Customizable Framework for Arithmetic Synthesis

Matsunaga, Taeko
Fukuoka Industry, Science & Technology Foundation

Matsunaga, Yusuke
System LSI Research Center Kyushu University

出版情報：SASIMI2004, pp.315-318, 2004-10. Sasimi Workshop
バージョン：
権利関係：

Customizable Framework for Arithmetic Synthesis

Taeko Matsunaga

FLEETS
Fukuoka Industry, Science & Technology Foundation
t_matsunaga@fleets.jp

Yusuke Matsunaga

System LSI Research Center
Kyushu University
matsunaga@slrc.kyushu-u.ac.jp

Abstract— Design of arithmetic units has been an important issue which can dominate performance of the whole circuits. Recent logic synthesis tools can implement arithmetic units from RTL descriptions by utilizing parameterized design components predefined in arithmetic libraries. This paper reviews the current status in arithmetic synthesis, and some issues, especially on customizability, are pointed out to be tackled for better performance. Further work is still needed in arithmetic synthesis, and it is helpful to have an framework which eases new approaches to be integrated. Requirements for such framework are discussed and a possible synthesis flow within this framework is shown.

I. Introduction

Design of arithmetic units has been an important issue which can dominate performance of the whole circuits. Many hardware algorithms and architectures for arithmetic units have been proposed so far, and still now[1][2]. In performance critical areas, such as processor design, designers often implement them by hand based on the accumulated know-how that has been built up over many years. In such cases, requirements for arithmetic units are usually defined beforehand.

On the other hand, when people design circuits using logic synthesis tools from specifications at register transfer level(RTL), the type and the number of necessary arithmetic units can vary. For example, an expression $a * b + c$ may need one adder and one multiplier, or one multiply-add module. How to implement an arithmetic module can not be decided uniquely even if a type of an arithmetic module is decided. An arithmetic module can have several different architectures with various characteristics such as area and delay. An architecture which is the best choice according to one measure, may not be the best under another measure. In those situations, it is very difficult for designers to decide which operation should be tuned and how to tune, so some effective tools are required. Further work is still needed in arithmetic synthesis, and it is helpful to have an framework which eases such work to be tried.

This paper focuses on arithmetic synthesis and its framework. The current status in existing tools is overviewed and the issues to be tackled are considered. Requirements for its framework are picked up, and a possible synthesis flow within this framework is shown.

II. Arithmetic synthesis in logic synthesis flow

In this section, current status of arithmetic synthesis in logic synthesis flow is overviewed and several issues to be addressed are considered.

A. Current status in arithmetic synthesis

Arithmetic synthesis task in logic synthesis flow is to generate arithmetic units which realize the functions of corresponding arithmetic operations appeared in RTL descriptions. Early logic synthesis tools had poor capability to implemented arithmetic operations. They converted arithmetic operations into logic expressions, and resulted in larger and slower implementations. Recent logic synthesis tools can generate better results by using special libraries for parameterized arithmetic units where many implementations of arithmetic units are stored. Here, we call such libraries arithmetic libraries. Arithmetic libraries are integrated into logic synthesis tools by default or optionally[3]. The amount and quality of reusable components for arithmetic units is very important to get better results. Users can build original parameterized components and integrate them into the library. Those components can be reused in synthesis flow.

Another approach to improve results of arithmetic synthesis is to group arithmetic operations in RTL descriptions, and implement them using carry save adders(CSA)[4][5]. Many arithmetic operations are based on addition and reduced to multi-operand additions. Using carry-save adders can reduce the number of necessary carry-propagate adders(CPA), and the maximal delay can be improved. This approach has been implemented within existing tools and better results are reported.

B. Considerations

In this section, we pick up two points to be considered for arithmetic synthesis capability to be more improved.

B1 Customizability for arithmetic libraries

As stated previously, the amount and quality of reusable components for arithmetic libraries is very important to get better results. Another important point is customizability, that is, capability to integrate one's original components into arithmetic libraries. Existing tools have this capability in case that such components are represented as HDL

descriptions. For example, an n-bit ripple carry adder is represented in parameterized form as shown in Fig.1.

```
module full_adder( output Co,
    output Sum,
    input A,
    input B,
    input Ci );
  assign Sum = (!A & !B & Ci)|(!A & B & !Ci)|(A & !B & !Ci)|(A & B & Ci);
  assign Co = (!A & B & Ci)|(A & !B & Ci)|(A & B & !Ci)|(A & B & Ci);
endmodule // full_adder

module rca (sum, co, a, b, ci);
  parameter SIZE = 4;
  output [SIZE-1:0] s;
  output   co;
  input [SIZE-1:0] a;
  input [SIZE-1:0] b;
  input    ci;

  wire [SIZE:0] c;

  assign c[0] = ci;
  assign co = c[SIZE];

  genvar i;
  generate
   for (i=1; i < SIZE+1; i=i+1) begin:u
   full_adder fa(.Co(c[i]),.Sum(sum[i-1]),.A(a[i-1]),.B(b[i-1]),.Ci(c[i-1]));
    end
  endgenerate

endmodule // rca
```

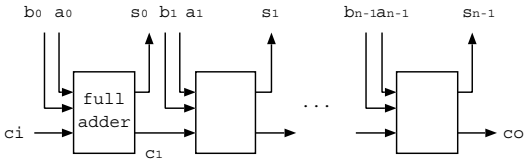

Fig. 1. Parameterized HDL description of ripple carry adder

Many arithmetic units have regular structures, and can be expressed by using generate-statements in VHDL or verilog.

But there are some cases where an appropriate architecture of an arithmetic unit can not be represented statically. One case is occurred from consideration of bit-wise timing constraints. In [6], an algorithmic approach is proposed to generate a parallel-prefix adder which has a minimal delay for a given profile of input arrival times. Bit-wise consideration of timing constraints is a promising approach to achieve better performance. But each timing constraint is decided dynamically in synthesis flow. Programs can generate an architecture dynamically under various environments, but existing tools can not integrate them flexibly in their synthesis flow.

B2 Selection problem

Another issue in arithmetic synthesis is selection and reselection problem; that is, how to select one among those alternatives, and how and where to find candidates to be changed.

Arithmetic synthesis usually has been done hierarchically. Circuit structures at technology independent level are defined first, and then mapped to netlist whose components are cells in the specified target technology library. First decisions at technology independent level are done without enough technology information available. Estimations at that level may lead to a wrong decision, and the first selection should be changed to another one to satisfy given constraints.

These issues seem not severe now. Logic synthesis tools have been much improved, so there is no problem without tight performance constraints. In timing critical cases, designers are often aware of circuit structure at RTL in mind, and sometimes design arithmetic circuits separately and embedded to other circuit parts.

In both cases, the above issues have less significance. But there exist some other cases where designers hardly recognize RTL structures. One case is RTL structure generated through behavioral synthesis tool. There can be lots of arithmetic modules to be tuned, where it is difficult for designers to handle them effectively. So, some effective strategies for automatic selection/reselection are needed.

One approach is that the total flow from behavior to cells can be considered to tackle the problem. When constraint violations have occurred, only cell sizing may solve the problem, but it may be a good idea to change behavior, which means scheduling and binding are changed. This can not be done at RTL. More accurate estimation at higher levels may reduce occurrence of reselection. But behavioral synthesis often link to less technology dependent information. So, more flexible interface is required between logic synthesis and behavioral synthesis.

III. Framework for arithmetic synthesis

A. Requirements for framework

According to the previous considerations, we pick up the following requirements for a better framework.

- User-defined programs can be dynamically linked and used in synthesis flow. This property enables users to integrate their own algorithms which generate appropriate architectures under dynamic conditions. Programs other than module generators, such as technology mapper, can also be linked.
- The total flow from behavior to cells can be considered within a framework. Arithmetic synthesis can be connected to behavioral synthesis method which utilizes customized functional units[7]

B. Overall synthesis flow

Fig.2 and 3 show an overall synthesis flow within this framework.

B1 Inputs and outputs for arithmetic synthesis

Inputs for arithmetic synthesis are the following three items:

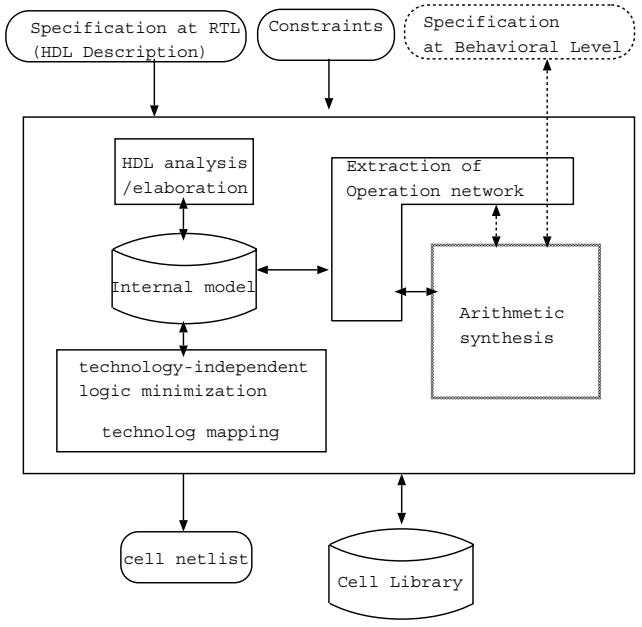

Fig. 2. Arithmetic synthesis in logic synthesis flow

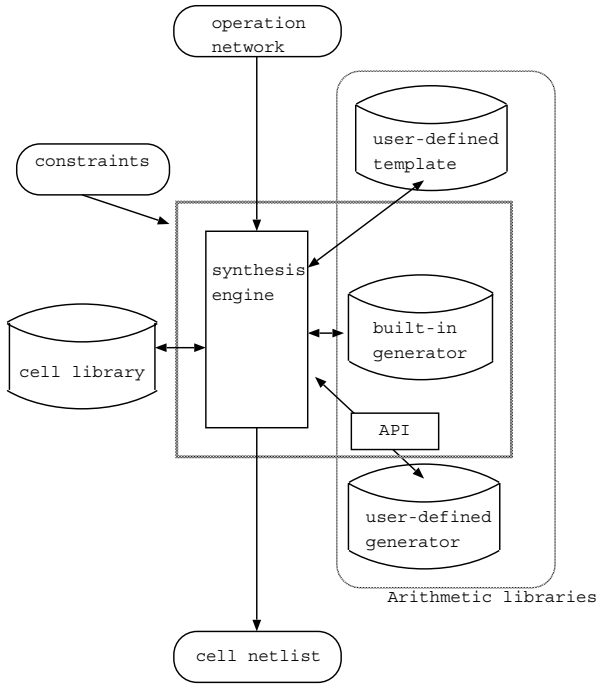

Fig. 3. An arithmetic synthesis system

- Operation network:
  Operation network is a directed acyclic graph which consists of operation nodes and edges. Every edge has bit width as an attribute. Fig.4 shows an example of operation network. In this case, two expressions, $op1 = a+b+c+d$ and $op2 = b+c+d+e$ are expressed in an operation network. Operation network can represent one and more expressions which may have common sub-expressions, and represents the function to be implemented as an arithmetic unit.

  Operation network can also be an interface to behavioral synthesis.

- Cell library: target cell library

- Constraints(optional): area, timing, power, and so on

Arithmetic synthesis tool accepts above three types of inputs, decides an architecture, and outputs netlist whose components are mapped to cells in the given target libraries. Characteristics such as area and delay values, or estimated values may also be output.

B2 Synthesis engine and arithmetic libraries

The arithmetic synthesis tool consists mainly of two part, synthesis engine and arithmetic libraries.

Synthesis engine should include the following capabilities:

- analysis on operation network

- transformation from operation network to structures at technology independent level

- transformation from structures at technology independent level to cell netlist (technology mapping for arithmetic modules)

- estimation for implementation

- selection of candidates

Decision from an operation network to cell netlist is done hierarchically. For example, Fig.4 shows two candidates for the operation network, which consist CSA and CPA. Since operation network including multiple operations is not assumed to be transformed into CSA structures, other components such as (4;2) compressors can be candidates to construct an architecture.

Candidates to be linked at each level are generated by using three arithmetic libraries.

- Built-in library which the framework has by default.

- User defined templates: template-type descriptions for architecture written in HDL.

- User defined programs: user-defined generator for reusable architecture which can not be represented as HDL code. Some interface mechanism is needed to be linked dynamically to synthesis engines.

IV. Summary and conclusion

In this paper, several issues on current arithmetic synthesis are reviewed and requirements for framework are discussed to tackle these issues. The framework which satisfies the described requirements can be a platform where new architectures or algorithms are plugged in. To evaluate the effectiveness of this framework, dynamically linked mechanism should be implemented.

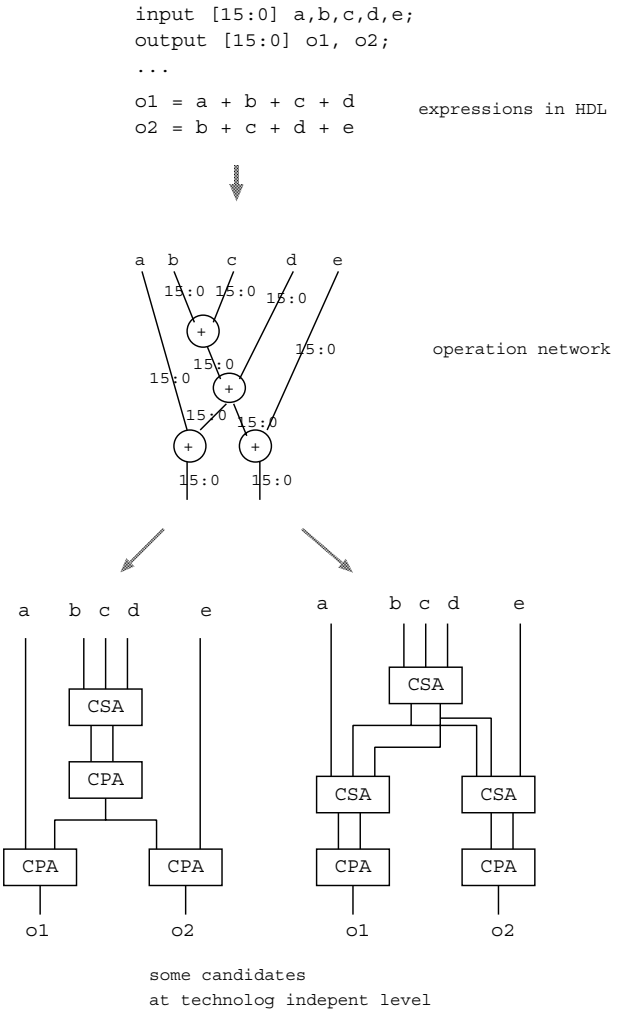

Fig. 4. Transformation of expressions in HDL to architectures

V. Acknowledgment

This research was supported in part by a grant of Fukuoka project in the Cooperative Link of Unique Science and Technology for Economy Revitalization (CLUSTER) of Ministry of Education, Culture, Sports, Science and Technology(MEXT).

References

[1] Israel Koren, "Computer Arithmetic Algorithms", A.K.Peters Ltd.

[2] Michael J. Flynn, Stuart F. Oberman, "Advanced Computer Arithmetic Design", Wiley Interscience

[3] DesignWare BuildingBlock IP, http://www.synopsys.com/products/designware/buildingblock.html

[4] T. Kim, W. Jao, and S. Tjiang, "Circuit Optimization Using Carry-Save-Adder Cells", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.17, No.10, pp.974-984, October 1998.

[5] Junhyung Um and Taewhan Kim, "An Optimal Allocation of Carry-Save-Adders in Arithmetic Circuits", IEEE Transactions on Computers, Vol.50, No.3, pp.215-233, March 2001.

[6] Jianhua Liu, Shuo Zhou, Haikun Zhu, and Chung-Kuan Cheng, "An Algorithmic Approach for Generic Parallel Adders", in Proceedings of ICCAD'03, pp.734-740, Nov.2003.

[7] Tsuyoshi Sadakata and Yusuke Matsunaga, "A Behavioral Synthesis Method Considering Complex Operations", In Proceedings of SASIMI 2004.