Achieving Performance Speed-up in FPGA Based Bit-Parallel Multipliers using Embedded Primitive and Macro support

B. Khurshid and R. N. Mir

***Abstract*— Modern Field Programmable Gate Arrays (FPGA) are fast moving into the consumer market and their domain has expanded from prototype designing to low and medium volume productions. FPGAs are proving to be an attractive replacement for Application Specific Integrated Circuits (ASIC) primarily because of the low Non-recurring Engineering (NRE) costs associated with FPGA platforms. This has prompted FPGA vendors to improve the capacity and flexibility of the underlying primitive fabric and include specialized macro support and intellectual property (IP) cores in their offerings. However, most of the work related to FPGA implementations does not take full advantage of these offerings. This is primarily because designers rely mainly on the technology-independent optimization to enhance the performance of the system and completely neglect the speed-up that is achievable using these embedded primitives and macro support. In this paper, we consider the technology-dependent optimization of fixed-point bit-parallel multipliers by carrying out their implementations using embedded primitives and macro support that are inherent in modern day FPGAs. Our implementation targets three different FPGA families viz. Spartan-6, Virtex-4 and Virtex-5. The implementation results indicate that a considerable speed up in performance is achievable using these embedded FPGA resources.**

***Keywords*— Fixed point arithmetic, FPGA primitives, VHDL, Instantiation based coding, Look-up table.**

I. Introduction

The multiplier circuit is one of the fundamental components used in digital signal processing (DSP) [1] [2] [3] [4]. The field of DSP has always been driven by the advancements in scaled very-large-scale-integration (VLSI) technologies. The goal of digital design is to maximize the performance while keeping the cost down [5]. In the context of general digital design, performance is measured in terms of the amount of hardware circuitry and resources required; the speed of execution (throughput and clock rate); and the amount of power dissipated. There is always an application-driven tradeoff between these parameters. It is, therefore, desirable to have an efficient realization of these circuits for use in different DSP systems [6] [7].

DSP algorithms have traditionally been implemented using general purpose processors or DSP processors. However, with current trend moving back towards hardware intensive processing it becomes important for the designers to give a serious thought to the underlying implementation platform [8]. Applications demanding an increased performance mainly use application integrated circuits (ASIC) or structural ASICs [2]. The main attraction with ASICs is that the architecture can be developed specifically to meet the performance requirement. However, the non-recurring engineering (NRE) costs associated with ASICs have cornered their use only to high-volume markets. Field programmable gate arrays (FPGAs) provide an alternate approach to ASICs. They avoid the high NRE costs by giving the user the flexibility to configure the device in field [4], [9]. Some other advantages include large-scale integration [4], [10], lower energy requirements [11], [12] availability of several on-board intellectual property (IP) cores [13] etc.

Design for FPGAs differs dramatically from general VLSI design [14]. The design process proceeds through phases like design entry, synthesis, translation, mapping and place & route (PAR). Design entry is the only manual phase in the entire design flow. Therefore, using FPGAs as an implementation platform requires programming of the desired functionality using some hardware descriptive language (HDL), as it is the most widely used design entry method [15]. The rest of the design process is automated and there is a strong computer aided design (CAD) support for synthesis and implementation. However, sophisticated CAD tools are often not good enough to meet some design constraint if an arbitrary coding style is used [16]. A popular guideline that has been followed for writing functional synthesizable HDL codes is the RTL guideline, where RTL stands for register transfer level, signifying that data transfer should occur through registers only. These guidelines adhere to synchronous design practices and signify the regulation of data flow, and how data is being processed [17] rather than what part of the FPGA fabric processes the data. In effect, such codes are purely inferential and strongly rely on the software environment that distributes the logic as per the design goal. Thus, in order to effectively use embedded primitive and macro resources the design entry needs to be modified.

There has been subsequent work regarding implementation of multipliers on FPGAs [17-33]. These mainly focus on modifying the multiplier architecture to achieve performance improvement. However, there has been very limited effort in improving the performance by using embedded FPGA resources [34-36]. In this paper we carry

Manuscript received March 31, 2015. Received in revised form May 5, 2015.

B. Khurshid is with the National Institute of Technology, Srinagar, J & K, India (e-mail: burhan_07phd12@nitsri.net).

R. N. Mir is with the National Institute of Technology, Srinagar, J & K, India (e-mail: naaz310@nitsri.net).

doi: 10.11601/ijates.v4i2.115

out technology dependent optimizations of fixed-point multipliers by modifying the coding strategy at the design entry phase. This is achieved by writing functional and synthesizable codes that involve direct primitive and macro instantiations. This requires detailed information about the FPGA target family that is being used and the primitives that are supported. In our study different primitives have been used and system functionality has been distributed in a way that utilizes these components with perfect mappings rather than writing a functional code and allowing the synthesizer to distribute the logic through inferences. The study focuses on Spartan-6, Virtex-4 and Virtex-5 families. Detailed analysis is carried out and it is concluded that by using primitive instantiations a subsequent improvement in performance can be achieved. This is achieved without having to alter the data-time relation of the algorithm under consideration. The only tradeoff is that the design entry gets complicated.

The rest of the paper is as follows. Section II briefly discusses the fixed-point bit-parallel multipliers that have been considered in this work. Section III lists the primitives that have been used in this work. A brief description about each primitive is provided. Section IV carries out the actual synthesis and implementation. Conclusions are drawn in section V and references are listed at last.

II. BIT-PARALLEL MULTIPLIERS

In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. Bit-parallel multipliers process one whole word of the input sample each clock cycle and are ideal for high-speed applications. The multiplication process is carried out as shown in figure 1. In this paper three different bit-parallel multipliers are considered viz. Parallel ripple-carry array multipliers; Parallel carry-save array multipliers and Baugh-Wooley multipliers. The details of these multipliers could be found in [5]. The operands in each case are assumed to be in fixed-point 2's complement representation. Such a representation ensures a correct final result even if there is an intermediate overflow [5].

III. FPGA PRIMITIVES

Primitives are the components that make an FPGA. The exact nature of a primitive may vary from family to family. In this section we briefly describe the primitives that are used in this work. These belong to the Spartan-6, Virtex-4 and Virtes-5 families.

A. *BUFG* [38]

This design element is a high-fan-out global clock buffer that connects signals to the global routing resources for low skew distribution of the signal. BUFGs are typically used on clock nets as well other high fan-out nets like sets/resets and clock enables. The primitive is supported by all the three families under consideration.

B. *FDSE* [38]

FDSE is a single D-type flip-flop with clock enable and synchronous set. The synchronous set input, when high, overrides the clock enable input and sets the output high during the low-to-high clock transition. The data is loaded into the flip-flop when set is low and clock enable is high during the low-to-high clock transition. The primitive is supported by all the three families under consideration.

C. *LUT4_L* [38]

This design element is a 4-bit look-up table (LUT) with a local output that is used to connect to another output within the same configurable logic block (CLB). The primitive is supported by all the three families under consideration.

D. *LUT6_2* [38]

This design element is a 6-input, 2-output LUT that can implement any two 5-input logic functions with shared inputs, or implement a 6-input logic function and a 5-input logic function with shared inputs and shared logic values. The primitive is not supported by the Virtex-4 logic family.

E. *CARRY4* [38]

This primitive represents the fast carry logic for a slice. The carry chain consists of a series of four multiplexers and four XOR gates that connect to the other LUTs in the slice via dedicated routes to form more complex functions. The fast carry logic is useful for building arithmetic functions like adders, counters, subtractors etc. The primitive is not supported by the Virtex-4 logic family.

F. *MULT_AND* [38]

MULT_AND is an AND component used exclusively for building fast and smaller multipliers. The primitive is only supported by the Virtex-4 logic family.

G. *MUXCY_L* [38]

This primitive is a 2-to-1 multiplexer for carry logic and is used to implement a 1-bit high-speed carry propagate function. The primitive is only supported by the Virtex-4 logic family.

H. *XORCY* [38]

XORCY is a special XOR element with general output that generates faster and smaller arithmetic functions. The primitive is only supported by the Virtex-4 logic family.

I. *DSP48* [38]

This design element is a versatile, scalable, hard IP block that allows for the creation of compact, high-speed, arithmetic-intensive operations, such as those seen for many DSP algorithms. Some of the functions capable within the block include multiplication, addition, subtraction, accumulation, shifting, logical operations, and pattern detection. The primitive is supported by all the three families under consideration.

							A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
							B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
							A_7B_0	A_6B_0	A_5B_0	A_4B_0	A_3B_0	A_2B_0	A_1B_0	A_0B_0	
						A_7B_1	A_6B_1	A_5B_1	A_4B_1	A_3B_1	A_2B_1	A_1B_1	A_0B_1		
					A_7B_2	A_6B_2	A_5B_2	A_4B_2	A_3B_2	A_2B_2	A_1B_2	A_0B_2			
				A_7B_3	A_6B_3	A_5B_3	A_4B_3	A_3B_3	A_2B_3	A_1B_3	A_0B_3				
			A_7B_4	A_6B_4	A_5B_4	A_4B_4	A_3B_4	A_2B_4	A_1B_4	A_0B_4					
		A_7B_5	A_6B_5	A_5B_5	A_4B_5	A_3B_5	A_2B_5	A_1B_5	A_0B_5						
	A_7B_6	A_6B_6	A_5B_6	A_4B_6	A_3B_6	A_2B_6	A_1B_6	A_0B_6							
A_7B_7	A_6B_7	A_5B_7	A_4B_7	A_3B_7	A_2B_7	A_1B_7	A_0B_7								
P_{15}	P_{14}	P_{13}	P_{12}	P_{11}	P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Figure 1 Tabular form for Parallel Array multiplication

IV. SYNTHESIS AND IMPLEMENTATION

A. Methodology

The implementation in this work targets three different FPGA families viz. Spartan-6, Virtex-4 and Virtex-5. Only LX series has been considered as it is apt for general logic applications. The implementation is carried out for an input operand length varying from 4 to 32 bits. The parameters considered are resource utilization, timing and dynamic power dissipation. Resource utilization is considered in terms of on chip FPGA components used. Timing refers to the clock speed of a design and is limited by the setup time of the input/output registers, propagation and routing delays associated with the critical path, clock to output time associated with the flip flops and the skew between the launch (input) register and the capture (output) register. Timing analysis is done to provide information about the speed/throughput of the system. Dynamic power dissipation is related to charging and discharging of node capacitances along the different switching elements. To ensure a fair comparison, similar test benches have been used for all the implemented designs i.e. the input statistics remain the same in each case. The initial design entry is done using VHDL. The coding strategy is based on instantiation of different primitives listed in section III. However, for comparison we have also followed the conventional inferential approach. The constraints relating to the period and offsets are duly provided and a complete timing closure is ensured. The design synthesis, mapping and translation are carried out in Xilinx ISE 12.1 and the simulator database is then analyzed for on-chip resources, throughput and timing metrics. Power metrics are obtained using Xpower analyzer.

B. Experimental results

As mentioned earlier, for each implementation a traditional inferential coding strategy is followed. Synthesis based on this coding strategy utilizes the FPGA resources as general logic elements. This will serve as a standard against which other implementations will be compared. Metrics associated with the instantiation of various primitives are named as per the primitives used. Tables 1, 2 and 3 give a comparison of the on chip resources utilized by different primitives for an input word-length of 16 bits. The architectures considered are the bit-parallel RCA, CSA and BW multipliers. The target device is XC6SLX16 from Spartan-6.

It is observed that by instantiating primitives and macro blocks there is a subsequent reduction in the on-chip resources being utilized by a particular structure. This is achieved without having to modify any architectural details. The most area efficient structure is obtained with LUT6_2 primitive because of its ability to implement both sum and carry in a single LUT. LUT4-L uses two different 4 input LUTs to implement the sum and the carry parts in each processing cell of the array. The CARRY4 and DSP48 primitives provide fast carry logic for each row. Their inclusion prominently will affect the timing properties of the structure. However, there is still some reduction in the slices being utilized when compared to the basic structure generated through inferential coding style. Further analysis is carried out for different multiplier structures for varying word-lengths and different target families. The metrics obtained from the synthesizer database are then plotted as a function of operand word-lengths and are presented in figures 2, 3 and 4. For simplicity we have considered only the occupied slices in each case. Virtex-4 family does not support the LUT6_2 primitive and hence does not appear in the plot. Further the fast carry logic in this family is implemented using a combination of MULT_AND and MUXCY_L primitives.

TABLE 1
RESOURCE UTILIZATION FOR RCA MULTIPLIER ON SPARTAN-6

On-chip resource	**Inferential coding style**	**LUT4_L**	**LUT6_2**	**CARRY4**	**DSP48***
Registers	32	32	32	32	32
LUTs	943	359	**286**	385	168
Slices	361	99	**82**	143	133

* DSP48 uses additional resources in the form of 16 DSP48A1 blocks

TABLE 2
RESOURCE UTILIZATION FOR CSA MULTIPLIER ON SPARTAN-6

On-chip resource	**Inferential coding style**	**LUT4_L**	**LUT6_2**	**CARRY4**	**DSP48***
Registers	32	32	32	32	32
LUTs	686	420	**343**	385	525
Slices	197	101	**92**	127	163

* DSP48 uses additional resources in the form of 16 DSP48A1 blocks

TABLE 3
RESOURCE UTILIZATION FOR BW MULTIPLIER ON SPARTAN-6

On-chip resource	**Inferential coding style**	**LUT4_L**	**LUT6_2**	**CARRY4**	**DSP48***
Registers	32	32	32	32	32
LUTs	583	492	**373**	407	500
Slices	206	128	**109**	134	174

* DSP48 uses additional resources in the form of 16 DSP48A1 blocks

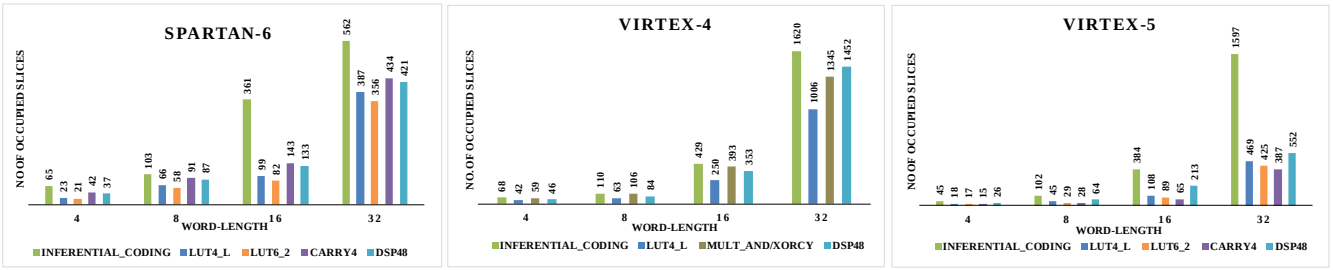

Figure 2 Resource utilization for RCA multiplier on different FPGA families

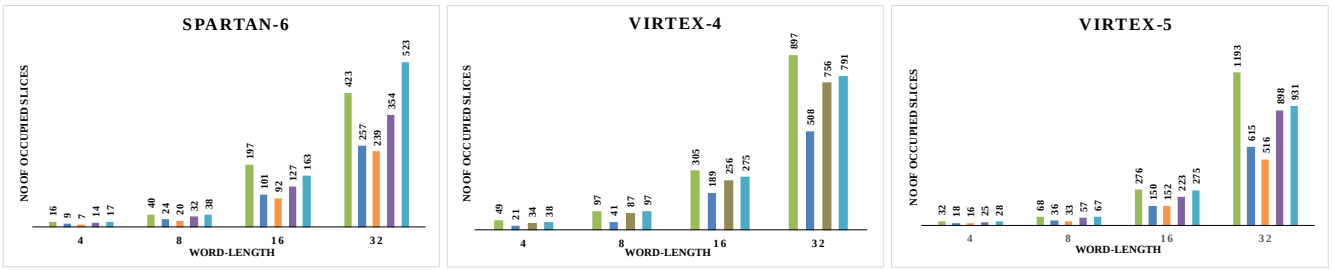

Figure 3 Resource utilization for CSA multiplier on different FPGA families

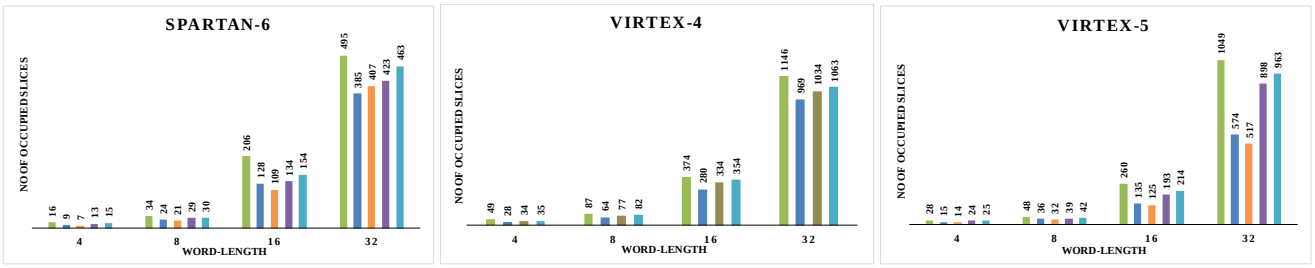

Figure 4 Resource utilization for BW multiplier on different FPGA families

It is observed that in each case there is a substantial reduction in the area when the structures are generated through instantiation of different primitive components. Also different primitives give different area performances depending upon the logic they implement. If area is the parameter of interest LUT6_2 gives the best performance.

The use of primitives LUT4_L and LUT6_2 although reduces the overall logic being used but the logic associated with the critical path of the structure is increased. This is indicated by the increase in the number of logic levels in the critical path. As a result the logic delay and the associated route delay increases. However, the fast carry logic associated with the CARRY4 primitive makes the addition process really fast resulting in reduced route delays. For Virtex-4 devices the fast carry logic is implemented using a combination of MULT_AND, MUXCY_L and XORCY primitives. The use of CARRY4 logic enhances the speed only in case of RCA multipliers as the critical path is limited by the rippling of the generated carry in each cell. However, with CSA and BW multipliers there is no rippling of the carry in the main structure. The only part of the multiplier that is enhanced using the fast carry logic is the vector merging adder (VMA). Tables 4, 5 and 6 provide a comparison of the maximum achievable clock rates post implementation for a word length of 16 bits. The target family is Spartan-6. The structures generated through instantiation of different primitives tend to have better timing closures in terms of the relationship between an external clock pad and its associated data-in or data-out pad. This is indicated by the offset-in and offset-out metrics from the timing database of the synthesizer. The values are included in the tables and are indicative of the fact that with primitive instantiations better timing behavior is achieved.

TABLE 4
TIMING ANALYSIS FOR RCA MULTIPLIER ON SPARTAN-6

Timing Parameter	**Inferential coding style**	**LUT4_L**	**LUT6_2**	**CARRY4**	**DSP48**
Maximum frequency (MHz)	30.67	23.186	23.89	38.7	**144.38**
Minimum available offset-in (ns)	5.112	5.018	2.568	**2.118**	2.543
Minimum available offset-out (ns)	11.727	8.488	10.047	6.782	**2.765**

TABLE 5
TIMING ANALYSIS FOR CSA MULTIPLIER ON SPARTAN-6

Timing Parameter	**Inferential coding style**	**LUT4_L**	**LUT6_2**	**CARRY4**	**DSP48**
Maximum frequency (MHz)	50.182	29.3	27.42	53.987	**167.87**
Minimum available offset-in (ns)	6.017	5.245	3.28	2.87	**2.521**
Minimum available offset-out (ns)	11.851	9.335	8.813	10.474	**4.78**

TABLE 6
TIMING ANALYSIS FOR BW MULTIPLIER ON SPARTAN-6

Timing Parameter	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48
Maximum frequency (MHz)	50.117	29.36	29.216	52.987	**165.43**
Minimum available offset-in (ns)	6.346	5.154	3.052	2.66	**2.543**
Minimum available offset-out (ns)	10.095	9.517	8.531	9.987	**4.87**

The results also indicate that CSA and BW multipliers have higher operating frequencies when compared to the RCA multiplier structures. Further analysis is carried out by plotting the maximum achievable speed against the operand word lengths for different structures and for different target families. The results are shown in figures 5, 6 and 7. Again for simplicity only the maximum achievable speeds have been considered.

It is observed from the plots that the use of fast carry logic results in faster execution and thus higher clock frequencies are achievable. The effect is more prominent in RCA multiplier as the carry rippling is completely eliminated.

Finally dynamic power dissipation for different structures is considered. Because an FPGA is programmable, it is only natural to look into minimizing the power dissipated. The dynamic power dissipation in a CMOS circuit is a function of the input voltage (V^2), the clock frequency (f_{clk}), the switching activity (α), the total capacitance seen by a particular node (C_L) and the number of elements used (σ). The analysis was done for a constant supply voltage and at maximum operating frequency for each structure. To ensure a reasonable comparison the test vectors provided during post route simulation were selected to represent the worst case scenario for data coming into the multiplier block. Same test bench was used for all the synthesized structures. The design node activity from the simulator database along with the power constraint file (PCF) was used for power analysis in the Xpower analyzer tool. Table 7 shows the power dissipated in various resources for RCA multiplier for operand length of 16 bits. The targeted device is Spartan-6. Tables 8 and 9 show the same metrics for CSA and BW structures.

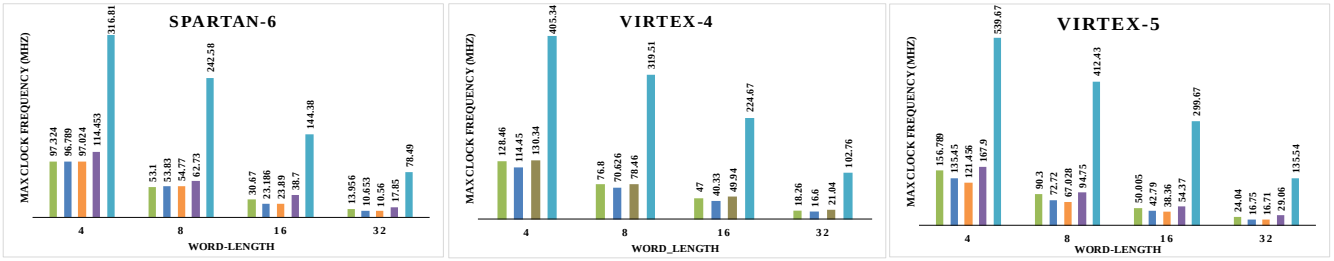

Figure 5 Maximum clock frequency comparisons for RCA multiplier on different FPGA families

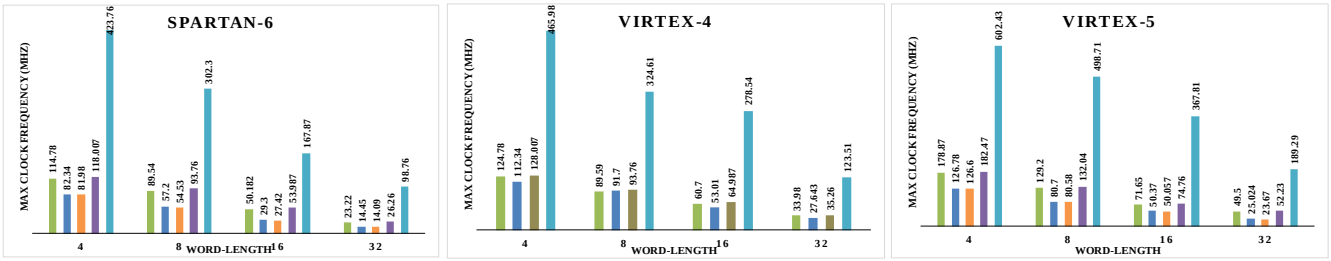

Figure 6 Maximum clock frequency comparisons for CSA multiplier on different FPGA families

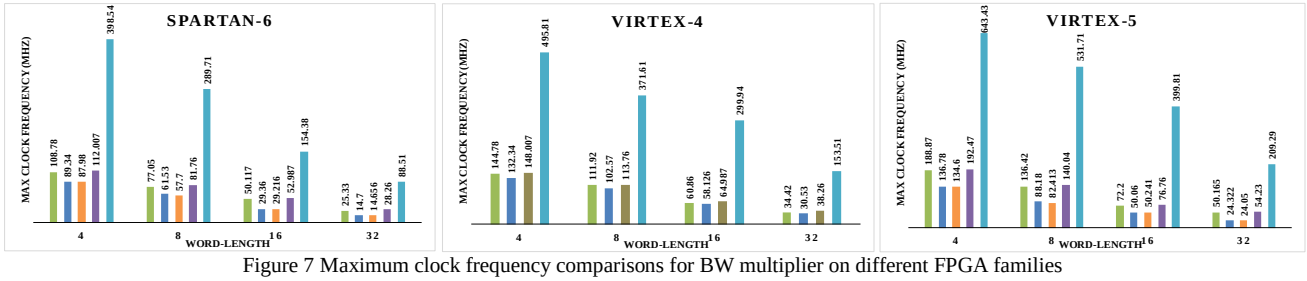

Figure 7 Maximum clock frequency comparisons for BW multiplier on different FPGA families

TABLE 7
POWER DISSIPATION FOR RCA MULTIPLIER ON SPARTAN-6

FPGA resource	Power dissipated (mW)				
	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48
Clock	0.54	**0.47**	**0.47**	0.64	2.8
Logic	2.34	1.64	1.78	1.4	**0.87**
Signals	4.41	1	**0.88**	1.16	1.23
I/Os	6.01	5.16	6.39	4.63	**2.54**
Total	13.3	8.27	9.52	7.83	**7.44**

TABLE 8
POWER DISSIPATION FOR CSA MULTIPLIER ON SPARTAN-6

FPGA resource	Power dissipated (mW)				
	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48
Clock	1.24	**0.7**	0.81	1.24	3.25
Logic	3.14	2.18	2.27	1.14	**0.69**
Signals	2.66	1.41	**1.2**	1.66	1.63
I/Os	5.44	5.35	5.12	4.44	**2.31**
Total	12.48	9.64	9.4	8.48	**7.88**

TABLE 9
POWER DISSIPATION FOR BW MULTIPLIER ON SPARTAN-6

FPGA resource	Power dissipated (mW)				
	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48
Clock	1.22	**0.63**	0.88	1.24	3.43
Logic	2.71	2.25	2.02	1.14	**0.67**
Signals	3.29	1.39	**1.11**	1.96	1.97
I/Os	4.99	6	5.75	4.42	**2.36**
Total	12.21	10.27	9.76	8.76	**8.43**

The power dissipated in the clocking resources varies with the clock activity (clock frequency) as provided in the PCF. Since each structure is operated at its maximum operating frequency, the power dissipated by the clock varies accordingly and has a maximum value for the multiplier based on CARRY4 and DSP48 primitives. However, the capacitance $C_{L,}$ which needs to be driven at each toggling node, varies with the type, fan-out, and capacitance of the logic and routing resources used in the design. The use of primitives through instantiations has a soothing effect on the fan-out of the non-clocking nets. This is indicated in table 10 where the average fan-out of non-clocking nets for different multipliers using different primitives has been enlisted for a 16-bit operand word-length. In addition, there is a reduction in the number of elements (σ) being utilized by different multiplier structures when designed using direct instantiation of primitives. Thus, the power dissipated in the logic is reduced and has a minimum value for CARRY4 and DSP48 primitives. The reduction in the power dissipation in the signals and I/Os is indicative of the fact that primitive instantiation also tends to relax the signal transition rates for the duration of operation.

Further analysis is carried out by plotting the total dynamic power dissipation as a function of input word-length for different multiplier structures and for different FPGA families. The results are shown in figures 8, 9 and 10.

TABLE 10
AVERAGE FAN-OUT OF NON-CLOCKING NETS FOR DIFFERENT MULTIPLIERS ON SPARTAN-6

Multiplier design	Average fan-out of non-clock nets				
	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48
RCA	5.52	2.94	2.32	**1.98**	--
CSA	4.75	2.71	2.22	**1.78**	--
BW	4.78	2.65	2.12	**1.66**	--

For DSP systems it is more appropriate to quantify the power efficiency through energy analysis [39]. This gives idea about the power requirements of a design at a lower level. Three energy related parameters are defined for different multiplier designs. These include Energy per operation (EOP), which is the average amount of energy required to compute one operation; Energy throughput (ET) which is the energy dissipated for every output bit processed and Energy density (ED) which is the energy dissipated per FPGA slice. Tables 11, 12 and 13 provide these metrics for different designs. The input operand length in 16 bits and the target device is from Spartan-6. In each case the critical path delay is taken as the approximate time to complete one operation. Further analysis is carried out by plotting the energy metrics as a function of operand word length for different multipliers. The plots appear in figures 11, 12 and 13. The target device in each case is XC6SLX16 from Spartan-6.

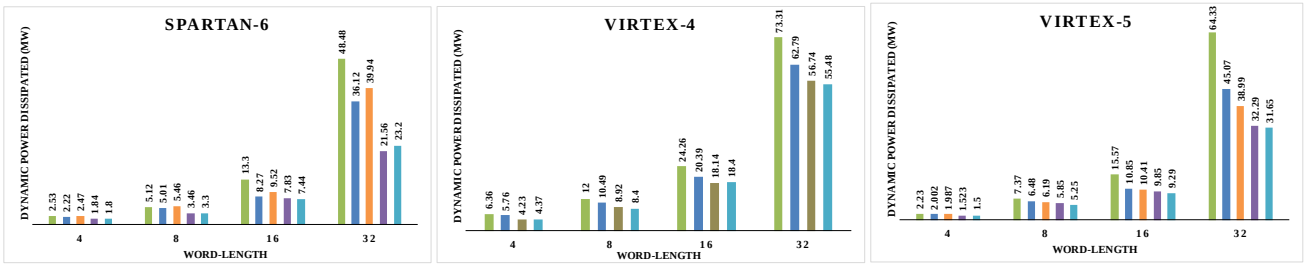

Figure 8 Dynamic Power dissipation comparisons for RCA multiplier on different FPGA families

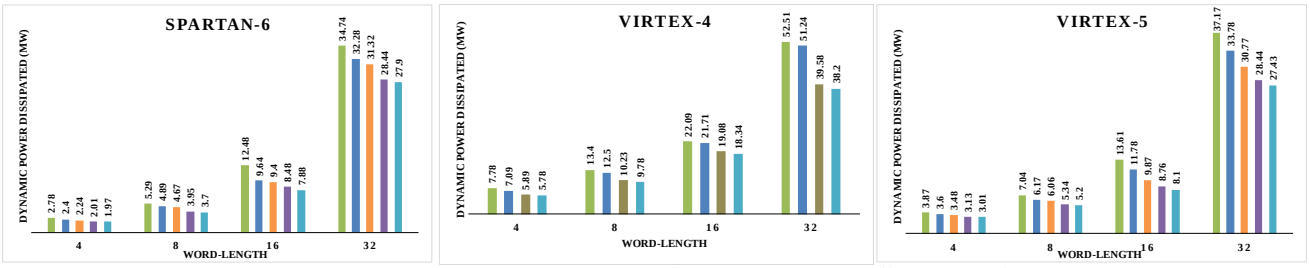

Figure 9 Dynamic Power dissipation comparisons for CSA multiplier on different FPGA families

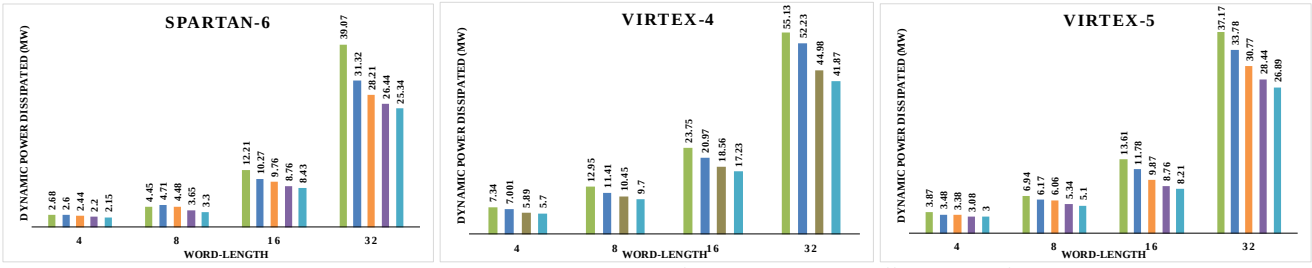

Figure 10 Dynamic Power dissipation comparisons for BW multiplier on different FPGA families

The plots clearly reveal that the structures based on primitive instantiations have high power efficiency. The energy requirement is minimum for the structures based on CARRY4 and DSP48 primitives. Also, note that the effect is more prominent for RCA multipliers as the entire structure is synthesized using the CARRY4 primitive, where as in the CSA and BW multipliers only the VMA part is based on the fast carry logic.

TABLE 12
ENERGY ANALYSIS FOR RCA MULTIPLIER ON SPARTAN-6

Energy parameter	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48*
EOP (pJ)	433.64	356.68	398.49	202.32	**51.530**
ET (pJ/bit)	27.103	22.29	24.90	12.645	**3.220**
ED (pJ/slice)	1.2012	3.602	4.859	1.4148	**0.387**

TABLE 12
ENERGY ANALYSIS FOR CSA MULTIPLIER ON SPARTAN-6

Energy parameter	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48*
EOP (pJ)	248.694	329.01	342.8	157.07	**46.94**
ET (pJ/bit)	15.543	20.56	21.42	9.81	**2.93**
ED (pJ/slice)	1.2624	3.25	3.72	1.23	**0.28**

TABLE 13
ENERGY ANALYSIS FOR BW MULTIPLIER ON SPARTAN-6

Energy parameter	Inferential coding style	LUT4_L	LUT6_2	CARRY4	DSP48*
EOP (pJ)	243.62	349.79	334.06	165.33	**54.60**
ET (pJ/bit)	15.22	21.86	20.87	10.33	**3.41**
ED (pJ/slice)	1.18	2.73	3.06	1.23	**0.35**

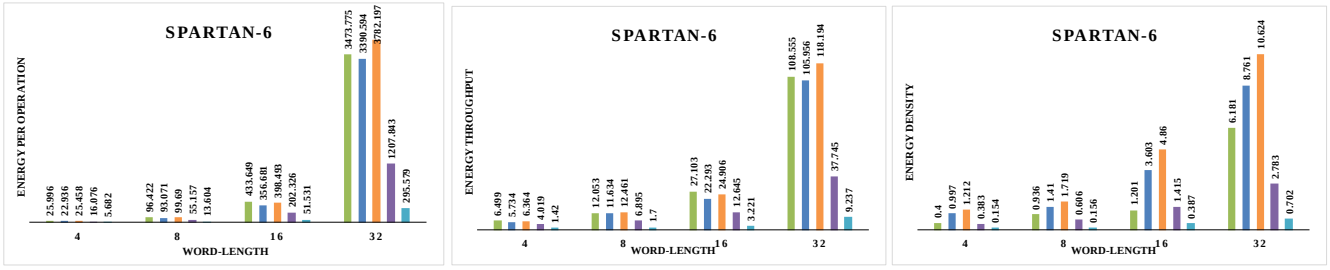

Figure 11 Energy analyses for RCA multiplier on Spartan-6 FPGA family

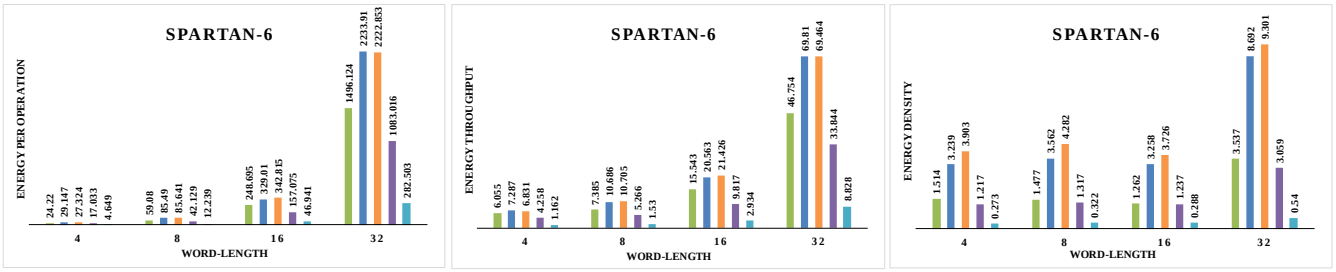

Figure 12 Energy analyses for CSA multiplier on Spartan-6 FPGA family

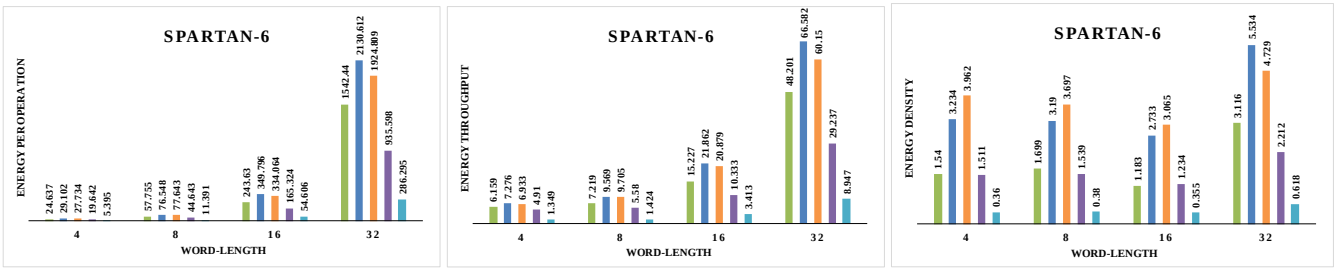

Figure 13 Energy analyses for BW multiplier on Spartan-6 FPGA family

V. CONCLUSIONS AND FUTURE SCOPE

This paper implemented the bit-parallel fixed-point multipliers in three different structures. The hardware implementations presented in this paper were based on the use of various in built primitives and macro blocks inherent to modern FPGAs. The analysis and the experimental results carried out in this paper clearly indicate that a considerable improvement in performance is indeed achievable by using these primitives. Further the design entry used in this paper was based on instantiations rather than inferences. By using a coding strategy based on instantiations the on-chip FPGA components can be used in a manner that fully utilizes their potential. Also a judicious choice of primitives will ensure that a particular performance parameter is enhanced as may be required by any particular application. This paper deliberately ruled out any architectural modification that may be carried out at the top level of the design. The idea was to present a clear cut analysis that will provide an insight about the performance speed-up that may be achieved by utilizing the huge primitive support provided by FPGA families. Currently the authors are working on achieving a performance speed-up by using a combination of architectural modifications and embedded primitives in FPGAs.

REFERENCES

[1]. G. L. Narayan and B. Venkataramani, " Optimization Techniques for FPGA based Wave Pipelined DSP Blocks," IEEE Transc.Very Large Scale Integr. (VLSI) syst., vol. 13, No. 7, pp. 783-792, July 2005.

[2]. M. A. Ashour and H. I. Saleh, "An FPGA Implementation guide for some different types of Serial-Parallel Multiplier Structures," Microelectronics Journal, vol. 31, pp. 161-168, 2000.
[3]. K. Compton, S. Hauck, "Reconfigurable Computing: A survey of Systems and Software," ACM Computing Surveys, vol. 34, No. 2, pp. 171-210, June 2002.
[4]. R. Tessier, W. Burleson, "Reconfigurable Computing and Digital Signal Processing: Past, Present and Future," Programmable Digital Signal Processors, Yu Wen Hue d, Marcel Dekker, pp. 147-186, 2002.
[5]. Keshab K. Parhi, "VLSI Digital Signal Processing Systems Design and Implementation," Wiley, 1999.
[6]. S. Shanthala and S. Y. Kulkarni, "VLSI Design and Implementation of Low Power MAC Unit with Block Enabling Technique," European Journal of Scientific Research, ISSN 1450-216X, vol. 30, No. 4, pp. 620-630, 2009.
[7]. K. H. Chen, Y. H. Chen and Y. S. Chu, "A Versatile Multimedia Functional Unit Design using the Spurious Power Suppression Technique," in Proc. IEEE Asian Solid-State Circuits conf., 2006, pp. 111-114.
[8]. Roger Woods, John McAllister, Gaye Lightbody and Ying Yi, "FPGA-based Implementation of Signal Processing Systems," Wiley, 2008.
[9]. Z. Guo, W. Najjar, F. Vahid and K. Vissers, "A Quantitative Analysis of the Speed up Factors of FPGAs over Processors," in Proc. Int. Symp. on FPGAs, ACM Press, 2004.
[10]. K. Underwood "FPGAs vs. CPUs: Trends in Peak Floating-Point Performance," in Proc. Int. Symp. on FPGAs, ACM Press, 2001.
[11]. G. Stitt, F. Vahid and S. Nematbakhsh, "Energy Savings and Speed ups from Partitioning Critical Software Loops to Hardware in Embedded systems," ACM Transc. Embedded Comput. Systems, vol. 3, pp. 218-232, 2004.
[12]. R. Tessier and W. Burleson, "Reconfigurable Computing for DSP: A Survey," Journal of VLSI Signal Processing, vol. 28, pp. 7-27, 2001, Kluwer Academic Publisher.
[13]. T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk and P. Y. K. Cheung, "Reconfigurable Computing: Architecture and Design Methods," in IEEE Proc. Comput. Digit. Tech., vol. 152, No. 2, March 2005.
[14]. K. S. Hemmert and K. D. Underwood, "Fast, Efficient Floating-Point Adders and Multipliers for FPGAs," ACM Transactions on Reconfigurable Technology and Systems, vol. 3, No. 3, Article 11, September 2010.
[15]. G. Quan, J. P. Davis, S. Devarkal and D. A. Buell, "High-Level Synthesis for Large Bit-Width Multipliers on FPGAs: A Case Study," ACM 2005.
[16]. Steve Kilts, "Advanced FPGA Design Architecture, Implementation, and Optimization," Wiley 2007.
[17]. Seetharaman Ramachandran "Digital VLSI Systems Design: A Design Manual for Implementation of Projects on FPGAs and ASICs using Verilog," Springer, 2011.
[18]. M. Shand, P. Bertin, and J. Vuillemin, "Hardware Speedups in Long Integer Multiplication," Computer Architecture News, vol. 19, No. 1, pp. 106–114, 1991.
[19]. L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," in ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, pp. 107–116, Feb. 1996.
[20]. F. de Dinchin and V. Lef`evre, "Constant Multipliers for FPGAs," in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, H.R. Arabnia (Ed.), CSREA Press, vol. I, pp. 167–173, June 2000.
[21]. T. Courtney, R. Turner, and R. Woods, "Multiplexer Based Reconfiguration for Virtex Multipliers," in Field-Programmable Logic and Applications. Proceedings of the 9th International Workshop, FPL 2000, pp. 749–758, 2000.
[22]. T. Courtney, R. Turner, and R. Woods, "An Investigation of Reconfigurable Multipliers for use in adaptive Signal Processing," in Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '00), IEEE Computer Society Press, pp. 341–343, April 2000.
[23]. A. F. Tenca, M. D. Ercegovac, and M. E. Louie, "Fast On-Line Multiplication Units Using LSA Organization," in Proceedings of the International Society of Optical Engineering (SPIE). Visual Communications and Image Processing. Real-Time Signal Processing, vol. 3807, pp. 74–83, 1999.
[24]. C. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, 13:14–17, 1964.
[25]. Z. Wang and W. C. Miller, "A new Design Technique for Column Compression Multipliers," IEEE Transactions on Computers, vol. 44:962–970, 2005.
[26]. F. Cheng and M. Theobald, "Design of Synchronous Variable Latency Pipelined Multipliers.," IEEE Transaction on Computers, vol. 49: 659-672,2005.
[27]. Z. Huang, "High Level Optimization Techniques for Low Power Multiplier Design" Ph.D. Thesis, University of California, los angels, 2003.
[28]. C. H. Chang and R. K. Satzoda, "A Low Error and High Performance Multiplexer-Based Truncated Multiplier," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 18, No. 12, December 2010.
[29]. S. S. Kidambi, F. E. Guibaly and A. Antoniou, "Area-Efficient multipliers for Digital Signal Processing Applications," IEEE Transactions on Circuits and Systems –II: Analog & Digital Signal Processing, Vol. 43, No. 2, February 1996.
[30]. J. E. Stine and O. M. Duverne, "Variations on Truncated Multiplication," Proceedings of the Euromicro Symposium on Digital System Design, 2003.
[31]. Y. M. Motey and T. G. Panse, "Hardware Implementation of Truncated Multiplier Based on Multiplexer Using FPGA," International conference on Communication and Signal Processing, April 3-5, 2013.
[32]. H. Park and E. E. Swartzlander, "Truncated Multiplications for the Negative Two's Complement Number System," 49th IEEE International Midwest Symposium on Circuits and Systems, San Juan, August 6-9, 2006.
[33]. J. Valls and E. Boemo, "Efficient FPGA Implementation of Two's Complement Digit-Serial/Parallel Multipliers," IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 50, No. 6, June 2003.
[34]. G. Zhou, L. Li and H. Michalik, "Area optimization of bit parallel finite field multipliers with fast carry logic on FPGAS," International Conference on Field Programmable Logic and Applications, 2008.
[35]. S. Gao, D. A. Khalili and N. Chabini, "Efficient Scheme for Implementing Large Size Signed Multipliers UsingMultigranular Embedded DSP Blocks in FPGAs," International Journal of Reconfigurable Computing Vol. 2009, Article ID 145130, Hindawi Publishing Corporation.
[36]. C. Ingemarsson, P. Kallstrom and O. Gustafsson, "Using DSP block pre-adders in pipeline SDF FFT implementations in contemporary FPGAs," 22nd International Conference on Field Programmable Logic and Applications, August 2012.
[37]. C. R. Baugh and B. Wooley, "A two's Complement Parallel Array Multiplication Algorithm," IEEE Trans. On Computers, vol. C-22, No. 12. Pp. 1045-1047, Dec. 1973.
[38]. http://www.xilinx.com
[39]. P. K. Meher, S. Chanderasekaran and A. Amira, "FPGA Realization of FIR Filters by Efficient and Flexible Systolization using Distributed Arithmetic," IEEE Transactions on Signal Processing, vol. 56, No. 7, July 2008.

AUTHOR PROFILES

B. Khurshid received the B.E. degree in Electronics and Communications Engineering from the Kashmir University, India, in 2008, the M.Tech degree in Communications and IT from National Institute of Technology, Srinagar, India in 2011. Currently he is pursuing his PhD in System design in the department of Computer Science and Engineering, NIT, Srinagar. His research interests include Reconfigurable architectures, Platform oriented solutions for arithmetic and DSP algorithms, Architectural and technology dependent optimizations targeted for FPGA platforms, etc. He has many publications in the related field and is a student member of IEEE. He is also a lifetime member of IETE.

R. N. Mir received B.E. (Hons) in Electrical Engineering from University of Kashmir (India) in 1985, M.E. in Computer Science & Engineering from IISc Bangalore (India) in 1990 and Ph D from University of Kashmir, (India) in 2005. She is currently a Professor in the department of Computer Science & Engineering at NIT Srinagar, India. She is the co-author of many scientific publications in international journals and conferences. Her current research interests include reconfigurable computing, security and routing in wireless ad-hoc networks and sensor networks