Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 7 Issue 05, May-2018

Implementation of 32-Bit Interlock Collapsing Alu Multipliers in VLSI using VHDL

Pratibha Pandey
M. Tech Student
Department of Electronics & Communication Engineering
Dr. C.V. Raman University Kota, Bilaspur
C.G., India

Akanksha Awasthi
Assistant Professor
Department of Electronics & Communication Engineering
Dr. C.V. Raman University Kota, Bilaspur
C.G., India

***Abstract* - An important area in computer architecture is parallel processing. Machines employing parallel processing are called parallel machines. A parallel machine executes multiple instructions in one cycle. However, parallel machines have a limitation, they cannot execute interlocked instructions. They are executed in seriallike any serial machine. It takes more than one cycle to execute multiple instructions causing performance degradation. In addition there is hardware underutilization as a result of serial execution in parallel machine.**

The solution requires a special kind of device called “Interlock collapsing ALU”. The Interlock Collapsing ALU, unlike conventional 2-1 ALU’s is a 3-1 ALU. The proposed device executes the interlocked instructions in a single instruction cycle, unlike other parallel machines, resulting in high performance. The resulting implementation demonstrates that the proposed 3-1 Interlock Collapsing ALU can be designed to outperform existing schemes for ICALU, by a factor of at least two. The ICALU is implemented in VHL. Its functionality is verified through simulation.

Keywords: ALU, Interlock collapsing, ICALU, parallel processing, computer, architecture, parallel machines.

1. INTRODUCTION:BACKGROUND:

Parallel machines cannot execute interlocked instruction concurrently.Interlocked instructions or instruction with dependencies cannot be executed concurrently in a parallel machine, thus degrading the performance of the machine. The thesis investigates a solution, called, “interlock collapsing”, to exec ute these interlocks concurrently. The solution requires a special kind of a device called the Interlock collapsing ALU. The Interlock collapsing ALU, unlike conventional 2-1 ALU’s, is a 3-1 ALU.

The proposed ALU, in addition to collapsing these interlocks also should be implemented in identical stages as the conventional ALU’s. A functional model of the ICALU is assumed initially. The functional model is optimized by optimizing the model’s individual blocks. The design and optimization of each block is discussed in separate chapters.

Finally, two parallel machines with and without the ICALU are compared with regard to their execution times. The effect of variation of percentage interlocks in a given code on the execution times and the percentage speed ratio of the parallel machines is studied.

The ICALU is implemented in VHDL. Its functionality is verified through simulation.

2. ICALU DATAFW MODELLO

- THE INTERLOCK COLLAPSING ALU UNIT:

In this chapter all the designed components are put together to implement the ICALU. Also, ALU1 is created using the designed components. Finally, the Interlock collapsing ALU unit is implemented which consists of both ALU1 and ICALU. The chapter also estimates the relative delay.

2.1 Reduced Icalu Model :

Resulting from the design of the various stages in the preceding chapters a reduced ICALU is obtained. The result was the elimination of the multiplexers M2 and M3 and also better implementations of the Pre and Post-CLA Logic Blocks. The block diagram is shown in Fig 4.1. The program for ICALU is in the Appendix A.2.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

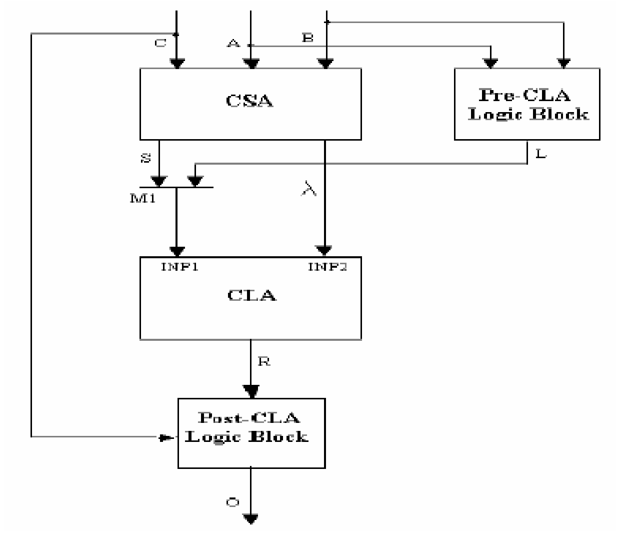

Fig. 2. 1 : Reduced Dataflow Model of Icalu

2.2 Alu1 Model :

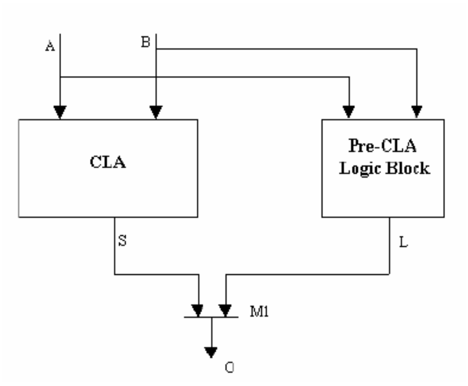

Fig 2.2 (Dataflow Model of ALU1)

The control signals for multiplexer are K_{12} and K_{13} and are set as follows :

I) CATEGORY 1 (ARITHMETIC) :
$K_{12} = 1$ and, $K_{13} = 0$;

Output of ALU1 = O = A ± B.

II) CATEGORY 2 (LOGICAL) :
$K_{12} = 0$ and, $K_{13} = 1$;

Output of ALU1 = O = A LOP B.

The values of control signals are summarized in Table 4.1 :

CATEGORY	K12	K13	O
1	1	0	A ± B
2	0	1	A LOP B

Table 2.3 : Output table for ALU1

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

2.2.2 Implementation :

The ALU1 is implemented using the block diagram above. The components CLA and PREBLK are the adder and the logic block respectively, for ALU1. The program for entity ALU1 is shown in A.3.

2.3 Interlock Collapsing Alu Unit :

The Interlock collapsing ALU unit consists of ALU1 and the ICALU operating in parallel. The block diagram of the Interlock collapsing unit was shown in Chapter 1, Fig 1.7. The program for entity ICUNIT is shown in A.2.

2.4 Estimation Of Relative Delay Between Alu1 And Icalu :

In this section the relative delay between the ALU1 in Fig 4.2 and the ICALU in Fig 4.1 is estimated. The relative delay is the difference between the delay of ALU1 and the ICALU. The delay is required to find out the instruction cycle length. The delay of a device can be estimated by taking a logic gate count from the input to the output. Only the delay between both ALU's considered because all other stages in their respective paths are identical, hence they also have identical delays.
Now, compare Fig 4.1 (ICALU) and Fig 4.2 (ALU1).

By elimination, it is deduced that the ICALU has two additional stages when compared to the ALU1 which are :

i) The CSA and,

ii) The Post-CLA Logic Block.

The procedure is :

1) The CLA and multiplexers are common to both the ALU's. Hence they can be eliminated.
2) The extra stages in the ICALU path are the CSA and the Post-CLA Logic Stage.
3) The Pre-CLA Logic stages are not considered because in case of ALU1 it is parallel with the CLA stage and has lesser stages than the same. Where as, in case of the ICALU it is in parallel and has the same delay as the CSA.

The logic delay of both stages are :

I) CSA :

To estimate this consider (3.13a) and (3.14) which represent the input-output transformations of the CSA sum and carry respectively. Both are in parallel.

$$\text{SUM} = S_i = A_i \vee B_i \vee C_i,$$

$$\lambda_{i+1} = K_2\, A_i\, B_i + K_1\, B_i\, C_i + K_1\, A_i\, C_i + K_3\, C_{i+1}.$$

(3.13a) and (3.14) can each be implemented in one gate delay using custom-built CMOS libraries. A ± 3 X 4 AO gate can serve this purpose ('+' represents AND-OR and '-' represents AND-OR-INVERT). The delay of this gate is assumed to be 1 gate stage as that of any other gate in the assumed libraries.

II) LOGIC DELAY OF POST-CLA LOGIC BLOCK :

Similarly, (3.9) (shown below) can be implemented in one gate delay by the AO gate.

$$L_i = \overline{L_{li}}\, KPRE1 + \overline{L_{ri}}\, KPRE1 + L_{li}\, L_{ri}\, KPRE2 + L_{li}\, L_{ri}\, KPRE3 \quad (3.9)$$

Thus

total relative gate delay of the ICALU over the ALU1 =

Logic delay due to CSA stage + Logic delay due to Post-CLA Logic Stage =1+1 = 2.

2.5 Determination of Instruction Cycle Lengths of a Machine With And Without Icalu :

The average instruction length is calculated to find out the speed of the machine. The instruction cycle length varies for each instruction. Hence an average instruction length has to be calculated. It is sufficient to take the average of only frequently executed instructions. The following discussion shows how the instruction lengths can be calculated for a given instruction. But first, Fig 2.4 is redrawn again.

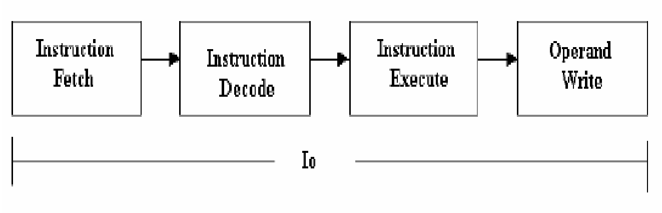

Fig2.4.5 (Phases of Instruction execution process)

Fig 4.3 represents the instruction path of serial machine. instructions given as I_0, or the basic instruction cycle time.

been discussed in Chapter 1.

The time to execute an The individual stage have

2.4.1 Without ICALU :

For a parallel machine there are two such paths in parallel. Fig 4.4 shows instruction execution (considering non-interlocked case) in a parallel machine with respect to time.

Fig 4.4 shows the instruction cycle of a parallel machine for a two-operand instruction pair shown below. The upper cycle in the figure represents execution of instruction 1. The instruction time is the same as the basic instruction cycle time, I_0. Execution of Instruction 2 is shown in the lower half. It starts a memory write cycle after the first instruction, because memory cannot be accessed simultaneously. It

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

shifts to the right by 1MW. The x's in figure represents an idle cycle.

ADD R1, R2 / Executed by ALU1 /

ADD R3, R4 / Executed by ALU2 /

The ID2 is smaller than ID1 by one memory access because we already have R2, fetched by Instruction 1. This compensates for the delay in start of execution of Instruction 2 and thus the execution cycles of both the instructions start at the same time. After the EX cycle is complete, Instruction 2 has to wait for 1MW for Instruction 1 to complete its memory access.

Instruction 2 takes a further 1MW to complete its cycle. Thus from the figure it can clearly be seen that the instruction time of a parallel machine is lengthened by 1MW.

2.4.1 With ICALU :

The instruction cycle in figure is for the pair given below :

ADD R1, R2

ADD R1, R3

The operation is almost similar to that of an ordinary parallel machine except that there is no memory access for ALU1. Hence the memory access starts once the ICALU completes it's execution which is two additional logic or gate delays more than the 2-1

ALU. Hence its instruction cycle time increases to $I_0 + 2$ D (D – Unit gate delay or the delay of one gate).

MW can be treated as three gate delays for CMOS memories. Substituting this value average instruction length can be calculated.

3. PERFORMANCE ANALYSIS

PERFORMANCE ANLAYSIS

In this chapter the performance of a Non-ICALU and that of a parallel machine with the ICALU is compared. Table 5.1 shows the average instruction lengths of a machine with ICALU and a Non-ICALU parallel machine for the interlocked and Non-interlocked categories. The average instruction lengths were calculated by taking the average of instructions lengths obtained for all possible interlocked and non-interlocked pairs (See Appendix B). The average instruction length is the time taken to execute an instruction pair, that is two consecutive instructions.

CATEGORY	AVERAGE INSTRUCTION LENGTH (NON – ICALU)	AVERAGE INSTRUCTION LENGTH (WITH ICALU)
Non–interlocked	$I_{PAVE1} = I_0 + 3.5$	IICAVE1 = $I_0 + 4.17$
Interlocked	IPAVE1 = 2I0	IICAVE2 = $I_0 + 2.63$

Table 5.1 : Average Instruction Lengths for machines with and without ICALU

Using the values in the table, the total execution time for each machine can be calculated, for a given number of instructions.

1) COMPARISON OF TOTAL EXECUTION TIME :

The total execution time of a parallel machine is given as :

TNI NNI + TI NI

Where,

T_{NI} = Time taken to execute a Non-Interlocked pair.

N_{NI} = Number of Non-Interlocked pairs.

T_I = Time taken to execute an Interlocked pair.

N_I = Number of Interlocked pairs.

Further,

$N = 2 (N_{NI} + N_I)$

$N_{NI} = ((100 – X) / 100) N / 2$, and

$N_I = (X / 100) N / 2$.

Where,

N = Total number of instructions to be executed.

X = Percentage of interlocked pairs.

Now, (5.1) can be rewritten as :

$T_{NI} [((100 – X) / 100) N / 2] + T_I [(X / 100) N / 2]$

Now, consider the following for a program :

a) N = 100,

b) X = 50 %

c) I_0 = 25 Logic Delays, typically

The execution times for the

machines are :

I) NON-ICALU MACHINE :

From Table 5.1 :

$T_{NI} = I_{PAVE1} = I_0 + 3.5$.

(5.1)

(5.1a)

$T_I = I_{PAVE2} = 2I_0$.

Substituting in (5.1a), we get,

$T_1 = (I_0 + 3.5) 25 + (2I_0) 25$

= 1962.5 Logic Delays.

II) MACHINE WITH ICALU :

Again from Table 5.1 :

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

$T_{NI} = I_{ICAVE1} = I_0 + 4.17.$
$T_I = I_{ICAVE2} = I_0 + 2.63.$

Substituting in (5.1a), we get,

Total execution time for 50 pairs of instructions,
$T_2 = (I_0 + 4.17) 25 + (I_0 + 2.63) 25$

= 1419.78 Logic Delays.

The machine with ICALU takes fewer logic delays than the Non-ICALU machine.

Chart 5.1 is a plot of (5.1a) with N constant (100) and varying X between 0 and 100 percent. It can be seen that the performance of the Non-ICALU machine degrades, where as the performance of the machine with ICALU is almost constant as X increases. This is because the Non-ICALU has to execute more and more instructions in serial. In the next section Percentage Speed Ratio is calculated.

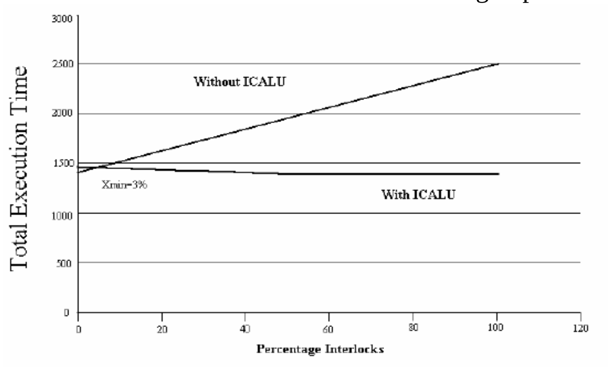

Fig 5.1 Percentage Interlocks Vs Total Execution Time

2) PERCENTAGE SPEED RATIO :
Percentage Speed Ratio of Machine 2 over Machine 1 is defined as :
[(T1 - T2) / T1] x 100 (5.2)

Percentage Speed Ratio reflects the time saved by one machine over the other.
Using (5.1a) in (5.2), we get,
$[(T_{NI1} – T_{NI2}) (100 – X) + (T_{I1} – T_{I2}) X] / [T_{NI1} (100 – X) + T_{I1} X]$
(5.2a)
Hence,
Percentage Speed Ratio of machine with ICALU over the Non-ICALU machine for the previous case (that is X = 50%) ≈ 28

Similarly, for (say) X = 75% :

Percentage Speed Ratio ≈ 37.
Thus the Percentage Speed Ratio increases as X increases.

Chart 5.2 shows variation of Percentage Speed Ratio with interlock percentage

(X). It can be seen clearly how Percentage Speed Ratio increases as interlock percentage

(X) increases.

From chart we can see that at X ≈ 3%, the gain of the machine with ICALU is zero. Below this point the gain is negative, that is the machine with ICALU is slower than the machine Non-ICALU machine. This point can also be obtained by setting Percentage Speed Ratio to zero in (10.2a).

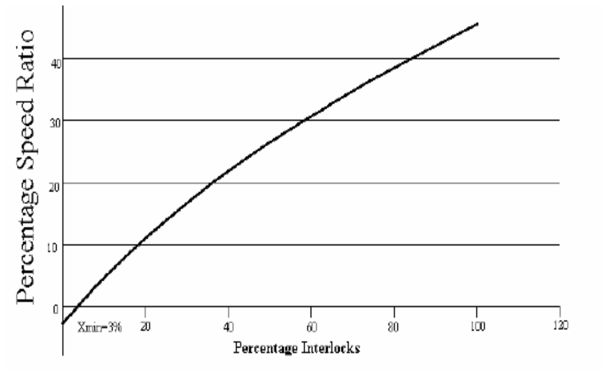

Fig : 5.2 (Percentage Interlock Vs. Percentage Speed Ratio.)

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

4 TESTING PROCEDURES TESTING

The ICUNIT has two outputs, result of ALU1 and that of ALU2.The testing of the ICUNIT was done by categories. They are as follows :

1) CATEGORY 1 (ARITHMETIC FOLLOWED BY ARITHMETIC) :

Since there are three operands, the four sub categories are :

i) All positive numbers.
ii) Two positive numbers.
iii) One positive number.
iv) None positive.

2) CATEGORY 2 (LOGICAL FOLLOWED BY ARITHMETIC) :

The sub categories are :

i) Logical AND followed by Arithmetic.
ii) Logical OR followed by Arithmetic.
iii) Logical XOR followed by Arithmetic.
iv) Logical NAND followed by Arithmetic.
v) Logical NOR followed by Arithmetic.
vi) Logical XNOR followed by Arithmetic.

3) CATEGORY 3 (ARITHMETIC FOLLOWED BY LOGICAL) :

The sub categories are :

i) Arithmetic followed by Logical AND.
ii) Arithmetic followed by Logical OR.
iii) Arithmetic followed by Logical XOR.
iv) Arithmetic followed by Logical NAND.
v) Arithmetic followed by Logical NOR.
vi) Arithmetic followed by Logical XNOR.

4) Category 4 (Logical followed by Logical) :

Category 2 and 3 cover all possible categories here. Hence only one subcategory is considered (say) :

Logical AND followed by Logical AND.

5. SIMULATION RESULTS

The simulation is conducted by assigning values to the variables in the design entities. The simulation is done through Modelsim XE II/starter 5.6e-Custom Xilinx Version. In Active-HDL a test run (simulation cycle) lasts for 100ns. The waveforms (resulting from the simulation) are displayed in waveform editor. The following pages show the simulation cycle as displayed by waveform editor.

The figures shown in the following pages depict the results of various categories of interlocked instructions explained in Chapter 6. A, B and C represents the three inputs to the ICUNIT. K1, K2, K3,…, K14 represents the di fferent control signals. The figures show consecutive simulation cycles. Their values are shown in hexadecimal in each cycle.

ARITHMETIC FOLLOWED BY ARITHMETIC OPERATIONS

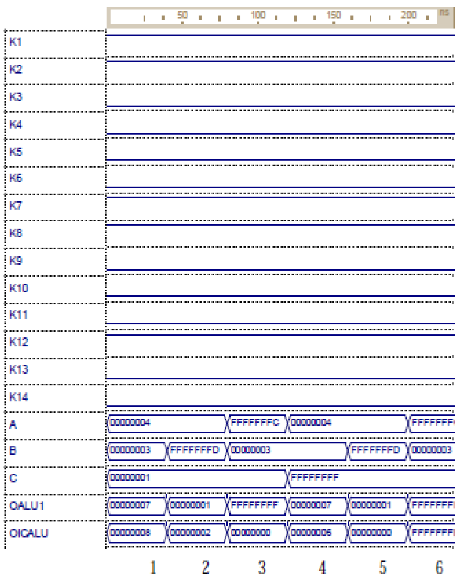

Fig 7.1

1. A + B + C
2. A – B + C
3. –A + B + C
4. A + B – C
5. A – B – C
6. – A + B – C

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

ARITHMETIC FOLLOWED BY LOGICAL OPERATIONS

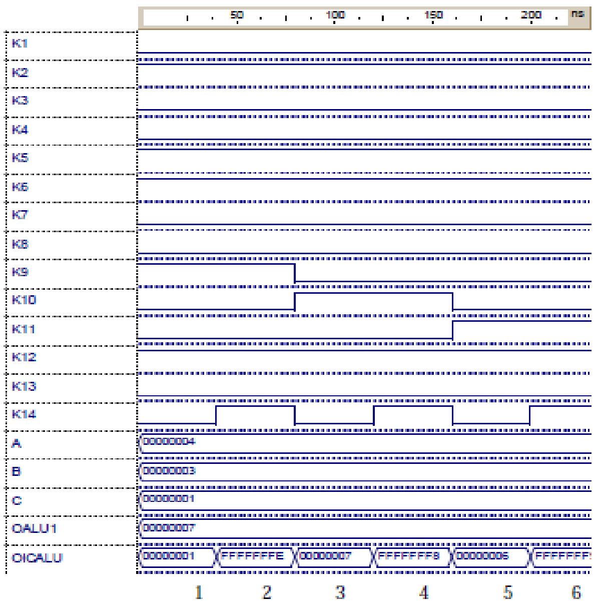

Fig 7.2

1. A + B and C
2. A + B nand C
3. A + B or C
4. A + B nor C
5. A + B xor C
6. A + B xnor C

LOGICAL FOLLOWED BY ARITHMETIC OPERATIONS

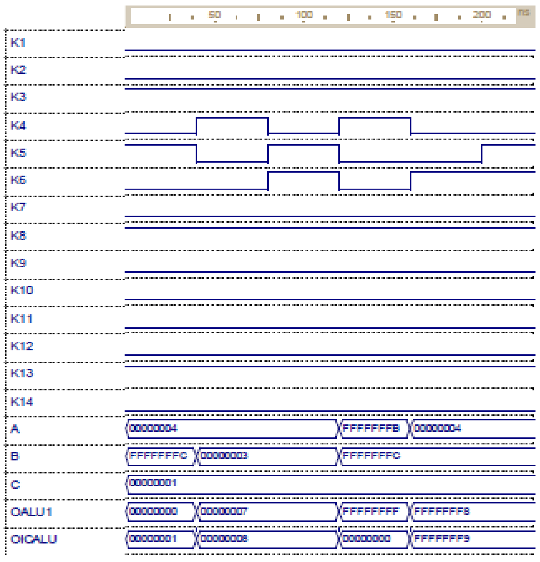

Fig 7.3

1. A and B + C
2. A or B + C
3. A xor B + C
4. A nand B + C
5. A nor B + C
6. A xnor B + C

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

LOGICAL FOLLOWED BY LOGICAL OPERATIONS

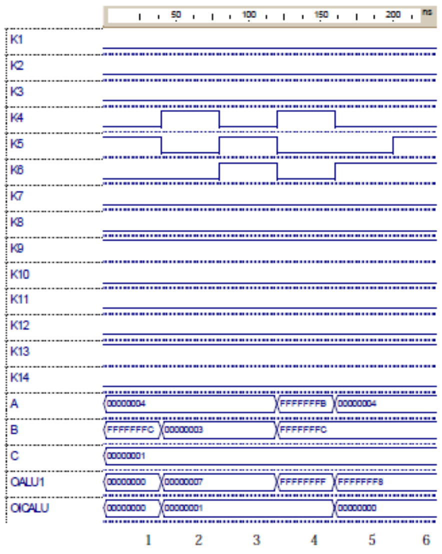

Fig 7.4

1. A and B and C
2. A or B and C
3. A xor B and C
4. A nand B and C
5. A nor B and C
6. A xnor B and C

CONCLUSION

The objective of the thesis, execution of interlocked instructions in one instruction cycle. This was achieved by ICALU successfully designed and implemented using VHDL. Its functionality was verified through simulation.

The ICALU can be implemented in just 2 logic delays more than that of a conventional 2-1 ALU. The performance of an ordinary (Non-ICALU) parallel machine and the machine with the ICALU incorporated in it, was compared.

The following is concluded from the performance analysis :

- The Percentage Speed Ratio of the machine with the ICALU over the Non-ICALU machine depends only on the amount of interlocked instructions in the code and not on the total number of instructions.
- The Percentage Speed Ratio increases as the number of interlocked instructions increase. This is due to the degradation in performance of Non-ICALU machines.
- Assuming an average of (50-75)% interlocks in a given code, the Percentage Speed Ratio obtained is between (23-37)%, which implies that the ICALU, when incorporated in a parallel machine saves up to a third of the total execution time of the Non-ICALU machine.

REFERENCE

[1] J. Phillips, S. Vassiliadis, "High-Performance 3-1 Interlock Collapsing ALU's," *IEEE Transactions on Computers*, vol. 43, no. 3, pp. 257-268, Mar., 1994
[2] D. W. Ruck, S. K. Rogers, M. Kabrinsky, M. E. Oxley, and B. W. Sutter, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function,"IEEE Trans. Neural Networks, vol. 1, no. 4, pp. 296-298, Dec. 1990.
[3] S. Vassiliadis, J. Phillips, and B. Blaner, "Interlock collapsing ALU's,"IEEE Trans. Comput., vol. 42, no. 7, pp. 825-839, July 1992.
[4] H. Ling, "High speed binary adder,"IBM J. Res. Develop., vol. 25, no. 3, pp. 156-166, May 1981.
[5] M. J. Flynn and S. Waser,Introduction to Arithmetic for Digital Systems Designers. CBS College Publishing, 1982, pp. 215-222.
[6] R. M. Keller, “Lookahead Processors,” ***Computing Surveys,***Vol. 7, No. 4, pp. 514-537, December 1973.
[7] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," <i>IBM J. Res. Develop.</i>, pp. 25-33, Jan. 1967.
[8] R D Acosta , J Kjelstrup , H C Torng, An instruction issuing approach to enhancing performance in multiple functinal unit processors, IEEE Transactions on Computers, v.35 n.9, p.815-828, Sept. 1986
[9] JAIN R.P . Digital Electronics , Printice hall
[10] The Low Carb VHDL Tutorial ,Bryan Mealy 2004

(This work is licensed under a Creative Commons Attribution 4.0 International License.)