Design and Analysis of High Speed Wallace Tree Multiplier Using Parallel Prefix Adders For VLSI Circuit Designs

UNDER THE GUIDANCE Of
V.V.G.S RAJENDRA PRASAD(HOD)[1]Head Of the Department
Assistant Professor[1], [234]UG scholar
T.ANAND KUMAR[2], S.SURYA[3], CH. JITHENDRA KUMAR [4], V. SURESH KUMAR[5]
DEPARTMENT OF ELECTRONICS COMMUNICATION ENGINEERING
R K COLLEGE OF ENGINEERING
VIJAYAWADA, INDIA
veguntav@gmail.com, anandk85260@gmail.com,suryas4924@gmail.com, vaddisureshkumar164@gmail.com, chalapatijithendrag@mail.com,

Abstract**: Arithmetic and logic unit has been the most significant unit in any electronic devices. In the recent advancement, for an arithmetic and logic unit to be significant it needs to have an efficient algorithmic operation such as Multiplications and addition. Multiplication is the most essential operation in digital signal processing, artificial intelligence, neural networks, and machine learning. In Digital era, filters can be used as memory elements to store data. The most commonly used adders to implement digital filters, are the full adders. Here in this paper the implementation of digital filter is done with PPA. Among various PPA, the sklansky adder is more efficient than others. When implementing kogge stone adder, it experiences a larger hardware complexity. So that it is necessary to reduce the hardware complexity the architecture of sklansky adder. The comparative analysis made among various parameters such as numbers of logic gates and delay. The above proposed architecture tends to reduce the hardware complexity using various parameters**.

I.INTRODUCTION

Generally there are some basic processes to design a digitalfilter. The design filter needs to be redesign the frequencies,calculate the parameters and have to adjust the filter at eachtime. In some rare cases, the redesign requires filters to beexchanged with their filter types or with its length to fitaccording to their requirements.A complicated DSP system consists of multiple adders andmultipliers[2]. The effective design of DSP machine enchancethe performance of the system. An adder, which is afundamental component is frequently employed in manynetworks that are used in system like controllers andprocessing chips[3]. In this system a performance is improvedby the running ability of adder and multiplier.

In Digital filters can use FIR or IIR filter systems forimplement and there adaptive algorithm to update theirparameters but in practice, FIR structured digital filters can beeffortlessly done with high performance so that it has beingmore widely used. We also use FIR digital filter for ourstudies[5]. Normally FIR filter will compute the sum ofinput signal by analyzing the different types of multipleand accumulate performance.

Normal serial adder has been replaced by the parallelprefix adder. In PPA there are four adders here we usesklansky and kogge stone adder in FIR filter.

Digital filters are very important part of DSP. In fact their extraordinary performance is one of the key reasons that DSP has become so popular. Filters have two uses: signal separation and signal restoration. Signal separation is needed when the signal has been contaminated with interference, noise or other signals. For example imagine a device for measuring the electrical activity of a baby's heart (EKG) while in the womb. The raw signal will be likely to be corrupted by the breathing and the

heartbeat of the mother. A filter must be used to separate these signals so that they can be individually analyzed. Signal restoration is used when the signal has been distorted in some way. For example, an audio recording made with poor requirement may be filtered to better represent the sound as it actually occurred. Another example is of debuting of an image acquired with an improper focused lens, or a shaky camera. These problems can be attacked with either digital or analog filters. Which is better? Analog filters are cheap, fast and have a large dynamic range both in amplitude and frequency. Digital filters in comparison are vastly superior in the level of performance that can be achieved. Digital filters can achieve thousand of times better performance than an analog filter. This makes a dramatic difference in how filtering problems are approached. With analog filters, the emphasis is on handling limitations of the electronics such as the accuracy and stability of the resistors and capacitors. In comparison digital filters are so good that the performance of the filter is frequently ignored. The emphasis shifts to the limitations of the signals and the theoretical issues regarding their processing. Multiplying a variable by a set of known constant coefficients is a common operation in many digital signal processing (DSP) algorithms. Compared to other common operations in DSP algorithms, such as addition, subtraction, using delay elements, etc., multiplication is generally the most expensive. There is a trade-off between the amount of logic resources used (i.e. the amount of silicon in the integrated circuit) and how fast the computation can be done. Compared to most of the other operations, multiplication requires more time given the same amount of logic resources and it requires more logic resources under the constraint that each operation must be completed within the same amount of time. A general multiplier is needed if one performs multiplication between two arbitrary variables. However, when multiplying by a known constant, we can exploit the properties of binary multiplication in order to obtain a less expensive logic circuit that is functionally equivalent to simply asserting the constant on one input of a general multiplier. In many cases, using a cheaper implementation for only multiplication still results in significant savings when considering the entire logic circuit because multiplication is relatively expensive. Furthermore, multiplication could be the dominant operation, depending on the application.
In this thesis, we will propose several algorithms which run in software, but the solutions that these algorithms produce enable one to efficiently implement constant coefficient multiplication in hardware. Given the set of constant coefficients, said algorithms search for good hardware realizations.

II. LITERATURE SURVEY

2.1 Design of Fast FIR Filter Using Compressor and Carry Select Adder

Speed and area are now a day's one of the fundamental design issues in digital era. To increase speed, while doing the multiplication or addition operations, has always been a basic requirement of designing of advanced system and application. Carry Select Adder (CSA) is a fastest adder used in many processors to accomplish fast arithmetic function. Many different adder architecture designs have been developed to increase the efficiency of the adder. It is very commonly known that per second any processors performed millions of work functions in semiconductor industry. So when we do designing of multipliers, one of the main standards is performing speed that should be taken in the mind. In this paper, we propose a technique for designing of FIR filter using multiplier based on compressor and carry select adder. Performance of all adder designs is implemented for 16, 32 and 64 bit circuits. These structures are synthesized on Xilinx device family.

2.2 Implementation of Kogge Stone Adder for Signal Processing Application

Low power system design has turn out to be a significant performance goal. The Finite Impulse Response Filter is an efficient component for digital signal processing applications. Adders and multipliers plays an essential role in implementation [R1] of FIR filter. The proposed FIR Filter is designed by using Kogge stone adder and booth multiplier. Kogge stone adder is a high speed adder which is used for designing high performance circuits and Booth multiplier is a multiplication algorithm which is used to perform multiplication by using 2‟s complement notation. Thus the proposed design can reduce the delay and power consumption of FIR filter. The proposed FIR filters are designed with the help of verilog Hardware Description Language and synthesized using Xilinx ISE 12.4 tool.

2.3 Implementation of programmable fir filter using dadda multiplier and parallel prefix adder

Digital filters have a magnificent role in various applications related to signal processing. It is the performance of the filters that made DSP popular. Filtering is usually done to obtain a desired output by manipulating the input data. Various types of filters are used to manipulate the data that helped in creating different applications to benefit the world. The major part in any filter design is the multiplication block as the performance of any filter depends on how the multiplication is performed. There are various multipliers of which Dadda multiplier is one, the significance of this is that it makes use of very few gates to perform multiplication. Keeping this in view a programmable FIR filter design has been carried out and implemented using the MAC (Multiply and Accumulate) unit[8]. In this, the multiplication is done by Dadda multiplier and PPA is used to carry out addition. Therefore, this project helps in bringing an efficient FIR filter which could be used for many DSP applications

2.4 Design of Fast FIR Filter Using Compressor and Carry Select Adder by Deepak Kumar Patel, Raksha Chouksey, Dr. Minal Saxena

Speed and area are now a day's one of the fundamental design issues in digital era. To increase speed, while doing the multiplication or addition operations, has always been a basic requirement of designing of advanced system and application. Carry Select Adder (CSA) is a fastest adder used in many processors to accomplish fast arithmetic function. Many different adder architecture designs have been developed to increase the efficiency of the adder. It is very commonly known that per second any processors performed millions of work functions in semiconductor industry. So when we do designing of multipliers, one of the main standards is performing speed that should be taken in the mind. In this paper, we propose a technique for designing of FIR filter using multiplier based on compressor and carry select adder. Performance of all adder designs is implemented for 16, 32 and 64 bit circuits. These structures are synthesized on Xilinx device family.

2.4 Implementation of Kogge Stone Adder for Signal Processing Applications by A.Abinaya, M.Maheswari

Low power system design has turn out to be a significant performance goal. The Finite Impulse Response Filter is an efficient component for digital signal processing applications. Adders and multipliers plays an essential role in implementation[R1]of FIR filter. The proposed FIR Filter is designed by using Kogge stone adder and booth multiplier. Kogge stone adder is a high speed adder which is used for designing high performance circuits and Booth multiplier is a multiplication algorithm which is used to perform multiplication by using 2‟s complement notation. Thus the proposed design can reduce the delay and power consumption of FIR filter. The proposed FIR filters are designed with the help of verilog Hardware Description Language and synthesized using Xilinx ISE 12.4 tool

2.5 Implementation of programmable fir filter using dadda multiplier and parallel prefix by S. Madhavi, K. Rasagna, N. Kavya, M. Sindhu

Digital filters have a magnificent role in various applications related to signal processing. It is the performance of the filters that made DSP popular. Filtering is usually done to obtain a desired output by manipulating the input data. Various types of filters are used to manipulate the data that helped in creating different applications to benefit the world. The major part in any filter design is the multiplication block as the performance of any filter depends on how the multiplication is performed. There are various multipliers of which Dadda multiplier is one, the significance of this is that it makes use of very few gates to perform multiplication. Keeping this in view a programmable FIR filter design has been carried out and implemented using the MAC (Multiply and Accumulate) unit[8]. In this, the multiplication is done by

Dadda multiplier and PPA is used to carry out addition. Therefore, this project helps in bringing an efficient FIR filter which could be used for many DSP applications.

III.EXISTING SYSTEM

Multiplication involves the generation of partial products, one for each digit in the multiplier, as in Figure3.These partial products are then summed to produce the final product. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length [2]. We used the following algorithm to implement the multiplication operation for unsigned data.

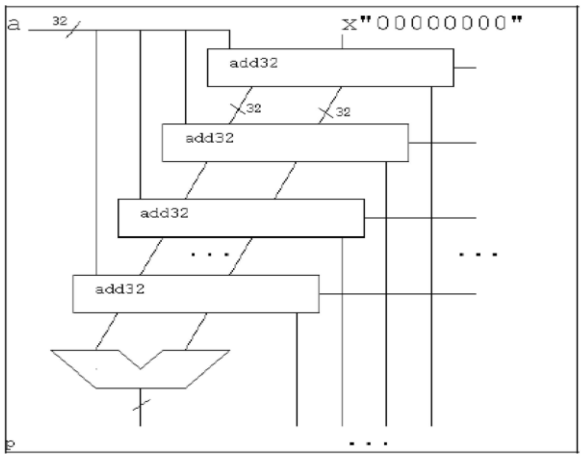


Figure 1. A partial schematic of the multiplier

3.1 5 MULTIPLICATION ALGORITHM

Let the product register size be 64 bits. Let the multiplicand registers size be 32 bits. Store the multiplier in the least significant half of the product register. Clear the most significant half of the product register.

Repeat the following steps for 32 times:

1. If the least significant bit of the product register is "1" then add the multiplicand to the most significant half of the product register.
2. Shift the content of the product register one bit to the right (ignore the shifted-out bit.)
3. Shift-in the carry bit into the most significant bit of the product register. Figure 5.Shows a block diagram for such a multiplier.

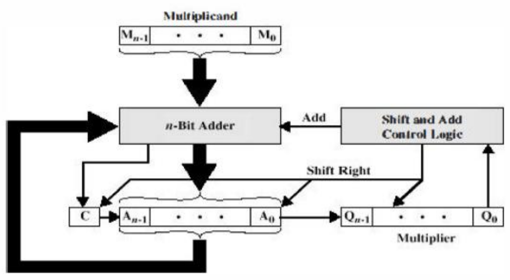
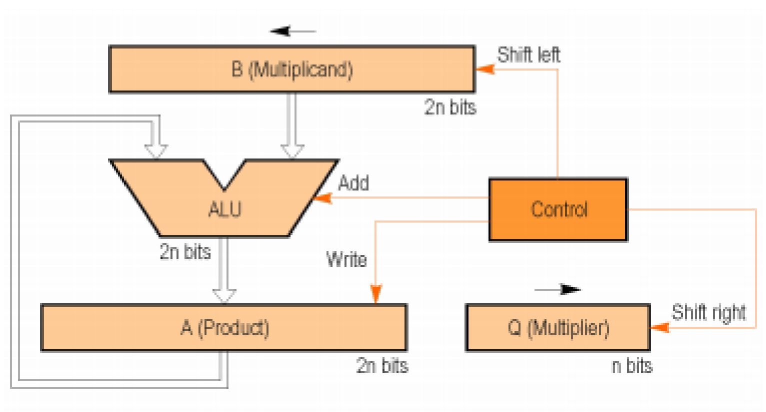


Figure 2. Multiplier of two n-bit values

Shift-and-add multiplication is similar to the multiplication performed by paper and pencil. This method adds the multiplicand X to itself Y times, where Y denotes the multiplier. To multiply two numbers by paper and pencil, the algorithm is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results.

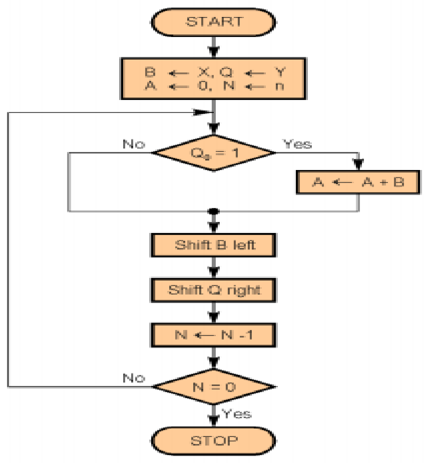
In the case of binary multiplication, since the digits are 0 and 1, each step of the multiplication is simple. If the multiplier digit is 1, a copy of the multiplicand (1 × multiplicand) is placed in the proper positions; if the multiplier digit is 0, a number of 0 digits (0 × multiplicand) are placed in the proper positions.

Consider the multiplication of positive numbers. The first version of the multiplier circuit, which implements the shift-and-add multiplication method for two n-bit numbers, is shown in Figure



First version of the multiplier circuit.

The 2n-bit product register (A) is initialized to 0. Since the basic algorithm shifts the multiplicand register (B) left one position each step to align the multiplicand with the sum being accumulated in the product register, we use a 2n-bit multiplicand register with the multiplicand placed in the right half of the register and with 0 in the left half.



The first version of the multiplication algorithm.

Figure shows the basic steps needed for the multiplication. The algorithm starts by loading the multiplicand into the B register, loading the multiplier into the Q register, and initializing the A register to 0. The counter N is initialized to n. The least significant bit of the multiplier register (Q0) determines whether the multiplicand is added to the product register. The left shift of the multiplicand has the effect of shifting the intermediate products to the left, just as when multiplying by paper and pencil. The right shift of the multiplier prepares the next bit of the multiplier to examine in the following iteration.

IV. PROPOSED SYSTEM

FIR filter is design using shift and add multiplier and ripple carry adder. In Ripple Carry Adder(RCA), delay increases linearly with the bit length.It takes longer computation time.
The operation required for designing an FIR filter in direct implementation is called the Multiply and Accumulate operation (MAC). In this type of operation, the co-efficient of the filter is directly multiplied with the variable and then added to get the final result. The following expression explains the MAC operation:

$$y = \sum_{k=0}^{N-1} h_{n-k} x_k \quad (3)$$

I n case of 1 tap filter, the filter co-efficient h0 is directly multiplied with variable x0 and the result is assigned to the output. In a 4-tap filter, the filter co-efficient are multiplied with corresponding variables, the result of 4 multipliers are added and assigned to the result.

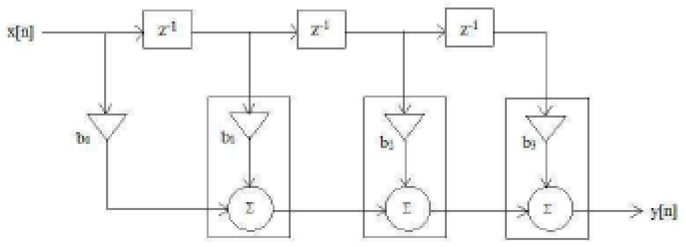


Fig 1. 4-tap FIR Filter (Direct form)

Fig. 1 shows the data flow graph diagram for a 4-tap FIR filter. When the number of taps increases the longest path delay also increases. Also, number of multipliers and adders required in the architecture also increases. The data flow graph shown in Fig. 1 is in direct form

The design of the Multiplier/Result module was performed at the RTL level. A block diagram of this module is shown in Figure 3-7 and the register assignment details.The input to the module is the multiplier, B_in, which is loaded into bits 0 to 7 of the register when the LOAD_cmd is asserted. The adder block outputs (Cout and adder_out) are also inputs to the module. The outputs of the module are, LSB, bit 0 of the register, RB, bits 8 to 15 of the register, and RC, bits 0 to 15 of the register. LSB is fed back to the controller to determine the next state, while RB is fed into the adder in order to be summed with the multiplicand. RC is the final multiplication result and is considered valid only when the controller asserts the STOP signal. If the Multiplier_Result receives a SHIFT_cmd without a prior ADD_cmd, the register will be shifted logically to the right. If the ADD_cmd precedes the SHIFT_cmd, bits 1 to 7 of the register will be placed into positions 0 to 6 while the 9 inputs bits from the adder will be placed into positions 7 to 15 of the register. This is essentially equivalent to storing the adder results and then shifting the entire register.

The main function of the adder block is to sum two bytes together. The main building block of the adders described in the following paragraph is the Full Adder. This block is designed structurally and accepts 2 inputs while generating 1 sum output and 1 carry output. The 8-bit ripple carry adder is composed of 8 individual full adders connected in a chain. In this case, the carry out of each full adder is the carry in of the following full adder. The limiting speed factor in this approach is the delay from the first full adder to the outputs of the final full adder.

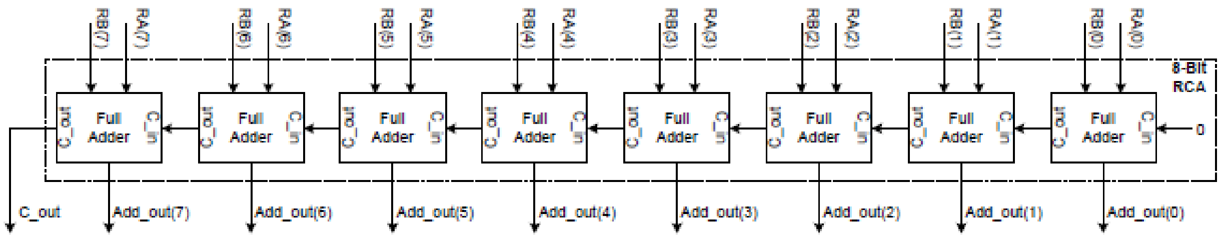


Figure Assistant Professor[1], [234]UG scholar

FIR filter is design using shift and add multiplier and parallel prefix adder..

The operation required for designing an FIR filter in direct implementation is called the Multiply and Accumulate operation (MAC). In this type of operation, the co-efficient of the filter is directly multiplied with the variable and then added to get the final result. The following expression explains the MAC operation:

$$y = \sum_{k=0}^{N-1} h_{n-k} x_k \quad (3)$$

In case of 1 tap filter, the filter co-efficient h0 is directly multiplied with variable x0 and the result is assigned to the output. In a 4-tap filter, the filter co-efficient are multiplied with corresponding variables, the result of 4 multipliers are added and assigned to the result.
Fig. 1 shows the data flow graph diagram for a 4-tap FIR filter. When the number of taps increases the longest path delay also increases. Also, number of multipliers and adders required in the architecture also increases. The data flow graph shown in Fig. 1 is in direct formThe design of the Multiplier/Result module was performed at the RTL level. A block diagram of this module is shown in Figure 3-7 and the register assignment details. The input to the module is the multiplier, B_in, which is loaded into bits 0 to 7 of the register when the LOAD_cmd is asserted. The adder block outputs (Cout and adder_out) are also inputs to the module. The outputs of the module are, LSB, bit 0 of the register, RB, bits 8 to 15 of the register, and RC, bits 0 to 15 of the register. LSB is fed back to the controller to determine the next state, while RB is fed into the adder in order to be summed with the multiplicand. RC is the final multiplication result and is considered valid only when the controller asserts the STOP signal. If the Multiplier_Result receives a SHIFT_cmd without a prior ADD_cmd, the register will be shifted logically to the right. If the ADD_cmd precedes the SHIFT_cmd, bits 1 to 7 of the register will be placed into positions 0 to 6 while the 9 inputs bits from the adder will be placed into positions 7 to 15 of the register. This is essentially equivalent to storing the adder results and then shifting the entire register.

4.1 PARALLEL PREFIX ADDERS

Parallel prefix adder (PPA) is a multi-bit carry-propagate adder which is used for parallel addition of two multi-bit numbers. PPA extend the generated and propagated logic of the carry look-ahead adder to perform addition even faster . As the basic schematic structure of the various PPA, perspective architecture is analyzed, it consists of three stages : pre-processing stage, prefix computation stage and final processing stage. Let consider each stage in more detail.

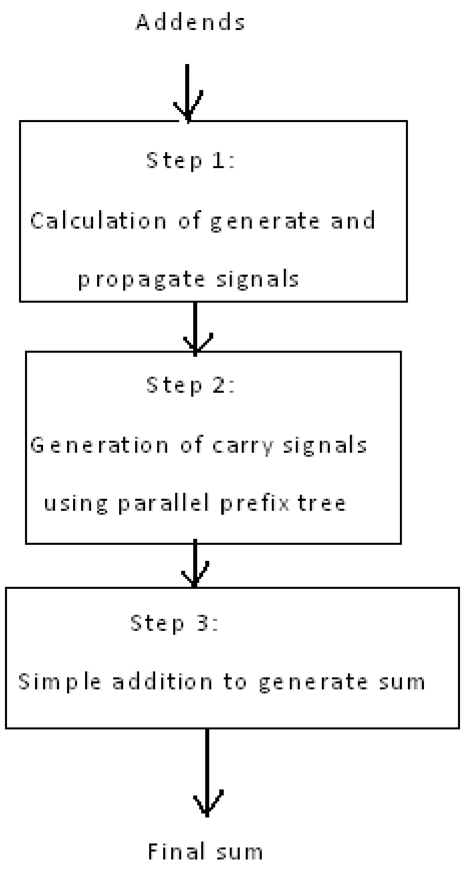

Addends
Step 1:
Calculation of generate and propagate signals
Step 2:
Generation of carry signals using parallel prefix tree
Step 3:
Simple addition to generate sum
Final sum

Fig 4.1. Addition procedure using Parallel Prefix tree structures

In every bit (i) of the two operand block, the two input signals (ai and bi) are added to the corresponding carry-in signal (carryi) to produce sum output (sumi) The equation to produce the sum output is:

Sumi = ai ^ bi ^ carryi (1)

Computation of the carry-in signals at every bit is the most critical and time – consuming operation. In the carry- look ahead scheme of adders (CLA), the focus is to design the carry-in signals for an individual bit additions. This is achieved by generating two signals, the generate (gi) and propagate (pi) using the equations:

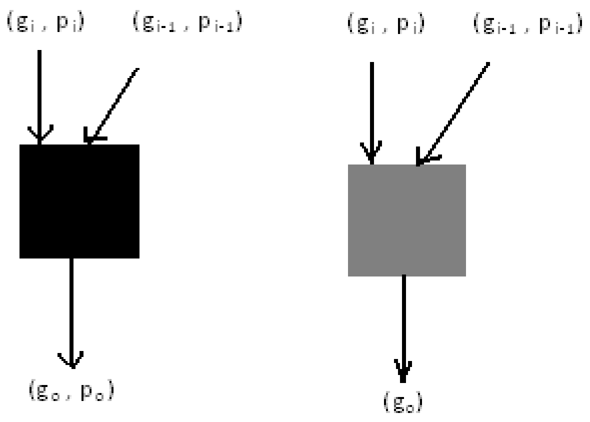
Gi = ai ^ bi (2)

Pi = ai ^ bi (3)

The carry in signal for any adder block is calculated by using the formula

Ci+1 = gi V (pi) (4)

Where ci must be expanded to calculate ci+1 at any level of addition
Parallel Prefix adders compute carry-in at each level of addition by combining generate and propagate signals in a different manner. Two operators namely *black* and *gray* are used in parallel prefix trees are shown in fig 2(a), fig 2(b) respectively.



(a) black operator (b) gray operator

Fig 4.2 Operators used in Parallel Prefix trees

The black operator receives two sets of generate and propagate signals (gi , pi),(gi-1 ,pi-1), computes one set of generate and propagate signals (go , po) by the following equations:
The gray operator receives two sets of generate and propagate signals (gi, pi),(gi-1 ,pi-1), computes only one generate signal with the same equation as in equation (5).

It is readily apparent that a key advantage of the tree-structured adder is that the critical path due to the carry delay is on the order of *log 2N* for an N-bit wide adder. The arrangement of the prefix network gives rise to various families of adders. For a discussion of the various carry-tree structures, see [1, 3]. For this study, the focus is on the Kogge-Stone adder , known for having minimal logic depth and fanout. Here we designate BC as the black cell which generates the ordered pair in equation (1); the gray cell (GC) generates the left signal only, following . The interconnect area is known to be high, but for an FPGA with large routing overhead to begin with, this is not as important as in a VLSI implementation.

V. SIMULATION RESULTS

Simulation Result of Multiplier:

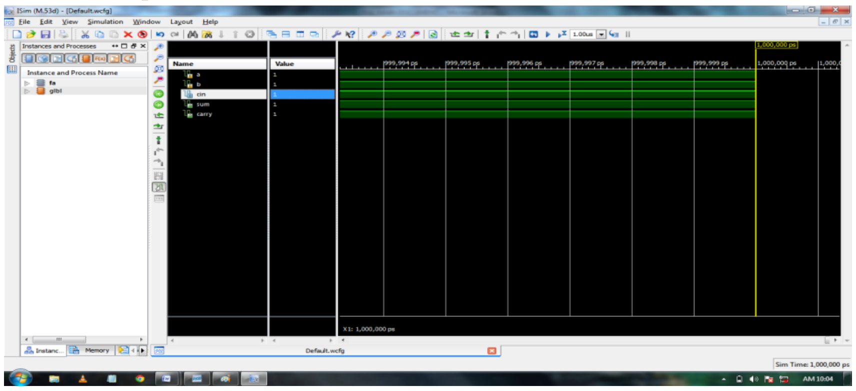


Fig 8.1 Simulation Result of Multiplier

Here we can give the inputs as A=6, B=3, clk as 1 and rst as 0, then the final output is 16.

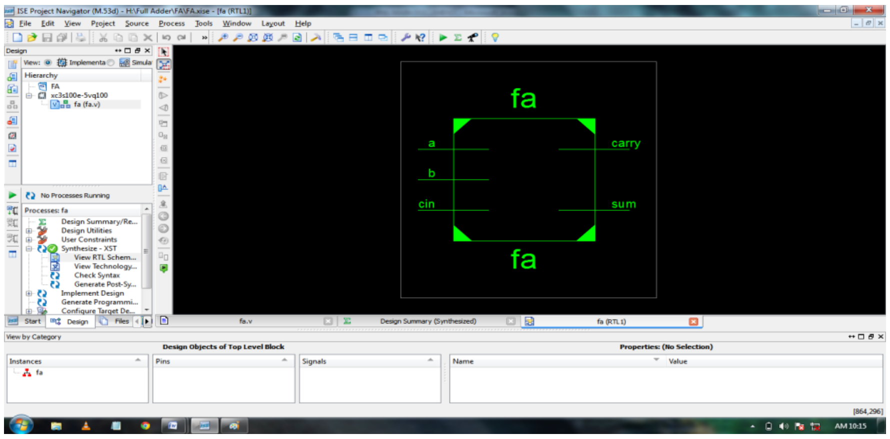
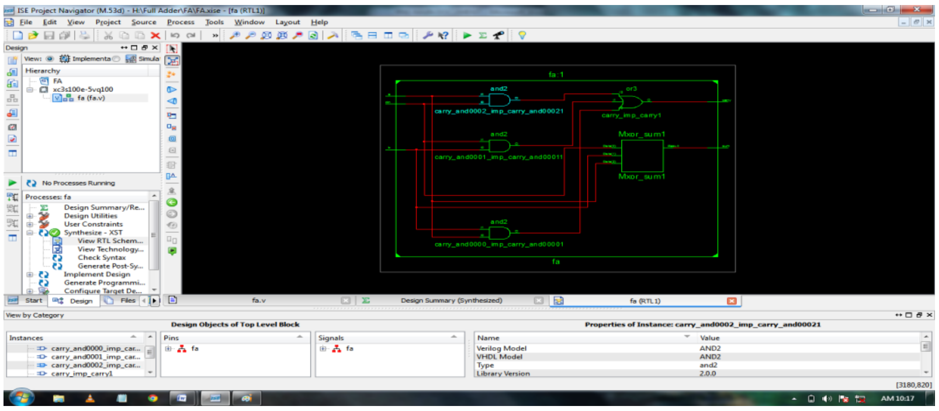
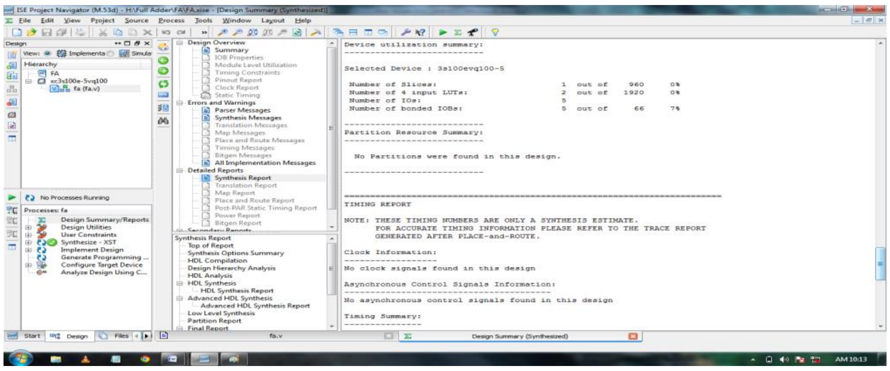


Fig 8.2 RTL Schematic of Multiplier


fa:1
and2
or3
carry_and0002_imp_carry_and00021
carry_imp_carry1
and2
Mxor_sum1
carry_and0001_imp_carry_and00011
Mxor_sum1
and2
carry_and0000_imp_carry_and00001
fa

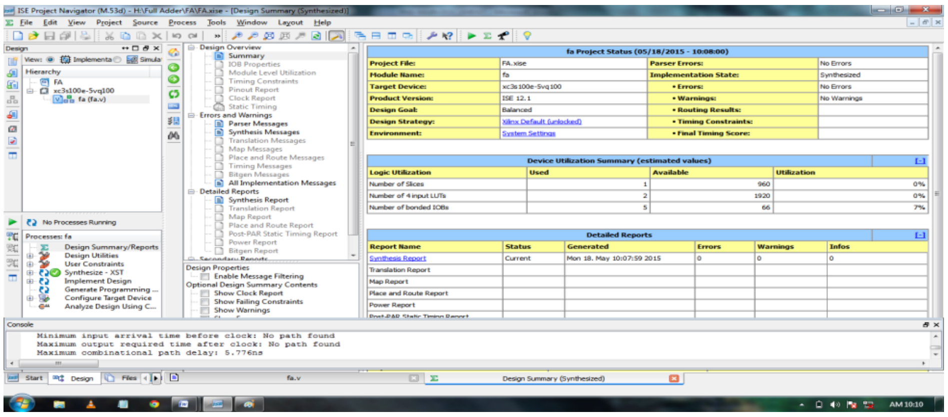
Name	Value
Verilog Model	AND2
VHDL Model	AND2
Type	and2
Library Version	2.0.0

Fig 8.3 Internal diagram of RTL schematic



```
Device utilization summary:
---------------------------

Selected Device : 3s100evq100-5

 Number of Slices:                       1  out of    960     0%
 Number of 4 input LUTs:                 2  out of   1920     0%
 Number of IOs:                          5
 Number of bonded IOBs:                  5  out of     66     7%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
No clock signals found in this design

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
```

Fig 8.5 Estimation of Delay



fa Project Status (05/18/2015 - 10:08:00)			
Project File:	FA.xise	Parser Errors:	No Errors
Module Name:	fa	Implementation State:	Synthesized
Target Device:	xc3s100e-5vq100	• Errors:	No Errors
Product Version:	ISE 12.1	• Warnings:	No Warnings
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	1	960	0%
Number of 4 input LUTs	2	1920	0%
Number of bonded IOBs	5	66	7%

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Mon 18. May 10:07:59 2015	0	0	0
Translation Report					
Map Report					
Place and Route Report					
Power Report					
Post-PAR Static Timing Report					

```
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 5.776ns
```

Fig 8.6 Estimation of Area

Synthesis Report:

Table 8.1 Comparison between power and delay

SYSTEM	POWER(w)	Delay(ns)
Existing(Wallace tree multiplier)	0.00583	110.386
Proposed method	0.00190	38.386

VI. CONCLUSION

The FIR filter was designed and result shown in Xilinx ISE 14.7. The result of the FIR filter using KSA is compared with the FIR filter using sklansky adder. After comparison we came to know that FIR filter using sklansky adder architecture is faster compared to FIR filter using KSA.

VII .REFERENCES

[1] Haichen Zhao, Shaolu Hu, Linhua Li, Xiaobo Wan. "NLMS Adaptive FIR Filter Design Method".

[2] B. Ramkumar and Harish M Kittur, "Low-Power and Area Efficient Carry Select Adder", IEEE Transsactions on Very Large Scale Integration (VLSI) Systems, VOL. 20, No. 2 Feb 2012.

[3] Deepak Kumar Patel, Raksha Chouksey, Dr. Minal Saxena "Design of Fast FIR Filter Using Compressor and Carry Select Adder", 2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN).

[4] A.Abinaya , M.Maheswari. "Implementation of Kogge Stone Adder for Signal Processing Applications", International Journal of Advanced Research in Computer and Communication Engineering Vol. 8, Issue 5, May 2019.

[5] S. Madhavi, K. Rasagna, N. Kavya, M. Sindhu. "Implementation of programmable fir filter using dadda multiplier and parallel prefix adder",IEEE Xplore Compliant Part Number:CFP18N67-ART; ISBN:978-1-5386-2456-2.

[6] V. Jamuna, P. Gomathi and A. Arun, "Design and Implementation of FIR Filter Architecture using High Level Transformation Techniques" Indian Journal of Science and Technology.

[7] M.Moghaddam, M. B. Ghaznavi-Ghoushchi. "A New Low-Power, Lowarea, Parallel Prefix Sklansky Adder with Reduced Inter-Stage Connections Complexity".

[8] Aung Myo San, Alexey N. Yakunin "Reducing the Hardware Complexity of a parallel prefix adder".