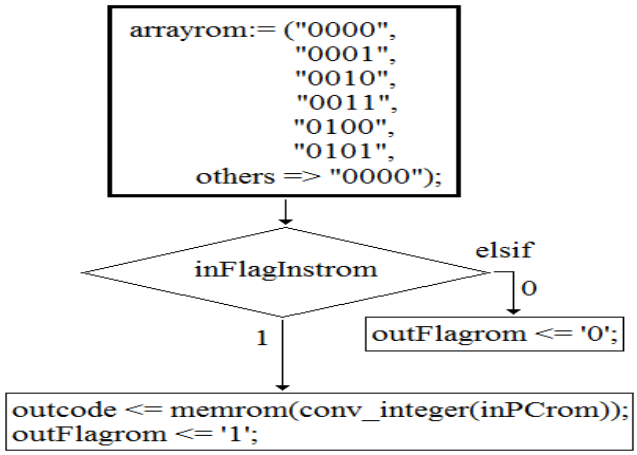

Figure 26: Flow diagram "ReadCode"

Program memory is very simple it is a constant array "arrayrom" from VHDL. Datas are accesed by PC as pointer toward such address memory. That module is part of Unit Control (UC) and it is included on State Machine as "*read code for the next instruction*" from Figure 7. Until this point, it has nine entities delivered as well as his respective architecture. Although in total were eleven module for implement algorithm, some of them as 12 bits accumulator and 12 bits shift have similar architecture with the respective 8 bits modules. In addition, the Iteration Counter module did not shown, however it has similar architecture with the PC. Fig. 1 has inside the top-level entity has input and output. Placed inside of such box, which is the symbol for entity and CPLD also, are all modules. At this level, the connections were made, with every one modules previously outlined. In order to get common busses such data and address it were used signals defined in VHDL syntax, one, four, eight and twelve bits respectively. Furthermore other four modules attached also, although such modules do not are parts of BCD algorithm, however they used to get display. Those four modules are, (1) timer to get two frequency for display, (2) counter ring, (3) mux in order to display decimal format in seven segments and (4) coder from BCD to seven segments.

XIV. TOOLS

The main approach and concern on this work is academic, however a simple assessment on performance and scope of device used, can be carry out. This exercise was focussed and addressed to teach and shows how can be designed small and simples threads, from configurable and Hardware Description Languages (HDL) tools. Device under assesment is ispMACH 4256ZE CPLD (LC4256ZE-5TN144C), from Lattice Semiconductor. Such device has 256 macro-cells, and some data from performance are show in the next section. Software for VHDL design is ispLever Classic from Lattice Semiconductor [5]. In the following tables, results are shows.

XV. VHDL CODES

Now in this section it will show list of codes VHDL. These codes, it can be obtain by e-mail and then instaled inside of a project. That implies structural style in VHDL and environment ispLever Classic under which was developmed [6], but is not exclusive. It can be used other development environment.

ac8bit06.vhd	mux06.vhd
ac12bit06.vhd	packagep06.vhd
coderNibbles06.vhd	pcinc06.vhd
compadd06.vhd	ReadPort06.vhd
contIter06.vhd	shift8bit06.vhd
contring06.vhd	shift12bit06.vhd
div06.vhd	sust06.vhd
init06.vhd	topp06.vhd
ReadCode06.vhd	

List 1: VHDL codes

Here some VHDL codes of modules are delivered, which are not part in the solution of algorithm, for example, (a) "coderNibbles06", (b) "contring06", (c) "div06", and (c) "mux06". These modules were for down to frequency and to multiplex seven segment display.

XVI. REPORTS FROM SINTHESIS FROM VHDL PROJECT

The next Tables are results sinthesis, implementation and fitter report from the development tools and of course, it is can be found from such report.

Table 3: Project_Summary

Project Name	processorv07
Project Path	C:\...\processorv07
Device	M4256_96
Package	144
GLB Input Mux Size	33
Available Blocks	16
Speed	-5.8
Part Number	LC4256ZE-5TN144C
Source Format	Pure_VHDL

Table 4: Compilation_Times

Prefit Time	0 secs
Load Design Time	0.05 secs
Partition Time	0.23 secs
Place Time	0.00 secs
Route Time	0.00 secs
Total Fit Time	00:00:01

Table 5: Design_Summary

Total Input Pins	11
Total Logic Functions	201
Total Output Pins	45
Total Bidir I/O Pins	2
Total Buried Nodes	154
Total Flip-Flops	188
Total D Flip-Flops	177
Total T Flip-Flops	8
Total Latches	3
Total Product Terms	791
Total Locked Pins	57
Total Unique Output Enables	2
Total Unique Clocks	3
Total Unique Clock Enables	10
Total Unique Resets	1
Fmax Logic Levels	2

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

Table 6: Device_Resource_Summary

	Device Total	Used	Not Used	Utilization
Dedicated Pins				
Clock/Input Pins	4	1	3	25
Input-Only Pins	10	2	8	20
I/O / Enable Pins	2	2	0	100
I/O Pins	94	53	41	56
Logic Functions	256	201	55	78
Input Registers	96	0	96	0
GLB Inputs	576	433	143	75
Logical Product Terms	1280	604	676	47
Occupied GLBs	16	16	0	100
Macrocells	256	201	55	78
Control Product Terms:				
GLB Clock/Clock Enables	16	15	1	93
GLB Reset/Presets	16	0	16	0
Macrocell Clocks	256	3	253	1
Macrocell Clock Enables	256	88	168	34
Macrocell Enables	256	0	256	0
Macrocell Resets	256	5	251	1
Macrocell Presets	256	0	256	0
Global Routing Pool	356	201	155	56

Maybe could come to be tediouse read this Tables, however they have important datas from which it is possible do assesment for performance of such type of solutions for algorithm.

XVII. Conclusions

Here has been outlined the building of a thread by using, as an example, the algorithm to translate a number given in natural binary format to decimal (BCD) format. In order to shows and teach how is go on and developed the process it is possible down to external frequence to see slowly the process and then up to it frequency. External 12 leds array can be connected to see activity of 12 bits register as well as 8 bit register, of course, three seven segments display. Furthermore, one led for each flag it is can connected to see its activity. That processor has excellent performance from a teaching point of view. Actually this is a thread, and so, can be done many others threads. Apendix A has bus architectiue.

References

[1] Intel. "Tutorial: Intel® Threading Building Blocks". Intel. Document Number 319872-009US. URL: http://www.intel.com/. "TBBtutorial.pdf". Intel: Developer Zone. https://software.intel.com/en-us/articles/intel-threading-building-blocks-tutorial-pdf.

[2] Sajjan G. Shiva, "Computer Organization, Design, and Architecture". Boca Raton, FL, 33487, USA. Ed. CRC Press Taylor and Francis Group 2014, pp. 185-214. International Standard Book Number 13: 978-1-4665-8554-6 (Book: Hardback). Purchased Book. It is not available from web site http://www.taylorandfrancis.com/. Only referenced.

[3] Lattice Semiconductor. User's Guide. ispMACH 4256ZE Breakout Board Evaluation Kit. March 2012 Revision: EB65_01.1. http://www.latticesemi.com/. Available from web site: http://www.latticesemi.com/en/Products/DevelopmentBoardsAndKits/ispMACH4256ZEBreakoutBoard.aspx.

[4] Lattice Semiconductor. Data Sheet DS1022. ispMACH 4000ZE Family. 1.8V In-System Programmable Ultra Low Power PLDs. August 2013. http://www.latticesemi.com/. From URL: http://www.latticesemi.com/en/Products/FPGAandCPLD/ispMACH4000ZE.aspx.

[5] Peter Alfke and Bernie New. Application Note. Serial Code Conversion between BCD and Binary. XAPP 029 October 27, 1997 (Version 1.1). Xilinx. Online available as XAPP 029.

[6] Lattice Semiconductor. Habel-HDL Reference Manual (ispLever Classic) 2003. Hillsboro, OR 97124. http://www.latticesemi.com/. Available from URL: http://www.latticesemi.com/en/Products/DesignSoftwareAndIP/FPGAandLDS/ispLEVERClassic.aspx

Dr. Gelacio Castillo Cabrera, Professor from Escuela Superior de Cómputo, Instituto Politécnico Nacional. Digital Systems and Computer Architectura are some of his courses taught at the Institute. Some of his published articles are (1): "CMOS prototype for retinal prosthesis applications with analog processing", and (2): "Performance evaluation of an architecture for the characterization of photo-divices: design, fabrication test on CMOS technology", (3): "Prótesis de retina: Innovación para el futuro", (4): "Procesamiento biológico: el desafío actual más importante". Many other of his works are find in proceeding. His main interest in research are "design of werable devices", "analog devices" and how teach them.

M. en C. Martha P. Jiménez V, Professor from Escuela Superior de Cómputo, Instituto Politécnico Nacional. Her main interest is on research for methodology for teaching mathematical in higher education. Many others of her published works are find in proceedings. She has some institutional books as "Matematicas discretas".

M. en C. Aurora Apericio C, Professor from Escuela Superior de Ingeniería Mecánica y Eléctrica, Instituto Politécnico Nacional. Her main interest is on research for analog devices and control applications. Some of her courses taught are on "analog systems" and "Control System". She has some institutional books as "Material de Apoyo para la Asignatura de Control". Many other works are find in proceedings.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.