Pipeline Thread Construction Using VLSI

R.Mahalakshmi, Mr.C.Mukuntharaj, Dr.K.Ramasamy

Abstract— This paper describes the design of reconfigurable architecture that is to be used for image processing, multimedia processing and wireless communication application. This reconfigurable architecture provides high computing power, flexibility and scalability through processing level pipe-lining and improved functionality. The main aim of this paper is to perform many processing with in a small number of clock cycles and thus increasing the pipelining speed..

Index Terms— Functionality, Image processing application, Pipelining, Processing array, Processing Element, Processing level pipelining and Reconfigurable architecture

1 INTRODUCTION

Currently image processing technology, multimedia communication and wireless communication application has been widely used. This developing technology requires high computing power, flexibility, reliability and scalability. An application specific integrated circuits (ASIC) satisfies the high computing power but it is inflexible and scalable. General purpose multiprocessors are flexible but it insufficient to provide computing power. Yun Yang [1] [2]Reconfigurable architecture has been proposed. The proposed architecture retrieves the shortcoming of existing system through processing level pipelining and improved functionalities. Reconfigurable architecture supports processing level pipelining and thus provides flexibility and scalability to many processes.

This project focus on the development of a synthesisable VHDL design for processing level pipelining. Generally pipeline is a set of processing elements connected in serial, where the output of one element is the input to the next element. These elements are executed in parallel or time sliced manner. There are many kinds of pipelining, are processing level pipelining, instruction level pipelining, graphics pipelining, software pipelining and so on. Graphics pipelining is to apply the windowing, clipping, colouring, light calculation and rendering technique to the image, it is possible to image processing application. Software pipelining is used for web development. Instruction level pipelining is a RISC processor which performs many instructions with in a same cycle. Processing level pipelining means processing many functions within a small number of clock cycles. Here the processing level pipelining is used to reconfigure the architecture, which provides high computing power, high speed and flexibility.

Here the simulation of processing level pipelining is implemented using VHDL tool. VHDL is a hardware description language which describes the hardware as different levels such as behavioural, structural and logical level. The main thing of VHDL is, it is a strongly typed language, can not mix other types.

2 ARCHITECTURE

The overall architecture consists of input switch, output switch, local memory, processing array. The processing array consists of number of inter-linked processing elements. The input and output switch is nothing but a multiplexer. It just useful for getting input and producing output.

- *R.Mahalakshmi is currently pursuing masters degree program in computer and communication engineering in P.S.R.Rengasamy college of engineering for women, India. E-mail: mahalak.7@mail.com*
- *C.Mukuntharaj is currently working as a assistant professor in electronics and communication engineering in P.S.R.Rengasamy college of engineering for women, India. E-mail: mukuntharaj@psrr.edu.in*
- *K.Ramasamy iscurrently working as Principal and Professor in electronics and communication engineering in P.S.R.Rengasamy college of engineering for women,India. Email: ramasamy@psrr.edu.in*

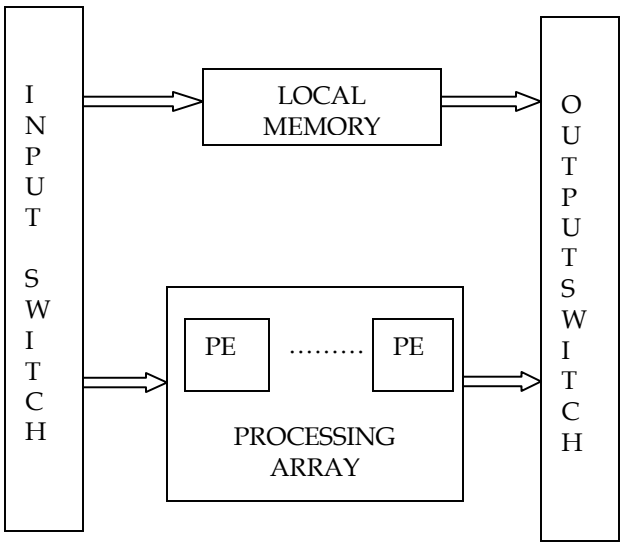

Fig: 1 Overall Architecture.

The local memory is used to store the temporary data. The processing element is capable of performing basic arithmetic and logical operation. The input 8 bit data is given to the both local memory and processing array. The local memory stores the data before beginning of operation. The processing array consist of number of processing element connected in serial, each processing element performs the basic operations based on image processing and multimedia application. There are 28 basic operations are discussed here. After processing each operation, the output of 8 bit data was produced. The overall architecture shown in fig 1.

2.1 Processing Element

Each processing element is capable of performing basic arithmetic functions, and has a small amount of local storage. Each processing element has two 8-bit registers, named A and B. Register A acts as an accumulator, storing the result of arithmetic operations.

The current value stored in register A which is accessible to the neighboring processing units following each clock cycle. The B register supplied additional operands, which load the value of an adjacent. Accumulator values can be used in future calculations. There is also a stack that is shared between A and B registers. It has a small local storage. The synthesis of one PE is shown in Figure.2 this schematic shows the large number of logic elements that are required for every PE in the system.

Internal Signals of processing element:

1. Accumulator A (8 bits):

The arithmetic and logical operations are stored in accumulator.

2. Register B (8 bits):

Register B is used to provide an additional operand for arithmetic and logical operations.

3. Stack (16x8bits):

The stack gives FILO storage if additional storage is required. It is possible to push values to the stack from A or B, and pop values from the stack to A or B.

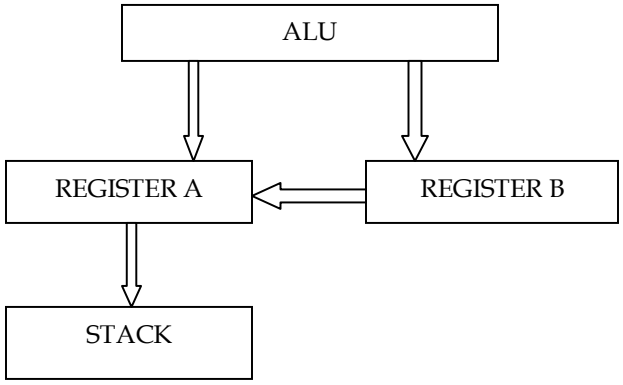

Fig2: Processing Element

4. Enable Bit:

Each processing element is simultaneously performing an identical operation. Enable signal is internal to the PE, and is set depends upon the value of accumulator A, through the Zero Flag. There is a single instruction to re enable all PEs, ensuring a known configuration.

5. Zero Flag:

The zero flag is set when the value of accumulator A is zero. It is possible to disable PEs.

The following table describes the function of processing elements. This table has three fields, the first field is op-code, second one is instruction which is the basic instruction of image processing and multimedia processing, the last field is description of instruction which performs, depends upon the op-code the instructions are performed.

Table: 1 Processing Element Instruction Set

Op-code	Instruction	description
0	NOP	No operation
1	Copy BA	Copy from B to A
2	Copy AB	Copy from A to B
3	OR	Bitwise Or
4	AND	Bitwise And
5	XOR	Bitwise Xor
6	Not of A	Inverse of A
7	Sum	Add B to A
8	Sub	Sub B to A
9	Abs A	Absolute value of A
10	Clr A	Clear A
11	Clr B	Clear B
12	Load up	Load from PE above into B
13	Load down	Load from PE below into B
14	Load left	Load from PE left into B
15	Load right	Load from PE right into B
16	swap	Swap value of A and B

17	Double A	Double the value A
18	Halve A	Halve the value A
19	Max	Maximum of A and B
20	Min	Minimum of A and B
21	Push A	Push the value of A to stack.
22	Pop A	Pop the value of A from stack.
23	Push B	Push the valve of B to stack.
24	Pop B	Pop the valve of B from stack.
25	Inc A	Increment A
26	Dec A	Decrement A
27	Disable Z	Disable PE with a=0
28	Enable All	Enable all PEs
29	SLT	Set Less Than

This table describes the 29 operations, among 29 operations AND, OR, XOR, NOT A are logical operations, remaining are general arithmetic operations. The push and pop are stack operations.

2.2 Processing Array

Processing array consist of a large number of linked processing elements. There are different configurations of processing elements generated in each corner or unique edge of the grid structure, where N is the dimension of the processing array, that is, an N-size array contains N2 processing elements. This prevents a processing element from attempting to read inputs from elements that do not exist. For example the "RT" (Right-Top) processing element does not have an in right or in above port instantiated. In that situation, there are not appropriate neighboring ports to connect to.

Table: 2 Generation of processing array

```
Algorithm1:

Generation of Processing Elements :
G1 : for m in N1 downto 0 generate In X direction
G2 : for n in N1 downto 0 generate In Y direction
This example display one of the nine configurations.
LT: if ((m=0) and (n=0)) generate

Generate Left Top unit .
apu0 : pu port map(
operation =>gInstruction , Same for all Pes
clk=>clk ,
dummy=>gDummy(N←n+m),
output=>dout (m , n )

Read by adjacent units .
in_down=>dout ( m , n +1) ,
Only two directional connections .
in_right=>dout(m+1,n),
in_load=>dout(m+1,n),
For loading data out_load=>gOut, Unique to LT
);
```

```
end generate LT;
end generate G2;
end generate G1;
```

A three PE by three PE array has to be used to simply the operation. For basic operations, while larger systems have been synthesized when required. In addition to the required changes to the directional input port configuration for each of these nine cases, the Left Top (LT) element, and the Right Bottom (RB) element are used to output and input image data for the entire grid, and they are connected to the outer structure of the processing array directly.
This algorithm is similar for all cases of processing element. The required processing elements are generated through a pair of nested generate statements. When the synthesizer repeats the process through the nested loops, PEs of the required configuration are generated and linked via an array of output signal

3 SIMULATION RESULT

3.1 Processing Element Output

3.1.1 Load up to b

This simulation shows that the value of in_up loaded on to the b register. The b register value moved to the accumulator "a" register.

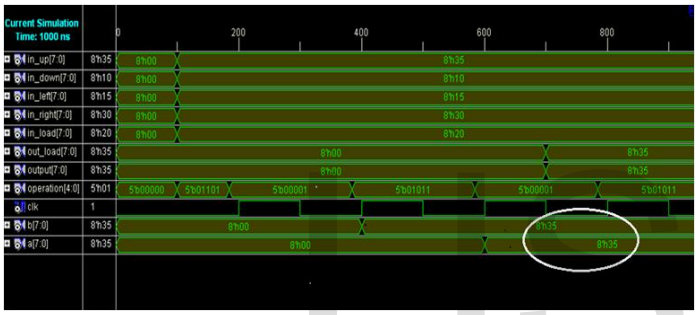

Fig: 3.1 output of processing element

Table: 3.1 Input ports and its value

Input ports	values
In_up	8h'35
In_down	8h'10
In_left	8h'15
In_right	8h'30
In_load	8h'20

3.1.2 Load left to b

This simulation shows that the value of in_load loaded on to the b register. The final result stored in accumulator "a" register. Input and output values are listed in the table 3.1 and 3.2

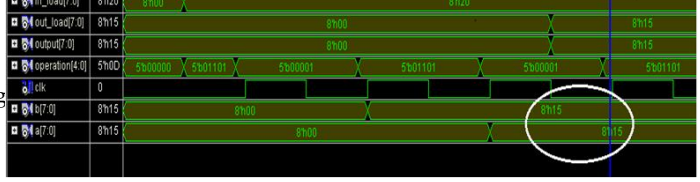

Fig: 3.1 output of load left to b

Table: 3.2 Output ports and its value

Output port	Load up to b	Load left to b
Reg b	8h'35	8h'15
Reg a	8h'35	8h'15

3.1.3 Overall Processing Element Output

This simulation shows that the overall output of processing element. It consists of two registers for storing the result. The two registers are accumulator register and b register. The final output stored in the accumulator "a" register. The circle shows that, when the op code is 11001, the value of "a" register gets increased. Likewise the general 29 operations performed.

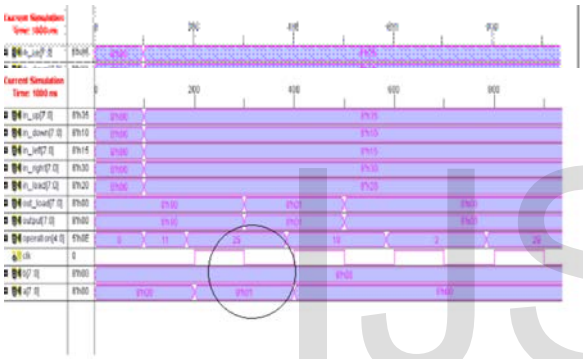

Fig: 3.1 output of processing element

3.2 Processing Array output

This simulation shows that the 28 operations of the processing array. Here the result stored in the b register. Then the final result stored in the accumulator "a" register. The circle shows that processing level pipeline is achieved after some basic operations (load, shift and move).

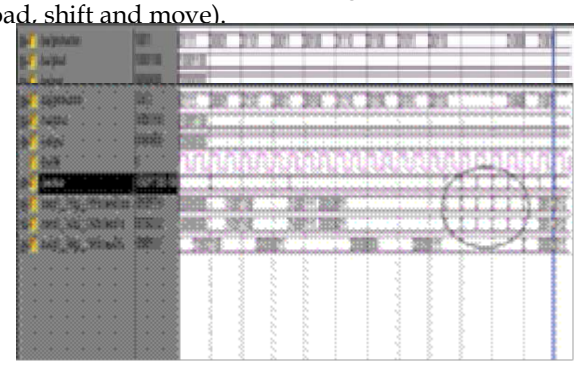

Fig: 3.2 output of processing array

In addition operation, the each process is added with previous process. I.e. during arithmetic operation the result of the previous one is added to the next processing. It took single pulses for single processing. This is known as processing level pipe-lining. The purpose of processing level pipe-lining is, it perform many processing within a short number of pulses.

3.3 Synthesis Result

RTL Top Level Output File Name: PA.ngr

Top Level Output File Name: PA

Output Format: NGC

Optimization Goal: Speed

Keep Hierarchy: NO

Design Statistics

IOs: 247

Cell Usage :

BELS: 99563
BUF: 98
GND: 1

LUT2: 7440
LUT3: 8427
LUT3_D: 1350
LUT4: 64700
#LUT4_D: 675
LUT4_L: 5625

MUXCY: 5175
MUXF5: 4271

VCC: 1
XORCY: 1800

Flip-flops/Latches: 32400
FDE: 32175
FDPE: 225

Clock Buffers: 1
BUFGP: 1
IO Buffers: 21
IBUF: 13
OBUF: 8

4 Conclusion

Reconfigurable architecture achieves processing level pipe-lining and improves functionalities for multimedia and image processing application. This implemented system shows that a system consisting of a parallel processing array. It is made up of a large number of interconnected Processing Units and it achieves high level of theoretical image processing performance, for performing many processing in a small number of clock cycles. This sort of architecture would be especially effective when operating on larger image sizes, as many common image processing tasks can be performed at a reduced order compared to serial processing.

With the current instruction set, commercially available FPGA hardware does not offer sufficient logic area to implement this generalized a parallel architecture on a realistic scale, as even relatively small processing arrays will consume the entirety of

available logic resources of current FPGA hardware. Unfortunately, with current technology, there are still significant compromises that must be made to the generalized capabilities of individual processing elements, to ensure it is possible to implement this kind of massively parallel architecture at useful image sizes. The chosen instruction set was relatively complex compared to other research approaches, and it is possible that further simplification of the PEs abilities in favor of a greater level centralized logic would allow for improved scaling.

5 FUTURE WORK

This project has demonstrated the possibility of developing a general purpose image processing architecture through the application of massively-parallel grid architecture. In this case the architecture's utility was benchmarked through the use of the Xilinx iSim simulator. It is expected that future work would include the synthesis of this design onto physical FPGA hardware. And introducing of instruction level pipe-lining also expected, further to increase the speed of pipelining.

REFERENCES

[1] Yun Yang,"Three-dimensional Image Processing VLSI System with Network- on-chip System and Reconfigurable Memory Architecture",IEEE Transactions on Consumer Electronics, Vol. 57, No. 3, August 2011.

[2] Ashish khodwe, Chandrashekhar Bhoyar,"Design and Implementation of FPGA Based Bidirectional Network-on-Chip Router through Virtual Channel Regulator",International Journal of Application or Innovation in Engineering & Management (IJAIEM), Volume 2,Issue 6, June 2013 .

[3] Gert-Jan van den Braak "GPU-CC: a Reconfigurable GPU Architecture withCommunicating Cores" SCOPES '13, June 19-21, 2013

[4] Hyo-Eun Kim, Student Member, IEEE, Jae-Sung Yoon, Student Member, IEEE, Kyu-Dong Hwang, Student Member, IEEE, Young-Jun Kim, Student Member, IEEE, Jun-Seok Park, Student Member, IEEE, and Lee-Sup Kim,"A Reconfigurable Heterogeneous Multimedia Processor for IC-Stacking on Si-Interposer" IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 22, NO. 4, APRIL 2012

[5] Mr. Sumedh. S.Jadhav, Prof. C.N.Bhoyar "Implementation of Embedded Multiprocessor Architecture Using FPGA " International Journal of Scientific & Engineering Research, Volume 3, Issue 1, January-2012.

[6] A. Ruta, R. Brzoza-Woch, and K. Zielinski, "On Fast Development of FPGA-based SOA Services," Design Automation for Embedded Systems, vol. 16, pp. 45–69, Mar. 2012.

[7] A. Nieto, V. Brea, D. L. Vilariño, and R. R. Osorio, "Performance Analysis of Massively Parallel Embedded Hardware Architectures for Retinal Image Processing,"EURASIP Journal on Image and Video Processing, vol. 2011, p. 10, Sept. 2011.

[8] A. Fijany and F. Hosseini, "Image Processing Applications on a Low Power Highly Parallel SIMD architecture," in Aerospace Conference, 2011 IEEE, pp. 1 –12, march 2011.

[9] B. Krill, A. Ahmad, A. Amira, and H. Rabah, "An Efficient FPGA-based Dynamic Partial Reconfiguration Design flow and Environment for Image and Signal Processing IPcores," Signal Processing: Image Communication, vol. 25, pp. 377–387, June 2010.

[10] T.Kurafuji, M.Haraguchi, M.Nakajima, T.Gyoten, T.Nishijima, H.Yama- saki, Y. Imai, M. Ishizaki, T. Kumaki, Y. Okuno, et al., "A Scalable Massively Parallel Processor for Real-time Image Processing,"inSolid-State Circuits Conference Digest of Technical Papers(ISSCC), 2010 IEEE International, pp. 334–335, IEEE, 2010.

[11] N. Roudel, F. Berry, J. Serot, and L. Eck, "A New High-Level Methodology for Programming FPGA-Based Smart Cameras," in Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on, pp. 573 –578, Sept. 2010.

[12] J. Xiong and Q. Wu, "An Investigation of FPGA Implementation for Image Processing," in Communications, Circuits and Systems (ICCCAS), 2010 International Conference on, pp. 331 –334, july 2010.

[13] Y.Hu and H.Ji, "Research on Image Median Filtering Algorithm and Its FPGA Implementation," in Intelligent Systems, 2009.GCIS'09.WRIGlobal Congress on, vol. 3, pp. 226 –230, may 2009.

[14] M. Prieto and A. Allen, "A Hybrid System for Embedded Machine Vision using FPGAs and Neural Networks," Machine Vision and Applications, vol. 20, no. 6, pp. 379– 394, 2009.

[15] Pei-Yin Chen, Chih-Yuan Lien, and Chi-Pin Lu, "VLSI Implementation of an Edge- Oriented Image Scaling Processor" IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 17, NO. 9, SEPTEMBER 2009.