High Throughput Implementation Of 64 Bit Modified Wallance Mac Using Multioperand Adders

BADISA SOWMYA SRI
M.Tech Student, Dept of ECE
Avanthi Institute of Engineering & Technology
Visakhapatnam, A.P, India

S.V.SUDHEER KUMAR
Assistant Professor, Dept of ECE
Avanthi Institute of Engineering & Technology
Visakhapatnam, A.P, India

Abstract: **Although redundant addition is widely used to design parallel multioperand adders for ASIC implementations, the use of redundant adders on Field Programmable Gate Arrays (FPGAs) has generally been avoided. The main reasons are the efficient implementation of carry propagate adders (CPAs) on these devices (due to their specialized carry-chain resources) as well as the area overhead of the redundant adders when they are implemented on FPGAs. This project presents different approaches to the efficient implementation of generic carry-save compressor trees.**

In computing, especially digital signal processing, the multiply–accumulate operation is a common step that computes the product of two numbers and adds that product to an accumulator. The hardware unit that performs the operation is known as a multiplier–accumulator (MAC, or MAC unit); the operation itself is also often called a MAC or a MAC operation. Power dissipation is one of the most important design objectives in integrated circuit, after speed. Digital signal processing (DSP) circuits whose main building block is a Multiplier-Accumulator (MAC) unit. High speed and low power MAC unit is desirable for any DSP processor. This is because speed and throughput rate are always the concerns of DSP system. MAC unit consists of adder, multiplier, and an accumulator it preserves a unique mapping between input and output vector of the particular circuit. In this MAC operation is performed in two parts Partial Product Generation (PPG) circuit and Multi-Operand Addition (MOA) circuit

Keywords: **Multiplier–Accumulator (MAC); Partial Product Generation (PPG); Multi-Operand Addition;**

I. INTRODUCTION

For many of the DSP and video processing applications the multi-input addition is an important operation. Using trees of carry-propagate adders multi-input addition has traditionally been implemented on FPGAs. Because to compressor trees the traditional lookup table that is LUT structure of FPGAs is not amenable so due does that thing this approach has been used in ASIC technology and these also used to implement parallel multiplication and multi-input addition. To map compressor trees onto the general logic of an FPGA we developed a greedy heuristic in this method. To design parallel multi operand adders for ASIC implementations although redundant addition is widely used, the use of redundant adders on Field Programmable Gate Arrays that is simply define as FPGAs has generally been avoided.

Generally the Binary multi operand adders are arranged in two ways that is one in an array of rows and the second one as in a tree-like structure. Where to reduce m operands into a final one each row of adders reduces one further operand in an array configuration due to that m levels of adders are required. In a m-operand adder tree the number of logic levels is either $\log_2(m)$ nor $\log_2(m) - 1$ levels for a signed-digit or a carry-save adder tree. The tree configurations are usually preferred because though the array a more regular routing the hardware cost of both configurations is similar.

Where the multiplier unit is an inevitable component in many of the digital signal processing (DSP) applications are has involving multiplications. Modified Wallace multiplier unit is used for high performance digital signal with the processing systems. The DSP applications include many of the filtering, convolution, and inner products. Most of the digital signal processing methods use nonlinear in the functions such as discrete cosine transform (DCT) or which can discrete wavelet transforms (DWT). Because they are basically accomplished by repetitive application of multiplication and addition, the speed of the Multiplication and addition arithmetic determines the execution speed and performance of the entire calculation. Multiplication-and-accumulate operations are typical for digital filters. Therefore, the functionality of the Multiplier unit enables high-speed filtering and other processing typical for DSP applications.

II. OVERVIEW

The first semiconductor chips held one transistor each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Now known retrospectively as "small-scale integration" (SSI), improvements in technique led to devices with hundreds of logic gates, known as large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and hundreds of millions of individual transistors

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like Ultra-large-scale Integration (ULSI) were used. But the huge number of gates and transistors available on common devices has rendered such fine distinctions moot. Terms suggesting greater than VLSI levels of integration are no longer in widespread use. Even VLSI is now somewhat quaint, given the common assumption that all microprocessors are VLSI or better

III. MAC OPERATION

The Multiplier-Accumulator (MAC) operation is the key operation not only in DSP applications but also in multimedia information processing and various other applications. As mentioned above, as like the MAC unit consist of multiplier, adder and register/accumulator. In this paper, we used 64 bit modified Wallace multiplier. The MAC inputs are obtained from the memory location and given to the multiplier block. This will be useful in 64 bit digital signal processor. The input which is being fed from the memory location is 64 bit. When the input is given to the multiplier it starts computing value and the output will be126 bits for the given 64 bit input and hence.

The function of the MAC unit is given by the following equation:

$$F = \sum P_i Q_i \qquad (1)$$

The output of carry save adder is 127 bit i.e. one bit is for the carry that is define as 126bits+ 1 bit. Then, the output is given to the accumulator register. In this design is parallel in Parallel Out that is PIPO where the accumulator register used. All the output values in parallel since the bits are huge and also carry save adder produces, where the input bits are taken in parallel and output is taken in parallel PIPO register is used. Fed back as one of the input to the carry save adder the output of the accumulator register is taken out. The figure 1shows the basic architecture of MAC unit.

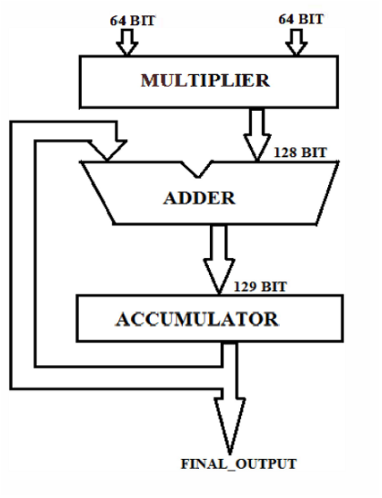


Figure 1: Basic Architecture of MAC unit

IV. VLSI

VLSI stands for "Very Large Scale Integration". This is the field which Involves packing more and more logic devices into smaller and smaller areas.

- Simply we say Integrated circuit is many transistors on one chip.
- Design/manufacturing of extremely small, complex circuitry using modified semiconductor material.
- Integrated circuit (IC) may contain millions of transistors, each a few mm in size.
- Applications wide ranging: most electronic logic devices.

V. VLSI DESIGN FLOW

Digital Circuit

Digital ICs of SSI and MSI types have become universally standardized and have been accepted for use. Whenever a designer has to realize a digital function, he uses a standard set of ICs along with a minimal set of additional discrete circuitry. Consider a simple example of realizing a function as

$$Q\,n{+}1 = Q\,n + (A\,B)$$

Here on, A, and B are Boolean variables, with Q n being the value of Q at the nth time step. Here A B signifies the logical AND of A and B; the ‘+’ symbol signifies the logical OR of the logic variables on either side. A circuit to realize the function is shown in Figure. The circuit can be realized in terms of

VLSI and systems

These advantages of integrated circuits translate into advantages at the system level:

- ✓ Smaller physical size. Smallness is often an advantage in itself-consider portable televisions or handheld cellular telephones.
- ✓ Lower power consumption. Replacing a handful of standard parts with a single chip reduces total power consumption. Reducing power consumption has a ripple effect on the rest of the system: a smaller, cheaper power supply can be used; since less power consumption means less heat, a fan may no longer be necessary; a simpler cabinet with less shielding for electromagnetic shielding may be feasible, too.
- ✓ Reduced cost. Reducing the number of components, the power supply requirements, cabinet costs, and so on, will inevitably reduce system cost. The ripple effect of integration is such that the cost of a system built from custom ICs can be less, even though the individual ICs cost more than the standard parts they replace.

Understanding why integrated circuit technology has such profound influence on the design of digital systems requires understanding both the technology of IC manufacturing and the economics of ICs and digital system

VI. OPERATORS

Arithmetic Operators:

These perform arithmetic operations. The **+** and **-** can be used as either unary (-z) or binary (x-y) operators.

Example: reg[3:0] a, c, f, g, count;

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)
%	(modulus)

f = a + c;

g = c - n;

count = (count +1)%16; //Can count 0 to 15.

Relational Operators:

Relational operators compare two operands and return a single bit 1or 0. These operators synthesize into comparators.

<	(less than)
<=	(less than or equal to)
>	(greater than)
>=	(greater than or equal to)
==	(equal to)
!=	(not equal to)

Example :if (x = = y) e = 1;

Else e = 0;

// Compare in 2's compliment; a>b

reg [3:0] a,b;

if (a[3]= = b[3]) a[2:0] > b[2:0];

Else b[3];

Wire and reg variables are positive Thus (-3'b001) = = 3'b111 and (-3d001)>3d110. However for integers 1< 6.

Bit-wise Operators:

Bit-wise operators do a bit-by-bit comparison between two operands. However set "Reduction Operators".

~	(bitwise NOT)
&	(bitwise AND)
|	(bitwiseOR)
^	(bitwise XOR)
~^ or ^~	(bitwise XNOR)

Example : module and2 (a, b, c);

input [1:0] a, b;

output [1:0] c;

assign c = a & b;

end module

Logical Operators:

Logical operators return a single bit 1 or 0. They are the same as bit-wise operators only for single bit operands. They can work on expressions, integers or groups of bits, and treat all values that are nonzero as "1". Logical operators are typically used in conditional (**if** ... **else**) statements since they work with expressions.

!	(logical NOT)
&&	(logical AND)
||	(logical OR)

Example :

wire[7:0] x, y, z; // x, y and z are multibit variables.

reg a;

. . .

if ((x == y) && (z)) a = 1; // a = 1 if x equals y, and z is nonzero.

else a = !x; // a =0 if x is anything but zero.

Reduction Operators

Reduction operators operate on all the bits of an operand vector and return a single-bit value. These are the unary (one argument) form of the bit-wise operators above.

&	(reduction AND)
|	(reduction OR)
~&	(reduction NAND)
~|	(reduction NOR)
^	(reduction XOR)
~^ or ^~	(reduction XNOR)

Example :

```
module chk_zero(a,z);
input [2:0] a;
output z;
assign z = ~| a;
Endmodule
```

Shift Operators

Shift operators shift the first operand by the number of bits specified by the second operand. Vacated positions are filled with zeros for both left and right shifts (There is no sign extension).

<<	(shift left)
>>	(shift right)

Example:

```
assign c = a << 2; /* c = a shifted left 2 bits;
vacant positions are filled with 0's */
```

Concatenation Operator:

The concatenation operator combines two or more operands to form a larger vector

{ }	(concatenation)

Example:

```
Wire [1:0] a, b; wire [2:0] x; wire [3;0] y, Z;
assign x = {1'b0, a}; // x[2]=0, x[1]=a[1], x[0]=a[0]
assign y = {a, b}; /* y[3]=a[1], y[2]=a[0], y[1]=b[1],
y[0]=b[0] */
assign {cout, y} = x + Z; // Concatenation of a result
```

Replication Operator:

The replication operator makes multiple copies of an item.

{n{item}}	(n fold replication of an item)

Example :

```
wire [1:0] a, b; wire [4:0] x;
Assign x = {2{1'b0}, a}; // Equivalent to x = {0, 0,a }
Assign y = {2{a}, 3{b}}; //Equivalent to y = {a,a,b,b}
```

Operator Precedence:

The below table shows the precedence of operators from highest to lowest. Operators on the same level evaluate from left to right. It is strongly recommended to use parentheses to define order of precedence and improve the readability of your code.

a) Operator Priority:

A clear understanding of the operator precedence makes room for a compact design description. But it may lead to ambiguity and to inadvertent errors. Whenever one is not sure of the operator priorities, it is better to resort to the use of parentheses and ensure clarity and accuracy of expressions.

Entering Synthesis Options

Synthesis options enable you to modify the behavior of the synthesis tool to make optimizations according to the needs of the design. One commonly used option is to control synthesis to make optimizations based on area or speed. Other options include controlling the maximum fan-out of a flip-flop output or setting the desired frequency of the design.

To enter synthesis options, do the following:

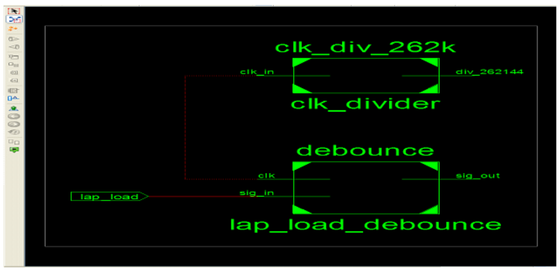
1. In the Hierarchy pane of the Project Navigator Design panel, select stopwatch.vhd (or stopwatch.v).

2. In the Processes pane, right-click the **Synthesize** process, and select **Process Properties**.

3. Under the Synthesis Options tab, set the Netlist Hierarchy property to a value of **Rebuilt**.

4. Click **OK**.

The RTL Viewer allows you to select the portions of the design to display as schematic. When the schematic is displayed, double-click on the symbol to push into the schematic and view the various design elements and connectivity. Right-click the schematic to view the various operations that can be performed in the schematic viewer.



Processes available for synthesis using the Synplify and Synplify Pro software are as follows:

• ***View Synthesis Report***

Lists the synthesis optimizations that were performed on the design and gives a brief timing and mapping report.

• ***View RTL Schematic***

Accessible from the Launch Tools hierarchy, this process displays the Synplify or Synplify Pro software with a schematic view of your HDL code.

• ***View Technology Schematic***

Accessible from the Launch Tools hierarchy, this process displays the Synplify or Synplify Pro software with a schematic view of your HDL code mapped to the primitives associated with the target technology.

Entering Synthesis Options and Synthesizing the Design

To synthesize the design, set the global synthesis options as follows:

1. In the Hierarchy pane of the Project Navigator Design panel, select stopwatch.vhd (or stopwatch.v).
2. In the Processes pane, right-click Synthesize, and select Process Properties.
3. In the Synthesis Options dialog box, select the Write Vendor Constraint File box.
4. Click OK to accept these values.
5. Double-click the Synthesize process to run synthesis.

Note: This step can also be done by selecting stopwatch.vhd (or stopwatch.v), clicking Synthesize in the Processes pane, and selecting Process > Run.

Analyzing the Signals

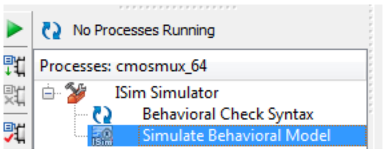
Now the DCM signals can be analyzed to verify that they work as expected. The CLK0_OUT must be 50 MHz and the CLKFX_OUT should be approximately 26 MHz. The DCM outputs are valid only after the LOCKED_OUT signal is high; therefore, the DCM signals are analyzed only after the LOCKED_OUT signal has gone high.

ISim can add markers to measure the distance between signals. To measure the

CLK0_OUT, do the following:

1. If necessary, zoom in on the waveform using the local Zoom toolbar buttons.

2. In the local waveform viewer toolbar, click the Snap to Transition toolbar button.


No Processes Running

Processes: cmosmux_64

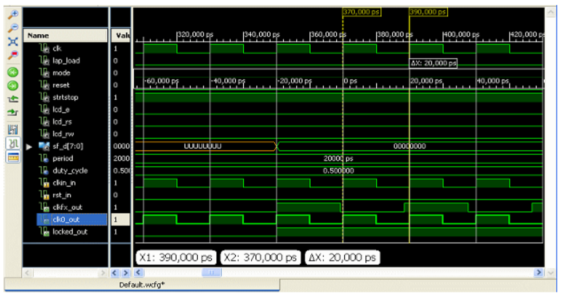
ISim Simulator

Behavioral Check Syntax

Simulate Behavioral Model

3. Click on the first rising edge transition on the CLK0_OUT signal after the LOCKED_OUT signal has gone high, then drag the cursor to the right to the next rising edge transition of the CLK0_OUT signal.

4. At the bottom of the waveform window, the start point time, end point time, and delta times are shown. The delta should read 20.0 ns. This converts to 50 MHz which is the input frequency from the test bench, which in turn is the DCM CLK0 output.



5. Measure CLKFX_OUT using the same steps as above. The measurement should read 38.5 ns. This equals approximately 26 MHz.

Adding Dividers

xilinx has the capability to add dividers in the Wave window to make it easier to differentiate the signals. To add a divider called DCM Signals, do the following:

1. Right-click anywhere in the signal section of the Wave window. If necessary, undock the window and maximize the window for a larger view of the waveform.

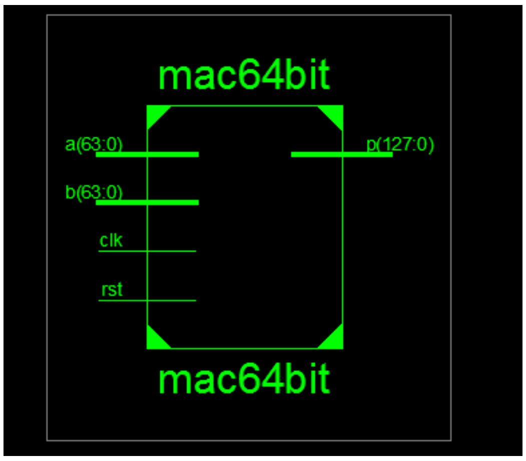
2. Select **Insert Divider**.

3. Enter **DCM Signals** in the Divider Name box.

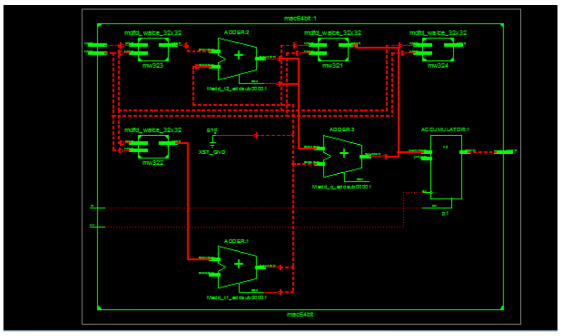
4. Click **OK**.

VII. RESULTS

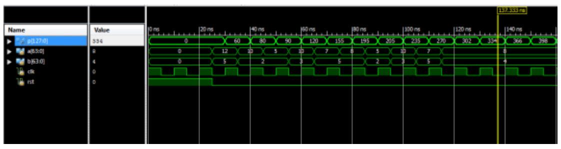
Schematics:



Rtl Schematics:



Waveforms:



VIII. CONCLUSION

The Design of high performance 64 bit Multiplier-and-Accumulator (MAC) was implemented in this paper. The total MAC unit operates at a frequency of 215 MHz's with a total power dissipation of 155.532 mW. Since the delay of 64 bit MAC is less, this design can be used in the system which requires high performance in processors involving large number of bits of the operation. The functionality of the MAC is verified using XILINX ISE 12.3i and synthesized using XILINX synthesizer.

IX. REFERENCES

[1] Javier Hormigo, Julio Villalba, Member, IEEE, and Emilio L. Zapata, " Multioperand RedundantAdders on FPGAs" ieee transactions on computers, vol. 62, no. 10, october 2013

[2] B. Cope, P. Cheung, W. Luk, and L. Howes, "Performance Comparison of Graphics Processors to Reconfigurable Logic: ACase Study," IEEE Trans. Computers, vol. 57, no. 4, pp. 433-446,Apr. 2010.

[3] S. Dikmese, A. Kavak, K. Kucuk, S. Sahin, A. Tangel, and H. Dincer, "Digital Signal Processor against Field Programmable Gate Array Implementations of Space-Code Correlator Beam former for Smart Antennas," IET Microwaves, Antennas Propagation, vol. 4, no. 5,pp. 573-577, May 2010.

[4] S. Roy and P. Banerjee, "An Algorithm for Trading off Quantization Error with Hardware Resources for MATLAB-based FPGA Design, "IEEE Trans. Computers, vol. 54, no. 5, pp. 666-676, July 2005.

[5] F. Schneider, A. Agarwal, Y.M. Yoo, T. Fukuoka, and Y. Kim,"A Fully Programmable Computing Architecture for Medical Ultrasound Machines," IEEE Trans. Information Technology in Biomedicine, vol. 14, no. 2, pp. 536-540, Mar. 2010.

[6] J. Hill, "The Soft-Core Discrete-Time Signal Processor Peripheral[Applications Corner]," IEEE Signal Processing Magazine, vol. 26,no. 2, pp. 112-115, Mar. 2007.

[7] J.S. Kim, L. Deng, P. Mangalagiri, K. Irick, K. Sobti, M. Kandemir,V. Narayanan, C. Chakrabarti, N. Pitsianis, and X. Sun, "An Automated Framework for Accelerating Numerical Algorithms on Reconfigurable Platforms Using Algorithmic/Architectural Optimization," IEEE Trans. Computers, vol. 56, no. 12, pp. 1654-1665, Dec. 2007.

[8] H. Lange and A. Koch, "Architectures and Execution Models for Hardware/Software Compilation and their System-Level Realization," IEEE Trans. Computers, vol. 57, no. 10, pp. 1363-1355, Oct. 2010.

[9] L. Zhuo and V. Prasanna, "High-Performance Designs for Linear Algebra Operations on Reconfigurable Hardware," IEEE Trans. Computers, vol. 55, no. 6, pp. 1055-1051, Aug. 2006.

[10] C. Mancillas-Lopez, D. Chakraborty, and F.R. Henriquez, "Reconfigurable Hardware Implementations of Tweakable Enciphering Schemes," IEEE Trans. Computers,, vol. 57, no. 11, pp. 1545-1561, Nov. 2010.

[11] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA," IEEE Trans. Computers, vol. 55, no. 11, pp. 1476-1513, Nov. 2006.