IMPLEMENTATION OF EFFICIENT CONSTANT MULTIPLIER ARCHITECTURE FOR RECONFIGURABLE FIR FILTER

[1]N.S.S.PRIYA, [2]G.SATISH KUMAR

[1]M.TECH VLSID, DEPT OF E.C.E, KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY,KORANGI,ANDHRAPRADESH,INDIA,533461

[2]ASSOSIATE PROFESSOR, KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY,KORANGI,ANDHRAPRADESH,INDIA,533461

Abstract: Multiplication with constants is one of the most important operations in digital signal processing including digital filters and linear transformations like the fast Fourier transform. Runtime reconfiguration of the multiplied coefficients, also called reconfigurable single constant multiplication, is an efficient method to reduce resources by time-multiplexing and resource sharing. There are several algorithms to realize RSCM on ASICs by fusing multiple single constant multipliers with multiplexers. This may lead to large multiplexers who do not fit well to the structure of FPGAs, so alternative methods are needed. There is one FPGA specific algorithm (called ReMB method) to generate RSCMs by utilizing an FPGA specific basic element based on 4-input Look-Up Tables (LUT) in Virtex 4 FPGAs. This paper investigates an extension of this heuristic algortihm utilizing the 6-input LUTs in recent FPGAs. It is shown that a reduction of the required basic elements of 18% on average can be achieved by this approach. Moreover, convergence in terms of finding a valid RSCM solution can be guaranteed. This was not the case in the original algorithm.

Key words: Clock gating (CG), clock network synthesis, low-power design, multi-bit flip-flop (MBFF).

I.INTRODUCTION

Multiplication with constants is one of the most frequent operations in digital signal processing (DSP). At the same time, FPGAs have a growing market in DSP applications which were formerly dominated by application specific integrated circuit (ASIC) implementations. Reasons for this trend are the flexibility provided by the re-programmability of FPGAs and increasing ASIC manufacturing costs. The costs of the re-programmability of FPGAs are that FPGA

designs are typically larger, slower and consume more power than an equivalent ASIC realization [1]. Therefore, optimized implementations of DSP algorithms for FPGAs are getting more and more important. This is one of the reasons why embedded multipliers are present in the fabric of FPGAs. Nevertheless, the drawback of those fixed coarse-grained blocks is their inflexibility in word size and their limited quantity. Limited quantity is particularly critical in industrial applications when low-cost FPGAs with only few embedded multipliers have to be chosen and other parts of a design are competing for DSP resources. Thus, alternative logic-based methods for constant multiplication are required which are independent of this embedded special purpose hardware but are, on the other hand, efficient enough to narrow the gap to an ASIC realization. Therefore, optimizing the implementation of constant multiplication as shift-add-based circuit is well studied. However, switching between a given limited set of constant multiplications during run-time instead of using larger generic multipliers is important, too. Reconfigurable constant multipliers are used to realize hardware efficient run-time adaptable filters, e.g., for adaptive control and video coding applications. Specifically an application with tight reconfiguration time and resource constraints is presented, which motivates the necessity of highly optimized RCMs on FPGAs. There, an FPGA is used as co-processor in the control loop of a particle accelerator.

In addition to that, RCMs can be directly integrated into optimized time-multiplexed realizations of linear DSP transforms like DCT and FFT implementations and can be used in the context of time-multiplexed resource sharing of linear systems in general. Further applications are multi-stage filters for decimation or interpolation, like polyphase finit impuls response (FIR) filters. Therefore, the implementation of run-time reconfigurable constant multipliers using multiplexers is a well studied research field, too. However, most of the previous methods to generate RCMs were optimized to be used on ASICs. On FPGAs long routing delays have to be avoided by inserting registers in the data path. That is why previous RCM solutions perform poor when they are directly applied to FPGAs. As FPGA implementations of DSP applications are getting more and more important, the algorithms presented in this work particularly focus on implementations of RCMs on FPGAs.

2. LITERATURE SURVEY

There are many approaches that tackle constant multiplier design, but not all of them are suitable for runtime reconfiguration to any constant value. For instance, the shift and add method

produces efficient designs for most of the cases, although the resulting architectures and their features (i.e., area, delay, etc.) strongly depend on the constant value [Gustafsson et al. 2006; Nguyen and Chattejee 2000]. Other ideas are based on storing the results of partial products in lookup tables (LUT), which are added to compose the final multiplication value [Chapman 1996; Meher 2010; Wirthlin 2004]. In Wirthlin [2004], the application of such methods to FPGA designs are thoroughly studied. The main advantage of this method is that the architecture is fixed disregarding the value of the constants: only the values of the tables change. Thus, such multipliers are easier to design or to generate automatically. Therefore, they can be straightforwardly reconfigured if a different constant is required. Also, their areatime-power figure is known, and the same multipliers structure is utilized, independently of the constant value. In this article, we select this approach as the starting point to implement an on-the-fly reconfigurable constant multiplier. Our proposed circuit enhances the regular LUT-based constant multiplier from Wirthlin [2004], enabling real-time reconfiguration of the constant with no restrictions on the possible constant values. The latter point is achieved by dedicating a portion of the multiplier to compute the contents of the LUTs, given a particular constant value. Thus, the multiplier is capable of reconfiguring itself. The circuit is able to self-reconfigure without implementing any standard FPGA reconfiguration techniques. Therefore, it can perfectly replace conventional multipliers in those applications where one of its operands changes its value only at particular time steps, being constant during long intervals of time (i.e., n clock cycles or n data computations). Domains that are suited well for this situation are cryptography, gain control, channel equalization, etc.

3. RELATED STUDY

Run-time reconfiguration can be achieved by the insertion of multiplexers into SCM and MCM solutions. As this is a generalization of the basic SCM and MCM problem, the problem of finding a minimal reconfigurable SCM/MCM solution is NP-complete, too. Nevertheless, there are solutions to solve the problem of finding reconfigurable SCMs, called ReSCM in the following. An FPGA-specific algorithm is presented as ReMB method in [3] and was further analyzed and extended in [4]. An ReSCM is constructed from basic structures that fit into the basic logic elements (BLE) of FPGAs. Moreover there are three ASIC-optimized solutions. Tummeltshammer et al. [5] suggest merging several optimized SCM graphs by a recursive algorithm called DAG fusion which fuses two SCM graphs with minimal hardware effort.

Multiplexers are inserted to switch between the different constants. More than two coefficients can be included by recursively adding the related SCMs to the existing ReSCM in the same way. Chen et al. exploit similarities between different coefficients using a canonical signed digit (CSD) representation to realize ReSCMs. The number of required adders is reduced by common subexpression elimination (CSE) through searching and fusing identical patterns in the CSD representation of constants. Multiplexers are inserted to switch between the different shifts and interconnections to realize a specific constant. Faust et al. describe an algorithm which provides not only solutions for ReSCM but also for reconfigurable multiple constant multiplication (ReMCM). The authors also use an adder graph based approach with special focus on minimal logic depth.

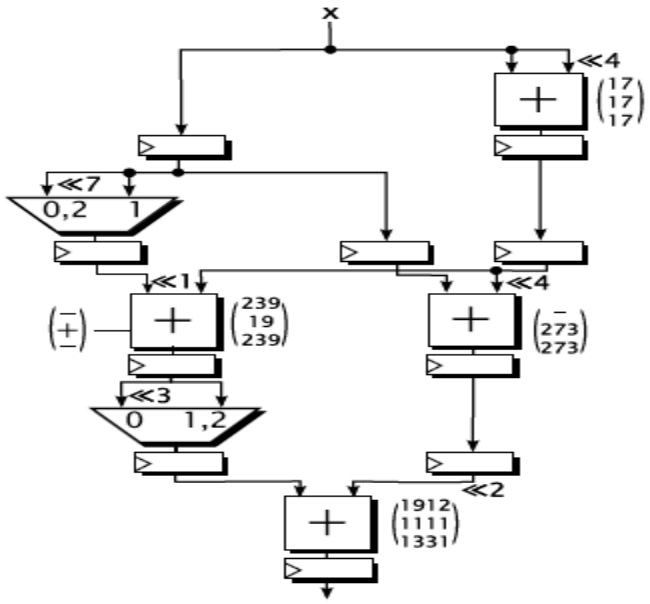


Fig.3.1. Reconfigurable single constant multiplier which can be switched between the constants 1912, 1111, 1331.

IV.PROPOSED SYSTEM

The input to the algorithm is pipelined adder graphs (PAGs) generated with the reduced pipelined adder graph (RPAG) heuristic [4]. In general, the presented fusion is not limited to RPAG generated circuits as pipelined MCM input. However, RPAG was chosen as it proved to outperform state-of-the-art MCM methods like Hcub [2] when these are optimally pipelined [5]. The results of RPAG are adder graphs representing multiplier-less pipelined constant multipliers using additions, subtractions and bit-shifts only. The main idea of multiplier-less multiplication

as applied in RPAG is to compose a constant multiplication of an addition of shifted inputs. This is beneficial because a constant shift is only a wire in hardware. All constants can be formally represented as A-operation [12], which is defined as

$$\mathcal{A}_q(u,v) = |2^{l_1}u + (-1)^{sg}2^{l_2}v|2^{-r}$$

with q = (l1; l2; r; sg), where u and v are the input constants, l1, l2 and r are shift factors and the sign bit sg 2 f0; 1g denotes whether an addition or subtraction is performed. A multiplication by 17 could for example be realized as an addition of the input with the input left-shifted by 4 (multiplication by 16). This can be seen in the leftmost example in Fig. 2. In the following subtractor, 17 times the input is subtracted from 256 times the input, which corresponds to a constant multiplication by 239. Finally, this intermediate result is left-shifted by three to get the final result of 1912 times the input. If the constant to multiply with is known in advance, this kind of realization is much cheaper in terms of resources than implementing a generic multiplier [15]. In order to automatically generate such constant multipliers, RPAG is backward-exploring reachable intermediate constants, called predecessors by evaluating the A-operation. This leads to a step-wise constant composition, starting with the required output constants. The goal of the heuristic is to select predecessors which result in the lowest number of intermediate constants in the preceding stage and which reduce the adder depth. Two more examples for such a circuit of a pipelined SCM realization can be found in Fig. 2, which are used as running example. The stage s denotes the pipeline depth of each realized constant.

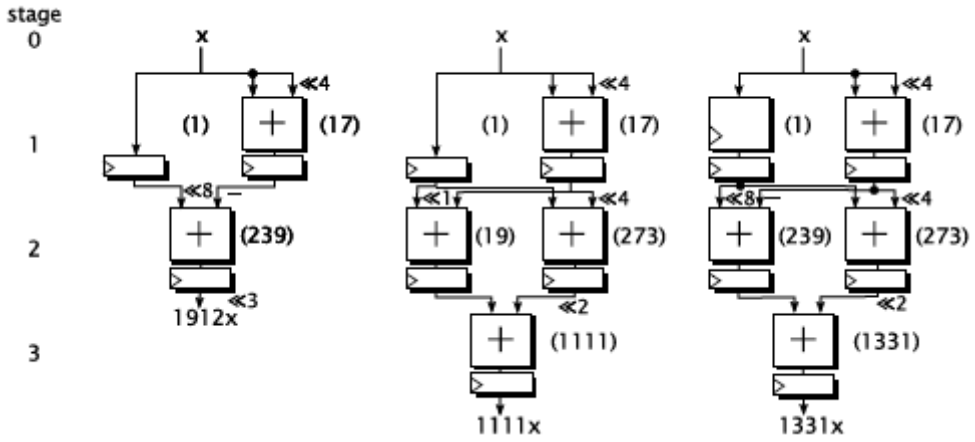


Fig.4.1. RPAG solutions for the constants 1912, 1111, 1331.

To do so, all combinations of intermediate values are evaluated and their costs are calculated separately and stored in a cost matrix. Multiplexers can appear at the inputs of the successive stage in the following cases:

1) input has a different shift value

2) input has a different source

3) both of 1) and 2)

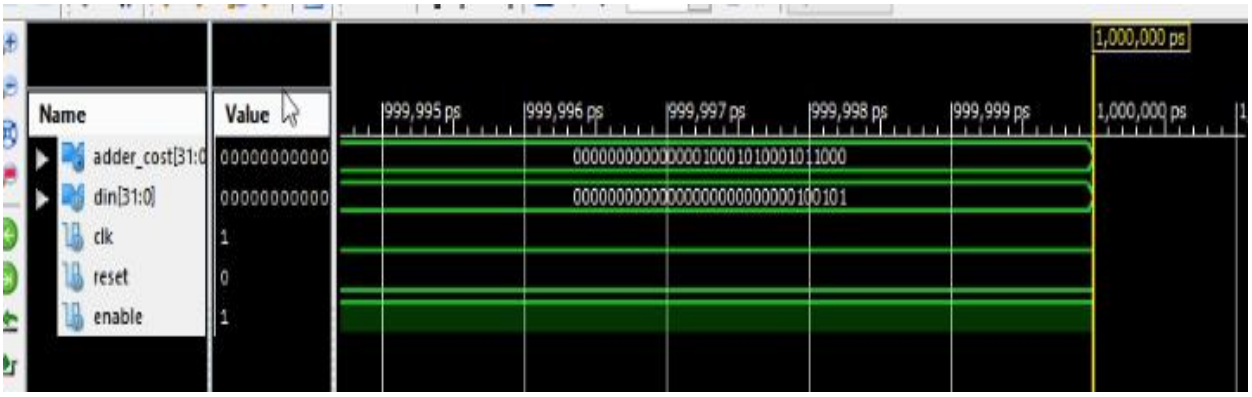
As described before, the target is to select the overall best mapping M for the specific stage s. This selection will be the source for the determination of the next preceding stage s □ 1. The procedure is repeated until the input (stage 0) is reached. A simplified pseudo-code of the generalized fusion process is given in Listing 1. It assumes that the overall best solution and costs are globally known. It is started with the constant mapping M of the output stage, the preceding stage s, the search width w (unlimited for the optimal search) and the costs of the current path current cost, which is zero in the beginning. Compared to the algorithm presented in [10] the algorithm was generalized, such that it can be used both as heuristic and in an optimal way. In contrast to an arbitrary search through the whole search space, which was done in the former version, the search is now improved and based on a sorted cost matrix. More details on the search width are provided.

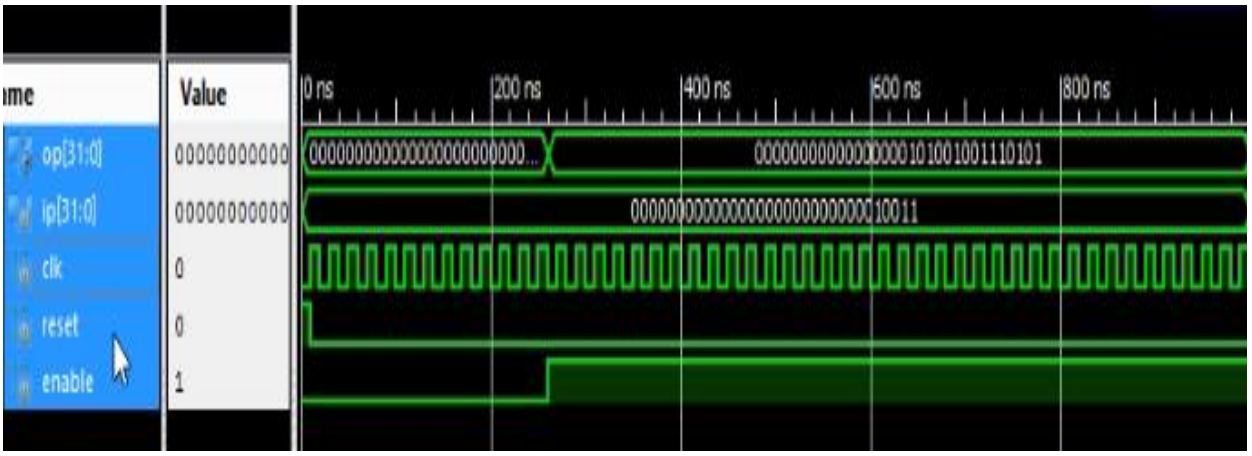
```
1  Fuse (M,s,w,current_cost)
2  if s > 0
3    C = evaluate_fusion_cost(M);
4    C = sort(C);
5    if current_cost+min(C)>=current_best_cost
6      return; -- subtree cut
7    else
8      for i = 0...w
9        if current_cost+cost(C(i))>=current_best_cost
10         return; -- normal b&b cut
11       else
12         M = mapping(C(i));
13         current_cost += cost(C(i));
14         Fuse (M,s-1,w,current_cost);
15       end if;
16     end for;
17   end if;
18 else
19   if current_cost < current_best_cost
20     current_best_cost = current_cost;
21     best_solution = current_solution;
22   end if
23 end if;
```

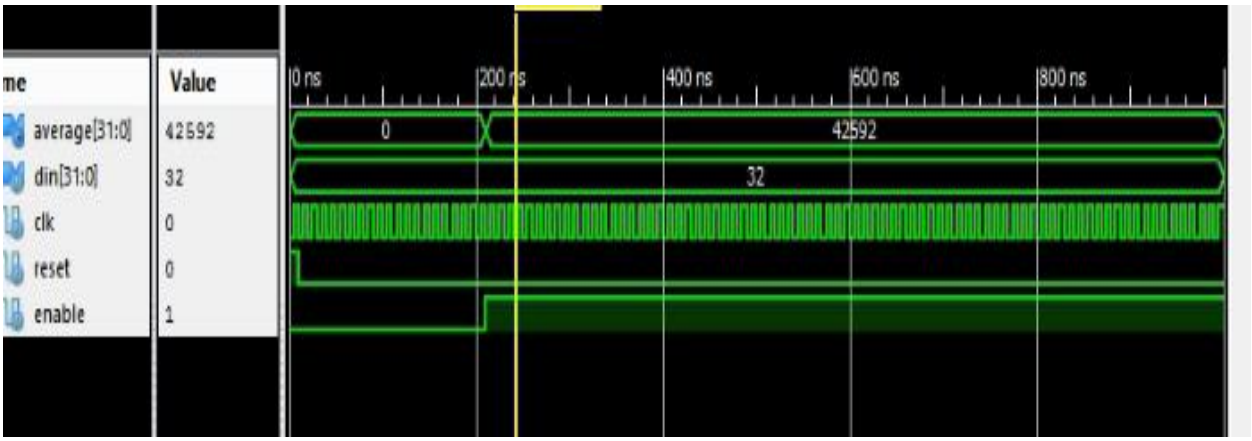
V. SIMULATION RESULTS

The results for the previous work and results for PAG fusion RMCM FIR filters for 2 to 5 configurations (conf.) are listed in TABLE III. In this case the number of configurations corresponds to the number of different FIR filter coefficient sets. The ICAP resource consumption and maximum clock frequency are noted as a range as the applied RPAG optimization heavily depends on the numeric coefficient values. In addition, a FIR filter using CoreGen multipliers together with RAM was evaluated. It can be seen that the resulting circuits of the proposed method provide the fastest reconfiguration time with a better resource consumption for 2 to 4 configurations. The large increase in slices from 4 to 5 configurations can be directly traced back to the increase of LUT costs for the 5-input multiplexers (cf. Fig). For 5 to 10 configurations it depends on the required reconfiguration time, if the reconfigurable FIR filter using distributed arithmetic or LUT multipliers together with reconfigurable LUTs or the ICAP implementation should be used. An implementation with CoreGen multipliers is only

valuable when very fast reconfiguration times and at the same time a large number of configurations are required.







Conclusion This paper presented an algorithm to generate pipelined runtime reconfigurable constant multipliers. It is based on the merging of pipeline optimized adder graphs generated

with the RPAG heuristic. Multiplexers are introduced to switch between different constant multiplications in the merged reconfigurable adder graph. Besides the generation of reconfigurable pipelined single constant multipliers (RPSCM) the shown algorithm is also able to merge reconfigurable pipelined multiple constant multipliers (RPMCM). The results of the proposed algorithm were compared to pipelined results from the ASIC-optimized ReSCM algorithm DAG fusion by the help of synthesis results. These synthesis results were obtained by automatically generated VHDL implementations of the regarded RPSCMs. The results of the proposed algorithm only need 77% of the number of slices on average compared to the pipelined solutions of DAG fusion. The differences in the solutions were analyzed by means of some resulting RPSCM graphs. That way the advantages of the proposed FPGAspecific optimal algorithm could be pointed out.

REFERENCES

[1] IEEE standard for floating–point arithmetic. IEEE Standards Committee, Oct. 2006.

[2] F. Y. Busaba, T. Slegel, S. Carlough, C. Krygowski, and J. G. Rell. The design of the fixed point unit for the z990 microprocessor. InProc. ACM Great Lakes14th Symposium on VLSI, pages 364– 367, Apr. 2004.

[3] M. F. Cowlishaw. Decimal floating-point: Algorism for computers. InProc. IEEE16th Symposium on Computer Arithmetic, pages 104– 111, July 2003.

[4] M. A. Erle and M. J. Schulte. Decimal multiplication via carry-save addition. InProc. IEEE Int'l Conference on Application-Specific Systems, Architectures, and Processors, pages 348–358, June 2003.

[5] M. A. Erle, E. M. Schwarz, and M. J. Schulte. Decimal multiplication with efficient partial product generation. In Proc. IEEE17th Symposium on Computer Arithmetic, pages 21–28, June 2005.

[6] R. D. Kenney and M. J. Schulte. High-speed multioperand decimal adders.IEEE Trans. on Computers, 54(8):953–963, Aug. 2005.

[7] R. D. Kenney, M. J. Schulte, and M. A. Erle. High-frequency decimal multiplier. InProc. IEEE Int'l Conference on Computer Design: VLSI in Computers and Processors, pages 26–29, Oct. 2004.

[8] T. Lang and A. Nannarelli. A radix-10 combinational multiplier. InProc.40th Asilomar Conference on Signals, Systems, and Computers, pages 313–317, Oct. 2006.

[9] R. H. Larson. High-speed multiply using four input carrysave adder. IBM Tech. Disclosure Bulletin, 16(7):2053–2054, Dec. 1973.

[10] N. Ohkubo and M. Suzuki. A 4.4 ns CMOS 54x54–bit multiplier using pass-transistor multiplexer. IEEE Journal of Solid State Circuits, 30(3):251–256, Mar. 1995.