MINI PROJECT

IMPLEMENTATION OF 16 BIT MULITPLIER USING BOOTH'S ALGORITHM

Authors: Sudarshan Sharma 15EC30034
Preetish 15EC30023

November 23, 2019

Contents

INTRODUCTION

Multiplication involves two basic operations creation of partial products and their accumulation.Consequently there are two ways to increase speed of multiplication to reduce the number of partial products and increase their accumulation.
In this experiment we consider using Booth's algorithm one of the very first algorithms which involves reducing the number of partial products.
The objective of the project is to design the architecture for the multiplication of two signed binary numbers in 2's complement form using Booth's Algorithm and Implementing Verilog design for the same.

ALGORITHM

RADIX 2 ALGORITHM:
a.Let x be multiplier(twos complement),A be multiplicand (twos complement)and y be the recoded multiplier
b.initialize y to zero.
c.The algorithm works as per the following conditions :
1. If x_n and x_{n-1} are same i.e. 00 or 11 perform arithmetic shift by 1 bit.
2. If x_n,x_{n-1} = 10 do y= y + A and perform arithmetic shift by 1 bit.
3. If x_n,x_{n-1} = 01 do y= y-A and perform arithmetic shift by 1 bit.
4.For x_0 we will consider x_{-1} to be equal to 0

x_i	x_{i-1}	Operation	Comments	y_i
0	0	shift only	string of zeros	0
1	1	shift only	string of ones	0
1	0	subtract and shift	beginning of a string of ones	$\bar{1}$
0	1	add and shift	end of a string of ones	1

Figure 1: Encoding for Radix-2

Booth's Algorithm can handle two's complement multiplication well and if unsigned numbers are to be multiplied we must add a zero to the left of multiplier to ensure correctness of result.These are very inconvenient while designing a synchronous multiplier and inefficient when there are isolated ones.So we can use radix 4 by examining 3 bits at a time

RADIX-4 MODIFIED BOOTH ALGORITHM:

1.Let x be multiplier,A be multiplicand and y be recoded multiplier

2.Bits x_i and x_{i-1} recoded into y_i and y_{i-1},x_{i-2} serves as reference bit

3.Separately - x_{i-2} and x_{i-3} recoded into y_{i-2} and y_{i-3} -x_{i-4} serves as reference bit

4.Thus groups of three bits each overlap with right most, being $x_1x_0x_{-1}$ the next one being $x_2x_1x_0$ and so on

The below table gives the rules for radix-4 booths algorithm While using the above algorithm

x_i	x_{i-1}	x_{i-2}	y_i	y_{i-1}	operation	comments
0	0	0	0	0	$+0$	string of zeros
0	1	0	0	1	$+A$	a single 1
1	0	0	$\bar{1}$	0	$-2A$	beginning of 1's
1	1	0	0	$\bar{1}$	$-A$	beginning of 1's
0	0	1	0	1	$+A$	end of 1's
0	1	1	1	0	$+2A$	end of 1's
1	0	1	0	$\bar{1}$	$-A$	a single 0
1	1	1	0	0	$+0$	string of 1's

Figure 2: Radix-4

we have to make sure that n is even.

ARCHITECTURE/IMPLEMENTATION

According to the Boothś Multiplier Algorithm we first build an Radix-2 encoding based multiplier for 16 bit. Fig. 3 describes the algorithm in details. and one can easily infer from the algorithm used, Moreover we have described the algorithmic approach for the problem.

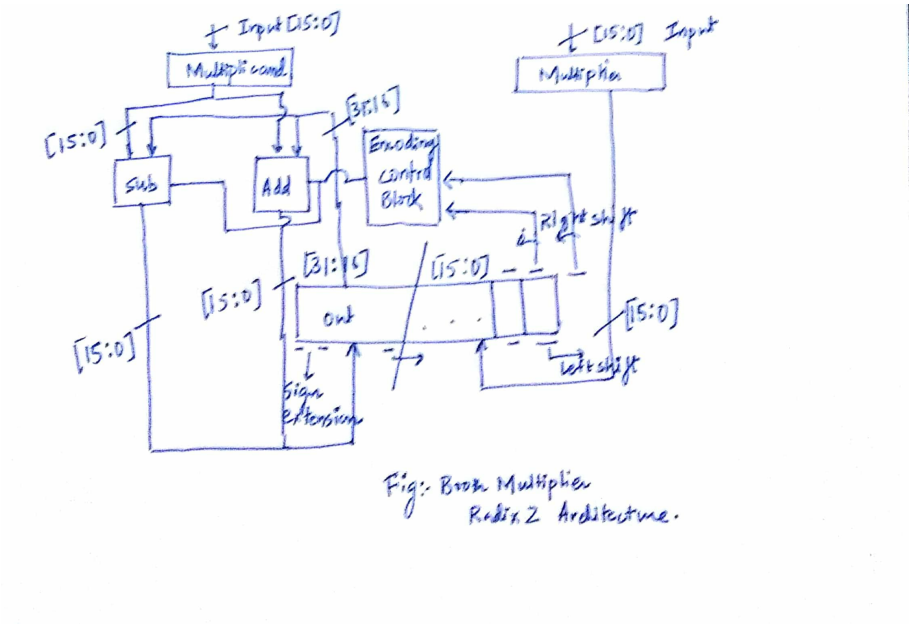

Figure 3: Radix-2 architecture

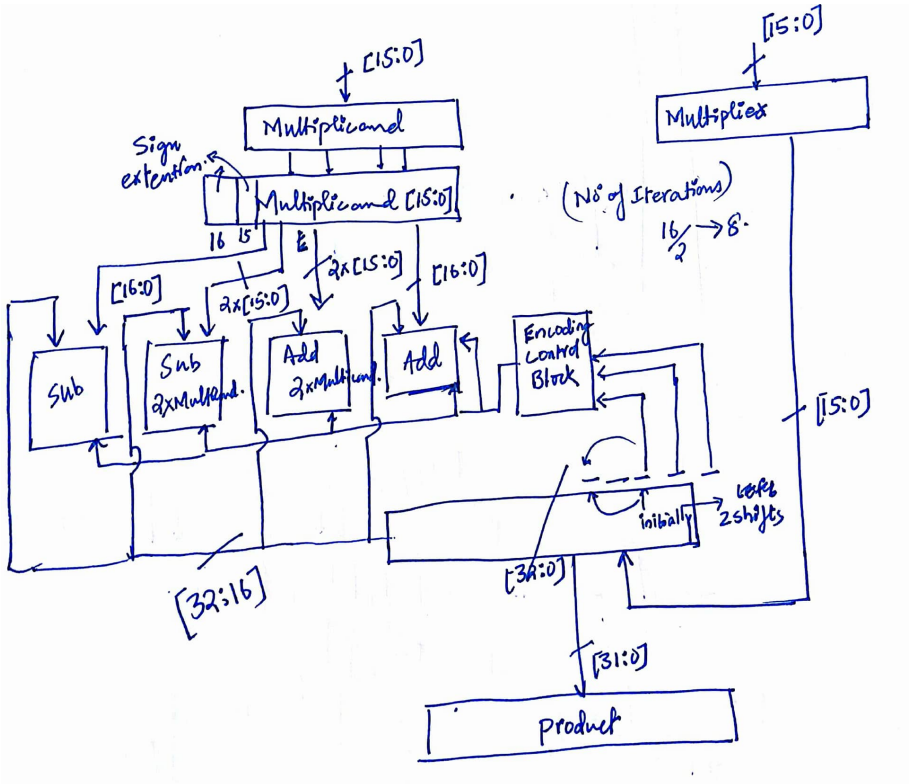

Figure 4: Radix-4 architecture

ADDITIONAL FEATURES

The most time consuming block of the multiplier are the adders and the subtractors, we built the multiplier based on three different kinds of adder a Carry-look ahead adder, Carry select adder and Inbuilt add operator in Xilinx ISE i.e Carry Chain Adders in the FPGA chip. The Carry Lookahead Adder and Carry Select Adders are highly hardware hungry but are relatively faster than the Ripple Carry Adders. We present an comparative analysis of the multiplier based on hardware resources and frequency of operation based on the adders used.

Carry Lookahead Adder CLA

The typical circuit of carry lookahead adder is shown below we incorporated an CLA in the adder and subtractor module of the multiplier and observed the Maximum Frequency of Operation and the hardware utilization. The table shows the comparison of the multiplier performance and area. Moreover, We have cascaded four 4 -bit CLA module to obtain the adder and subtractors.

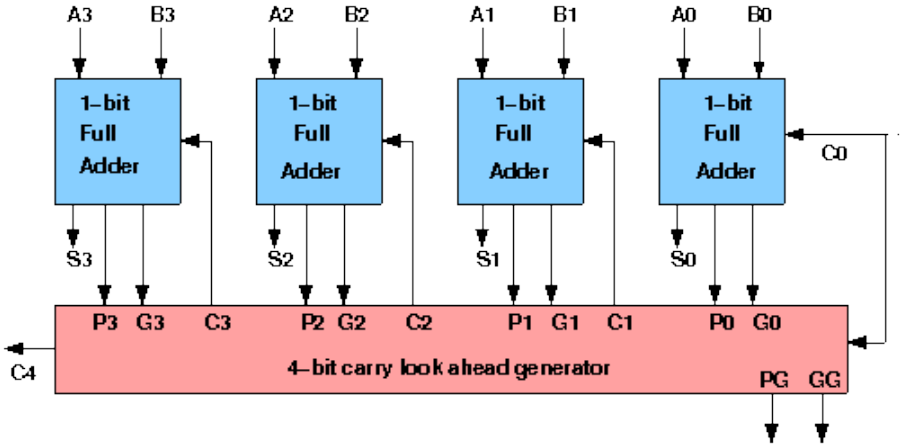

Figure 5: 4-bit Carry Lookahead Adder CLA

Carry Select Adder CSA

We got the inspiration to implement a carry select adder through [3]. The Carry select adders are the parallel ripple carry adders evaluated for two values of carry 0 and 1.After the two results are calculated, the correct sum, as well as the correct carry-out, is then selected with the multiplexer once the correct carry-in is known. Carry Select Adder is shown below in the Figure 6. The comparative study between adders is shown in the table.

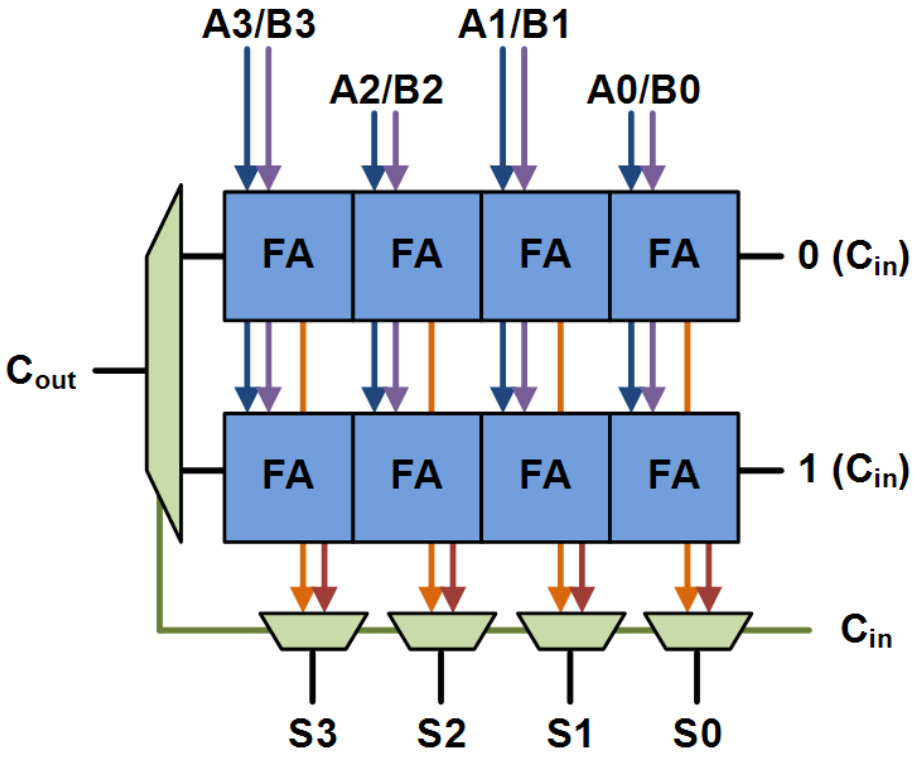

Figure 6: 4-bit Carry Select Adder CSA

Carry Chain Adders

The simple carry chain based adders continue to be the fastest implementations as they exploit specific carry propagation resources. The latter was introduced into modern FPGAs to furnish high performance carry computations through dedicated routing and fast logic gates like multiplexers, thus reducing the critical path of the most common arithmetic and logic operations executed in digital applications.

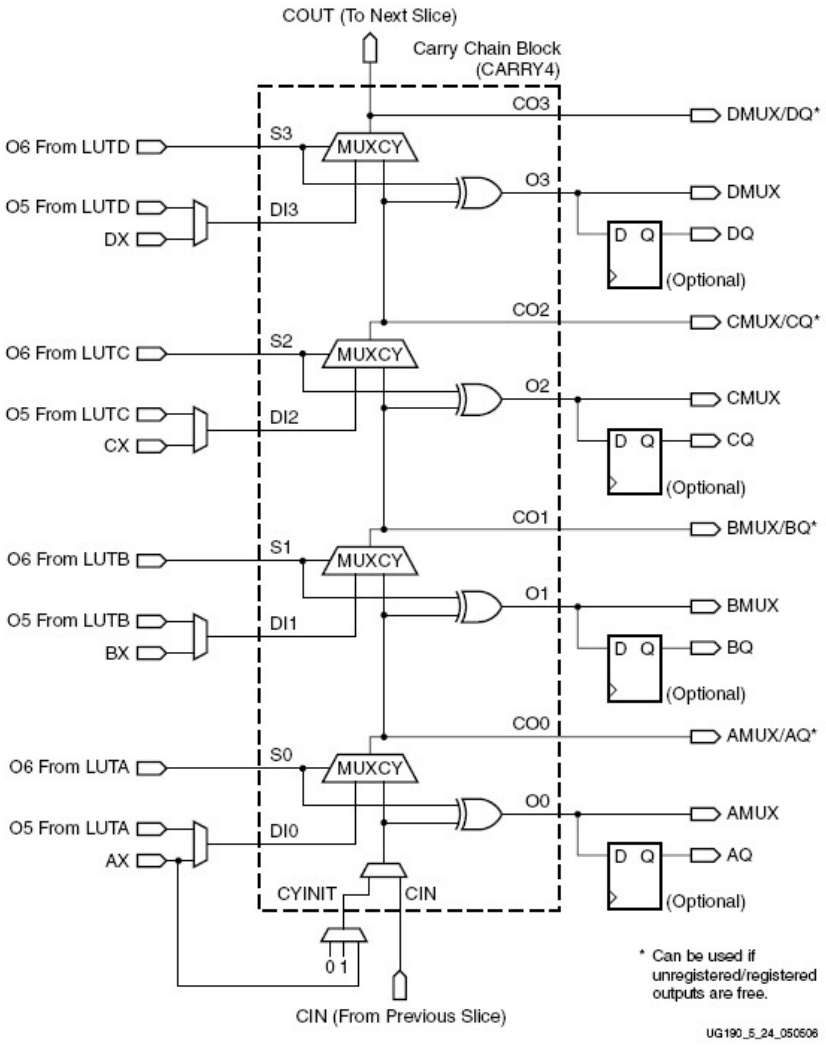

Figure 7: 4-bit Carry Chain Adder in FPGA Slices

OBSERVATIONS/RESULT

Device Utilization Summary

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                 37 out of  93,120    1%
    Number used as Flip Flops:               37
    Number used as Latches:                   0
    Number used as Latch-thrus:               0
    Number used as AND/OR logics:             0
  Number of Slice LUTs:                      61 out of  46,560    1%
    Number used as logic:                    61 out of  46,560    1%
      Number using O6 output only:           51
      Number using O5 output only:            0
      Number using O5 and O6:                10
      Number used as ROM:                     0
    Number used as Memory:                    0 out of  16,720    0%
    Number used exclusively as route-thrus:   0

Slice Logic Distribution:
  Number of occupied Slices:                 23 out of  11,640    1%
  Number of LUT Flip Flop pairs used:        62
    Number with an unused Flip Flop:         34 out of      62   54%
    Number with an unused LUT:                1 out of      62    1%
    Number of fully used LUT-FF pairs:       27 out of      62   43%
    Number of slice register sites lost
      to control set restrictions:            0 out of  93,120    0%
```

Figure 8: Resource Utilization of Radix 2 Boothś Multiplier with Carry Chain Adders

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                 37 out of  93,120    1%
    Number used as Flip Flops:               37
    Number used as Latches:                   0
    Number used as Latch-thrus:               0
    Number used as AND/OR logics:             0
  Number of Slice LUTs:                      84 out of  46,560    1%
    Number used as logic:                    84 out of  46,560    1%
      Number using O6 output only:           65
      Number using O5 output only:            0
      Number using O5 and O6:                19
      Number used as ROM:                     0
    Number used as Memory:                    0 out of  16,720    0%
    Number used exclusively as route-thrus:   0

Slice Logic Distribution:
  Number of occupied Slices:                 38 out of  11,640    1%
  Number of LUT Flip Flop pairs used:        84
    Number with an unused Flip Flop:         59 out of      84   70%
    Number with an unused LUT:                0 out of      84    0%
    Number of fully used LUT-FF pairs:       25 out of      84   29%
    Number of slice register sites lost
      to control set restrictions:            0 out of  93,120    0%
```

Figure 9: Resource Utilization of Radix 2 Boothś Multiplier with CLA Adders

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                    37 out of  93,120    1%
    Number used as Flip Flops:                  37
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                        106 out of  46,560    1%
    Number used as logic:                      106 out of  46,560    1%
      Number using O6 output only:              80
      Number using O5 output only:               0
      Number using O5 and O6:                   26
      Number used as ROM:                        0
    Number used as Memory:                       0 out of  16,720    0%
    Number used exclusively as route-thrus:      0

Slice Logic Distribution:
  Number of occupied Slices:                    45 out of  11,640    1%
  Number of LUT Flip Flop pairs used:          107
    Number with an unused Flip Flop:            79 out of     107   73%
    Number with an unused LUT:                   1 out of     107    1%
    Number of fully used LUT-FF pairs:          27 out of     107   25%
    Number of slice register sites lost
      to control set restrictions:               0 out of  93,120    0%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
```

Figure 10: Resource Utilization of Radix 2 Boothś Multiplier with CSA Adders

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                    36 out of  93,120    1%
    Number used as Flip Flops:                  36
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                        116 out of  46,560    1%
    Number used as logic:                      116 out of  46,560    1%
      Number using O6 output only:             106
      Number using O5 output only:               0
      Number using O5 and O6:                   10
      Number used as ROM:                        0
    Number used as Memory:                       0 out of  16,720    0%
    Number used exclusively as route-thrus:      0

Slice Logic Distribution:
  Number of occupied Slices:                    37 out of  11,640    1%
  Number of LUT Flip Flop pairs used:          117
    Number with an unused Flip Flop:            88 out of     117   75%
    Number with an unused LUT:                   1 out of     117    1%
    Number of fully used LUT-FF pairs:          28 out of     117   23%
    Number of slice register sites lost
      to control set restrictions:               0 out of  93,120    0%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
```

Figure 11: Resource Utilization of Radix 4 Boothś Multiplier with Carry Chain Adders

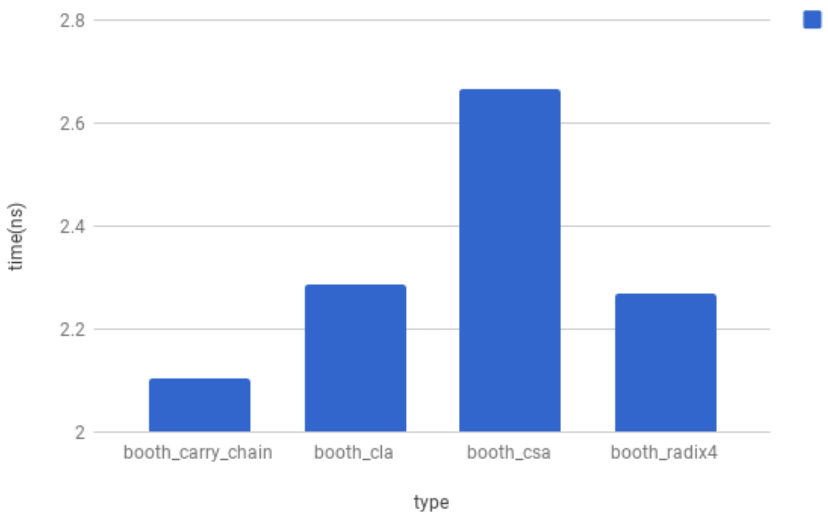

Figure 12: Best time performance of different implementations

DISCUSSION

As we can easily observe from the analysis presented that the Carry Chain Adder implementation in the FPGA Hardware yields the best performance time compared to the common fast implementations like the Carry look Ahead Adder (CLA) and Carry Select Adders (CSA). The viable reason being that the Carry chain adders are built in the hardware and are for the specific purpose, rather other implementation build the hardware from the available resources in the FPGA. Considering the hardware utilization as we can easily predict the hardware hungry nature of the CLA and CSA Implementation.

We started the Booth's Multiplier implementation through a very naive way using may latching mechanism to obtain the add shift or the sub shift operation, later we realised the register bit merge operator present the verilog tool. It reduced the code size and enabled testing of various adder implementation and tests.

REFERENCES

- *Adapted from Computer Arithmetic Algorithms, Israel Koren, UMass. Copyright 2008 Koren, UMass and A.K. Peters.*

- *D. BOOTH, ANDREW. (1951). A signed binary multiplication technique. Quarterly Journal of Mechanics and Applied Mathematics. 4. 10.1093/qjmam/4.2.236.*

- *L. P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication," in IEEE Transactions on Computers, vol. C-24, no. 10, pp. 1014-1015, Oct. 1975.*

- *Singh, Dr Jaikaran & Tiwari, Mukesh & Shrivastava, Sandeep. (2011). Implementation of Radix-2 Booth Multiplier and Comparison with Radix-4 Encoder Booth Multiplier. International Journal on Emerging Technologies 2(1): 14-16(2011). 2. 14-16.*

- *Singh, Kulvir & Kumar, Dilip. (2012). Modified Booth Multiplier with Carry Select Adder using 3-stage Pipelining Technique. International Journal of Computer Applications. 44. 35-38. 10.5120/6334-8710.*