Application Mapping Methodology for Reconfigurable Architecture

Rahul K. Hiware and Dinesh Padole

Abstract This paper suggests mapping of application on course grain reconfigurable architecture. The Data flow graph (DFG) in form of Op Codes of an application is stored in Memory i.e. Configuration memory. The CGRA reconfigure itself according to DFG and does parallel multi-processing.

Keywords Reconfigurable architecture · CGRA · Multicore

1 Introduction

To change the shape, model or to reform is known as reconfiguration. Reconfiguration provides rearrangement of elements or settings of a system contributing to restructuring as a whole or application based. In field of research in computer architectures and software systems, Reconfigurable computing has made its important place. An application can be significantly speeded by putting the computationally concentrated parts of an application onto the reconfigurable hardware. The advantages make us realize the significance of software along with hardware walking in parallel to save computation time, complexity and cost [1]. In era we are approaching, a single chip is able to comprise more than 100 million transistors. Despite the great flexibility current general-purpose processor systems possess still they will not reach their full potential they can provide. In contrast, exceptionally high performance has been achieved by application specific integrated circuits (ASICs) by aiming every application on

R.K. Hiware (✉) · D. Padole
Department of Electronics Engineering, G.H. Raisoni College of Engineering, Nagpur, India
e-mail: rahulhiware@gmail.com

D. Padole
e-mail: dinesh.padole@raisoni.net

S.C. Satapathy and S. Das (eds.), *Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 1*, Smart Innovation, Systems and Technologies 50, DOI 10.1007/978-3-319-30933-0_2

custom circuitry. However, it is difficult to design and implement a custom chip for every application because of the vast expense.

The gap between ASICs and general-purpose computing systems can be filled by the hardware that can be reconfigured. In 1960s, the basic concept was proposed, due to which reconfigurable computing system have become easy using technological upgradation in chip design. The availability of highly dense VLSI devices that programmable switches makes possible to implement the flexible hardware architectures. Many reconfigurable systems consist of a general-purpose processor, tightly or loosely coupled with system. These systems can device definite functionality of applications on reconfigurable system afore on the general purpose processor, providing considerably better performance. The general-purpose processors in such systems are only useful for data collection and synchronization and not for the major computational power. A lot of attention has been attracted because of the high performance and high flexibility delivered by the reconfigurable computing systems as compared to an ASIC. Moreover, in recent years such system can achieve high performance for a range of applications, such as image processing, pattern recognition and encryption.

2 Methodology

Reconfigurable systems due to their blend of flexibility and efficiency have drawn increasing attention lately. Reconfigurable designs have limitation of flexibility to a specific algorithm domain. The reconfigurable systems defined in two categories one is fine-grained where the functionality of the hardware is definite at the bit level and coarse grained where functionality of the hardware is definite at the word level. System design for coarse-grained reconfigurable architectures required some High-level design tools. This paper proposes a method for mapping applications onto a coarse grained reconfigurable architecture. This is a heuristic method which tackles this complex problem in four phases: Translation, Clustering, Scheduling and Allocation [2]. In this paper a coarse-grained reconfigurable architecture shown in Fig. 1, is proposed to demonstrate the proposed mapping method.

The proposed system consists of 4 ALUs. These ALU can be either any simple ALU or it could be even advanced processor. For simplicity, here we have taken simple four bit ALU. These ALUs are connected in 2 × 2 cross bar pattern. To map DFG on ALU there is ALU selector which is an encoder named as chip selector. The memory here is used to store DFG/opcodes of an application. This memory is known as configuration memory.

User provides the interface in terms of application DFG/opcodes as input which is stored in memory [3]. Based on the user control the counter shifts the memory which selects the respective ALU to take the op code segment, and perform the operation based on the data. Op code format is as shown Fig. 2. The DGF or

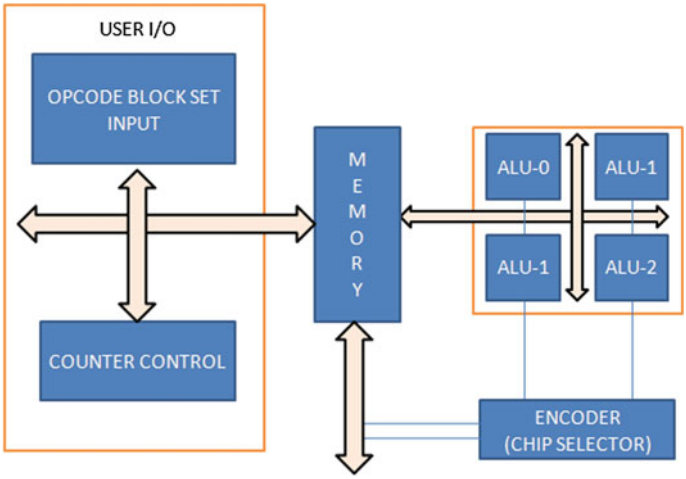

Fig. 1 Experimental CGRA architecture

opcodes for the given system are stored in configuration memory as per the format given below. Bits D7-D6 are used to select the ALU, which is nothing but input to the chip selector. Bits D1-D0 is for writing a data/result in memory or read data from memory. Rests of the bits D5-D2 are used to select operation to be performed by the selected ALU.

3 Simulation Result and Conclusion

Multi-processing is performed as shown on op code (Fig. 2) based transition phase but this reconfiguration has demerits of data transformation/flow looped with arbiter hence we suggest a programming element based alternative. This coarse grained architecture comprises of four parts: In the translation phase, an input program can be written in a high-level language and translated into a control dataflow graph; and some transformations and simplifications could be done on the control data flow graph. In the clustering phase, the data flow graph can be divided into clusters and mapped into Arithmetic Logic Units (ALUs). The structure of ALU is the main concern of this phase and the inter ALU communication is not in the consideration. In the scheduling phase the graph can be obtained from the clustering phase to know whether the scheduling taking the maximum number of ALUs into account. The scheduling algorithm should minimize the number of clock cycles used for the application under the constraints of reconfiguration. In the allocation phase, variables can be allocated to memories or registers, and data moves could be scheduled; where the main concern in this phase could arise in target architecture in terms of

D7	D6	D5	D4	D3	D2	D1	D0
X	X	@	@	@	@	Z	Z

ZZ - memory read/write cycle
01 - Read cycle
11 - Write cycle

@@@@- ALU operation selection
0000 - addition
0011 - Multiplication

xx - 2:4 encoder bits - for multi core ALU selection
00 - ALU-0
01 - ALU-1
10 - ALU-2
11 - ALU-3

Fig. 2 Op code format and operations

```
1  module top_alu(out,a,in,clk,en,reset);
2  input [7:0]in;
3  input [15:0] a;
4  output reg [7:0]out;
5  input clk,en,reset;
6  wire flag;
7  wire [5:0] count;
8  wire [15:0] dataout;
9  wire [7:0] out_temp,out_temp1,out_temp2,out_temp3;
10 wire [3:0]dout;
11 sync_reset one(count,reset,clk,en);
12 mem_ram_sync two(count,a,dataout,in[1],in[0],reset,en,clk);
13 decoder2_4 three(in[7:6],dout,clk,en);
14 ALU four_A(out_temp,dataout[15:8],dataout[7:0],in[5:2],clk,dout[3],flag);
15 ALU four_B(out_temp1,dataout[15:8],dataout[7:0],in[5:2],clk,dout[2],flag);
16 ALU four_C(out_temp2,dataout[15:8],dataout[7:0],in[5:2],clk,dout[1],flag);
17 ALU four_D(out_temp3,dataout[15:8],dataout[7:0],in[5:2],clk,dout[0],flag);
18 always@(posedge clk)
19    begin
20    if(en==1)
21    out<=out_temp;// !out_temp1 or out_temp2 or out_temp3;
22    else
23    out<=8'd0;
24    end
25 endmodule
```

Fig. 3 HDL code of proposed CGRA

area. Figure 3 shows HDL program for top level entity which is a proposed experimental architecture shown in Fig. 1.

Figure 4 shows RTL view of proposed system showing all elements of the system and Fig. 5 shows simulation result.

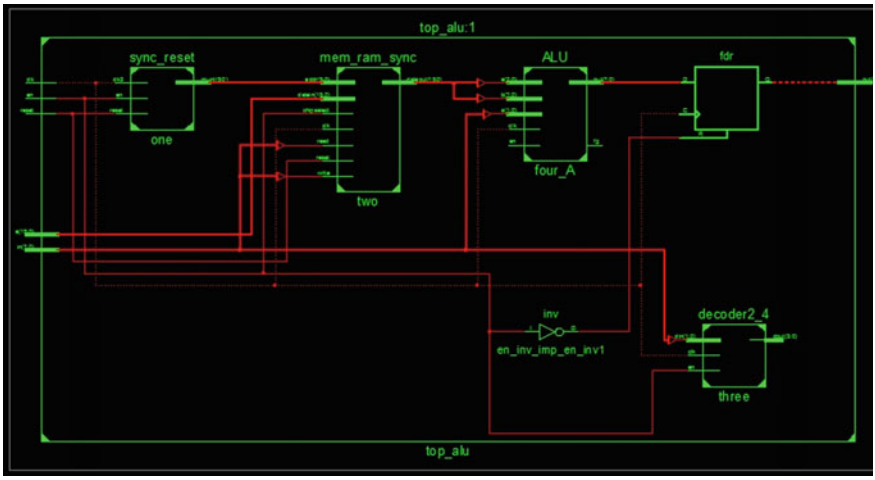

Fig. 4 RTL view of proposed CGRA

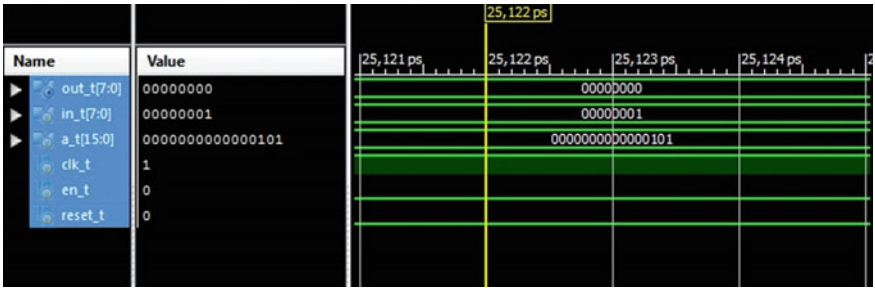

Fig. 5 Simulation result

References

1. Li, Z.: Configuration management techniques for reconfigurable computing. Northwestern University, June 2002
2. Guo, Y.: Mapping Applications to a Coarse-grained Reconfigurable Architecture. University of Twente, Enschede (2006)
3. Padole, D., Hiware, R.: Configuration memory based dynamic coarse grained reconfigurable multicore architecture. In: IEEE Region 10. Conference TENCON 2013 Xi'an, pp. 1–5, Oct 2013

http://www.springer.com/978-3-319-30932-3

Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 1

Satapathy, S.C.; Das, S. (Eds.)
2016, XVI, 602 p. 258 illus., Hardcover
ISBN: 978-3-319-30932-3