International Journal of Advanced Technology and Innovative Research

www.ijatir.org

ISSN 2348–2370
Vol.11,Issue.06,
June-2019,
Pages:264-269

Implementation of FIRs using Reconfigurable Constant Multiplication

S. ASHOK[1], L.M.L NARAYANA REDDY[2]

[1]PG Scholar, Dept of ECE, PBR VITS, Kavali, AP, India.
[2]Assistant Professor, Dept of ECE, PBR VITS, Kavali, AP, India.

Abstract: This article introduces a new heuristic to generate pipelined run-time reconfigurable constant multipliers for field programmable gate arrays (FPGAs). It produces results close to the optimum. It is based on an optimal algorithm which fuses already optimized pipelined constant multipliers generated by an existing heuristic called reduced pipelined adder graph (RPAG).Switching between different single or multiple constant outputs is realized by the insertion of multiplexers. The heuristic searches for solutions that result in minimal multiplexer overhead. Using the proposed heuristic reduces the run-time of the fusion process, which raises the usability and application domain of the proposed method of run-time reconfiguration. An extensive evaluation of the proposed method confirms a 9%–26% FPGA resource reduction on average compared to previous work. For reconfigurable multiple constant multiplication, resource savings of up to 75% can be shown compared to a standard generic lookup table based multiplier. Two low level optimizations are presented, which further reduce resource consumption and are included into an automatic VHDL code generation based on the Flo Po Co library.

Keywords: FPGAs, Reconfigurable Constant Multipliers, FIRs.

I. INTRODUCTION

In this thesis optimization methods to implement run-time reconfigurable constant multipliers (RCMs) on field programmable gate arrays (FPGAs) are proposed. The performance, hardware effort, reconfiguration time and power consumption of resulting circuits are evaluated. The resulting solutions add some important trade-off points to the design space of RCM on PGAs and make new applications possible. 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. 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 in 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 accelerator.

The Multiplication with constant coefficients is an essential operation in digital signal processing. Initially one of the reasons to put embedded multipliers or DSP blocks into the fabric of field-programmable gate arrays (FPGAs) was to reduce the performance gap between application specific integrated circuits (ASICs) and FPGAs. Nevertheless, the price to pay for those fixed coarse-grained blocks is their inflexibility in word size and limited quantity. Limited quantity is particularly critical in industrial applications, when cheaper and rather small FPGAs with only few DSP blocks have to be chosen due to price pressure. Thus, logic-based constant multiplication methods are needed. Optimizing the implementation of this operation is well studied. Switching between a given set of constants of such multipliers during run-time instead of using larger generic multipliers is important to realize hardware efficient run-time adaptable filters; discrete cosine transformation and fast Fourier transform implementations [4] as well as multistage filters for decimation or interpolation like poly phase finite-impulse response (FIR) filters.

II. OVERVIEW OF EXISTING METHOD

A. Field Programmable Gate Arrays

Field programmable gate arrays (FPGAs) are array-based integrated circuits, which can be programmed and re-programmed on-site. They are regularly structured as two-dimensional array originally consisting of basic logic elements (BLEs) and an interconnecting network. Programmability is achieved by small programmable memories, which change the logic function of BLEs as well as their interconnection. The price for this programmability is that FPGA designs are larger, slower and consume more dynamic power than an equivalent application specific integrated circuit (ASIC) realization. However, FPGAs are widely used with a growing market in the aerospace and consumer electronics and automotive industry. An FPGA can thusly be programmed to implement any logical circuit, only limited by its required hardware resources a low price are a great advantage to keep the financial risk for small and medium volume developments low. This is not the case for ASICs having large non-recurring engineering costs. It shows only a small part of an FPGA which typically consists of hundreds of thousands BLEs. Each BLE consists at least of a function generator realized as look-up table (LUT) and a memory element. The input/output ports (IO) as well as the BLEs are connected to the routing network with connection blocks. Some simplified example connections are shown. Moreover, there are programmable routing switches and interconnection complexity.

B. Architecture of FPGA

Normally FPGAs comprise of

1. Programmable logic blocks which implement logic functions.
2. Programmable routing that connects these logic functions.
3. I/O blocks that are connected to logic blocks through routing interconnect and that Make off-chip connections.

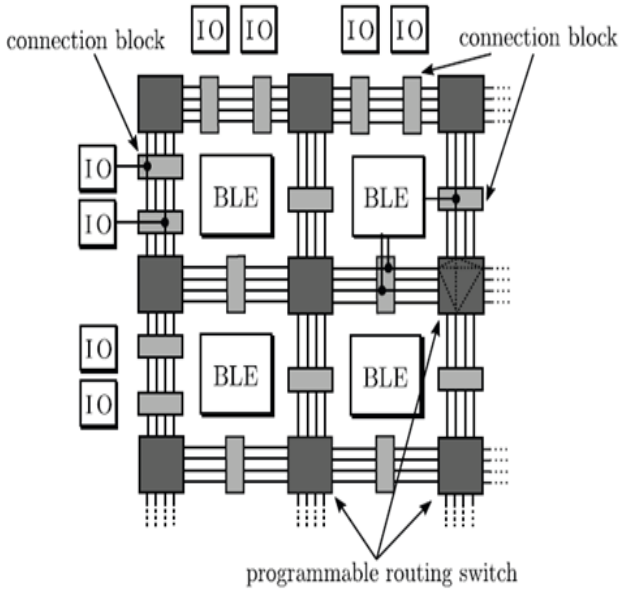

Fig 1: Simplified overview of FPGA Layout.

C. Routing Delay

Regarding the delay of a circuit, the programmability of FPGAs comes at a price. While in non-programmable integrated circuits local connections have a negligibly small contribution to the overall delay, in FPGAs such a connection can include several routing elements. Moreover, programmable parts within the connection blocks and programmable routing switches add additional delay. A common way to overcome this drawback is to add registers to the initial circuit to split up the combinatorial parts. Introducing registers has to be done systematically to keep the functionality of the original circuit. This is achieved by placing registers in the combinatorial path such that the number of introduced registers on each path from an input to an output is equal. This procedure called pipelining increases the latency of the circuit while Keepin its functionality. An overhead of required FPGA resources may occur due to a massiv register insertion. On the other side, it is very likely that unused registers in the FPGA's BLEs, which are already used for logic, will be taken. This was shown, e. g., in by a speedup of 111% of a non-pipelined circuit on a Cyclone II FPGA with a pipelining overhead of only 6 %. The more recent FPGAs presented above should provide even better results, as they have double the number of registers per BLE. Power Consumption With an increase in FPGA resources and maximum clock frequency, which can be observed for each new FPGA generation, the power consumption of an FPGA gets more and more important. The power consumed by an FPGA can be separated into the two components static power and dynamic power. Static means, that the power consumption is independent of the switching activity in the device. The main source of static power consumption is transistor leakage current, which tends to increase with decreasing technology size.

D. Routing Reconfiguration Using Logic Multiplexers On FPGAs

The programmable routing of an FPGA is fixed during run-time. However, multiplexers realized with BLEs can be used to change signal routing during run-time. This is especially interesting in the context of switching between the different circuit alternatives for resource sharing like, e. g., in multiplexer-based run-time reconfigurable constant multipliers (RCMs), which are the topic in Chapter 4. Besides their mapping into the soft logic (slices/ALMs) as shown in the following, the mapping of multiplexers onto otherwise unused DSP blocks has been investigated , too. A Virtex 6 slice consists of four 6-input LUTs, which can be used as any 6-input logic function . Hence, each LUT can be used as an up to 4:1 1-bit multiplexer. Moreover, the Virtex 6 slice includes two 2:1 multiplexers (MUXF7) to switch between two of the LUT results, which extends the usage to two up to 8:1 1-bit multiplexers. Finally, there is another 2:1 multiplexer (MUXF8) to switch between the outputs of the two MUXF7s. This means, four LUTs (= one slice) are required to build an up to 16:1 1-bit multiplexer as shown in Two of those 16:1 multiplexers can be combined to a 32:1 1-bit multiplexer utilizing only one additional LUT and so on.

Using Primitives [36] in the VHDL description makes it possible to use the slices exactly in that way.

E. Power Consumption

The power consumption for FPGAs can be estimated using the vendor tools or measured using high precision amplifiers and an oscilloscope. Estimation is done by summing up the power of used resources based on their switching activity. Moreover, the resulting data depends on an FPGA specific capacitance model provided by the FPGA vendors. It was shown by Becker et al. by a comparison to a power measurement that the data gained by the vendor tool Xilinx Power Estimator (XPE) is reliable. Therefore, XPE is used for the analysis of run-time reconfigurable circuits. For this purpose a netlist after Place and Route is simulated using ModelSim using random input data. Then, the simulated switching behavior as well as the netlist are used as input for the power analysis in XPE. However, XPE does not provide a method to estimate the power consumption of the partial reconfiguration process. While there are investigations how to model reconfiguration power, e.g. by Bonamy et al, a power estimation tool for PR power is still not available.

III. PROPOSED METHODOLOGY

This project introduces a new heuristic to generate pipelined run-time reconfigurable constant multipliers for field programmable gate arrays (FPGAs). It produces results close to the optimum. It is based on an optimal algorithm which fuses already optimized pipelined constant multipliers generated by an existing heuristic called reduced pipelined adder graph (RPAG).Switching between different single or multiple constant outputs is realized by the insertion of multiplexers. The heuristic searches for solutions that result in minimal multiplexer overhead. Using the proposed heuristic reduces the run-time of the fusion process, which raises the usability and application domain of the proposed method of run-time reconfiguration. An extensive evaluation of the proposed method confirms a 9%–26% FPGA resource reduction on average compared to previous work. For reconfigurable multiple constant multiplication, resource savings of up to 75% can be shown compared to a standard generic lookup table based multiplier. Two low level optimizations are presented, which further reduce resource consumption and are included into an automatic VHDL code generation based on the Flo Po Co library.

A. Pipelined Adder Graphs

The input to the algorithm are PAGs generated with the RPAG heuristic. 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, when these are optimally pipelined. 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 is performed. A multiplication by 17 could, for example, be realized as an addition of which is defined as

$$Aq(u,v) = |2^{l1}u+(-1)^{sg}2^{l2}v|2^{-r} \quad (1) \text{ with } q= (l1, l2, r, sg) \quad (1)$$

Where, u and v are the input constants, l1, l2, and r are shift factors and the sign bit $sg \in \{0, 1\}$ denotes whether an addition or subtraction the input with the input left-shifted by 4 (multiplication by 16). In the following subtraction, 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.

B. Improved Pipelined Adder Graph Fusion

Just like RPAG, the proposed PAG fusion is backward exploring. Starting with the constant mapping of the output stage all PAGs are fused stage by stage. The basic idea is to combine those intermediate values in the respective preceding stage to share the same adder, which leads to a minimal overhead of possibly necessary multiplexers or switchable adder/subtractors. 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 curretcost, which is zero in the beginning. Compared to the algorithm presented in [20] 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.

In the running example used here, the three SCM graphs generated with RPAG (see Fig. 2) are fused starting with the desired output mapping M = {1912; 1111; 1331}, meaning that the resulting circuit can be switched between these three values. This will be called an SCM circuit with three configurations in the following. The enumeration of all adder combinations of the second last stage consisting of {239} for the first, {19}, {273} for the second and {239}, {273} for the third SCM solution, respectively. For the constant 1912 only one adder is required in stage two, but two adders are required in the other SCM circuits. The cost evaluation is following the assumption that the multiplexers will be

realized as a cascade of 2 : 1 multiplexers. Thus, N −1 2 : 1 multiplexers are required to switch between N configurations, which leads to a contribution of each used multiplexer input of cost

$$MUX = N - 1/\ N \quad (2)$$

As a zero input can be realized by resetting the succeeding register, these inputs are not considered as multiplexer inputs as our implementation targets pipelined implementations. Cost Matrix for Stage 2 Fusion of the given example

	19, 239	273, 273	273, 239	19, 273
239	**1.33**	1.33	2	2
–	0.67	**0.67**	1.33	1.33

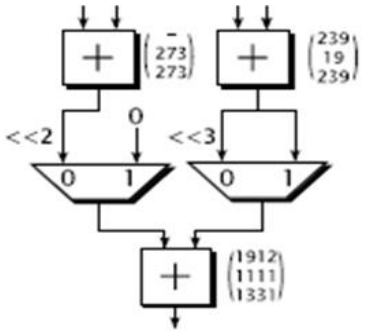

Fig 2: Result of combining 239, 19, 239, and -, 273, 273 as preceding adders.

The multiplexer cost for each mapping is stored in a multidimensional cost matrix C (line 3 in Listing 1). The cost matrix for the combinations of the current stage can be found in Table I in a 2-D representation. Thus, finding the cheapest mapping solution M for a specific stage reduces to finding the valid solution with the lowest sum of costs. An example for such a selection is given in Fig. 4.2. It corresponds to the highlighted selection in Table I with a total cost contribution of 1.33 + 0.67 = 2 2 : 1 multiplexers. The cheapest solution for a specific stage is not necessarily the best overall choice as it affects the costs in the preceding stages. So, to find the optimal solution, a full search over all possibilities is necessary.

IV. CONSTANT MULTIPLICATION ON INTEGRATED CIRCUITS

Multiplication with constants is an essential arithmetic operation and used in nearly any DSP algorithm. The implementation of this operation on integrated circuits (ICs) is thus a well studied research topic. Instead of using generic multipliers, constant multiplication is implemented multiplier-less using additions, subtractions and bit shifts. This is advantageous as bit shifts can be realized as wires on ICs and special properties in the constant's number representation can be individually exploited.

A. Look-Up Table Based Constant Multiplication

So far, the focus of the introduction of multiplier-less constant multiplication has been on implementations using additions and bit shifts. Alternatively,constant multiplication can be performed by dividing the multiplication into partial products. These partial products are then realized with LUTs or block RAMs. The basic concept known as KCM was described by Chapman, further extended by pipelining. This method is especially interesting for FPGAs, due to their LUTs in the BLEs and the presence of block RAM on nearly every recent FPGA. Again, the multiplication of a number x by a constant c is considered. As the sign bit for signed multiplication is important it is included in the derivation from the beginning. The two's complement representation of a signed number x with a width of Bx bit is

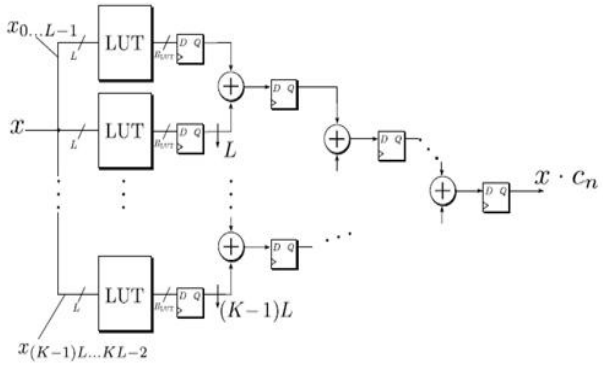

Fig3:LUT-based constant multiplication using pipelined adder tree.

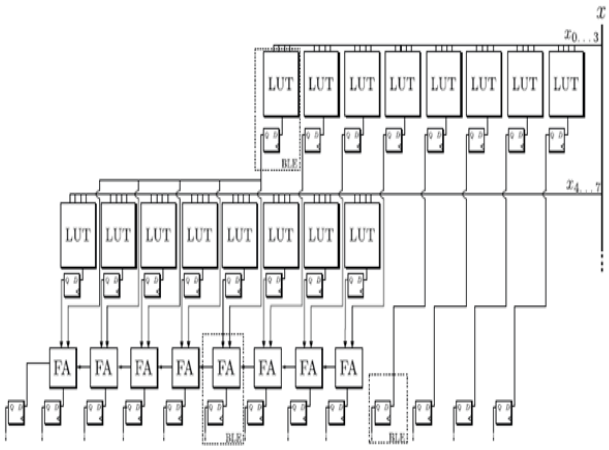

Fig 4: Example implementation of an 8x4 bit LUT-based signed multiplier.

B. Reconfigurable Constant Multiplication using LUTs

In this section, a reconfigurable constant multiplication method using look -up tables (LUTs) is presented. It is based on constant multiplication using LUTs known as KCM described by Chapman. Reconfiguration is achieved by changing the LUT contents of partial products during run-time. This is achieved by using the logic reconfiguration in Xilinx FPGAs prand was originally published. After a short introduction of related work, the architecture of the reconfigurable LUT multiplier is presented and compared to a generic multiplier implementation and constant multipliers reconfigured using Partial Reconfiguration (PR) and the Internal Configuration Access Port (ICAP). Finally, an application of the presented reconfiguration concepts is shown using run-time reconfigurable finite impulse response (FIR) filter architectures.

C. Reconfiguration Time

The time that is needed to reconfigure the multiplier is an important requirement in time-critical applications. The fastest reconfiguration can be provided by the generic multiplier architecture, whose coefficient can be changed within one clock cycle. For the presented reconfigurable KCM-based LUT multiplier design 32 reconfiguration clock cycles are required until the output of the multiplier is valid again (90.65 ns for the slowest design). This is very fast compared to the PR concept offered by Xilinx via the ICAP. There, the reconfiguration time depends on the size of the partial bit file replacing the currently running constant multiplier bit file. With a maximum reconfiguration clock rate of 100 MHz and 32 bit data with the reconfiguration time for a given partial bit file size.

V. SIMULATION RESULTS & DISCUSSIONS

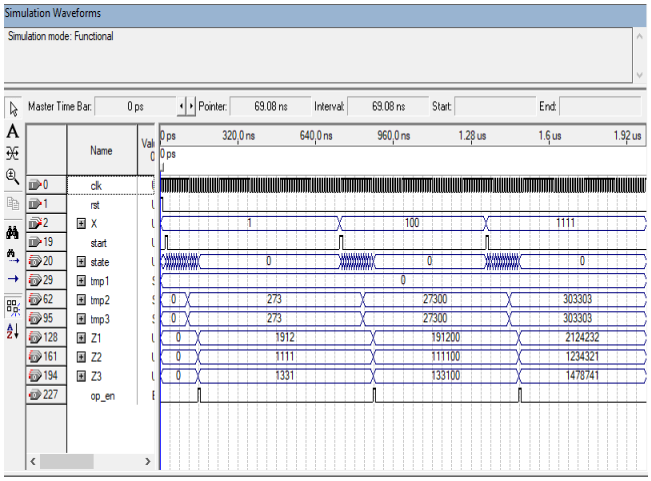

Fig 5: Result of the Existing system RPAG.

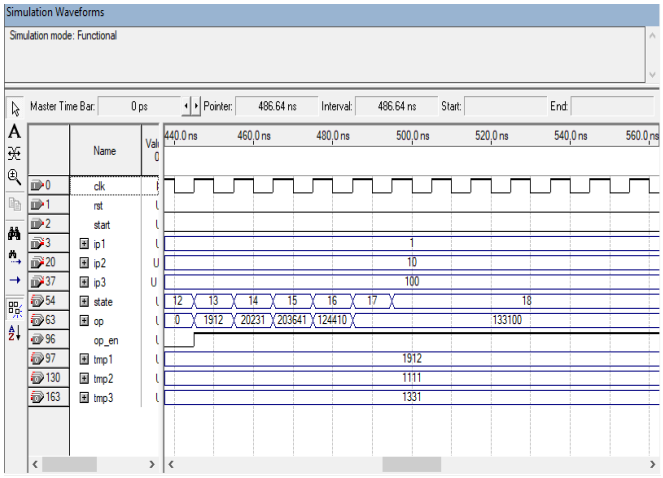

Fig 6: Result of the Proposed system FIR.

VI. CONCLUSION AND FUTURE WORK

In this section a run-time reconfigurable constant multiplication circuit based on reconfigurable LUTs was presented. It could be shown that the proposed realization can be beneficially used instead of a generic multiplier in terms of required hardware resources and provides a solution with short reconfiguration times compared to using constant coefficient multiplier IP cores and Partial Reconfiguration via the ICAP. An experimental evaluation showed that the CFGLUT reconfiguration with only 32 clock cycles of reconfiguration time provides important trade-off points in the design space for run-time reconfigurable constant multipliers on FPGAs. Moreover, two run-time reconfigurable FIR filter architectures using LUT reconfiguration were introduced and analyzed in this chapter. A direct integration of the presented FIR filter architecture into the adaptive control loop of a particle accelerator could be shown in. For this application no other FIR filter implementation than the reconfigurable KCM based multiplier presented here and was able to fulfill the strict reconfiguration time and FPGA resource constraints. Finally, decision support for selecting one of the presented architectures for an FIR filter with given filter length and word sizes was provided and supported by an experimental evaluation. A comparison of the presented reconfigurable FIR filter approaches to reconfigurable FIR filters using other Reconfiguration methods.

VII. REFERENCES

[1] S. S. Demirsoy, I. Kale, and A. G. Dempster, "Reconfigurable multiplier blocks: Structures, algorithm and applications," Circuits Syst. Signal Process., vol. 26, no. 6, pp. 793–827, 2007.

[2] L. Aksoy, P. Flores, and J. Monteiro, "Multiplierless design of folded DSP blocks," ACM Trans. Design Autom. Electron. Syst., vol. 20, no. 1, Nov. 2014, Art. no. 14.

[3] P. Lowenborg and H. Johansson, "Minimax design of adjustablebandwidth linear-phase FIR filters," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 53, no. 2, pp. 431–439, Feb. 2006.

[4] M. Garrido, F. Qureshi, and O. Gustafsson, "Low-complexity multiplierless constant rotators based on combined coefficient selection and shift-and-add implementation (CCSSI)," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 7, pp. 2002–2012, Jul. 2014.

[5] M. Faust, O. Gustafsson, and C.-H. Chang, "Reconfigurable multiple constant multiplication using minimum adder depth," in Proc. 44th Asilomar Conf. Signals Syst. Comput. Conf. Rec., Pacific Grove, CA, USA, Nov. 2010, pp. 1297–1301.

[6] U. Meyer-Baese, J. Chen, C. H. Chang, and A. G. Dempster, "A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters," in Proc. IEEE Asia Pac. Conf. Circuits Syst. (APCCAS), Singapore, Dec. 2006, pp. 1555–1558.

[7] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," IEEE Trans. Acoust. Speech Signal Process., vol. 32, no. 5, pp. 1037–1041, Oct. 1984.

[8] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," IEE Proc. Circuits Devices Syst., vol. 141, no. 5, pp. 407–413, Oct. 1994.

[9] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Extended results for minimum-adder constant integer multipliers," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), vol. 1. Scottsdale, AZ, USA, 2002, pp. I-73–I-76.

[10] J. Thong and N. Nicolici, "A novel optimal single constant multiplication algorithm," in Proc. ACM/IEEE 47th

Design Autom. Conf.(DAC), Anaheim,CA, USA, Jun. 2010, pp. 613–616.
[11] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," IEE Proc. G Circuits Devices Syst., vol. 138, no. 3, pp. 401–412, Jun. 1991.
[12] Y. Voronenko and M. Püschel, "Multiplierless multiple constantz multiplication," ACM Trans. Algorithms, vol. 3, no. 2, May 2007, Art. no. 11.
[13] (2016). SPIRAL-Project. [Online]. Available: http://www.spiral.net [14] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Seoul, South Korea, May 2012, pp. 49–52.
[15] M. Kumm, "Multiple constant multiplication optimizations for field programmable gate arrays," Ph.D. dissertation, Digit. Technol. Group, Elect. Eng. Comput. Sci., Univ. Kassel, Kassel, Germany, 2016.
[16] (2016). PAGSuite Project Website. [Online]. Available: http://www.uni-kassel.de/go/pagsuite
[17] P. Tummeltshammer, J. C. Hoe, and M. Puschel, "Time-multiplexed multiple-constant multiplication," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 26, no. 9, pp. 1551–1563, Sep. 2007.
[18] J. Chen and C.-H. Chang, "High-level synthesis algorithm for the design of reconfigurable constant multiplier," IEEE Trans. Comput.- Aided Design Integr. Circuits Syst., vol. 28, no. 12, pp. 1844–1856, Dec. 2009.
[19] K. Möller, M. Kumm, B. Barschtipan, and P. Zipf, "Dynamically reconfigurable constant multiplication on FPGAs," in Proc. Workshop Methoden Und Beschreibungssprachen Zur Modellierung und Verifikation Von Schaltungen Und Systemen (MBMV), Böblingen, Germany, 2014, pp. 159–169.
[20] K. Möller, M. Kumm, M. Kleinlein, and P. Zipf, "Pipelined reconfigurable multiplication with constants on FPGAs," in Proc. 24th Int. Conf. Field Program. Logic Appl. (FPL), Munich, Germany, 2014, pp. 1–6.
[21] M. Kumm, M. Hardieck, J. Willkomm, P. Zipf, and U. Meyer-Baese, "Multiple constant multiplication with ternary adders," in Proc. 23rd Int. Conf. Field Program. Logic Appl. (FPL), Porto, Portugal, Sep. 2013, pp. 1–8.

Author's Profile:

L.M.L Narayana Reddy working as Assistant Professor in the department of ECE, PBR Visvodaya Institute of Technology & Science, Kavali with total teaching experience of 8 years. He has guided 7 PG and 14 batches of UG students. His areas of interest include VLSI.

S. Ashok has received his B.Tech degree in Electronics and Communication Engineering from JNTU, Anantapur in 2014 and pursued M.Tech degree in VLSI from PBR VITS, affiliated JNTU, Anantapur in 2017.