

A FLEXIBLE MULTIPLICATION UNIT FOR AN FPGA LOGIC BLOCK

Kamal Rajagopalan, Peter Sutton

School of Computer Science and Electrical Engineering
The University of Queensland
Brisbane Queensland Australia

ABSTRACT

FPGAs are increasingly being applied to DSP applications but are often inefficient in space and time compared with dedicated DSP chips, particularly for multiplication-based operations. To improve FPGA arithmetic performance, a flexible multiplication unit and configurable carry logic circuitry suitable for incorporation into a FPGA logic block are proposed. The multiplier unit is based on a modified carry-save adder and along with the carry logic circuitry efficiently supports multiplication, addition and multiply-accumulate operations in serial or parallel form. Preliminary results indicate logic utilization for a multiplier implementation in such an FPGA is approximately a third that of the XC4000 architecture and half that of the Virtex architecture. Propagation delays are also reduced due to the use of dedicated inter-block interconnect for all sum and carry signals and flexible routing multiplexers.

1 INTRODUCTION

As FPGA device densities increase, field programmable gate arrays (FPGAs) are increasingly being applied to DSP applications. This is due to the enormous speed up possible over conventional digital signal processors (for some applications) usually due to parallel computation on multiple data streams. For some applications, however, FPGAs suffer in performance and logic utilization relative to digital signal processors due to the use of general purpose logic for computation as opposed to hardwired computation units. This is particularly the case for multiplication and multiply-accumulate operations.

Digital multipliers are needed in many system applications, including digital filters, correlators and other DSP applications. Multipliers often make ineffective use of a programmable part by consuming significant logic and routing resources. One solution is to use dedicated multiplier devices connected to FPGAs, however, this often results in performance degradation due to inter-chip communication delays.

Improvements can be made to FPGA performance by modifying device architectures to embed multiplier units (e.g [3]) or to add small amounts of supporting logic as in Virtex [8] and ORCA [6]. These approaches provide increased performance but inefficiencies still arise, particularly for multiplications, due to the use of relatively slow general purpose resources (e.g. lookup-tables (LUTs) and general purpose routing) for some time-critical operations or signal propagations. In this paper, a new FPGA logic block architecture is proposed which embeds a flexible multiplication unit in each logic block to provide support for multiplication, multiplication with addition or pure addition. A configurable carry unit allows the generation of a carry to support either multiplication or addition. Dedicated interconnect between

adjacent blocks allows for greatly reduced delay times. Floating point operations are also supported through the incorporation of programmable multiplexers able to efficiently implement barrel shifting operations.

The remainder of this paper is organised as follows. Section 2 discusses multiplication in FPGAs. Section 3 describes the new multiplication unit and logic block architecture. A brief evaluation of the architecture compared with the Xilinx XC4000 and Virtex architectures is presented in section 4 and some conclusions are drawn in section 5.

2 MULTIPLICATION IN FPGAS

2.1 Existing Dedicated Multiplier Architectures

Lee and Flynn [5] have constructed a carry built-in architecture to implement arithmetic operations through multi-ported look up tables. The carry architecture is built-in with a multi-port LUT with additional support for carry logic by the use of a bypass multiplexer. They have achieved 5 times greater throughput density for particular applications such as variable multiplier, FIR filter, Viterbi decoder and Jacobi Iteration method. Due to the additional logic necessary they have achieved this result at the cost of a 10% area penalty, however the use of LUTs instead of dedicated arithmetic logic imposes a performance penalty.

Haynes and Cheung [3] described a technique to implement a multiplier with the aid of new flexible array block (4×4 multiplier). An array of these blocks is capable of being configured to perform any $4m$ bits \times $4n$ bits signed/unsigned binary multiplication. The blocks are designed to be embedded within a conventional FPGA structure to increase the functionality of the device by freeing valuable general reconfigurable resources, particularly when used in the area of image processing. In general, the lack of a regular CLB structure reduces the flexibility of the design. When these blocks are used within an FPGA structure then a special routing architecture is required in order to overcome this problem.

2.2 Commercial FPGA Architectures

Typical multipliers widely implemented in FPGAs are ripple carry array multipliers (RCM) and carry-save multipliers (CSM) since tree multipliers such as Wallace [7] and Dadda [2] multipliers result in irregular structures. Wallace tree multipliers are considered to be faster than other multipliers, but due to poor way the irregular structure maps to available FPGA routing resources, this theoretical performance advantage can not be realized in FPGAs. Ripple carry array multipliers are regular in structure. This repeatability of basic cells in the array and local interconnections make ripple carry multipliers suitable for realization in most

FPGAs. Due to the long carry propagation paths, ripple carry multipliers have the drawback of long delay times. The carry-save technique can be applied to reduce propagation delays to a limited extent. In general then, carry-save multipliers are used in FPGAs due to their regular structure and faster computation. Some specific comments about implementing addition and multiplication in various commercial architectures are made below.

2.2.1 Altera Flex

In Altera Flex 8k, Flex 10k and Apex [1] the LUT and carry logic can act in parallel to perform 2-bit addition operations. The Altera carry chain structure provides a fast carry forward between logic elements (LEs). The carry-in signal from a lower-order bit moves forward into the higher order bit via the carry chain, and feeds into both the LUT and the next portion of the carry chain. This feature allows the Altera devices to implement fast counters, adders and comparators. Lookup tables are used when implementing multipliers which does impact performance when compared with dedicated multiplication logic.

2.2.2 Xilinx XC4000

In the Xilinx XC4000 [8], LUT and carry circuitry acts in parallel to perform the addition of two bits. In the case of multiplication, the LUT and carry circuitry are used to implement $m \times n$ multipliers. Since there is no dedicated support for multipliers, XC4000 requires additional LUTs to implement multipliers. For example: XC4000 requires on average $1.14n^2$ CLBs to implement an $n \times n$ multiplier [4] (73 CLBs [8] are required to implement an 8×8 multiplier). The architecture functional block proposed in this paper resembles the XC4000 CLB structure, however, the architecture is more efficient for multiplication operations than the

XC4000 due to additional logic and dedicated interconnects at the cost of a small area penalty.

2.2.3 Xilinx Virtex

In Xilinx Virtex FPGAs [8], LUT and carry logic act in serial to perform the addition of two bits. LUTs are used to generate a carry selection signal. Each CLB consists of two slices, where each slice consists of 2 LUTs, carry logic and a flipflop. Each slice contains two MULT_AND gates to generate partial products so as to implement the multipliers efficiently. MULT_AND is an AND component used exclusively for building fast and smaller multipliers instead of “wasting” a LUT. More CLBs are required for the implementation of larger multipliers however. The flexible architecture proposed in this paper allows the implementation of both small and large multipliers with fewer functional blocks.

2.2.4 Lucent ORCA

In ORCA [6] FPGAs, multipliers are implemented based on an add/pass-shift form of the parallel multiplier. The add/pass-shift works by adding the partial product and multiplicand when the multiplier bit is 1 otherwise the partial product is passed on without addition. They can be inefficient in implementing smaller multipliers due to large blocks and the complex routing structure.

3 ARCHITECTURE DESCRIPTION

The proposed architecture is an island style FPGA made up of logic blocks, which we have named *functional blocks* (FB). Each functional block consists of three 4-LUTs, two configurable flip-flop/latches (F/L), Flexible Multiplier Unit (FMU), Programmable Multiplexers (PM), Special Programmable Multiplexer (SPM), Configurable carry circuitry (CF) and carry circuitry as shown in

Figure 1. Each FB has its own dedicated inputs and outputs for fast execution of arithmetic operations. Carry and sum signals for the FMU are controlled by a programmable multiplexer (PM). A special programmable multiplexer (SPM) is provided to support efficient implementation of barrel shifters (described further in section 3.4). Two configurable flip-flops/latches also make up the functional block.

3.1 Flexible Multiplier Unit

The Flexible Multiplier Unit (FMU) is built with a 1x1 Multiplier Cell (MC), full adder, Internal Multiplexer (IM) and Configuration Controlled Multiplexers (CM) as shown in Figure 2. The MC unit is a modified version of carry save adder for generating partial products, sum signal and the carry signal for implementing smaller multipliers efficiently. The MC is used as a basic cell for implementing large multipliers. The IM Multiplexer supports the MC for carry generation and carry propagation depending upon the previous sum signal. The CM multiplexers are controlled by configuration bits in order to select appropriate sum and carry output signals.

Programmable Multiplexers (PM) are used to select appropriate input carry signals and sum signals from adjacent functional blocks as shown in Figure 1. Dedicated interconnect lines (i.e. without switches) are

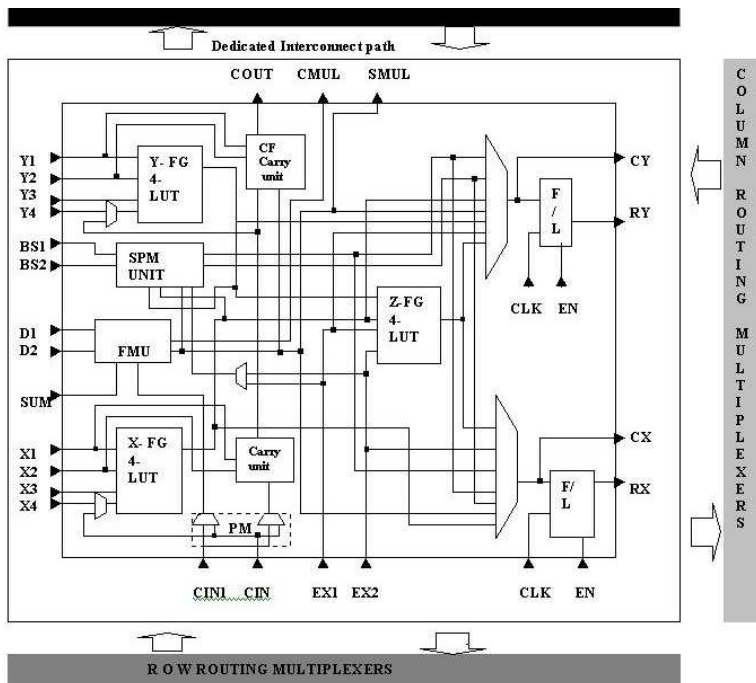


Figure 1. Functional Block of the proposed architecture

used to route carry and sum signals between adjacent blocks. Dedicated interconnect allows for faster signal propagation and therefore faster computations. This is particularly important for larger bit-width computations.

The FMU is described as *flexible* due to the ability to implement either parallel or serial multipliers efficiently through the use of the PMs and the dedicated interconnect.

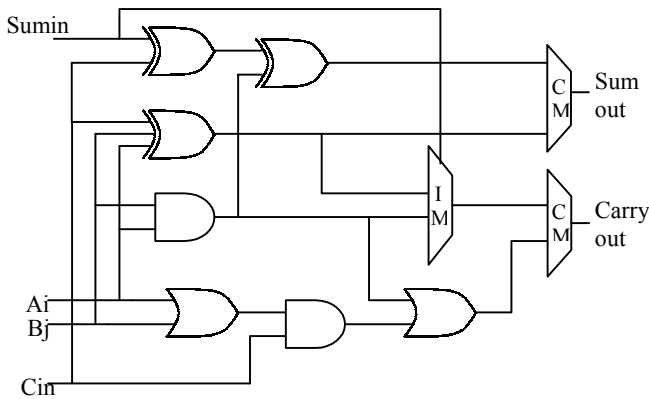


Figure 2. Configurable FMU.

3.2 Configurable Carry Circuit

The carry circuit is also designed in a manner to support the FMU to generate or propagate the carry according to whether the functional block is configured as an adder or multiply and add or multiplier circuit. The carry circuit (shown in Figure 3) consists of two components – the choice of which is determined by the configuration data for the output multiplexer. Multiplier Fast Carry (MFC) logic is used to support multiplier designs. The multiplier carry circuitry is similar to the FMU carry circuitry. Adder Fast Carry (AFC) logic is the traditional carry logic used in a full adder. This flexibility improves the performance of the architecture for arithmetic based designs.

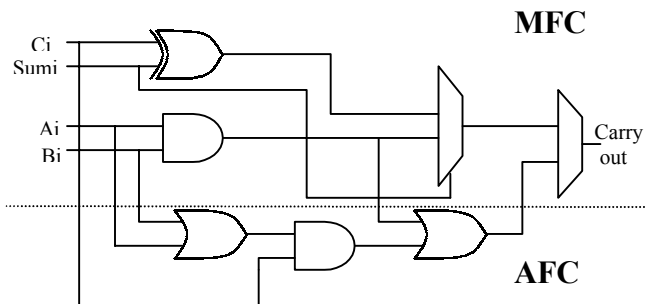


Figure 3. Configurable Carry Circuit

3.3 Dedicated Interconnect

With today's shrinking process dimensions and increasing logic performance, routing delays now have a major impact on overall FPGA performance. Conventional FPGA architectures suffer in potential performance when implementing complex arithmetic operations due to the general purpose routing resources and

switches that must be used to route signals from one logic block to another. This is one reason why architectures such as XC4000 (some versions) and Virtex have dedicated carry propagation circuitry.

The architecture proposed here extends this dedicated carry propagation circuitry to better support multiplication as well as addition (and similar operations). The dedicated interconnects, which are between adjacent FBs, are mostly free of RC delay from programmable switches. This also makes timing performance estimation easier since it is simpler to characterise dedicated interconnect as opposed to estimating the performance of routing wire segments and switches.

When implementing multipliers, adders and some other arithmetic operations, *only* the dedicated interconnect lines need to be used, thus avoiding general purpose routing switch delays.

3.4 Floating Point Arithmetic Support

Floating point multiplication is more difficult to implement in hardware due to the need to shift the mantissa by an appropriate number of bits and adjust the exponent. In hardware, a floating point multiplier generally consists of a fixed-point multiplier and other extra circuitry for sign checking, pre and post shifting of the exponents. Shifting can be performed with barrel or logarithmic shifters. In general, barrel shifters are built with a linear chain of multiplexers with input bits and control signals.

When implemented in conventional lookup-table (LUT) based FPGAs, these multiplexers are implemented using LUTs resulting in poor utilization of the device and slow performance. To avoid this problem, an additional programmable multiplexer is provided between the multiplier unit and carry circuit for efficient implementation of barrel shifters. This enhances the area-efficiency and performance of the proposed architecture for floating-point arithmetic operations. Further investigations are planned into architectural enhancements to better support floating point arithmetic in FPGAs.

4 ARCHITECTURAL EVALUATION

An accurate measure of the performance (both speed and area) provided by the new architecture requires CAD tools for performing technology mapping, placement and routing of designs onto the new architecture. These tools are not available, so we provide an approximation of the expected gain of the new architecture compared to Xilinx's XC4000 and Virtex architectures.

4.1 Area Estimation

In order to compare the area of the proposed functional block (FB) with Xilinx XC4000 and Virtex CLBs, VHDL descriptions of the proposed FB and an XC4000 CLB/Virtex CLB (both with and without RAM) were created¹ simulated and synthesized to determine gate counts (which are assumed to be proportional to the block area). Table 1 shows the relative sizes and clocking performance of the new architecture's functional blocks and Xilinx

¹ The design created for the XC4000 CLB and Virtex CLB of course does not necessarily correspond to the actual implementation of the CLB. The design is based on the information available in Xilinx's databook [8].

XC4000 and Virtex CLBs. (Simulation has been performed with ModelSim. Synthesis has been performed with Leonardo Spectrum targeting the TSMC 0.25 micron technology.)

Table 1: Synthesis Report for XC4000 and Virtex CLBs and proposed FB in TSMC 0.25 technology (without configuration and RAM logic)

Parameters	XC4000 (CLB)	Virtex (2 slice CLB)	New Architecture (FB)
No of Gates*	2828	9850	3663
Clock Speed (MHz)	202	224	272.9
Critical Path delay (ns)	4.94	4.47	3.88

*: As reported by synthesis tool. (e.g. one NAND2X1 cell = 17 “gates”.)

Table 2 shows the number of FBs used in the new architecture and the number of CLBs used in XC4000 and Virtex for an 8 bit barrel shifter, an 8×8 multiplier and an 8 bit counter. In general, the new architecture requires 3.6 times fewer logic blocks (2.8 times fewer equivalent gates) than the XC4000 to implement a multiplier and 1.6 times fewer logic blocks (4.3 times fewer equivalent gates) than Virtex (2 slice per CLB) to implement a barrel shifter.

In summary, for a logic block area penalty of 29% over the XC4000 architecture (when considered equivalently without RAM and configuration logic), multipliers can be implemented with a 65% reduction in logic utilization.

Table 2: Comparison of logic block utilisation used for design implementation. (Without configuration and RAM logic)

Circuit	XC4000	Virtex (2 slice per CLB)	New Architecture
	No. of CLBs (000 gates)	No of CLBs (000 gates)	No. of FBs (000 gates)
8 bit bshift	16 (45.2)	8 (78.8)	5 (18.3)
8x 8 multiply	73 (206.4)	16 (157.6)	20 (73.2)
8 bit counter	4 (11.3)	3 (29.6)	2 (7.3)

4.2 Delay Estimation

It is difficult to compare the delay performance of these two architectures when implementing a particular function (e.g. multiplication). CAD tools can provide estimates of performance of particular existing FPGAs (e.g. XC4000) but it requires detailed knowledge of the FPGA design and manufacturing process, along with a detailed layout level design of the proposed architecture using similar design and process constraints in order to enable a comparable estimate of the performance of the new architecture.

Even then, the performance of a multiplier in XC4000 FPGAs is very dependent on the particular routing resources used during the routing stage of the design. Notwithstanding these limitations, it is still possible to make some qualitative statements about the relative performance.

The main factor influencing timing performance is signal propagation delay. Although some versions of the XC4000 have dedicated carry propagation circuitry, the implementation of a multiplier in XC4000 also requires the use of general purpose routing resources. Signals routed along such paths will have to pass through several switch blocks with each switch introducing significant delay. The dedicated interconnect provided in the proposed architecture allows the implementation of multiplier circuits without using general purpose routing resources and hence with fewer switch elements in the signal paths.

5 CONCLUSION

This paper presents a novel FPGA logic block architecture containing a flexible multiplier unit with configurable carry circuitry in addition to a conventional LUT based logic block. This design provides for efficient (in terms of both area, or logic utilization, and delay) implementation of multipliers and similar arithmetic circuits. The architecture also provides dedicated interconnect between blocks. A programmable multiplexer allows for the efficient implementation of barrel shifters, thus enhancing the efficiency for floating point arithmetic operations as well as for fixed-point arithmetic operations. The proposed architecture is suitable for complex DSP applications, and when compared with the Xilinx XC4000, results in a 65% reduction in logic utilization.

REFERENCES

- [1] Altera, “Programmable logic device family data book”, 2000.
- [2] Dadda.L, “Some Schemes for Parallel Multipliers”, *Alta Frequenza*, Vol.34, pp 349-356, 1965.
- [3] Haynes, S.D., and Cheung, P.Y. “Configurable Multiplier Blocks for use within a FPGA”, *IEEE Trans.Computers*, Vol .3, No.1, 1998, pp 638-639
- [4] Haynes, S.D., Ferrari, A.B. and Cheung, P.Y.K., “Flexible Reconfigurable Multiplier Blocks Suitable for Enhancing the Architecture of FPGAs,”, in *Proceedings of Custom Integrated Circuit Conference (CICC)*, San Diego, California, May 16-19 1999, pp. 191-194
- [5] Lee, H. and Flynn, M. “Coarse Grained Carry Architecture for FPGA”, *Proceedings of the ACM/SIGDA international symposium on FPGAs*, Feb 10 2000, Monterey, CA.
- [6] Lucent Technologies Inc, “Create Multiply Accumulate Functions in ORCA FPGAs”, Article from *Synario Design Automation*, 1996.
- [7] Wallace C.S., “ A suggestion for fast multipliers”, *IEEE Trans.Electronic Computers*, Vol.EC-13, pp.14-17, 1964.
- [8] Xilinx Corporation Inc, “Programmable logic Databook”, 2000.