16bit RISC processor with 8x8 low power Radix-4 Booth encoded Dadda Tree Multiplier

Wang Cao
University of Michigan – Ann Arbor
wangcao@umich.edu

Bohan Wang
University of Michigan – Ann Arbor
bohanw@umich.edu

Matthew Mojick
University of Michigan – Ann Arbor
mwojick@umich.edu

Fan Wu
University of Michigan – Ann Arbor
fanwudy@umich.edu

Yayan Zhao
University of Michigan – Ann Arbor
yayanzh@umich.edu

ABSTRACT

This paper describes an 8x8 Booth encoded multiplier with Dadda tree partial product reduction region and low power adder cell. It also features modified Sklansky tree adder in consideration of different arrival time in redundant output. It is integrated as part of the datapath of 16 bit RISC processor implemented in IBM 0.13um technology. The speed of the multiplier is decent with 1.7ns delay and an area of 73.6um x 89.6um. The program counter and controller is synthesized and auto placed and routed. The overall processor is verified successfully, and runs at 250MHz maximum clock frequency.

Keywords

Multiplier; booth encoding; Dadda tree; two-stage RISC microprocessor

1. INTRODUCTION

The multiplier is a key component in the microprocessor, especially for DSPs. The speed improvement on the multiplier will be crucial to improve the overall performance of a microprocessor for signal and graphic processing. On the other hand, due to the increasingly severe constraints imposed on power, demands on power requires novel architecture and circuit-level design to address the need. We propose an 8x8 Radix-4 Booth encoded Dadda tree multiplier with low power full adder cell.

The report is organized as follows: Section 2 describes our multiplier design in details and evaluates its performance. The multiplier integration with baseline CPU is presented in Section 3. Section 4 summarizes processor performance and addresses design problems and improvements.

2. MULTIPLIER

There are three steps for our multiplier. The block diagram is shown in Figure 1. In the first step, the partial products are generated. In the second step, the partial products are accumulated and reduced to two rows (sums and carries). The final step is to sum these two rows using a fast tree adder.

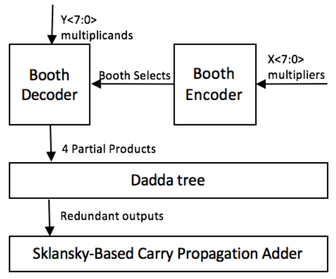


Figure 1. Multiplier structure

2.1 Partial Product Generation

2.1.1 Booth encoder and decoder circuit

Radix-4 booth encoding is efficient in reducing partial products to be accumulated from n to $\frac{n}{2}$. The circuit diagrams for Booth encoder and Booth decoder are shown in Figure 2 and 3. The Single, Double and Neg signals are generated in Booth encoder and used to generate partial products in decoder circuit.

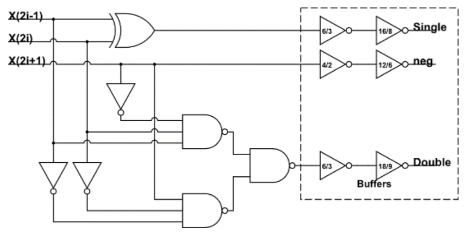


Figure 2. Booth encoder circuit

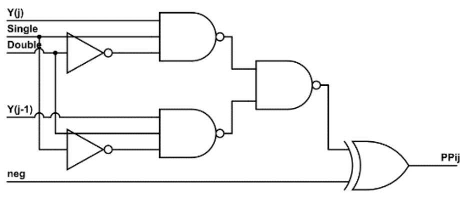
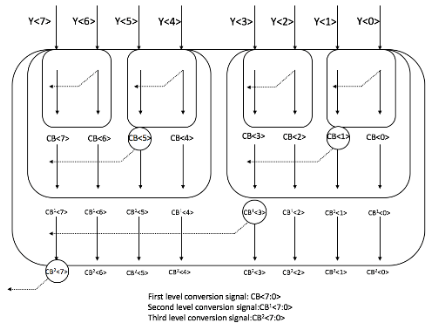


Figure 3. Booth decoder circuit

2.1.2 The 2's Complementer

Despite of the advantages of booth encoding method, sign extension prevention and negative encoding of booth encoding will cause the formation of additional partial product row, which is the last Neg signal. This will require more hardware, as well as increase the delay time of the multiplier. In order to remove the additional partial product, we generate the 2's complement of the multiplicand for the last row while select signals are generated and use Neg signal to choose between Y and -Y. And the output goes into Booth decoder to generate the last partial products.

The conversion of a number to its 2's complement form is performed in our 2's complementer. The conventional method flips everything bit of the number and plus one. As introduced in [1], Our method finds the first '1' from the least significant bit and then flip all bits to the left of this '1' bit. A tree structure is used find this '1' bit as shown in Figure 4. Each bit is assigned a conversion bit for the bit to the left of it. The "conversion bit" (CB) will dominate the all other CBs to the left of it. i.e. CB'<3>=1 means CB'<4-8>=1.



Y<7>
Y<6>
Y<5>
Y<4>
Y<3>
Y<2>
Y<1>
Y<0>
CB<7>
CB<6>
CB<5>
CB<4>
CB<3>
CB<2>
CB<1>
CB<0>
First level conversion signal: CB<7:0>
Second level conversion signal: CB'<7:0>
Third level conversion signal: CB²<7:0>

Figure 4. The tree to find the rightmost '1' bit

The implementation could be simply a tree of OR gates connecting conversion bits. After optimization on the critical path, by changing some OR gates to NAND and NOR gates the number of stages reaches optimal. The delay overhead introduced by the 2's complementer is small since it runs in parallel with the booth encoder as shown in Figure 5. The Neg signal is used to select multiplicand in its original or 2's complement form. The savings in the partial product reduction tree also compensate the delay and the area overhead.

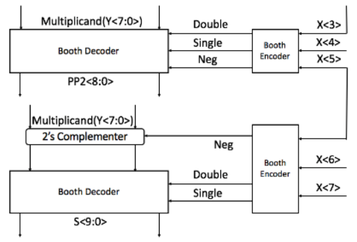


Figure 5. The 2's complementer runs in parallel with the booth encoder

2.2 Reduction tree

After the 4 partial products are generated, we have to reduce the number of terms to 2 for the final addition stage. We have implemented this in the form of a Dadda tree, which is essentially a variation of a Wallace tree. It has been shown that the Dadda tree structure is faster than a standard Wallace tree [2]. Unlike a Wallace tree, Dadda tree only performs as many additions per stage as needed. For instance, initially, the max number of bits per weight is 4. The first reduction stage reduces this to 3, and the next to 2. This reduction strategy is shown in Figure 6, with the sign extension strategy based on [1]. Different arrival times of the sum and carry signals have also been considered by connecting them to the fast or slow inputs of the next stage in order to optimize delay.

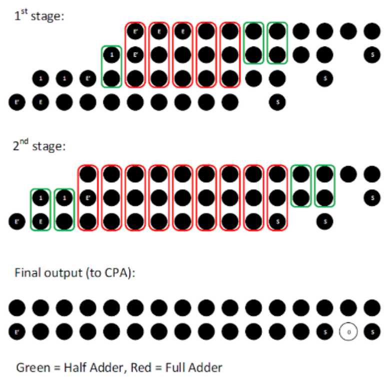


Figure 6. Dadda tree with 2's complementer, 14 full adders and 7 half adders

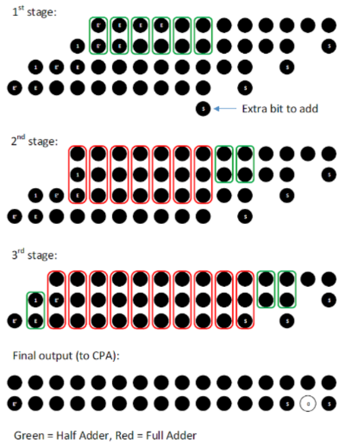


Figure 7. Dadda tree without 2's complementer, 17 full adders and 11 half adders

When comparing the partial product accumulation with a design without our 2's complementer, the number of full and half adders is by 3 and 4, respectively. This is because we do not have to add an additional '1' for the fourth partial product. The Dadda tree reduction strategy that would be needed with the additional 1 can be seen in Figure 7. The overhead associated with the 2's complementer is therefore compensated by the decrease in adders needed in this stage.

In addition to implement Dadda tree and modified 16-bit adder which combines ripple-carry and Sklansky tree, we implement smaller yet robust full adder cell that further reduces power consumption on a circuit level.

2.2.1 Low Power Full Adder

Standard static CMOS mirror adder which consists solely of pull-up and pull-down network with 28 transistors(require inverter to flip output) has the most robust operation, balanced generation of both sum and Cout, less glitch and decent speed since no contention occurs when charging or discharging any node and no voltage drop at output. However it costs large area and therefore consumes more power. This is particularly discouraged in multipliers since computations are intense when summing partial products and area is expensive in integration. One particular challenge is to compromise output swing when reducing power and area. It becomes more difficult as the supply voltage has been shrinking. After we explored and compared several full adder cells, we choose a full adder cell mixing CMOS transmission gate and pass transistor-based MUX.

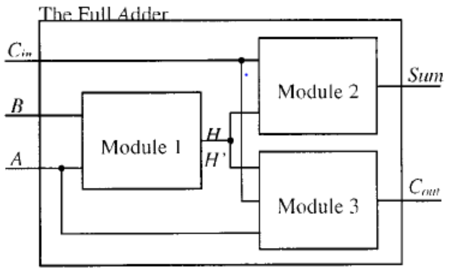


Figure 8. main modules of a full adder cell[2]

A more systematic way to implement full adder is to improve the generation of "half sum"[3], H = A XOR B, since both SUM and COUT can be expressed as SUM = H XOR C_{in} = HC_{in}'+H'C_{in}, COUT = AH'+C_{in}H. Some of the possible choices are very efficient in area and power, such as a 10T Static energy recovery full adder, and 12 T MUX-based full adder topology [4]. The major problem is significant voltage drop that when A, B, and carry-in are high, which makes cascading these cells for the column addition difficult. As [4] shown, maximum voltage drop can be at least twice of threshold voltage. If these cells are cascaded, it is risky for such circuit to fail when input is dropped below the switching point, especially with 1.2V supply. We then decide to adopt the following adder design, which has less transistor count, decent speed, reduced leakage and switching power as well as full output swing.

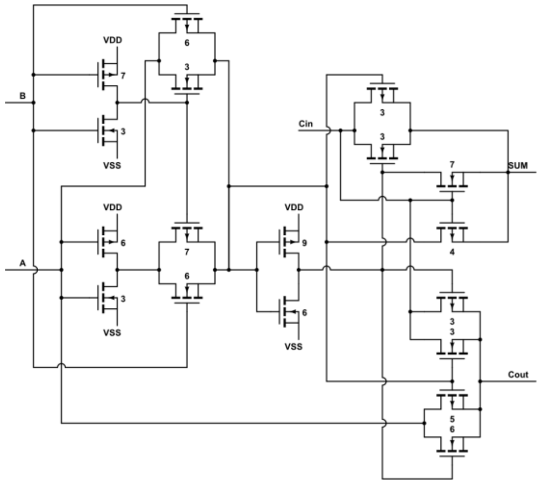


Figure 9. Full Adder schematic

As [3] shown, such topology has very asymmetric sizing. Each transistor is sized with iterated simulation to isolate the single transistor causing the longest delay for a given input pattern. In our process, minimum width is 280nm. We compared the documented sizing in the literature as well as reduced sizing of transmission gate, and it shows a 120ps performance improvement with the proposed size. In term of circuit, first module is implemented as transmission gate-based XOR. As discussed above, half sum logic is the key to improve power efficiency and area since static CMOS XOR is cumbersome and large. Since H' is generated by inverting H, the maximum voltage drop at output is V_{th}. If cascading, all signals will be restored by static inverter to full swing. However, performance is skewed from topology, and the longest path being from input B to Sum, while carry propagation is aggressively reduced to only 64ps. When routing Dadda tree, the longest path is connected to the shortest path input of next adder, so the skew performance can be mitigated.

Table 1. Comparison between proposed adder and mirror adder

	CMOS Mirror Adder	Proposed Adder
Total power consumption	3.86uW	2.73uW
B to Sum delay	180ps	310ps
Cin to Cout delay	160ps	64ps

This cell is compared with the CMOS mirror adder in terms of performance and power, and the result are shown in the Table above. We use the same set of 1000 randomized input patterns for 2us, which shows 30% total power improvement without too much performance penalty. In addition, output driver may be desired if fanout and wiring is huge in more complex and bigger multiplier since reduced drive strength will slow down transition considerably.

2.3 Carry Propagation Adder

The design for the final carry propagation adder (CPA), shown in Figure 10, is based on the fact that inputs are arriving to the adder at different times, due to the nature of the Dadda tree in the second stage. This means that bits that are in the middle weight categories will arrive at later times than the least and most significant bits. So instead of optimizing the speed for all of the bits, the first few least significant bits can be allowed to simply ripple-carry through, since the other bits haven't arrived yet anyway. So this saves power without sacrificing performance. But once the rest of the bits have arrived, we want to speed up the completion of the carry propagation, so at this point it switches over to a Sklansky tree adder. This was chosen since it is one of the faster tree adders, and, compared to a full 16-bit Sklansky adder, our design will not have as much fanout as it usually does. Also, compared to a full 16-bit Sklansky adder, the inclusion of the ripple carry chain in the beginning reduces the number of black cells from 17 to 11 (the number of gray cells are the same).

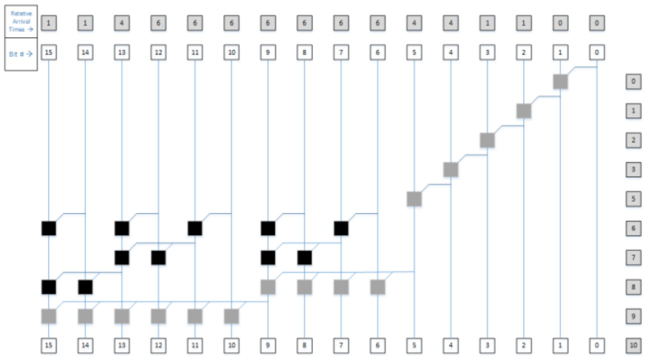


Figure 10. Block diagram of CPA

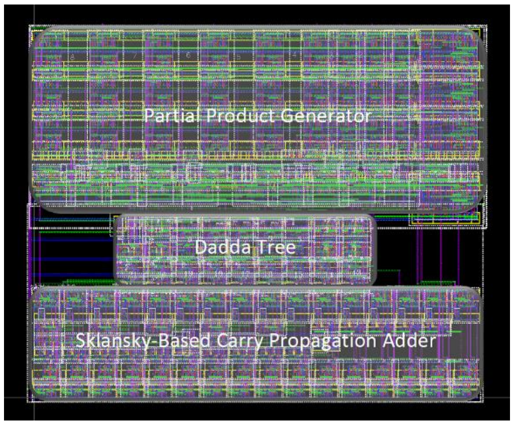

Partial Product Generator
Dadda Tree
Sklansky-Based Carry Propagation Adder

Figure 11. Layout of the multiplier

2.4 Functionality Verification

An exhaustive simulation is performed on our multiplier before layout. A nested for loop generates multiplier and multiplicand. For each test input, the comparison between multiplier output and calculated output is made and a count variable would increment when these two outputs differ. The following figure shows sample waveform for this simulation

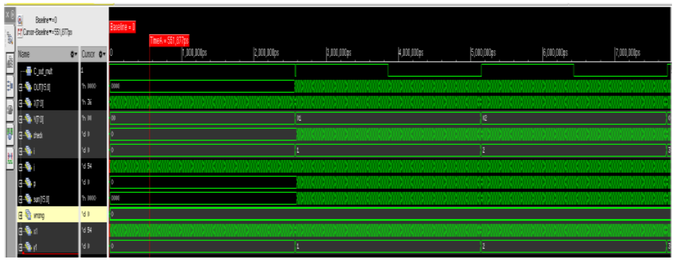


Figure 12. Multiplier Verilog simulation

2.5 Performance Comparison

Table 2 compares the synthesized multiplier design with 2's complementer with Artisan TSMC 0.13um 1.2V technology [1] and our design in the same technology node and voltage.

Table 2. Comparison between our design and multiplier design from the literature

	Area(um^2)	Delay(ns)	Power(mW)
Our design	6594 (73.6x89.6)	1.7	0.3
Multiplier from [1]	4030	3.03	6.388
Conventional [1]	4578	3.42	7.275

Our design features a really fast speed over the one synthesized, at the cost of area. After careful analysis of our multiplier floorplan, the multiplier could be smaller but with more complicated wiring. Comparing in numbers, our multiplier design saves over 90 percent of the total power. This result is measured in HSPICE by integrating supply current multiplied by supply voltage in a given time interval. We are not convinced by this result since the reference paper does not mention how they measured the power.

3. BASELINE CPU

3.1 Register File

Our 16-word × 16-bit register file is in master-slave configuration with one write port and two read ports. Each master latch will drive 16 slave latch. Two read signals can be inserted to read the output of the slave latch from two read ports for Rsrc and Rdest. TX gates are used on the output bus so that only one bus is needed. Critical path is sized for better speed performance. Clock inverters and output buffers are also sized based on logic effort considering the large load caps.

3.2 ALU

Adder, multiplier, shifter and logic operations (AND, OR, XOR) are implemented in ALU. It features a carry-select adder, which accelerates the critical path compared to carry-bypass adder by precomputing both the possible inputs and using multiplexer to choose between them, at the cost of area. Different carry-select group partitions are simulated through Hspice and we finally decide (3, 4, 4, 5) for optimal speed. The first carry-in signal is also the select signal to choose between addend B for ADD operation and $\bar{B}$ for SUB operation and CMP operation. NAND tree zero detector is implemented to generate zero flag. Carry over-flow (C) flag and greater than (N) flag are also generated in this block.

Bitwise operations (AND, OR, XOR) share the logic with adder for compact and efficient design and an array of TX gate-based 4 to 1 multiplexer is implemented to choose between operations.

3.3 Shifter

Logarithmic shifter is finally implemented to achieve left shift 0 – 15 bits. We use TX gate for the shifter rather than the pass transistor for better maintaining the voltage level. Buffers for the control signals are sized considering the large fanout of the control signals.

3.4 Integration with the Multiplier

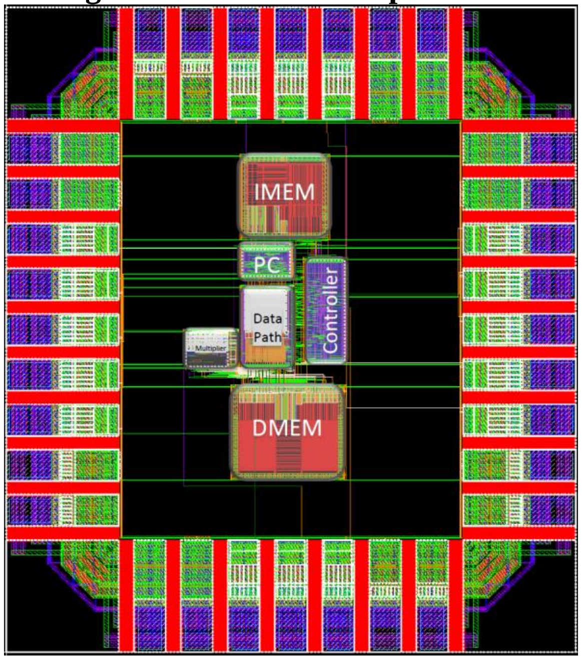


Figure 13. The dimensions of the integrated processor (excluding pads) are 363µm x 708µm, for a total area of 257,004µm^2.

3.5 PERFORMANCE EVALUATION

Table 1. Critical Path Delay of Major components

Component	Delay	Units
Register File(T_{CQ})	0.59	ns
Adder	1.74	ns
Shifter	0.54	ns
Multiplier	1.7	ns
Controller	2.2	ns
PC	1.31	ns

Each module is tested separately to better describe its performance, and the table above documents our data. After simulations on some provided test file the processor stably runs at 250MHz.

4. SUMMARY

The overall processor along with the multiplier have been tested to work functionally and robustly. In general it is a successful design, and it achieves the goal to reduce power and provide decent performance. There are several potential improvements though. To begin with, datapath is not optimized and can be a bottleneck of the overall performance. Carry select adder can be replaced with tree adder to provide faster arithmetic operation. The multiplier can also be improved by better floorplan, and logic gate cells can be optimized to increase speed.

5. ACKNOWLEDGMENTS

We are very thankful to Jaeyoung Kim for his generous help throughout the whole semester and Prof. Dennis Sylvester for his useful advices for all area of our project.

6. REFERENCES

[1] Jung-Yup Kang, and Gaudiot, J. A Simple High-Speed Multiplier Design. *IEEE Transactions on Computers 55*, 10 (2006), 1253-1258.

[2] Townsend, W., Swartzlander, Jr., E. and Abraham, J. A comparison of Dadda and Wallace multiplier delays. *Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, (2003).

[3]Shams, A., Darwish, T. and Bayoumi, M. Performance analysis of low-power 1-bit CMOS full adder cells. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 10*, 1 (2002), 20-29.

[4]Moradi, F., Wisland, D., Mahmoodi, H., Aunet, S., Cao, T. and Peiravi, A. Ultra low power full adder topologies. *2009 IEEE International Symposium on Circuits and Systems*, (2009).