

A COMPARATIVE STUDY
OF DIFFERENT
MULTIPLY
ACCUMULATE
ARCHITECTURE
IMPLEMENTATIONS ON
FPGA

USING DISTRIBUTED ARITHMETIC AND
RESIDUE ARITHMETIC

PREPARED BY MUSTAFA IMRAN ALI #230203
SUBMITTED TO DR. ALAAELDIN AMIN
SPRING 2004

Table of Contents

ABSTRACT	2
1. INTRODUCTION	2
2. FPGA ARCHITECTURE OVERVIEW	2
3. COMPARISON METHODOLOGY	3
4. DISTRIBUTED ARITHMETIC ARCHITECTURES	4
4.1 BRIEF OVERVIEW.....	4
4.2 ARCHITECTURE COMPONENTS	4
4.2.1 <i>parallel-in serial-out shift register chain</i>	4
4.2.2 <i>DA look-up tables and adders</i>	5
4.2.3 <i>ScalinG Accumulator</i>	6
4.3 COMPLETE ARCHITECTURE	6
5. RESIDUE ARITHMETIC ARCHITECTURES	7
5.1 BRIEF OVERVIEW.....	7
5.2 ARCHITECTURE COMPONENTS	7
5.2.1 <i>Modulo adders</i>	7
5.2.2 <i>mac channels</i>	8
5.2.3 <i>Forward converter</i>	8
5.2.4 <i>reverse converter</i>	9
5.3 COMPLETE ARCHITECTURE	9
6. DA-RNS ARCHITECTURE	9
6.1 OVERVIEW	9
6.2 MODULO SCALING ACCUMULATOR.....	10
6.3 COMPLETE ARCHITECTURE.....	10
7. COMPARISON AMONG ARCHITECTURES	10
8. CONCLUSION	11
BIBLIOGRAPHY	11

A Comparative Study of Different Multiply Accumulate Architecture Implementations on FPGA

USING DISTRIBUTED ARITHMETIC AND RESIDUE ARITHMETIC

ABSTRACT

The proposed theoretical benefits of DA and RNS for realizing the full potential of FPGA architecture for hardware implementation of MAC calculation and achieving large parallelism are tested in this work. The relative area and speed efficiencies of DA and RNS based hardware MAC implementation on FPGA were analyzed. It has been found that though the FPGA has support for efficient implementation of components required in these architectures, the DA approach is superior. Compared to RNS, DA approach can achieve near to maximum clock rates possible with a given FPGA technology using only basic 4-LUT based blocks and the fast ripple carry chains while the multi stage modulo adders required in RNS implementation are slow, even for small word lengths, and as such the accumulator stage becomes the performance bottleneck. Also, RNS architecture require a large area overhead in forward converters using a direct implementation of CRT, besides being a speed bottleneck due to the large modulo adders required.

1. INTRODUCTION

Distributed arithmetic (DA) based multiply accumulate (MAC) implementation has gained wide spread acceptance for field programmable gate arrays (FPGA) based digital signal processing computations [1]. The primary for this being the inherent support in FPGA architecture for implementing components of DA computation in an efficient way [2]. On the other hand, residue arithmetic (RA) based MAC architecture have been touted for their parallel multiply and add operations using a number of small moduli. The main RA advantage comes from being able to implement operations in small LUTs (for 4-6 bits moduli) and shorter carry chains in adders. Again, these features have led the research community to explore the benefits of RA for FPGA based implementations. There also exists the possibility of using residue number system (RNS) instead of two's complement (2C) for DA and as such gain advantages of smaller word lengths. The work presented in this report aimed to test these claims using a typical FPGA and to compare the relative efficiencies of DA, RNS and DA-RNS based approaches. Only the basic FPGA features were used in order to not bias the results to a particular FPGA architecture. In this work it was found that though there appear to be advantages of RA due to short word lengths, but there are implementations issues which negate these advantages or at best give results just comparable with 2C DA based implementation.

This report is organized as follows. In section 2 a brief overview of the chosen FPGA architecture is given. Section 3 describes the comparison methodology and implementation method. Section 4 gives the implementation details for DA architecture which is followed by details of RNS and DA-RNS architectures in sections 5 and 6. In section 7, the architectures are compared and finally section 8 concludes.

2. FPGA ARCHITECTURE OVERVIEW

The basic element of many commercial FPGAs consist of a block with a 4-input LUT and a output that may be registered or not. Besides this, most FPGAs offer dedicated high speed carry

logic for fast carry propagation chain implementations. The 4-LUTs and the associated programmable logic are often organized into groups of 4 to 10 elements. These groups are called configurable logic element (CLB) or a logic array block (LAB) in Xilinx [3] and Altera [4] FPGA architectures respectively. The CLBs are further divided into 2 slices of 2 basic elements each. Also, there is additional logic in each CLB that allows combination of 4-LUTs to form larger LUTs of up to 5 and 6 inputs. The routing architecture has support for fast local connections between adjacent CLBs to bypass the slower global connections.

The architecture chosen for comparing different MAC implementations in this work is the Xilinx Virtex FPGA (xcv50-5tq144). It is a 0.22 μ m 5 layer metal device that can achieve register-to-register clock performance of up to 200 MHz (device grade 6) [3]. It comes under the middle range category of FPGAs. The slice based CLB structure of the Virtex FPGA is shown in figure 1.

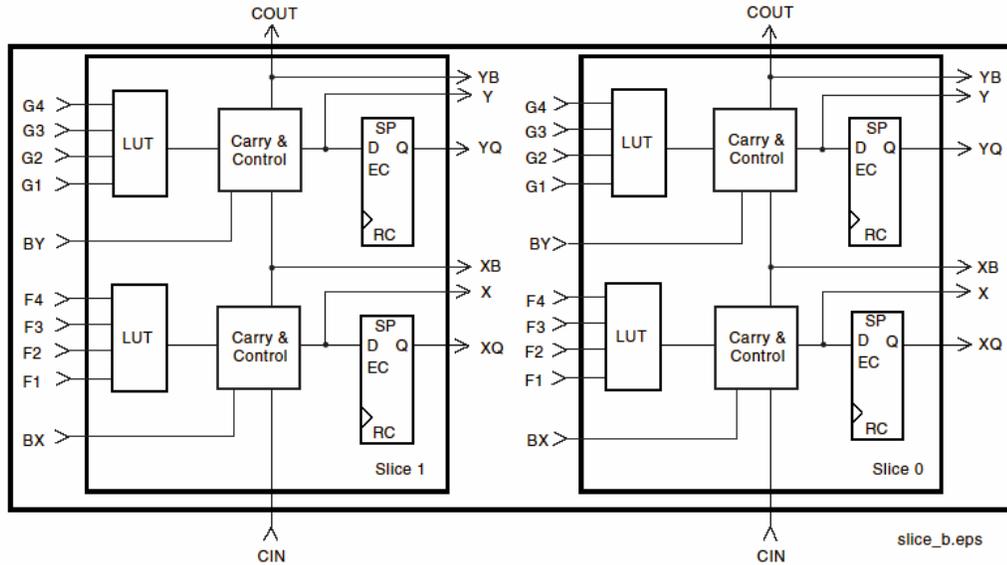


Figure 1: 2 Slice Virtex CLB

3. COMPARISON METHODOLOGY

Since the purpose of the comparison is to evaluate the efficiency of different MAC architectures for FPGA implementation comparison that is applicable to many different FPGAs, only the basic FPGA architecture elements that are common to most commercial FPGAs were used for implementations. Although, a large number of FPGAs today feature large dedicated blocks of SRAM, these were not used due to their architecture specific structure. Thus, the results of implementations are applicable to a fairly large degree to most other FPGAs as well.

The inputs descriptions for all designs were made using the schematic entry tool of the Xilinx ISE v6.1 environment. All designs were done using optimized macros for the chosen architecture. This was necessary to avoid tainting the results with mapping inefficiencies and optimize the implementations as close as possible for the given architecture. This does not lead to drastically different results across FPGA families because a given macro is automatically optimized for the chosen device and besides, every FPGA vendor provides a library of optimized macros for its devices. The implementation details in terms of FPGA primitives of each macro used is given in Appendix C. The synthesis was optimized for speed using a high optimization

effort. The architectures were implemented using both pipelined and non-pipelined components to observe the effect of register insertion on area and delay.

In the next three sections, implementations results are given. Appendix A and B give the detailed synthesis reports for each architecture components and schematics respectively.

4. DISTRIBUTED ARITHMETIC ARCHITECTURES

4.1 BRIEF OVERVIEW

The architecture for a serial (one bit at a time) DA computation is shown in figure 2. Parallelism can be obtained by adding more DALUTs and adders and shifting more than one bit at a time [5]. The basic serial architecture requires N clock cycles to process N-bits input operands, independent of the number of input operands.

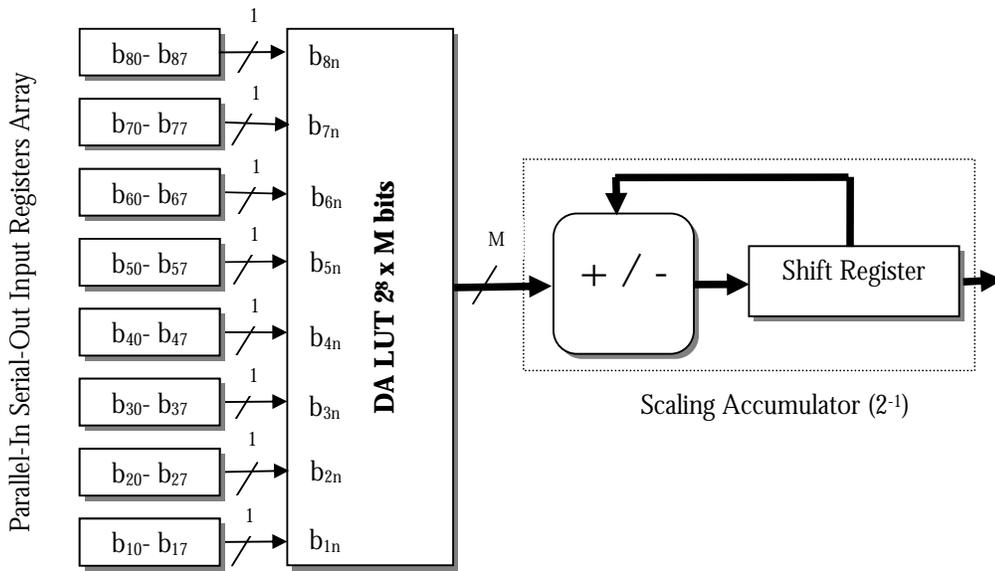


Figure 2: Architecture block diagram for a serial (one bit at a time) DA computation

4.2 ARCHITECTURE COMPONENTS

In this section, the implementations of the DA architecture components are described and then in the next section, implementation results for the complete architecture are given. For detailed, implementation information on each module and the complete architecture, Appendix A and B should be referred.

4.2.1 PARALLEL-IN SERIAL-OUT SHIFT REGISTER CHAIN

The DA architecture requires that all the input operands be available before the DA computations can take place. For this, the operands are loaded in the registers through a register chain. The operands are then shifted one, two or, in general, n bits at a time (LSB first or MSB first) depending upon whether a serial (one bit at a time) or parallel (n bits at a time) architecture is implemented. Structurally, shift registers are a chain of flip-flops with a MUX at the data input to select between the parallel data input or the output of the adjacent flip-flop (See appendix C). The logic for a single element is implemented completely in half a slice (1/4 of a CLB) and the chain is implemented in a series of consecutive CLB slices. A single 4, 8 or 16 bits shift register is implemented using the available macro. The shift chain is implemented by connecting the outputs and inputs of the consecutive shift registers. The implementation of 8 and 16 bits

parallel-in serial-out shift register chains of length 4 is given in Appendix. Chains longer than 4 registers are constructed by cascading. The area and delay values for 8-bits and 16-bits and 32-bits chains of length 4 are given in table 1.

Table 1

SHIFT REGISTER CHAIN	SLICES USED	SLICE FLIP-FLOPS	DELAY
8-bits	16	32	6.923ns
16-bits	32	64	6.923ns

4.2.2 DA LOOK-UP TABLES AND ADDERS

The complete NxM-bits DALUT, where M is the coefficient cumulative precision, is implemented by decomposing it into smaller 4-input DALUTs (4xM-bits ROMs) and using an adder tree to add the outputs of individual 4-input DALUTs. For example, a 8x12-bits DALUT is implemented using two 4x12-bits DALUTs. Each 4-input DALUT is implemented by using an array of basic 4x1-bit LUTs (4-LUTs). The number of 4-LUTs in the array depends upon word length used for the multiplication coefficient representation, i.e. 12-bits coefficients would require 12 4-LUTs in an array. The area and delay values for 4x12-bits and 4x16-bits DALUTs are given in table 2.

Table 2

DALUT	SLICES USED	SLICE FLIPS-FLOPS	DELAY
4x12 (unregistered outputs)	6	-	0.642ns
4x12 (registered outputs)	13	12	2.458ns ¹
4x16 (unregistered outputs)	8	-	0.642ns
4x16 (registered outputs)	17	16	2.458ns ¹

¹ Includes the setup time of output flip-flops

Adders are implemented using the dedicated fast carry chain logic. The width of the adders required depends upon the co-efficient precision used. The area and delay values for 12-bits and 16-bits adders are given in table 3.

Table 3

ADDER	SLICES USED	SLICE FLIP-FLOPS	DELAY
12-bits	6	-	2.331ns
16-bits	8	-	2.527ns

4.2.3 SCALING ACCUMULATOR

The output of the DALUT (or the adder tree stage) is added to the scaled output in the scaling accumulator. It is implemented using an M-bits adder with a N+M bits shift register at the output. The area and delay values for 20-bits and 32-bits scaling accumulators are given in table 4.

Table 4

SCALING ACCUMULATOR	SLICES USED	SLICE FLIP-FLOPS	DELAY
20-bits	17	21	5.340ns
32-bits	25	33	5.536ns

4.3 COMPLETE ARCHITECTURE

The area and delay values for 4-tap 8-bits input 12 bits coefficient, 8-tap 8-bits input 12 bits coefficient and 16-tap 16-bits input 16-bits coefficient DA architectures are given in table 5.

Table 5

NO.	DA CONFIGURATION	SLICES USED	SLICE FLIPS-FLOPS	CRITICAL PATH/SPEED
1	4-taps 8-bits input 12 bits coefficient	39	53	8.482ns/117.897MHz
2	4-taps 8-bits input 12 bits coefficient (fully pipelined)	51	77	7.943ns/125.897MHz
3	8-taps 8-bits input 12 bits coefficient	67	85	11.424ns/87.535MHz
4	8-taps 8-bits input 12 bits coefficient (one pipeline stage)	73	97	8.482ns/117.897MHz
5	8-taps 8-bits input 12 bits coefficient (fully pipelined)	86	121	7.943ns/125.897MHz
6	16-taps 16-bits input 16 bits coefficient (fully pipelined)	269	401	8.343ns/119.861MHz

The critical path in the DA architecture extends from the input shift register output through the DALUT, adder tree and finally through the adder stage in the scaling accumulator. Without the pipeline registers, the critical path delay is dominated by the adder delays. It was found that when the architecture is fully pipelined, the adder delays are completely masked out by the high fan-out loading delay incurred at the output of the shift register feeding the DALUT inputs. This effect of this can be seen in the table 5 showing that No. 6 has a larger critical path delay due to

higher loading (16) compared to No. 2 and 5 (12). If the loading factor is taken care of, then the critical path delay is dominated by the adder delays, which can be improved drastically by using the approach described in [6]. But unless the fan-out delays are improved upon, there will be no gain in using faster adder stages.

It can be observed from the implementation results that the speed up of DA architectures can scale-up almost linearly by employing parallelism with more than one bit at a time implementations using more hardware and pipelining. Adding parallelism is equivalent to replicating the basic structure as many times as required, each of which can operate independently without degradation in clock frequency due to pipelining. This inference is drawn from the results of configuration No. 2 and 5, since 5 is basically a replicated No. 2 configuration till the DALUT stage with an added adder stage. The operation frequency of both due to pipelining remains same. Also, each stage in the DA computation is just a single basic FPGA element wide and this leads to exploiting the maximum possible clock frequencies possible with a given FPGA device.

5. RESIDUE ARITHMETIC ARCHITECTURES

5.1 BRIEF OVERVIEW

The overall structure of RA based MAC implementation is shown in figure 3. The implementation of components in the architecture is described next.

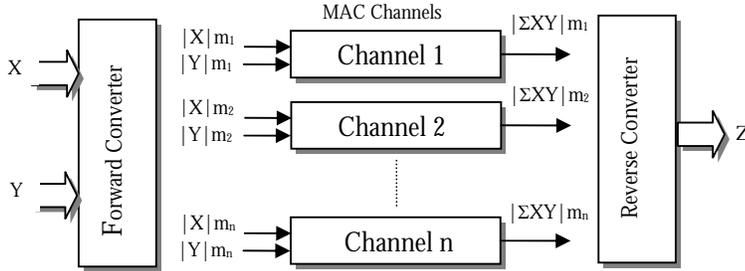


Figure 3: RNS based MAC architecture block diagram

5.2 ARCHITECTURE COMPONENTS

5.2.1 MODULO ADDERS

Modulo adders are the basic building blocks of the RNS based system. They are required in the forward converters, MAC channels as well as in reverse converters. For moduli greater than 2 bits, implementing adders using 4-LUTs is not possible and as such modulo adders were implemented using binary adders of width $\lceil \log_2 m \rceil$ for a modulo m adder using the approach described in [7]. A number of techniques are possible for modulo adder implementation and can even be specialized for different modulo types [8]. To keep the architecture implementation uniform across modulo channel and, since the speed is governed by the slowest implementation, a generic modulo adder was implemented using the fastest technique [7]. The modulo adder can be implemented as a three stage pipeline and the implementation results are given in table 6. It should be noted here that the delay of modulo adders are dominated by the last MUX stage due to the high fan-out loading of carry-or gate output signal and not by the adder delays which are only 1.988ns and 2.968ns for 5 and 25 bit adders. The fan-out delays are 2.200ns and 4.700ns for a fan-out of 10 and 50 in case of 5 and 25 bits modulo adders respectively

Table 6

MODULO ADDER	SLICES USED	SLICE FLIPS-FLOPS	DELAY
5-bits (non pipelined)	6	-	11.127ns
5-bits (pipelined)	13	12	7.619ns
25-bits (non pipelined)	26	-	14.607ns
25-bits (pipelined)	65	77	10.143ns

5.2.2 MAC CHANNELS

Since the multiplication is being done with a constant coefficient, the number of inputs is small and each multiplier can be implemented as a single table look-up. Each MAC channel consists of a 5x5 LUT for a constant coefficient multiplication and a modulo m accumulator. The critical path consists of the modulo adder stage, since it cannot be pipelined due to its recursive feedback nature. Implementing it as a multi stage block is not beneficial because though it will improve the overall clock frequency, but will require the pipeline to be stalled for every 3 cycles resulting in the same effective input sample rate. Implementation results for a 5-bit modulo channel are given in table 7.

Table 7

MAC CHANNEL	SLICES USED	SLICEFLIPS FLOPS	DELAY
5-bit	14	5	15.286ns

5.2.3 FORWARD CONVERTER

Implementation of forward converter involves table look-ups followed by a modulo adder which adds the LUT outputs. The input binary word is broken up into groups of four and each group serves as the input to a separate LUT that stores a value modulo m. For example, in the case of an 8 bits input, there are two groups of four bits used for the table look-up in two separate tables and the output is added in the modulo m adder. The LUT is implemented in a manner similar to DALUT implementation described in section 3.2.2. The implementation results for a non-pipelined and a fully pipelined forward converter stage for 8-bits input are given in table 8.

Table 8

CONVERTER TYPE	SLICES USED	SLICEFLIPS FLOPS	DELAY
8-bits inputs Non-pipelined	55	-	12.919ns
8-bits inputs Pipelined	115	110	7.710ns/129.702MHz

5.2.4 REVERSE CONVERTER

Reverse converter also requires table look-ups in the first stage, but the LUTs are larger as they output a modulo M value. The LUT outputs are added in a modulo M adder tree. The critical path is dominated by the last stage delay of the modulo M adders as described before. The implementation results of a fully pipelined reverse converter for 25-bits output are given in table 9. It can be seen that the reverse converter using a direct implementation of Chinese Remainder theorem requires a large amount of resources. The critical path is due to large fan-out loading delay in the last stage of the modulo adder.

Table 9

REVERSE CONVERTOR	SLICES USED	SLICEFLIPS FLOPS	DELAY
24-bits output Fully pipelined	458	458	10.143ns/98.590MHz

5.3 COMPLETE ARCHITECTURE

A five modulo architecture using $m=32, 31, 29, 27,$ and 25 for a 24 bits dynamic range has been implemented. The critical path is the 5-bit modulo MAC channel. The results are given in table 10.

Table 10

RNS MAC	SLICES USED	SLICEFLIPS FLOPS	DELAY
24-bits dynamic range	669	661	17.636ns

It is obvious from the results that unless a multi-stage pipelined modulo adder is used, the architecture speed is severely limited by the modulo adder stage. The modulo accumulator is the performance bottleneck. Also, up to 70% of the device resources are being used to implement the reverse converter, while the actual computation stage takes less than 3% of the resources per channel. In case of the reverse converter, the resource requirement will increase drastically if even higher precision for inputs and coefficients is required, requiring the use of more of smaller moduli or switching to larger moduli. For this reason, the reverse converters are usually implemented in other forms [9].

6. DA-RNS ARCHITECTURE

6.1 OVERVIEW

Each of the independent MAC channels in a RNS based design can be replaced by a DA architecture. The benefits of RNS to DA design are two folds: smaller number of shifts required to process the operands and a faster architecture due to small word lengths. The scaling accumulator has to be modified however to account for modulo shift operation implementation. Since the rest of the architecture remains the same as a RNS, and DA architecture has been described above, only the implementation of a modulo scaling accumulator is described next.

6.2 MODULO SCALING ACCUMULATOR

The scaling modulo accumulator has been implemented as described in [5]. The implementation results for a 5-bit modulo scaling accumulator are given in table 11. The critical path delay is quite long even for the optimized implementation compared to the non-optimized implementation. The critical path consists of a carry save adder followed by a carry propagate adder stage, the outputs of which are multiplexed to select the proper output according to the correction factors.

Table 11

DA-RNS ACCUMULATOR	SLICES USED	SLICEFLIPS FLOPS	DELAY
5-bit modulo	31	5	17.278ns

6.3 COMPLETE ARCHITECTURE

The implementation results for a DA-RNS MAC channel for 5-bit moduli are given in table 12. The critical path is through the modulo scaling accumulator. In the complete architecture, the DA-RNS channel replaces the RNS MAC channel. The implementation results are not presented but it is obvious that due to the greater critical path delay in the scaling modulo accumulator, the DA-RNS MAC will be slower than the RNS MAC. Also the area required will be greater, as can be seen from the resource requirements for a single DA-RNS MAC channel as given in section 4.2.

Table 12

DA-RNS CHANNEL	SLICES USED	SLICEFLIPS FLOPS	DELAY
5-bit modulo	71	60	17.327ns

7. COMPARISON AMONG ARCHITECTURES

The implementation results show that the theoretical benefits of RNS and DA-RNS based architectures are impaired by the high resource requirements of the converters. Also, the delay of the modulo accumulator and modulo scaling accumulators are quite large compared to same word length 2's complement scaling accumulators, due to the multi stage recursive nature, which does not lend itself to pipelining. For this reason, even the small word length computations achieved of RNS does speed up the computations significantly compared to 2C DA implementations. It has also been observed that implementation of large adders in FPGAs with fast carry chains is quite fast and the adder delay scales up less than linearly with increasing word lengths.

The comparison presented in this report was done using the 4-LUTs only. This is one of the reason why RNS architectures, which benefit from larger and faster memory, do not show any gains over the 2C DA architecture. A number of architectures presented in the literature use specific optimizations to make RNS architecture look better than the 2C DA architecture. These include using larger memory structures available in many FPGAs and choosing specific moduli sets that ease the forward and reverse conversion process, besides allowing the use of special cases of modulo adders that are implemented using only 2^n binary adders.

It is also seen that the delay results of both DA and RNS architecture can be improved by solving the high fan-out loading problem. One possible solution to the high fan-out problem is the duplication of logic at the fan-out point. The amount of duplication has to be determined empirically. Another solution is to manually route the high fan-out connections using long lines, if the tool is not handling this well.

In light of the implementation results it is clear that DA based architectures have an area, speed and simplicity advantage over RNS based implementations. Also, it is observed that efficient implementation of RNS does require larger memory blocks than the basic 4-LUTs. But in context of the comparison basis used in this project, RNS implementations using only the basic 4-LUTs were compared with DA implementation for the sake of a fair comparison. It is in this context, we can say that DA implementations are superior over RNS implementations when targeting FPGAs.

8. CONCLUSION

In this report, the implementations results of a few MAC configurations using DA, DA-RNS and RNS structures on FPGA were presented and commented upon in order to find out whether the theoretical synergy between these formulations and the underlying FPGA architecture is realizable to what extent. It was observed that though this synergy does exist, the DA configuration has clear advantages over a RNS implementation when using only the basic 4-LUTs and 5-LUTs.

BIBLIOGRAPHY

1. Distributed Arithmetic FIR Filters version 8.0, Xilinx Logic Core Product Specification, Xilinx Inc. www.xilinx.com
2. Techniques for Implementing Multipliers in Stratix, Stratix GX & Cyclone Devices, Altera Application Note 306, Altera Corp. www.altera.com
3. Datasheets, www.xilinx.com
4. Datasheets, www.altera.com
5. COE-586 Term Report I
6. Radhika S. Grover, Weijia Shang and Qiang Li, "A Faster Distributed Arithmetic Architecture for FPGAs", FPGA February '02, pp. 31-39
7. Melanie Dugdale, "VLSI Implementation of Residues Adders Based on Binary Adders", IEEE Trans. on Circuit and Systems- II: Analog and Digital Signal Processing, vol. 39, no. 5, pp. 325-329, May 1992.
8. Magdy A. Bayoumi, G. A. Jullien, W. C. Miller, "A VLSI Implementation of Residue Adders", IEEE TRANSACTIONS ON CAS, VOL. CAS-34, NO. 3, MARCH 1987
9. Neil Burgess, "Scaled and Unscaled Residue to Binary Conversion Techniques Using the Core Function", IEEE, 1997, pp. 250-257
10. G. C. Cardarilli, M. Re, and R. Lojacono, "RNS-to-Binary Conversion for Efficient VLSI Implementation", IEEE TRANSACTIONS ON CAS—I: FUNDAMENTAL THEORY AND APPLICATIONS, VOL. 45, NO. 6, JUNE 1998, pp. 667-669