Reconfigurable Multiplier for Virtex FPGA Family

Juri Põldre[1] and Kalle Tammemäe[2]

[1] Tallinn Technical University, Tallinn, Estonia
jp@pld.ttu.ee
[2] Tallinn Technical University, Tallinn, Estonia
nalle@cc.ttu.ee

Abstract. This paper describes integer multiplier design optimizations for FPGA technology. The changes in partial product generator component enable to infer CLB fast carry logic for building Wallace trees. This change increases speed and gives better resource allocation.

1 Overview

As usable gate count grows an important factor in designing system on chip is to be able to reuse the intellectual property (IP) [4]. Contemporary ASIC technology allows more than 20 Mgates per die. To utilize these gates efficiently a variety of tools have emerged that use the knowledge about architecture of design to create regular structures for arithmetic operators. Such regular structures are used in applications like signal processing, data compression and cryptography. The regular structure generators are available commercially - like the Module Compiler from Synopsys [3]. A free arithmetic module generator is represented here by a tool from Norwegian University of Science and Technology [5]. Multiplier can be partitioned into three separate units: partial product generator, Wallace tree and final adder [1]. The Wallace tree consists of full-adder and half-adder cells. The cells are however connected in such a way that the longest carry chain is equal to $\log_2$(argument_length). The structure also contains carry chains of intermediate lengths. The shortest common factor in these trees is two. This decomposition can be made in such manner that the carry propagation signal between adder cells is not required elsewhere.

Most FPGA providers include carry propagation circuits inside CLB [6]. These internal carry chains are optimized for building the fast ripple carry adders. The carry out signal of one stage is directly connected to carry in of the next within one CLB and cannot be accessed outside. If the Wallace tree is decomposed into length two carry chains we can map these chains directly to CLB.

Commercial structure generator tools allow to specify target technology. To use the FPGA library the tool creates vector versions of elements in target library. The tool tries to find the elements for datapath synthesis in target library. In case it finds none the respective cell is generated from more primitive cells. In

P. Lysaght, J. Irvine, and R. Hartenstein (Eds.): FPL'99, LNCS 1673, pp. 359–364, 1999.

case of FPGA we cannot directly infer full adder cell, but rather two full adder cells – a length 2 ripple-carry chain.

The other solution is to use some standard cell library and run technology translation on target netlist. This leaves us with non-optimal result because internal carry logic is not inferred. The penalty is in both area and speed.

With both the commercial and free module generators we do not have access to the routine that generates the tree. The output is gate-level VHDL or Verilog file. From that netlist it is rather hard to replace cells because we need topology info to decide which full adders to group into single CLB. The simplest solution here is to group all these cells into single CLB that have the carry chain used only internally.

The best solution can be achieved when generating the netlist directly for the target technology. We can create structural designs in VHDL using generate statements. Such design can also support carry in and carry out from Wallace trees what are otherwise connected to ground and removed by logic optimizer.

Using these carry signals we can create a serial-parallel multiplier with more degrees of freedom exploiting the bit widths of both arguments. Consider 1024×1024 bit multiplier for example. In first solution we can use carry-save accumulator and do it in 64 steps using 16×1024 multiplier as a building block. This can be done by both generators.

If we can access carries within Wallace tree we can further partition it into eight times more steps using 16×128 multiplier as a base block. The above-described block can now easily be included into behavioral design with arrays mapped to FPGAs internal RAM cells. Such multiplier design leaves us with large set of architectural solutions and fast synthesis time. In the following sections we will overview the structure of Wallace tree and generators. Then the methods described above are used to create the optimized netlist. First the structural netlist from technology translation is used directly. Then the necessary changes are introduced to group the cells together based on the carry chain connection constraint. Finally the structural VHDL generator is used. All designs are mapped to VIRTEX FPGA device and results are compared with respect to area and delay.

2 Wallace Tree and CLB

The Wallace tree compresses partial products into two components - Sum vector and Carry vector. The final result can be calculated by adding these vectors. If magnitude comparison is not needed then the intermediate results can be kept in carry-save format. The usual practice in modular exponent calculations is to use squaring method. The method breaks down the exponent into $2 * N$ multiplications, where N is the size of the argument. The $N \times N$ bit multiplication is further reduced by scanning second argument with some determined bit width, usually 8 or 16 bits. This core of these calculations in $k \times N$ multiplication what will be called $2kN$ times. During these calculations the result is kept in carry-save format and only added together at the end of exponent

calculation. The similar practice is used in DSP multiply-accumulate cycles. In all these calculations Wallace Tree becomes the main resource and constraint.

The Wallace Tree consists of full-adder and half-adder cells. There are several ways to connect these cells. For this article the generalized 4to2 adder cells are used. Each cell has 4 primary inputs and 2 primary outputs, carry in and carry out. This cell adds two carry-save digits. It has carry in and carry out signals to expand the calculations. This cell is like full adder, but all IO lines are duplicated. Using these blocks the adder tree of any size can be built much as with ordinary full adders. The main difference with full adders is that there is no timing path from carry inputs to the primary inputs. All carries are introduced at lower level only. The Wallace tree for 16 arguments is depicted in fig. 1.

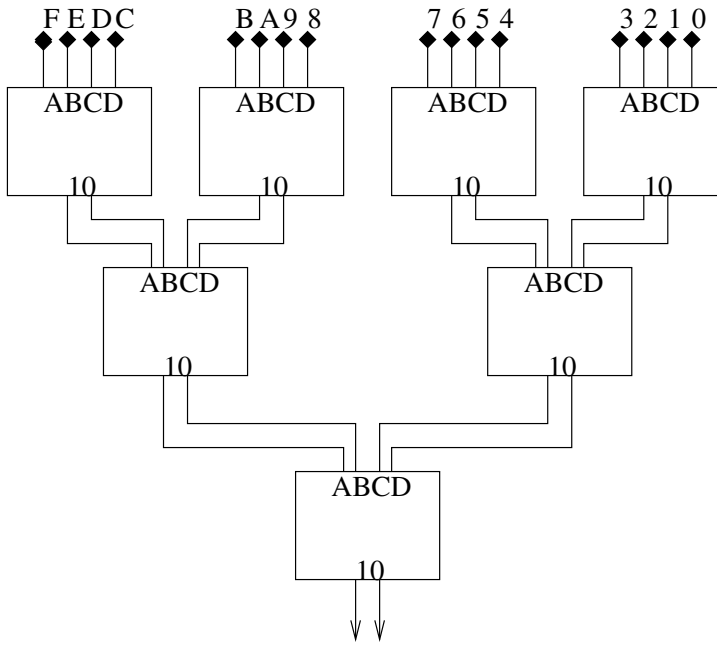

Fig. 1. Wallace tree

The expanded 4to2 adder cell is shown in fig. 2. The carry logic in FPGA CLB cell is depicted in fig. 3. Comparing these figures it is evident that the 4to2 adder can be grouped differently to use CLB fast carry logic. The only signal that is not available in CLB is the carry propagation between full adder cells - marked with the wide line in fig. 2.

The grouping is done in fig. 4. Notice the addition of M1 and M2 signals. As now we use the dedicated circuits to build carry chains then CLB resources are free. These can be used to include partial product generator in the cell. The M1

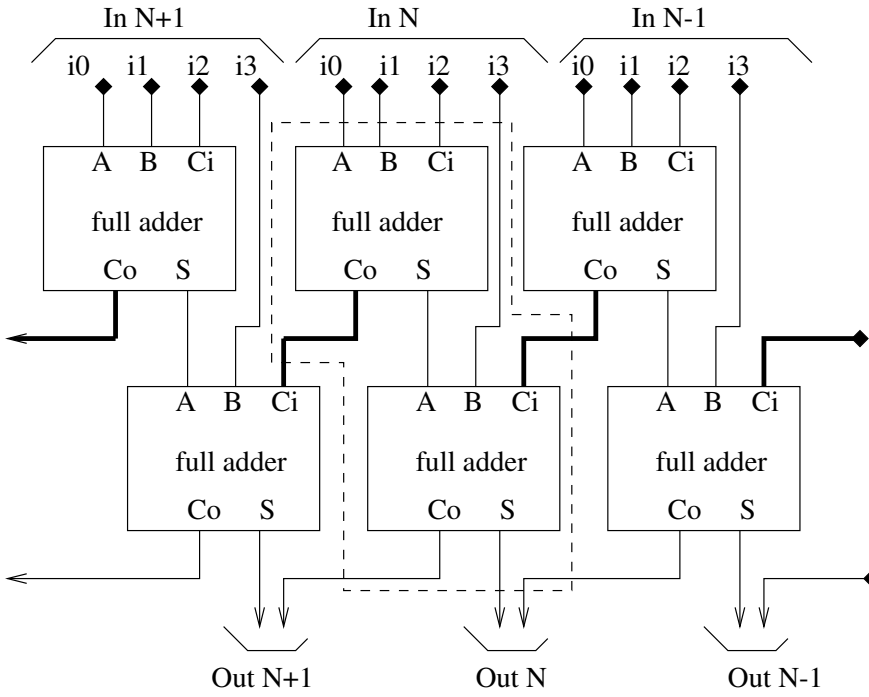

Fig. 2. 4to2 adder cell

and M2 can add any function to input lines. In case of ordinary partial product generator these can be AND-gates to mask products.

3 Tools and Tricks

The structure generators produce the output in structural Hardware Description Language (HDL) file. Synopsys Module Compiler (MC) starts with technology database. Let us first look at it.

3.1 Technology Library

The Synopsys has tool for adding components to technology library. With this tool it is possible to add two versions of full adder cell described in section 2 to technology library. The first is bare 4to2 cell and second is the cell with mask functions included. While adding these cells the designer has to be sure to set the area and delays less than respective full adder cells. After MC library creation the cell is included in datapath library. When building Wallace trees it is selected instead of plain full adder cell because area and delays are better.

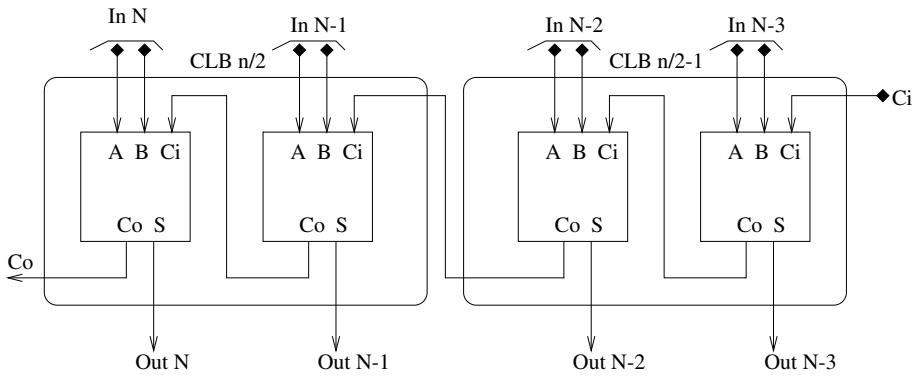

Fig. 3. FPGA CLB carry logic

3.2 Target Netlist Translation

In most cases the starting point is the produced HDL netlist. This netlist usually has instantiations of half and full adder cells. It is simple to change these cells to 4to2 sell as described in sec. 2. The algorithm is the following:

REPLACED ← NIL
for N in all nets
  if net is connected to carry out in full adder cell (A1) **and**
    A1 $\notin$ REPLACED **and**
    net is connected to carry in of another full adder cell (A2) **and**
    A2 $\notin$ REPLACED **and**
    net is **not** connected to any other nets **then**
      replace A1 and A2 with cell as in sec. 2
      REPLACED ← REPLACED $\cup$ A1
      REPLACED ← REPLACED $\cup$ A2
  end if
end for

The more complex algorithm would also study the other inputs of A1 and A2 for possibility to use additional free logic in LUT.

3.3 Structural Generator

VHDL language has the generate construct. It is a nice tool for creating structures. Using this it is possible to write generator that creates exactly the structure described in sec. 2. It is possible to keep the technology dependent part at bare minimum. In following section we will use this as a golden device to compare different solutions to it.

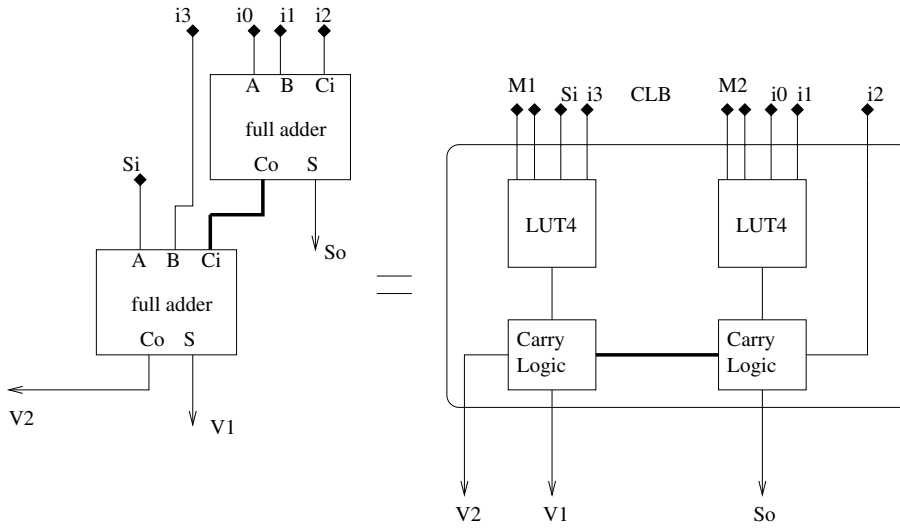

Fig. 4. 4to2 adder mapped to CLB

4 Results

The results of the experiments were not fully available for printing in proceedings but are provided in Web:
`http://www.pld.ttu.ee/~jp/FPL99/`

5 Conclusions

In this paper Wallace Tree component of integer multiplier for VIRTEX FPGA is described. The method of inferring fast carry logic is developed. Compared with current approaches the method gives better device resources utilization and yields faster multiplication.

References

1. David A. Patterson, John L. Hennessy Computer Architecture. A qualitative Approach Morgan Kaufmann Publishers, Inc., (1996)
2. Põldre, J., Tammemäe, K., Mandre, M.: Modular Exponent Realization on FPGAs. Proceedings of FPL98
3. Synopsys Module Compiler User Guide. Synopsys, Inc, ver. 1998.08
4. Michael Keating, Pierre Bricaud Reuse Methodology Manual Kluwer Academic Publishers, (1998)
5. `http://redback.fysel.ntmu.no/~pihl/iwlas98/index.html`
6. Product Specification. Virtex 2.5V. Xilinx, Inc., Nov. (1998)