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

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

A Review on VHDL Implementation of 32-BIT Interlock Collapsing ALU

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 serial like 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 VHDL. Its functionality is verified through simulation.**

Keyword- ALU, ICALU, VHDL, logic device, carry adder, arithmetic, instruction set, interlock etc.

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 execute 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 are studied.

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

2 PRELIMINARY DESIGN ISSUES OF ICALU

2.1 INTRODUCTION

Computers have markedly changed over the last decade. Features, performance, and memory sizes representing a computer that filled a room with equipment and cost millions of dollars a decade ago now sit on top of a desk. High performance computers are increasingly in demand in the areas of industrial automation, medical diagnosis, aerodynamics simulation, military defense, signal processing, artificial intelligence, expert systems and socioeconomic, among many other scientific and engineering applications. This revolution has been brought about by major improvements in computer architecture and processing techniques and the enabling technology of Very Large Scale Integration (VLSI).

This thesis involved developing a novel technique to speed up instruction execution in parallel computers. Before moving any further, an overview of basic computer components and its many related terms is necessary.

2.1.1 COMPUTER ARCHITECTURE

WHAT IS COMPUTER ARCHITECTURE ?
Computer architecture involves the design of various aspects of computer design such as memory design, bus structure, internal Central Processing Unit, instruction set and the hardware implementation of the machine.

The aim of a computer architect is to design a computer that meets the functional requirements as well as price and performance goals.

2.1.2 BASIC COMPUTER COMPONENTS

Computer architecture has changed incredibly over the years. One element has remained constant throughout the years, and that is the Von Neumann concept of computer design. Von Neumann architecture is composed of three distinct components(or subsystems) : a central processing unit (CPU), memory and input/output (I/O) interfaces. Fig 2.1 represents one of the several possible ways of connecting these components together.

- THE CPU
- CONTROL UNIT
- ALU
- MEMORY
- INPUT/OUTPUT INTERFACES

2.1.3 INSTRUCTION FORMAT

An instruction is a group of binary bits that tell the computer what has to be done. Any computer instruction has two parts ::

i) Opcode Opcode (stands for operation code) field determines the function of the instruction, i.e. it contains the operation that is to be performed by the CPU.

ii) Operand Operand is the data on which the intended operation is performed.

2.1.4 PARALLEL MACHINES

An important area in computer architecture is parallel processing. Machines (computers) employing parallel processing are called parallel machines. A parallel machine executes multiple instructions in parallel, in one cycle, compared to a serial machine (discussed so far) that can execute only one instruction. Thus a parallel machine is faster than a serial machine.

In a parallel machine, a number of execution units (ALU's) are connected in parallel, so that each unit is able to handle an instruction. But for practical reasons the number is limited to two. For example, if two such units are present in the processor, two instructions can be handled concurrently resulting in faster execution. Fig 2.5 shows a simple block diagram of a parallel machine unit.

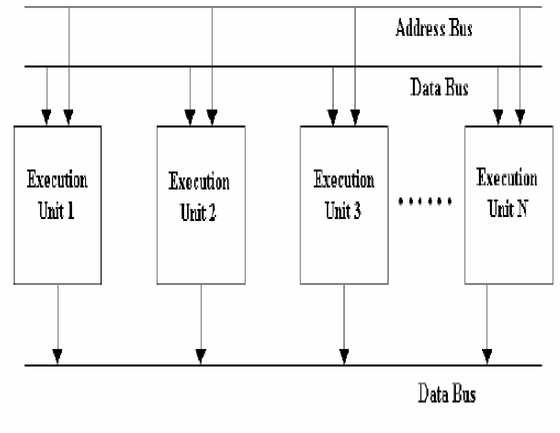

Fig 2.5 (A parallel unit)

- REGISTER ARRAY
- PC
- MAR AND MDR

However, parallel machines have a limitation, they cannot execute interlocked instructions (instructions with dependencies) in parallel [5, 6]. They are executed in serial like any serial machine. It takes more than one cycle to execute multiple instructions causing performance degradation in the machine. In addition, there is hardware underutilization as a result of serial execution in the parallel machine.

To improve performance it would be necessary to be able to execute these interlocked instructions in one cycle. Thus, interlock collapsing execution units in the form of multi-operand ALU's have to be employed.

2.1.5 OBJECTIVE

The thesis proposes design and simulation of a 32-bit 3-1 Interlock Collapsing ALU (ICALU), to allow the execution of two interlocked instructions in a single instruction cycle. This will improve the performance when it is degraded by data hazards. The device will be studied to find out if it meets it's objective which is to execute two interlocked instruction in one instruction cycle. The collapsing of interlocks will be confined to arithmetic and logical operations, on fixed point two's complement numbers.

2.1.6 INTERLOCKING IN PARALLEL COMPUTERS

WHAT ARE INTERLOCKED INSTRUCTIONS?

Instructions are said to be interlocked if an instruction depends on a previous instruction for its data so that they cannot be executed simultaneously. Consider the instruction pair of Fig. 2.6a.

i) ADD R2, R1 ; [R2] [R2] + [R1]

ii) ADD R3, R2 ; [R3] [R3] + [R2]

Fig 2.6a (An interlocked instruction pair)

Instruction 2 required the result of instruction 1 (stored in R2). Instruction 2 can be executed only after instruction 1 has been executed. Thus, instruction 2 is said to be dependent on instruction 1. The dependency prevents the simultaneous execution of the instructions.

i) ADD R1, R2 ; [R1] [R1] + [R2]

ii) ADD R4, R3 ; [R4] [R4] + [R3]

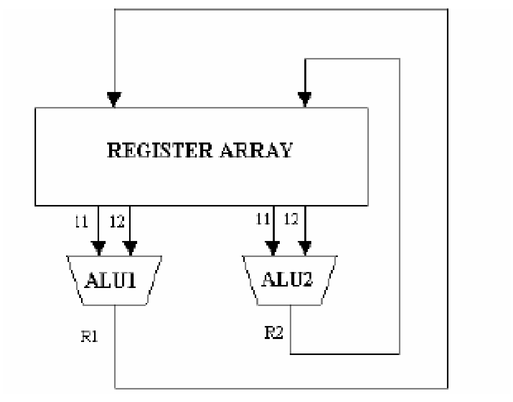

Fig 2.6b (A non-interlocked instruction pair)

Fig 2.6b is an example of a non-interlocked instruction pair. Instruction 2 does not require that instruction 1 be executed before it (instruction 2) is executed. Thus they can be executed simultaneously. Before moving on to the ICALU, consider the data flow of a parallel machine , which can execute two instructions concurrently Fig 2.7 shows the ALU unit for a parallel machine which consists of two 2-1 ALU's. The notation '2-1' stands for two input operands an d a single output (result). A 2-1 ALU has one 2-1 CLA (Carry Look Adder) to perform arithmetic operations and one logic stage to perform logical operations. The following explains the operation of the unit for both types of instructions.

I) NON-INTERLOCKED

ALU1 executes the first instruction and ALU2 executes the second simultaneously.

Thus, the total execution time is one cycle.

II) INTERLOCKED

Since, an interlocked instruction cannot be executed simultaneously, ALU1 executes both the instructions one after the other requiring two cycles.

To resolve these interlocks a solution had been proposed previously. This can be shown in Fig 2.8 in which the proposed dataflow of a implementation for relieving fixed point data dependency interlocks is shown.

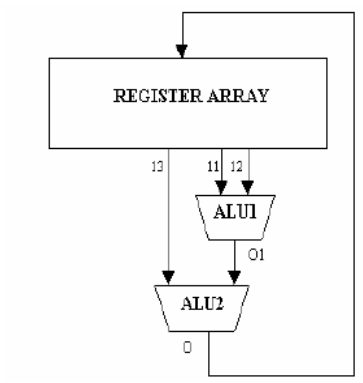

Fig 2.7 (Simplified view of the ALU unit for an ordinary parallel machine)

Two ALU's are concatenated as shown. It can result in the execution of a multi-operation instruction, however, it requires twice the execution time of a single ALU operation. An attempt to execute the interlock in a cycle could result in an increase in the cycle time of the machine and unnecessarily penalize all instruction executions, resulting in practically no performance gain.

2.1.7 THE ICALU

In order for an implementation to eliminate interlocks between instructions and to execute such instructions in parallel (in addition to execution of non-interlocking instructions in parallel), it is required to collapse the interlocks with the incorporation of

- ➢ Multiple execution units, and
- ➢ Multi-operand execution units.

Multiple execution units are required because more than one instruction is being executed at a time. The number of instructions that can be executed is assumed to be two here and hence the number of execution units (ALU's) is two.

Multi-operand execution units are required, since there are two interlocked sequential instructions. The first instruction is execute by a traditional ALU. Since the second instruction may be dependent on the first, the second ALU must be capable of performing the collapsed instruction of both the instructions, in parallel to the first ALU. The second ALU has three input operands, one in addition to that of first ALU.

1) DESCRIPTION AND WORKING

The ICALU is basically a 3-1 ALU. It has 3-1 CSA (Carry Save Adder) in addition to the 2-1 CLA to achieve the desired 3-1 arithmetic operation. The ICALU also has an extra logic when compared to the 2-1 ALU. The ICALU is implemented in the parallel machine by replacing ALU2 with the ICALU. The operation of the parallel machine employing the ICALU is explained as

FIG 2.8 (WM'S APPROACH TO COLLAPSING INTERLOCKS

i) NON-INTERLOCKED

The operation is the same as that for the parallel machine when the sequence is non-interlocked.

ii) INTERLOCKED

Consider the interlocked sequence of Fig2.6a. ALU1 executes the first instruction as usual. The ICALU collapses the two instructions into a single 3-operand instruction as shown in Fig 2.9 :

ADD R3, R2, R1 ; [R3] [R3] + [R2] + [R1]

Fig 2.9 (The Collapsed Instruction)

Thus, the above instruction is executed in a single cycle by the ICALU. In short, the ICALU operates on both the instructions when there is interlock and on the second when there is no interlock.

The design of ICALU is described and simulated in VHDL. VHDL is a language to describe or model hardware systems. The next section of this chapter gives a brief explanation on the same.
iii) LIMITATIONS:
At percentage of interlocked instructions(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.

3PRELIMINARY DESIGN ISSUES OF ICALU

3.1) PRELIMINARY DESIGN ISSUES OF ICALU

As discussed earlier, the ICALU performs a 3-1 operation in case of an interlocked sequence and a normal 2-1 operation in case of non-interlocked sequence. To design the ICALU we start with its functional requirements.

3.1.1) FUNCTIONAL REQUIREMENTS OF THE ICALU :

The functional requirements of the ICALU is divided into 2 modes :

1) Interlocked mode.
2) Non-interlocked mode.

1) MODE 1 (INTERLOCKED MODE) :

In Mode 1, the ICALU is in the Interlock mode, where it performs a three operand operation. Now, consider again an interlocked pair.

i)		ADD	
ii)		ADD	
A, B ; A, C ;			
[A]	[A]	+	[B]
[A]	[A]+		[C]

Fig. 3.1a (An interlocked instruction pair)

In Fig 2.10a an arithmetic operation follows an arithmetic operation. Similarly, there are other ways by which instruction can combine. The possible ways are categorized as follows:

i) Arithmetic followed by Arithmetic.

ii) Logical followed by Arithmetic.

iii) Arithmetic followed by Logical.

iv) Logical followed by Logical.

I) CATEGORY 1 : (ARITHMETIC FOLLOWED BY ARITHMETIC)

This category is represented by :

(A ± B ± C)

'±' addition or subtraction operation.

A – Operand 1

B – Operand 2

C – Operand 3

e.g.,	ADD	A,B
SUB		A,C

II) CATEGORY 2 : (LOGICAL FOLLOWED BY ARITHMETIC)

This category is represented by :

(ALOPB)±C

('LOP' – Logical Operation)

e.g., ANDA, B

ADDA, C

III) CATEGORY 3 : (ARITHMETIC FOLLOWED BY LOGICAL)

This category is represented by :

(A ± B) LOP C

e.g., ADDA, B

ANDA, C

IV) CATEGORY 4 : (LOGICAL FOLLOWED BY LOGICAL)

This category is represented by :

(A LOP B) LOP C

e.g., ANDA, B

XORA, C

2) MODE 2 (NON-INTERLOCK MODE) :

In Mode 2 the ICALU is in the 'Non-Interlocked' mod e. It performs a two operand operation. Consider a Non-Interlocked pair.

i) ADD

ii) ADD
A, B
D, C
[A][D]
[A]+[B]
[D]+[C]

Fig 3.1b (An non-interlocked instruction pair)

Since no interlock exists between the two instructions, no collapsing is required. Thus ICALU executes only instruction 2. The categories in this mode are

i) Arithmetic.

ii) Logical

I) CATEGORY 1 (ARITHMETIC)
This category is represented by : A±B
This category can be executed as a Mode 1 – Categor y 1 instruction if the third operand is forced to zero. It can be illustrated as :
(A±B±C) = (A ± B), when C = 0.

e.g., ADD A, B / Executed by ALU1/

SUB C, D

II) CATEGORY 2 (LOGICAL) :

This category is represented by :

A LOP B

Similarly this category can be executed as Mode 1 – Category 2 instruction, by forcing the third operand, C, to zero, as illustrated below :

(A LOP B) ± C = (A LOP B) ; when C = 0 e.g., ADD A, B / Executed by ALU1 / AND C, D
From the above discussion it follow that :

- Category 1 is a 3-1 arithmetic operation, which requires a 3-1 adder. This can be achieved by cascading a 3-1 CSA followed by a 2-1 CLA.

- Category 2 & 4 require a logic stage before the adder stage. This is the Pre-CLA Logic Block.

- Categories 3 & 4 require a logic stage after adder stage. This is the Post-CLA Logic Block.

- The Mode 2, Non-interlocked operations is just a subset of Mode 1, Interlocked operations. They are executed as Mode 1 operations by setting the third operand to zero and hence do not require any additional circuitry within the ICALU.

Taking into consideration, all the above requirements, a logical dataflow model of the ICALU is developed. It is shown in Fig 2.11.

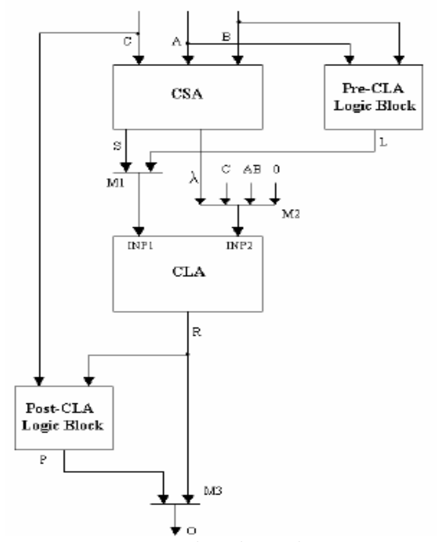

Fig 3.2 (Data flow form of ICALU)

3.6 Binary Adders And Arithmetic
The arithmetic unit of the ICALU is implemented by binary adders. An introduction to binary adders is necessary. Binary subtraction and two's complement numbers are also explained in this chapter.
3.6.1 BINARY ADDERS:

1) FULL ADDER :

A	B	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 3.7 : Truth table of a Full Adder

The expressions for full adder are :
$S = a \veebar b \veebar c$ (3.10)
Cout = a b + b Cin + a Cin (3.11)

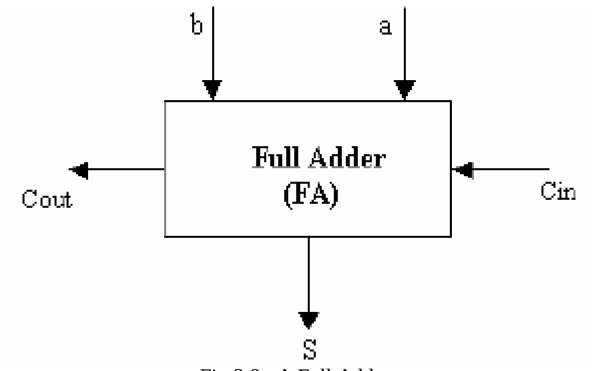

Fig 3.8 : A Full Adder

2) RIPPLE CARRY ADDER:

One of the most basic adders is the ripple carry adders. The addition is similar to that of paper and pencil addition. A block diagram to add two 4-bit binary numbers is shown in Fig2.2. The carry is allowed to ripple from one stage to another. However the ripple carry is the slowest, because the carry has to propagate from the least significant bit(LSB) to the most significant bit(MSB). Hence it is not used for larger adders.

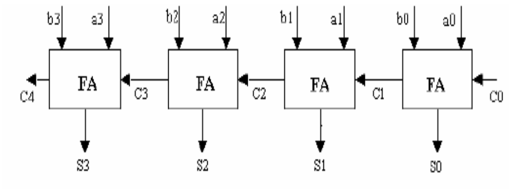

Fig 3.9: A 4-Bit Ripple Carry Adder.

4)CARRY SAVE ADDER (CSA):

The CSA is used when more than two numbers are to be added.

For example, Consider addition on three numbers (X,Y,Z).

	0101	X
	0011	Y
+	0100	Z
	0010	Partial sum
	1010	Saved Carry

In the next step, the sum and saved carry are added with each other.

	0010	Partial Sum
+	1010	Saved carry
	1010	Final sum

In the last step the CLA is used to add the partial sum with saved carry.

Fig 3.4 shows the block diagram of the addition process. The first stage is the CSA. The carry consists of a chain of full adders. A full adder is present for each significant bit position. Unlike the ripple adders, in carry look ahead adders carry is saved for the next stage.

y3 x3 z3 y2 x2
z2 y1 x1 z1
y0 x0 z0

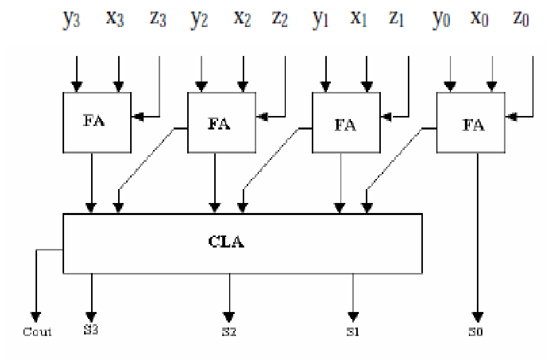

Fig 3.11 : A 4-Bit Carry Save Adder

3.7 Design of Csa Stage

In the previous chapters various adders were discussed. The CSA was also explained. The CSA stage for the ICALU not only has to generate the sum and carry for the next stage, but also the inputs for INPUT 2 of CLA. This is designed and implemented in this chapter.

3.7.1 DESIGN:

In the last chapter we saw the block diagram for three 4-bit inputs using a CSA . It can be extended to 32 bits. The equation for sum and carry are:

SUM = $S_i = A_i \vee B_i \vee C_i$ CARRY = $\lambda_{i+1} = A_i B_i + B_i$

for $(0 \leq i \leq 31)$ (3.13a)

$C_i + Ai\, C_i$,for $(1 \leq i \leq 31)$ (3.13b)

A_i, B_i, C_i are the i th bits of operands A, B & C respectively.

As explained in chapter 3 the carry out of CSA is designed to provide all inputs for INPUT 2 of CLA. It is given as:

$\lambda_{i+1} = K_2 A_i B_i + K_1 B_i C_i + K_1 A_i C_i + K_3 C_{i+1}$,for(1<= i <= 31)(3.14)

3.7.3 IMPLEMENTATION:

The (3.13a) and (3.14) are bit-wise expressions. They are the basic building blocks of the CSA. First, they are implemented as individual components (blocks). Later, they are instantiated the required number times to obtain the CSA.

3) IMPLEMENTATION OF CSA :

The entities SUM3_1 and CSA-CARY are the basic building blocks of the CSA. The sum is generated for bit positions 0 to 31 where as the carry is from 0 to 30. Since, the additions considered are in 2's complement the final carry (i.e., bit-position 31) is discarded. Thus, carry spans one bit positions lesser when compared to sum. The component SUM3_1 is instantiated 16 times for each bit position to obtain sum. The 'for generate' statement is used to repeat the instantiations for the desired number of times.

3.8) Design Of CLA Stage

The basic of a CLA was explained in chapter 5. using the same principle, it is extended to 32 bits in this chapter.

3.8.2) Implementation:

The 32-bit CLA can be implemented by creating an 8-bit CLA first. Later, the 8-bit CLA' are connected together to obtain the 32-bit CLA.

1) Implementation of 8-bit CLA:
The 8-bit CLA is obtained using the preceding expressions. The sum block for the CLA are obtained by instantiating the component SUM3_1. It has already been discussed in chapter 6. There are eight sum and carry blocks for each of the 8-bit positions. Two CLA's have been implemented here, CLA_1 and, CLA_2. Though, both are 8-bit, CLA_1 is slightly different from the other CLA's, since it does not have an input carry. It has been created as a separate component.
2 IMPLEMENTATION OF 32-BIT CLA:

The 32-bit CLA is obtained by installing the four 8-bit CLA's and generating the intermediate carries C_{16} and $C_{24,}$ appropriately. The final carry C_{32} is discarded, since all operations are in two's complement. The program for 32-bit CLA is shown in A.4.

3.9 DESIGN OF POST – CLA LOGIC BLOCK

This chapter deals with the design of the Post-CLA Logic Block of the ICALU. The Post-CLA Block performs logic operations between the third operand and the result of the operation on the first two operands. The Post-CLA Logic Block is not similar to the Pre-Block because the inverting inputs are not readily available.

Control Signal	*Description*
FAND2	AND Inputs R_i & C_i
FOR2	OR Inputs R_i & C_i
FXOR2	XOR Inputs R_i & C_i
FINV2	Inverts above operations.

Table 3.12 : Control signals to Post-CLA Logic Block.

3.9.1 DESIGN

The control signal format remains the same as that for the Pre-CLA Logic Block. But the control unit generates a separate set of signals for the Post-CLA Logic Block. Now, L_i represents the output of the ICALU.

FAD D	FAN D	FO R	FXO R	FIN V	OUTPUT (L_I)
1	0	0	0	0	Ri
0	1	0	0	0	Ri Ci
0	0	1	0	0	$R_i + C_i$
0	0	0	1	0	$R_i \veebar C_i$
0	1	0	0	1	$\overline{R\iota\, C\iota}$
0	0	1	0	1	$\overline{R_\iota + C_\iota}$
0	0	0	1	1	$\underline{R_\iota \veebar C}$

TABLE 3.13: OUTPUT TABLE OF POST-CLA LOGIC BLOCK.

3.9.2 IMPLEMENTATION :

The implementation of Post-CLA Logic Block is also similar to earlier implem entations, that is, CSA, Pre-CLA Logic Block, etc. First, the bit-wise component is implemented and later instantiated to obtain the Logic Block.

1) IMPLEMENTATION OF BIT-WISE LOGIC COMPONENT :
The bit-wise logic component, P_CLA_BCMP, is the implementation of (3.20). The program is shown in A.8.2.

2) IMPLEMENTATION OF POST-CLA LOGIC BLOCK :
The logic block is implemented by instantiating P_CLA_BCMP for bit positions 0 to 31. The program is shown in A.8.1. The program creates entity P_CLA_LOGBLK.

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 functional 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