IOSR Journal of Electronics and Communication Engineering (IOSRJECE)
ISSN : 2278-2834 Volume 1, Issue 1 (May-June 2012), PP 36-42
www.iosrjournals.org

Design of a Control logic in a Dynamic Reconfigurable System

Prasannakumar.A[1], Naga.V.Satyanarayana.Murthy[2]
[1]((Electronics and communication, Amrita University, India)
[2](Assistant Professor, Electronics and communication, Amrita University, India)

ABSTRACT : *In this paper, we propose an architecture for controlling Dynamic Reconfigurable systems. The processor instructions when compiled one by one produces very high delay overhead. If these instructions are converted into a combinational logic then the overhead can be reduced thereby making the system an efficient one. This paper presents a method for creating the control logic necessary for performing operations on instructions. The proposed design is simulated using Modelsim 10.0c.The area and power constraints are evaluated using Synopsys Design Compiler.*

Keywords - *Functional Unit, MIPS, Multistage Interconnection Network, Omega MIN, Reconfigurable system.*

I. INTRODUCTION

A reconfigurable system must be able to execute large number of instructions simultaneously into the reconfigurable logic. So a fast communication between the reconfigurable components can be achieved. In order to implement a low cost communication, huge interconnection networks have been used that guarantees fast routing and enhanced performance.

Designing a cost effective interconnection network is one of the major problems while designing reconfigurable functional processing units. A Multistage Interconnection Network (MIN) is composed of several stages of switches by which any network input can be connected to any output. In this work, we use Omega MIN for transferring data from the processor to the Functional Units thereby converting the sequence of instructions into a combinational logic.

In this paper, we propose a new architecture in which the instruction sequence are converted into frame formats that can used for performing basic operations in hardware. These operations are carried out using the Functional Units. The area and power analysis of the proposed architecture is also evaluated.

II. DYNAMIC RECONFIGURABLE SYSTEM

The Proposed system is a reconfigurable architecture (shown in Fig 3) by which the processor instructions are converted into a combinational logic. The instructions are converted into frames and the operation of a particular instruction is done using the Functional Units like ALU, LD/ST units and Multiplier blocks. The processor instructions are used to perform various types of operations. For performing the operations in hardware it is essential to convert the instructions into a machine language. This is done by converting the instructions into a frame. According to the type of operations required there can be various frame formats. They are

i) R-Type → Register Type instructions
ii) J-Type → Jump Type instructions
iii) I-Type → Immediate Type instruction

2.1 R Type Instruction Format

The frame format for this type of instruction is shown in Fig.1. The values in bracket indicate the width of the particular field. Here, the ‘opcode’ decides the type of processor instruction. ‘rs1’ and ‘rs2’ are the source registers. ‘rd’ is the destination register. ‘shamt’ field is used when any shifting operations is to be carried out. The ‘function’ field is used to select the type of operation.

opcode (6)	rs1 (5)	rs2 (5)	rd (5)	shamt (5)	function (6)

Fig.1 R-Type frame format

2.2 J-Type Instruction

The frame format for jump instruction is shown in Fig.2. The target field is 26 bits wide and is used in jump type instructions. This specifies the register where the operation must jump to.

opcode (6)	target (26)

Fig 2 J-Type frame format

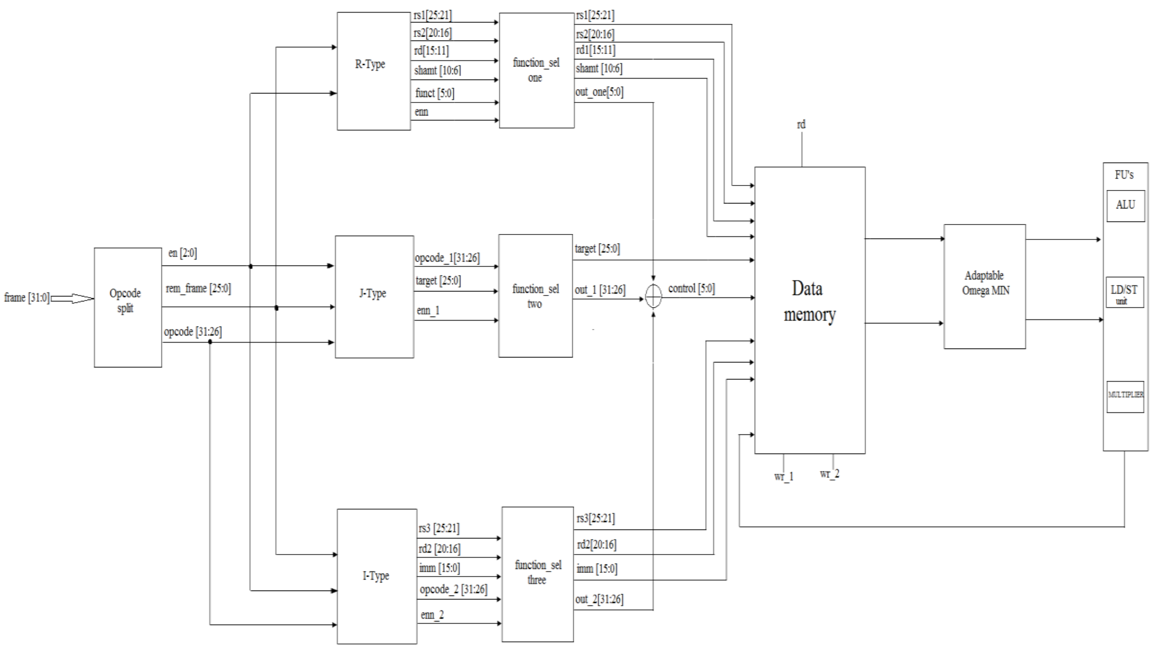

Fig.3 Proposed Dynamic Reconfigurable system

2.3 I-Type Instruction

The I-type frame format is shown in Fig.4. The register 'rs3' is a source register. 'rd2' indicates the destination register. The 'imm' is an immediate field where the data value for the given instruction is specified directly instead of the register value. The proposed architecture of Dynamic Reconfigurable system is shown in Fig 4 where the instructions are processed. The Omega MIN is used for transferring the data from the MIPS processor flow output to the Functional units available at the output ant the results are written back and stored in memory.

opcode (6)	rs3 (5)	rd2 (5)	imm(16)

Fig.4 J-Type frame format

2.4 Opcode Split

The opcode split block is shown in Fig.5. The instructions are first converted into a frame format which is of 32 bit length and is given as the input for the opcode split block. This block splits the frame into opcodes and 'rem_frame' (rem_frame) and an enable signal will be generated which will be used to select different type of instruction blocks.

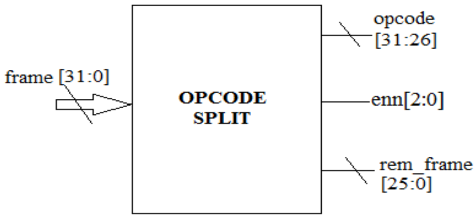

Fig.5 Opcode Split Block

2.5 R-Type Split Block

If the opcode detected is '000000' then the enable signal becomes '001' and the 'rem_frame' is given to the R-type block. Then the 'rem_frame' (26 bits) is divided into different r-type fields and this block will generate an enable 'enn' to indicate that the 'function_sel' block should be activated. For r-type instruction the function field in the 'rem_frame' decides the type of function to be performed by the instruction. The block for the R-type is shown in Fig.6.

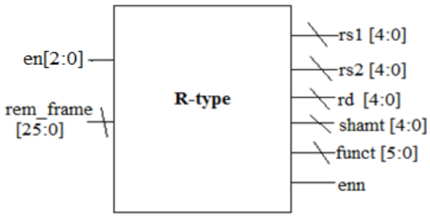

Fig.6 R-Type split block

2.6 J-Type Split Block

If the opcode corresponds to that of a J-type instruction which is '000010' or '000011' then the enable signal (en) becomes '010' and the rem_frame is given to the J-type block. Here the opcode which is obtained from the opcode split block is given to the J-type block and is responsible for detecting the type of function to be performed by the particular instruction. One thing to be noted is that the function field in r-type instruction is responsible for determining the type of operation to be performed, but for j-type it is the opcode itself. The rem_frame is given as target itself and the opcode is just passed to the output and stored in opcode_1. The 'enn_1' signal is activated to indicate that the function sel_two block should be activated. Fig .7 shows the J-Type split block.

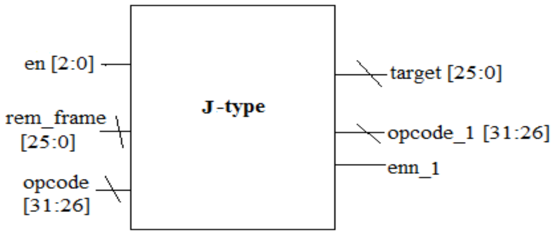

Fig.7 J-Type split block

2.7 I-Type Split Block

If the opcode corresponds to that of an I-type instruction then the 'rem_frame' and opcode is given to the I-type split block and the enable signal is made as '100'. Here also, the opcode decides the type of function and is stored in 'opcode_2'. The 'rem_frame' is split according to the I-type instruction frame format. Also enable 'enn_2' is activated to inform that the 'function sel_three' block should be enabled. Fig.8 shows the I-Type split block.

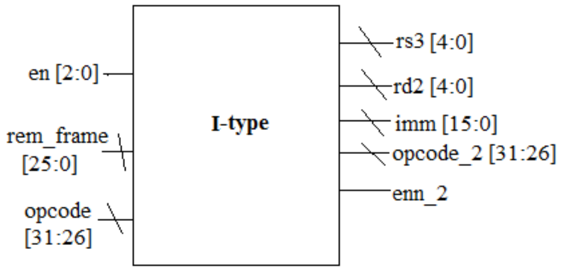

Fig.8 I-Type split block

2.8 Function_Select One

Fig.9 shows the function_select one block. The output of the r-type split block is given to the 'function_ sel one' block. In this block the function field values are detected and are assigned a particular binary value indicating the type of function. Then the corresponding function value is stored in 'out_one'. This block is activated only if the 'enn' value is '1' otherwise it is switched off.

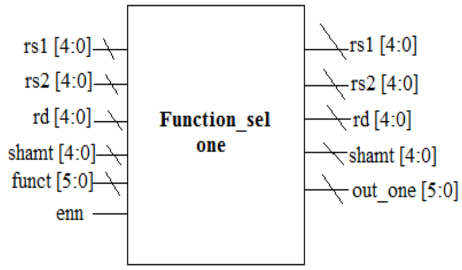

Fig.9 Function_selection one

2.8 Function_Select Two

Fig.10 shows the function_ select two block. The output of the j-type split block is given to 'function_sel two' block (shown in Fig.10) and the block is activated if the 'enn_1' value is '1'. The function of this block is to assign a binary value to each of the 'opcode_1' values since it determines the function to be performed. Then the 'opcode_1' values are stored in 'out_two'.

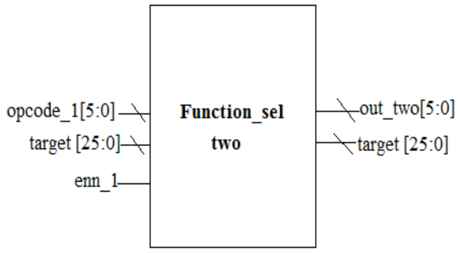

Fig.10 Function_select two

2.10 Function_Select Three

Fig.11 shows the function_select three block. The output of the I-type split block is given to the 'function_sel three' (shown in Fig.11) and it is activated when 'enn_2' value is '1'. Then similar to the 'function_sel two' block, the 'opcode_2' values are assigned a binary value. Then this 'opcode_2' is stored in 'out_three'.

For all the three 'function_sel' blocks the remaining fields in the frames are passed to the output for using it in the next stage.

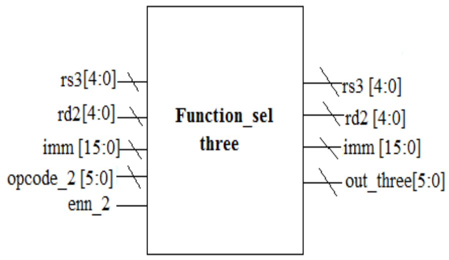

Fig.11 Function_ selection three

2.11 Data Memory, Adaptable Omega MIN and Functional Units.

The data memory is used for storing the values in the memory specified by the addresses. The memory has 8 bit width and 64 bit depth with 64 addresses pointing towards each of them. The Data memory together with Omega MIN and Functional Units is shown in Fig.12. The out values from the 'function_ sel one', 'function_sel two' and 'function_sel three' is 'XOR'ed and the final outp4ut 'control' is obtained. This value is given as selection input for the data memory. The control value decides the type of function to be performed. The data values for this particular function are accessed from the data memory, depending upon the address values specified in the instruction. This data is passed to the output of the data memory depending upon the enable value.

The data memory has three enable signals namely 'wr_1', 'wr_2' and 'rd'. When 'wr_1' enable is '1', then the address values are given to the memory and the data value is given as input. When 'rd' enable is '1', then the values are taken out from memory or read from memory. Then this value is routed via the Adaptable Omega MIN and mapped onto the Functional Units. The Functional Units perform different functions on the data values and when 'wr_2' is '1' then the results are returned and stored in the data memory itself.

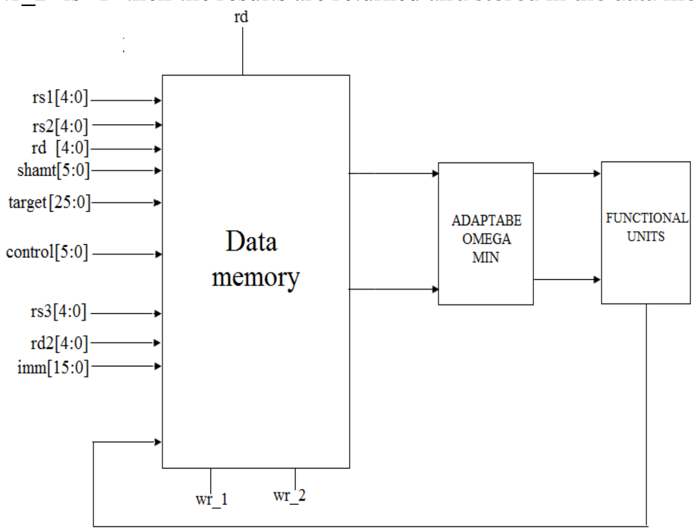

Fig.12 Arrangement of data memory, omega min and Functional Units

III. SIMULATION AND SYNTHESIS RESULTS

Simulation of all the blocks is done using Modelsim 10.0c and the synthesis is done using Design Compiler. Individual blocks were designed and integrated to form the proposed architecture.

3.1 Control Logic

The instructions are converted into frames first and these are given as input to the proposed architecture. The frames are detected for the type of instruction and the operation type. The out1, out2 and out3 which are obtained from function select blocks are 'XOR'ed to get the final control output. The 'XOR'ing is done to get a particular control output which will be used to fetch the data from the data memory.These data corresponds to the address which is present in the given input frame. The simulated result for the control logic is shown in Fig.13.

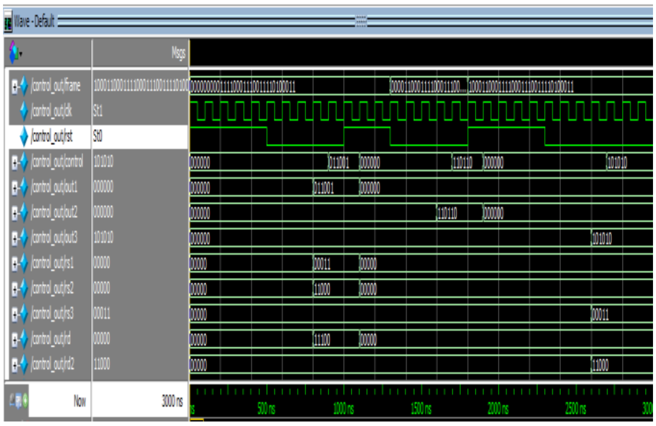

Fig.13 Simulation result of the Control logic.

3.2 Data Memory with ALU as Functional Unit

The data memory is designed which is of 8 bit width and 64 bit depth. The input data is taken as data1 and data2. The address for data1 is addr1 and for data2 is addr2. The data1, data2 and addresses are given as input when wr_1 is enabled. When rd (read) is enabled then data1 is read at out1 and data2 is read at out2. When wr_2 is made as '1' then both data1 and data2 are given as input to ALU and operation is done depending upon the opcode (Operation for ALU). Finally the result is written back and stored in memory corresponding to the addr3 value. The Functional Unit considered here is the ALU. The operation using this ALU is addition. If any operations like Load and store, Multiplication functions then the corresponding units are designed and integrated with the Data memory. The simulated waveform for the Data memory with ALU is shown in Fig.14.

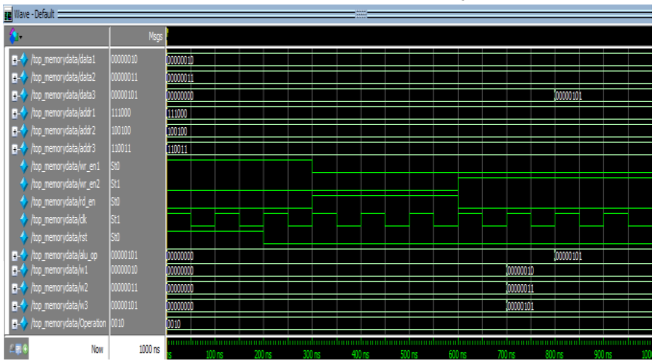

Fig.14 Simulation result of the Data memory with ALU.

Table 1 Synthesis Report of the Control logic

Parameters	Obtained Values
No of cells	163
total dynamic power	37.17uW
total area	12209.76 nm^2

IV. CONCLUSION

In this paper, a new logic for generating the control in a Dynamic Reconfigurable system is designed. As a future work the Omega MIN and other Functional units can be designed and integrated into a single block to complete the proposed architecture.

V. ACKNOWLEDGEMENT

This work was supported by AMRITA University, Bengaluru for the M.Tech Program. The author would like to thank Professor Mr. Sathish.K.Manocha and Asst professor Mrs.Vinodhini.M at AMRITA University, Bengaluru providing valuable feedback. The author would also like to thank AMRITA Management and ECE faculty for their support of further research program.

REFERENCES

Proceedings paper

[1] Ricardo Ferreira,Cristoferson Bueno, Marcone Laure,Monica Pereira and Luigi Carro "A Dynamic Reconfigurable Super VLIW Architecture for a Fault Tolerant Nanoscale Design" 4TH HiPEAC Workshop on Reconfigurable Computing,pp.7-16,January 2010

[2] A. Beck, M. Rutzig, G. Gaydadjiev, L. Carro, "Transparent reconfigurable acceleration for heterogeneous embedded applications," in DATE '08: Proceedings of the conference on Design, automation and test in Europe, pp.1208–1213, March 2008 [3] Tse-yun Feng, "A Survey of Interconnection Networks," IEEE, pp.12-27, December 1981.

[3] Berticelli Lo, T. Beck, A.C.S. Rutzig, M.B. Carro, L, "A low-energy approach for context memory in reconfigurable systems" IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW), pp.1-8, April 2010.

[4] K. Tanigawa, T. Zuyama, T. Uchida, and T. Hironaka, "Exploring compact design on high throughput coarse grained reconfigurable architectures," in FPL '08: Proceedings of the International Workshop on Field-Programmable Logic, pp. 126–135, September 2008.

Conference paper

[5] G. Mehta, J. Stander, M. Baz, B. Hunsaker, and A. K. Jones, "Interconnect customization for a coarse-grained reconfigurable fabric," *International Parallel and Distributed Processing Symposium (IPDPS)*, pp.1-8, March 2007.

Text book

[6] Antonio Carlos Schneider Beck Fl, Luigi Carro, *Dynamic reconfigurable architectures and transparent optimization techniques*, Springer Dordrecht Heidelberg London,2010.