An Effective Strategy of Area Reduction for Custom Instruction Based on Basic Cell

Haiming Liu
College of Information Science and Engineering
Hunan University
Changsha, China
liuhm87@163.com

Yuchun Ma
EDA Lab, Dept. of Computer Science and Technology
Tsinghua University
Beijing, China
myc@mail.tsinghua.edu.cn

***Abstract*-Area reduction is very important for CI (Custom Instruction) on reconfigurable processor. However, for FPGA (Field Programmable Gate Array) structure, resource sharing method is inefficient for area reduction of data-path. In the paper, we propose an effective strategy of area reduction for custom instruction. Firstly, we partitioned data-path of the final CI to lots of BCs (Basic Cells). Then we checked all the validity of BCs to make sure that each BC can be realized using single logic element of FPGA, and selected the unique BC set to overlap original data-paths. Finally, on the basis of BC partitioning, we adopt an approach of Merging CIs to reduce area cost of data-path, which is different from traditional method of resource sharing. Experiment results show that basic cell can really represent FPGA architecture, compared to a method of efficient resource sharing, the approach of merging CIs based on BC can lead to more than 11% average reduction of area.**

Keywords-custom instruction, FPGA, area reduction, merging

I. INTRODUCTION

With development of embedded system technology, general processors can't meet the demand of high performance and programmability in the future system. However, reconfigurable processors based on FPGA (Field Programmable Gate Array) can overcome the shortcomings by using CI (Custom Instruction). CI can extend code segments that have the highest execution frequency as specific instruction set. Therefore, CI can improve system's computing performance and meet requirements of programmability. At the same time, reconfigurable processors based on FPGA can meet the demand of the shrinking time-to-market window and custom demand of specific application.

Custom instruction can extend the original instruction to a set of special instructions. CI process is divided into two stages of identification and generation. Identification chiefly explores subgraphs from IR (Intermediary Representation) as CI candidates, which need to satisfy all constraints of system architecture. IR is generally extracted from some compiler, e.g. Trimaran. In [1] [2] [3] [4], various identification algorithms are used to rapidly indentify a set of CI candidates from DFG (Data Flow Graph). Generation stage mainly selects lesser subgraphs from candidates as final CIs. Criterions of selection are mainly about performance enhancement and area cost. Based on characterizing of basic blocks to explore CI candidates, Liang proposed an efficient custom instruction generation algorithm in [5] [6].

In the process of CI, Gate was considered as basic element of data-path in [1~6]. However, FPGA realizes the function of data-path by using logic elements. Therefore, we use BC (Basic Cell) as a basic element. We suppose that a basic cell (e.g. BC_1~BC_4 in Fig1) can be realized using one logic element. At the same time, area cost becomes more important as embedded application system becomes more complicated. Resource sharing was generally used to optimize area. However, we proposed an approach of merging CIs to reduce area of data-path in the paper. For example, Figure 1a represents the original data-path of two CIs, which is realized by using 4 LEs. But, Figure 1b represents the data-path after merging CIs, which needs 2 LEs to realize the function. Data-paths in dotted line are CIs (e.g. CI_1, CI_2), data-paths in solid line is called BCs (Basic Cells).

The contribution of paper is:

- Used basic cell as basic element in the process of CI, rather than gate level representation. Based on island-style FPGA architecture, we divided CI candidates into a unique set of BC (Basic Cell) to really represent structure of FPGA. Each basic cell can be realized on the single logic element of FPGA.
- On the basis of basic cell, proposed an approach of merging CIs to reduce area of data-path. Differed from resource sharing approach, we merged a pair of BCs from different CIs to form a new basic cell. The new BC can implement the function of former BCs and must be only realized using single logic element of FPGA.

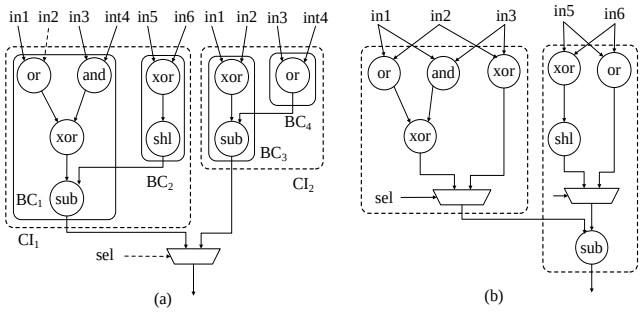


Figure 1. Example of Merging based on BC.

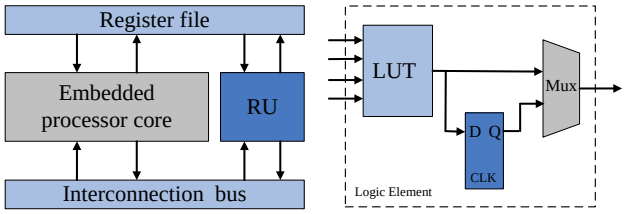


Figure 2. Architecture of reconfigurable processor.

II. FPGA-BASED RECONFIGURABLE PROCESSOR

Differed from general processor, reconfigurable processors mainly consist of an embedded processor core and a reconfiguration unit (RU), which is used to implement custom functions of users. In Figure 2a, an embedded processor core was extended with a RU based on FPGA. RU can gain input data from register file and output computing results to interconnection bus. The way of communication may facilitate data sharing between embedded processor and RU.

Figure 2b illustrates an inner structure of LE，which is the basic logic unit of FPGA. LEs mainly contain LUT (Look-Up Table), D flip-flop and multiplexers. LUT can realize combination logic function of up to *k* inputs. When LE needs to realize combination logic function, the Mux select the result of LUT to out. If want to finish timing logic function, the Mux needs to output the result of D flip-flop. Meanwhile LE can finish arithmetic operation (e.g. addition, subtraction) by using inner fast carry chains of FPGA.

III. BASIC CELL PARTITIONING

Based on off-the-shelf island-style architecture of FPGA, we use basic cell as basic element of data-path to really describe structure of FPGA. The process needs to 3 stages: (1) BC enumeration (2) BC validity checking, (3) BC overlapping.

A. BC enumeration

For island-style FPGA structure, LEs can implement combination logic function of up to *k* inputs by using LUTs. Meanwhile, LEs may realize the function of some arithmetic operation (e.g. addition, subtraction) by fast carry chain. RU can realize shift operation by configuring routing resource. Therefore, primitive operations (e.g. 1~ 6 in Figure 3a) were firstly classified to logic operations (e.g. and, xor, or), shift operations (e.g. shr, shl), arithmetic operations (e.g. add, sub). Then we adopted graph partitioning algorithm in [7] to enumerate all subgraphs as BC examples (e.g. BC_1, BC_2 in Figure 3a).

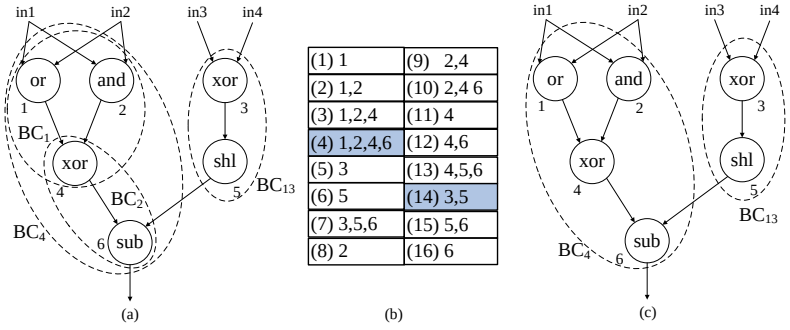
B. BC validity checking

A basic cell is valid if it can be mapped onto single logic elements. In other words, we only need a logic element to realize the function of basic cell (e.g. (1) ~ (16) in Figure 3b). Therefore, BCs must comply with a set of rules, which include checking of operations and external inputs/outputs. Operation checking is used to decide whether an operation may be included in the basic cell.

- There is only one arithmetic operation in the DFG of BC. Because one LE of FPGA can only implement the function of one arithmetic operation (e.g. sub, add) by using inner fast carry chain.
- If there are synchronously shift operation and arithmetic operation in BC, condition that shift operation executed after arithmetic operation is allowed. Because shifted value of arithmetic operation can be implemented by configuring FPGA's routing resource.
- If there are logic operation and arithmetic operation in one BC, logic operation must be executed before arithmetic operation. Because LE can realize arithmetic operation by using inner fast carry chain. Moreover result of LUT is used as partial sum.

Another rule is used to check external inputs/outputs of the BCs. The rule makes sure that can't violate the constraints of number of inputs/outputs. We suppose that input pin of LUT is *k*:

- Maximum number of input and maximum output is *k* and 1 respectively. Because a look-up table of LE only realizes combination logic of up to k inputs. Moreover there is output pin of LE.
- One input of arithmetic operation (e.g. add, sub) must be external input. In other words, one of inputs to arithmetic operation must be directly connected to one of external inputs to BC. Because carry-out may comes from one input of LUT when LE realize function of arithmetic operation.



(1) 1	(9) 2,4
(2) 1,2	(10) 2,4 6
(3) 1,2,4	(11) 4
(4) 1,2,4,6	(12) 4,6
(5) 3	(13) 4,5,6
(6) 5	(14) 3,5
(7) 3,5,6	(15) 5,6
(8) 2	(16) 6

Figure 3. Example of BC partitioning.

C. BC overlapping

We selected a unique set of valid BCs to overlap original DFGs of CI candidates by using a heuristic algorithm of graph overlapping in [5] (e.g. BC_1 and BC_2 in Figure 3c). The objective function is to minimize the number of BCs selected. Therefore the final BCs selected will be realized by using least number of LEs.

IV. MERGING CI BASED BC

For effectively mapping data-path of final CIs on FU, we proposed a strategy of merging CIs to reduce area cost of RU. Unlike existing technology mapping, the process can be executed on the high-level representation of the CIs, because this avoids constraints of hardware. Differed from way of resource sharing, which maximizes the common component of data-path, we treat mapping problem as graph coving problem, because this don't result in delay problem. According to whether introduce multiplexers, the strategy was divided into merging without using multiplexer and merging with using multiplexer.

A. Merging without using multiplexer

On basis of gaining a unique set of BCs, a pair of BCs from different CIs can be merged if merged cell can be realized on a logic element of FPGA. In other words, the new cell can also be treated as a valid BC. Before merging a pair of BCs, we firstly partition BCs into several corresponding *log* and *arith* components (e.g. *arith* and *log* in Figure 4). Arithmetic operations need to be classified to *arith* component, other operations (e.g. logic, multiplexer) need to be classified to *log* component. It is worth mention that multiplexer in *log* component has an external input, which be directly connected to one input of BC.

A pair of BCs may be merged if the *log* and *arith* components may satisfy the following condition：

- When both of BC_1 and BC_2 have *arith* components, a pair of BC_1 and BC_2 can be merged if $arith_1$ and $arith_2$ must are identical. The reason is that if $arith_1$ and $arith_2$ are different, the merged cell needs several LEs to realize the function. It will violate the condition of merging.

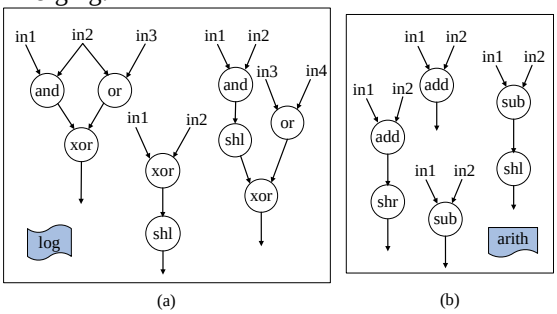


Figure 4. Example of *log* and *arith* component.

- When one of BC_1 and BC_2 has only *arith* component, BC_1 and BC_2 can be merged only if the *arith* component don't include shift operation. Because merged cell wants to achieve both functions of BC_1 and BC_2 only if the *arith* component may be convertible to an *invisibility* function (see condition 4). But *arith* component with shift operation can't be convertible to an *invisibility* function.
- When condition1 and condition2 are met, if BC_1 has a *log* component with multiplexer, which has an external input, then BC_1 and BC_2 can be merged by connecting output of *log* component of BC_2 to the external input of multiplexer (e.g. Figure 5a), because BC_1 and BC_2 from different CIs don't synchronously be executed.
- When condition1 and condition2 are met, BC_1 only has *log* component and BC_2 only has *arith* component. BC_1 and BC_2 can be merged by connecting output of *log* component of BC_1 to an input of *arith* component of BC_2 only if *log* component of BC_1 may be convertible to an *invisibility* function (e.g. Figure 5b). Merged cell realizes the function of BC_1 by setting external input of $arith_2$ to '0' and the function of BC_2 by converting log_1 component to an *invisibility* function.

Invisibility function represents that output value of component is identical to an input value of component. In other words, an input value is unchanged when all input values are propagated to output of component. To judge whether a component is converted to *invisibility* function, we continually try to set other input values to '0' or '1'. Then, all input values are propagated to output of component by manipulating of operation in component. If output value of component is identical to an input value by setting the remaining inputs, then the component can be convertible to an *invisibility* function.

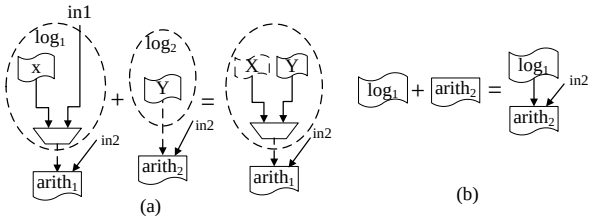


Figure 5. Example of merging without using multiplexer

B. Merging with using multiplexer

If a pair of BC_1 and BC_2 can't be merged without using a new multiplexer, then we try to merge them by using a new multiplexer. On the basis of partitioning, we connect output of one *log* component to an input of the new multiplexer, and connect output of another *log* component to another input of the new multiplexer (e.g. Figure 6a and Figure 6b). If BC_1 and BC_2 have *arith* component, then output of the used multiplexer need to be connected to the inner input of *arith* component (e.g. Figure 6c). Because add an external selection pin to control the new multiplexer, so we need to newly check validity of inputs/outputs constraints. If the number of inputs is less to constraint of BC, we will merge the pair of BCs by adding a new multiplexer.

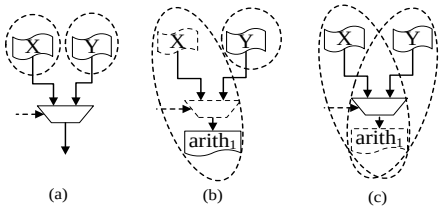


Figure 6. Example of merging with using multiplexer

V. EXPERIMENTAL RESULTS

For validating our strategy, we use benchmark of Trimaran. Firstly we partition CDFG of candidates to unique set of BCs, then use the model in [8] to estimate area cost. We use selection algorithm in [6] to generate CIs. Finally we adopt approach of merging CIs based on BC to reduce area of data-paths. The final CI set is realized on RU based on 4-LUT (e.g. Virtex-4 in Xilinx). The whole strategy is described by C++, and machine configuration is: AMD Athlon(tm) 64 X2 Dual core processor, 2G memory. Figure 7 shows whole framework of custom instruction.

Table 1 shows that area result after merging CI compared with results after an effective resource sharing. Column 1, column 2 and column 3 represents benchmark name, the number of CIs and the number of whole nodes respectively. Area of original, data-path after RS (Resource Sharing) and data-path after merging CI are column 4, column 5 and column 6 respectively. Results show, both of our strategy and RS can lead to notable area reduction. Although RS approach excel out strategy when a large number of common operation exist in CIs set (e.g. fir), our strategy can lead to more than 11% percent average reduction of area than RS method. Although the execution time of merging CI is longer than time of resource sharing, both can be finished in order of seconds.

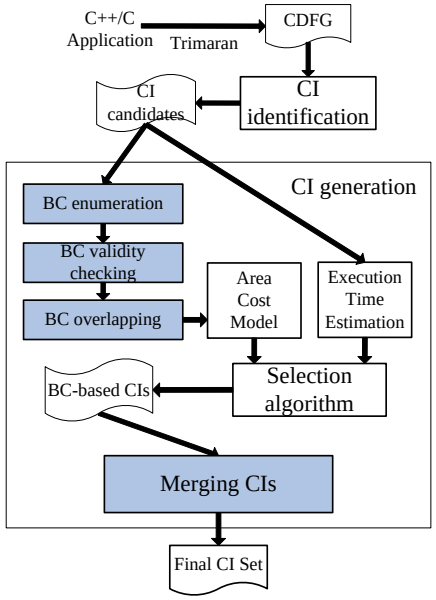


Figure 7. Flow of custom instruction

TABLE I. AREA RESULTS BETWEEN RS AND MERGING CI

Benchmark	Number of CIs	#Num	Number of LUT		
			Original	RS	Merging
adpcm	13	35	507	355	263
bmm	8	52	569	438	345
fir	10	58	695	527	543
idct	4	22	247	193	155
mm_dyn	5	32	346	295	251
sqrt	16	71	883	692	540

VI. CONCLUSION

In the paper, we proposed an effective strategy of area reduction for custom instruction. We firstly partition every CI candidates to a set of basic cells. BC can really represent structure of RU based on FPGA, especially for area cost. The area estimation value based on BC is very close to actual value of high level synthesis. On the basic of BC partitioning, we adopt an approach of merging CIs to reduce area of original data-path. Compared to an efficient method of resource sharing, the strategy can lead to more than 11% percent average reduction of area.

REFERENCE

[1] K. Atasu, L. Pozzi, and P. Ienne, “Automatic application-specific instruction-set extensions under microarchitectural constrains”. DAC, 2003, pp.256-261.

[2] Cong J, Fan Y, Han G, and Zhang Z, “Application-specific instruction generation for configurable processor architectures”. FPGA, 2004, PP.183-189.

[3] L. Pozzi, K. Atasu, and P. Ienne, “Exact and approximate gorithms for the extension of embedded processor instruction sets”. IEEE Trans. on CAD of Integrated Circuits and Systems, 2006, pp.1209-1229.

[4] I-Wei Wu, Cheng-Ping Chung, Jean Jyn-Jium Shann, “Area Effificent Instruction Set Extension Exploration with Hardware Design Space Exploration”. Journal of inforation science and engineering 27, 2011, pp.1641-1657.

[5] Guoqiang Liang, Yuchun Ma, Kang Zhao, Jinan Bian, “Efficient Custom Instruction Generation Based on Characterizing of Basic Blocks”. 17th IEEE International Conference on Computer Supported Cooperative Work in Design, 2013.

[6] Guoqiang Liang, Yuchun Ma, Kang Zhao, Jinian Bian, “Adaptive Custom Instruction Identification Algorithm based on Two-Step Partitioning of Basic Blocks”. The 10th IEEE International Conference on Embedded Computing, 2012.

[7] Kang Zhao and Jinian Bian, “Processor Accelerator Customization through Data Flow Graph Exploration”. IEICE Trans, 2011, pp.1540-1552

[8] H. Gao, Y. Yang, X. Ma, G. Dong, Analysis of the effect of LUT size on FPGA area and delay using theoretical derivations”. Proceedings of the Sixth International Symposium on Quality of Electronic Design, 2005, pp.370-374.

[9] http://www.trimaran.org/