Tradeoff Analysis of Integer Multiplier Circuits Implemented in FPGAs

M. A. Thornton
Mississippi State University
J. D. Gaiche
Acxiom Corporation
J. V. Lemieux
University of Arkansas

Abstract

Integer multiplication is a necessary operation for performing many tasks relevant to multimedia and telecommunications processes. Here, we discuss the results of an investigation into the effectiveness of automated synthesis tools as related to a sample of modern Programmable Logic Devices (PLDs). Although it is generally accepted that superior results in terms of required area and circuit delay can generally be obtained through manual implementation of such circuits, the exclusive use of automated synthesis tools based upon an original specification in terms of a Hardware Description Language (HDL) is presented here. The results of several different approaches to multiplier architectures are presented.

1 Introduction

Field Programmable Gate Array (FPGA) and Complex Programmable Logic Devices (CPLDs) devices have matured to the point that they present a viable alternative for the implementation of general arithmetic and digital signal processing (DSP) circuitry [4] [2]. DSP processing tasks have traditionally been realized using microprocessors, Application Specific ICs (ASIC) or special purpose DSP processors. However, the logic density and speeds of modern FPGA/CPLDs have now reached levels where they can be competitive with special purpose DSP devices. In terms of custom solutions, the FPGA/CPLD approach offers immediate availability compared to the relatively lengthy development and turn around time for the design and fabrication of an ASIC. As a result of these benefits, many developers are investigating the feasibility of porting DSP applications into FPGA/CPLD based circuitry.

Digital signal processing algorithms are generally multiplication intensive tasks and many different multiplication circuit architectures exist [1] [5] [9]. It is therefore crucial that the particular multiplier circuit employed must make efficient use of the internal FPGA/CPLD architecture for a DSP task to yield acceptable performance. Furthermore, the multiplication architectures which are utilized in ASIC's are not necessarily optimal for a FPGA/CPLD implementation because of their fixed internal structure. More information is needed to help determine which algorithms are best suited for FPGA/CPLD implementations and to further clarify the division of hardware and software functions in the circuitry being implemented. A survey of various multiplier circuits is presented in [1]. The motivation for this study is that dynamically reconfigurable computing solutions for DSP tasks will require automatically generated multipliers of various sizes and resource requirements. In some cases, throughput is enhanced with the use of many small but slow multipliers versus a large fast one [7].

The suitability of several multiplication algorithms for use in FPGAs is explored by describing the circuit structure using the Hardware Descriptive Language (HDL), *Verilog*, followed by simulation and synthesis runs. Each multiplier in this investigation is capable of multiplying two 16-bit integer numbers. Multiplier circuits that are investigated include a Scaling Accumulator Multiplier, a Addition Array Multiplier, a Computed Partial Product Multiplier, a Wallace Tree Multiplier, and a Partial Product Look-Up Table Multiplier. These particular architectures were chosen since they represent differing resource requirements and all can be scaled to various wordlengths.

The paper is organized as follows. First, a description of example multiplication circuits is presented. Next, the experimental results are presented and analyzed. Finally, conclusions and future efforts are addressed.

2 Multiplier Circuit Architecture

Descriptions of the integer multiplication architectures are provided in this section. As a point of reference, a purely behavioral description of a 16-bit multiplier is synthesized and results are given for the purpose of comparison to the other approaches considered here. An attempt was made to choose candidate architectures that represent diverse approaches. We have included structures that have traditionally yielded the best results in terms of ASIC implementation (addition tree and array based approaches), techniques that are known for speed but require large amounts of circuitry (combinational logic implementations such as the look-up table based approaches), and, word-serial implementations that require smaller amounts of circuitry but multiple clock cycles.

2.1 Behavioral

The behavioral multiplier is implemented using a *Verilog* statement of the form: `prod = mplier * mcand;`. When the synthesis tools encounter this statement, they default to built in multiplier models for the target device. Thus, these results are useful for comparing against the RTL level multiplier descriptions of the other architectures. The *Verilog* code for this model is shown in Figure 1.

Figure 1: *Verilog* Description of Behavioral Multiplier

```
module behmult(mcand,mplier,prod,clk,
         reset,din_valid,dout_valid);
// I/O Declarations
input   [15:0]  mcand, mplier;
input           clk, rest, din_valid;
output  [31:0]  prod;
output          dout_valid;
// Data Type Declarations;
reg     [31:0]  prod;
reg             dout_valid;

always @(posedge clk)
   if (reset == 1'b1)
      begin
        prod = 0;
        dout_valid = 1'b0;
      end
   else
      if (din_valid == 1)
         begin
           prod = mplier * mcand;
           dout_valid = 1'b1;
         end
endmodule
```

2.2 Scaled Accumulator

This is a "shift and add" approach to generating the product. The advantages are a small amount of circuitry, but a number of clock cycles equal to the wordsize of the multiplier operand.

2.3 Computed Partial Product

This circuit stores the multiplicand, twice the multiplicand and three times the multiplicand in three internal registers. These are generated very efficiently through the use of a single-bit left-shift and an adder. The multiplier word is then used two bits at a time to select the appropriate multiple of the multiplicand. As each successive bit pair from the multiplier word is brought in, the final product is accumulated in a shift register. This approach is a generalization of the scaled accumulator where a digit size of two bits versus a single bit is used. Figure 2 contains a block diagram of the computed partial product multiplier.

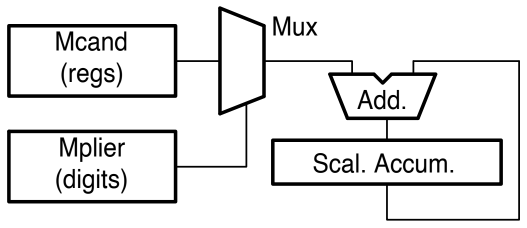

Figure 2: Block Diagram of Parallel Multiplier (array/tree type)

2.4 Addition Array

This is a array based multiplier circuit [6] using full and half single-bit adders. This approach is known for producing fast multipliers with a regular layout pattern and is popular in ASIC designs. It operates by generating all the partial products in parallel and adding them together using a rectangular array of single bit addition circuits. This is a fully parallel approach in that a product is returned at each clock cycle. Figure 3 contains a diagram of this type of multiplier.

2.5 Addition Tree

This approach accumulates the partial products through the use of a tree of 3:2 and 2:2 compressors (or single bit adders) [8] [3]. While these approaches often yield the fastest multipliers in ASIC designs, they are known for large area requirements and usually do not map well to the regular layouts inherent in FPGAs and CPLDs. In terms of operational functionality, this approach is very similar to the array multiplier described above with the difference being that a tree of adders is used to accumulate the partial products. Figure 3 also represents the addition tree multiplier.

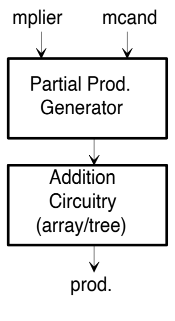

Figure 3: Block Diagram of Parallel Multiplier (array/tree type)

2.6 Memory Based Lookup Table

Here, external memory is used to store the product with the address serving as the concatenation of the multiplier and multiplicand. To reduce the size of the required memory, four partial products are stored versus the entire 32-bit product and programmable device logic is used to scale (shift) and accumulate the final products. Due to the small amount of logic required, this approach is generally limited in speed due to the memory access times. This approach also places additional demands on FPGA/CPLD I/O requirements since pins are needed to drive the memory device address and to read in the resulting data for scaling and accumulation.

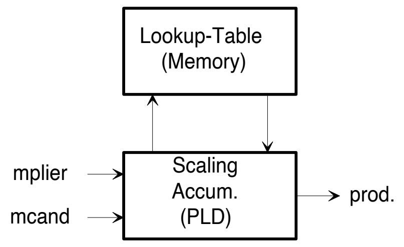

Figure 4: Block Diagram of Memory Based LUT Multiplier

3 Experimental Results

Table 1 contains results that were generated using the place and route tools from Altera MaxPlusII (ver. 9.1.2) and Xilinx Foundation Express (ver. 1.5). The Verilog descriptions were synthesized using the Synopsys FPGA Express tools bundled with MaxPlusII and Foundation, and the Synplicity Synplify tool (ver. 5.1.4). The Verilog models were tested using the PC-based Model Tech and the Unix-based Cadence Verilog-XL simulators. Estimated operating frequencies and logic resource utilization metrics are compiled based on the place and route tool log files. The place route tools were set to place approximately equal emphasis on area and delay. Since the various FPGA/CPLD manufacturers market devices with different internal architectures, each multiplier circuit is synthesized onto a variety of target devices.

In the data presented below, each multiplier type is designated using a three-letter acronym. `beh` is the behavioral multiplier, `aam` is the addition array multiplier, `atm` is the addition tree multiplier, `cpp` is the computed partial product multiplier, `sca` is the scaling accumulator approach, and, `lpp` is the look-up table based circuit. The table entries `io` and `lb` refer to inadequate device resources in terms of required I/O pins and logic blocks respectively. The columns labeled IOB, CLB, IO and LB contain the percentage of the device resources needed to implement the circuit. For those circuits that were implemented as a block of combinational logic with a register for the product, clock speed was computed as the inverse of the critical path delay plus the output register setup time. The frequencies for the `lpp` type circuits are given assuming a 10 ns SRAM, a 40 ns DRAM and a 120 ns EPROM are used for storing the partial products.

4 Conclusion

Various scalable multipliers have been synthesized and analyzed in terms of speed and area for implementation in programmable devices. It should be noted that the purpose here is not to compare the effectiveness of logic synthesis tools or programmable device architectures, but rather to get a feel for the trend in resource requirements given the various approaches. A fair comparison of the synthesis tools would require careful adherence to coding styles required by each tool and the to compare the architectures, details about internal features (such as dedicated carry-ripple logic) would need to be exploited in the HDL descriptions.

These results are useful for researchers in the area of reconfigurable computing where tradeoffs between speed and logic usage must be evaluated dynamically. The results can also be of use to designers of dedicated programmable logic circuits in the preliminary phases of resource estimation and the evaluation of candidate designs with respect to functional requirements.

References

[1] R. Andraka. *Multiplication in FPGAs. Web Page*, `http://users.ids.net/ randraka/multipli.htm`, 1999.

[2] Altera Corporation. Improving fixed-point dsp processor system performance with plds as a dsp coprocessor. *Altera Application Brief*, `http://www.altera.com/document/papers/cpdsp.pdf`, 1999.

[3] L. Dadda. Some Schemes for Parallel Multipliers. *Alta Freq.*, 34:349–356, 1965.

[4] S. K. Knapp. Using programmable logic to accelerate dsp functions. *Xilinx Application Brief*, `http://www.xilinx.com/appnotes/dspintro.pdf`, 1995.

[5] I. Koren. *Computer Arithmetic Algorithms.* Prentice Hall Publishers, Englewood Cliffs, New Jersey, 1993.

[6] S. D. Pezaris. A 40-ns 17-bit by 17-bit Array Multiplier. *IEEE Transactions on Computers*, C-20:442–447, 1971.

[7] J. Pihl. Tradeoffs Between Parallel and Serial Architectures in High Performance Digital Signal Processing. *Proceedings of the International Symposium on Integrated Circuits*, 1997.

[8] C. S. Wallace. A Suggestion for a Fast Multiplier. *IRE Transactions on Electronic Computers*, EC-13:14–17, 1964.

[9] S. Waser and M. J. Flynn. *Introduction to Arithmetic for Digital Systems Designers.* Holt, Rinehart and Winston, New York, 1982.

Table 1: Results Using Various FPGA/CPLDs

Xilinx		Synopsys Express			Synplicity Synplify		
Arch.	%IOB	% CLB	CLK (MHz)	Rate (MHz)	% CLB	CLK (MHz)	Rate (MHz)
V300PQ240-4 (Virtex)							
beh	40	5	21.4	21.4	4	42.4	42.4
aam	40	12	10.0	10.0	9	9.1	9.1
atm	40	11	18.8	18.8	11	15.8	15.8
cpp	31	5	22.7	2.8	4	42.0	5.3
sca	31	1	57.7	3.6	1	69.7	4.4
lpp	50	2	36.6	14.9/3.7/2.0	1	65.1	18.1/3.9/2.0
S30VQ100-3 (Spartan)							
beh	88	43	10.6	10.6	39	25.6	25.6
aam	88	68	4.7	4.7	86	4.3	4.3
atm	88	60	7.6	7.6	65	12.2	12.2
cpp	70	25	17.2	2.2	22	25.3	3.2
sca	68	12	27.4	1.7	6	35.4	2.2
lpp	109	9	io	io	5	io	io
S05XLPC84-4 (Spartan XL)							
beh	111	250	io,lb	io,lb	214	io/lb	io/lb
aam	111	381	io,lb	io,lb	295	io/lb	io/lb
atm	111	332	io,lb	io,lb	348	io/lb	io/lb
cpp	88	144	lb	lb	116	lb	lb
sca	86	72	44.9	2.8	39	52.4	3.3
lpp	137	54	io	io	32	io	io
4028EXHQ208 ex-2 (4000EX)							
beh	42	24	10.0	10.0	22	23.1	23.1
aam	42	38	4.0	4.0	28	8.2	8.2
atm	42	33	7.9	7.9	35	12.3	12.3
cpp	33	14	22.4	2.8	12	23.2	2.9
sca	33	7	32.0	2.0	3	35.8	2.2
lpp	51	5	22.4	11.8/3.5/1.9	3	38.3	15.1/3.8/2.0
40150XVHQ240 xv-09 (4000XV)							
beh	38	4	8.9	8.9	4	33.0	33.0
aam	38	7	3.4	3.4	5	10.2	10.2
atm	38	6	6.0	6.0	7	15.5	15.5
cpp	30	2	19.2	2.4	2	38.6	4.8
sca	29	1	17.0	1.1	1	61.5	3.8
lpp	46	1	11.4	7.8/3.1/1.8	1	59.2	17.6/3.9/2.0
3020APC68-6 (3000A)							
beh	115	414	io,lb	io,lb	607	io,lb	io,lb
aam	115	525	io,lb	io,lb	576	io,lb	io,lb
atm	115	648	io,lb	io,lb	637	io,lb	io,lb
cpp	91	337	lb	lb	343	lb	lb
sca	89	185	lb	lb	92	135.1	8.4
lpp	143	98	io	io	84	io	io
Altera		Synopsys Express			Synplicity Synplify		
Arch.	%IO	% LC	CLK (MHz)	Rate (MHz)	% LC	CLK (MHz)	Rate (MHz)
EPF10K20TC144-3 (Flex 10k)							
beh	64	71	9.2	9.2	48	26.0	26.0
aam	64	67	6.1	6.1	54	6.1	6.1
atm	64	67	10.4	10.4	lb	lb	lb
cpp	50	52	12.2	1.5	28	33.9	4.2
sca	48	13	20.0	1.3	15	49.5	3.1
lpp	81	10	17.2	10.2/3.4/1.9	8	55.2	17.2/3.9/2.0
EPF81188AQC208-2 (Flex 8k)							
beh	45	81	8.7	8.7	55	23.8	23.8
aam	45	77	5.3	5.3	61	4.7	4.7
atm	45	76	9.6	9.6	80	8.3	8.3
cpp	36	59	10.7	1.3	34	21.6	2.7
sca	35	15	17.5	1.1	14	34.7	2.2
lpp	56	12	16.5	9.9/3.3/1.8	9	41.5	15.6/3.8/2.0
EPF6010ATC100-1 (Flex 6k)							
beh	95	93	12.3	12.3	62	23.0	23.0
aam	95	89	7.1	7.1	lb	lb	lb
atm	95	87	13.4	13.4	lb	lb	lb
cpp	74	68	14.1	1.8	38	33.6	4.2
sca	73	17	25.4	1.6	16	41.3	2.6
lpp	io	io	io	io	io	io	io