International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 8, Issue 9 (September 2013), PP. 28-35

FPGA implementation of High Order FIR Filter Using Distributed Arithmetic operation

M.Lakshmanna[1], N.Praveenkumar, M.Tech(Ph.D)[2], G.Harikumar[3]
[1](*ECE Department, Stanley Stephen College of Engineering ,Kurnool, Indian).*
[2](*ECE Department,Assistant Professor of Stanley Stephen College of Engineering, Kurnool, Indian.*

ABSTRACT
The implementation of FIR filters on FPGA based on traditional method costs considerable hardware recourses, which goes against the decrease of circuit scale and the increase of system speed. A new design and implementation of FIR filters using Distributed Arithmetic is provided in this paper to slove this problem. Distributed Arithmetic structure is used to increase the resource usage while pipeline structure is also used to increase the system speed. In addition, the divided LUT method is also used to decrease the required memory units. The simulation results indicate that FIR filters using Distributed Arithmetic can work stable with high speed and can save almost 50 percent hardware recourses to decrease the circuit scale, and can be applied to a variety of areas for its great flexibility and high reliability.

Keywords:- Distributed Arithmetic; Fir; Pipeline; Lut; Fpga .

I. INTRODUCTION

In the recent years, there has been a growing trend to implement digital signal processing functions in Field Programmable Gate Array (FPGA). In this sense, we need to put great effort in designing efficient architectures for digital signal processing functions such as FIR filters, which are widely used in video and audio signal processing, telecommunications and etc. Traditionally, direct implementation of a K-tap FIR filter requires K multiply-and-accumulate (MAC) blocks, which are expensive to implement in FPGA due to logic complexity and resource usage. To resolve this issue, we First present DA, which is a multiplier-less architecture.

Implementing multipliers using the logic fabric of the FPGA is costly due to logic complexity and area usage, especially when the filter size is large. Modern FPGAs have dedicated DSP blocks that alleviate this problem, however for very large filter sizes the challenge of reducing area and complexity still remains. An alternative to computing the multiplication is to decompose the MAC operations into a series of lookup table (LUT) accesses and summations. This approach is termed distributed arithmetic (DA), a bit serial method of computing the inner product of two vectors with a fixed number of cycles. The original DA architecture stores all the possible binary combinations of the coefficients w[k] of equation (1) in a memory or lookup table. It is evident that for large values of L, the size of the memory containing the pre computed terms grows exponentially too large to be practical. The memory size can be reduced by dividing the single large memory (2Lwords) into m multiple smaller sized memories each of size 2k where $L = m \times k$. The memory size can be further reduced to 2L−1 and 2L−2 by applying offset binary coding and exploiting resultant symmetries found in the contents of the memories.

This technique is based on using 2's complement binary representation of data, and the data can be pre-computed and stored in LUT. As DA is a very efficient solution especially suited for LUT-based FPGA architectures, many researchers put great effort in using DA to implement FIR filters in FPGA. Patrick Longa introduced the structure of the FIR filter using DA algorithm and the functions of each part. Sangyun Hwang analyzed the power consumption of the filter using DA algorithm. Heejong Yoo proposed a modified DA architecture that gradually replaces LUT requirements with multiplexer/adder pairs. But the main problem of DA is that the requirement of LUT capacity increases exponentially with the order of the filter, given that DA implementations need 2Kwords (K is the number of taps of the filter).

This method not only reduces the LUT size, but also modifies the structure of the filter to achieve high speed performance. The proposed filter has been designed and synthesized with ISE 9.1, and implemented with a 4VLX40FF668 FPGA device.Our results show that the proposed DA architecture can implement FIR filters with high speed and smaller resource usage in comparison to the previous DA architecture.

II. FIR FILTER DESIGN

II(I)Finite Impulse Response Filter

In signal processing, there are many instances in which an input signal to a system contains extra unnecessary content or additional noise which can degrade the quality of the desired portion. In such cases we may remove or filter out the useless samples. For example, in the case of the telephone system, there is no reason to transmit very high frequencies since most speech falls within the band of 400 to 3,400 Hz. Therefore, in this case, all frequencies above and below that band are filtered out. The frequency band between 400 and 3,400 Hz, which isn't filtered out, is known as the pass band, and the frequency band that is blocked out is known as the stop band. FIR, Finite Impulse Response, filters are one of the primary types of filters used in Digital Signal Processing. FIR filters are said to be finite because they do not have any feedback. Therefore, if you send an impulse through the system (a single spike) then the output will invariably become zero as soon as the impulse runs through the filter.

A finite impulse response (FIR) filter is a filter structure that can be used to implement almost any sort of frequency response digitally. An FIR filter is usually implemented by using a series of delays, multipliers, and adders to create the filter's output. Figure1 below shows the basic block diagram for an FIR filter of length N.The delays result in operating on prior input samples. The hk values are the coefficients used for multiplication, so that the output at time n is the summation of all the delayed samples multiplied by the appropriate coefficients.
Where:

x[n] is the input signal, y[n] is the output signal, bi are the filter coefficients, and N is the filter order – an Nth-order filter has (N + 1) terms on the right-hand side; these are commonly referred to as taps.

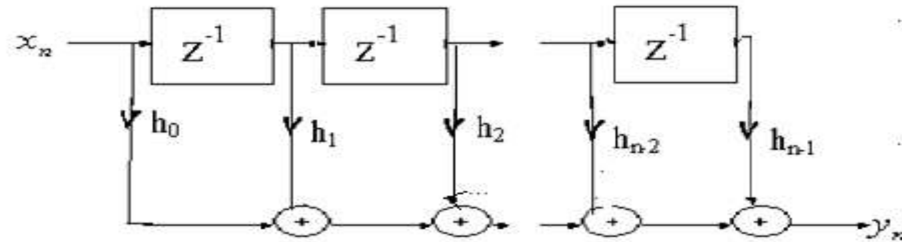


Fig1. The logical structure of an FIR filter

This equation can also be expressed as a convolution of the coefficient sequence b_i with the input signal:

$$y = \sum_{k=0}^{k-1} x[n-k]\ h_k$$

That is, the filter output is a weighted sum of the current and a finite number of previous values of the input. The process of selecting the filter's length and coefficients is called filter design. The goal is to set those parameters such that certain desired stop band and pass band parameters will result from running the filter. Most engineers utilize a program such as MATLAB to do their filter design. But whatever tool is used, the results of the design effort should be the same.A frequency response plot, verifies that the filter meets the desired specifications, including ripple and transition bandwidth, the filter's length and coefficients. The longer the filter (more taps), the more finely the response can be tuned.

III. DISTRIBUTED ARITHMETIC FIR FILTER USING FPGA ARCHITECTURES

III(i)Background

Traditional implementations of the finite impulse response (FIR) filter equation is

$$y = \sum_{k=0}^{k-1} x[n-k]\ h_k$$

Typically employ L multiply-accumulate (MAC) units. Implementing multipliers using the logic fabric of the FPGA is costly due to logic complexity and area usage, especially when the filter size is large. Modern FPGAs have dedicated DSP blocks that alleviate this problem, however for very large filter sizes the challenge of reducing area and complexity still remains. An alternative to computing the multiplication is to decompose the MAC operations into a series of lookup table (LUT) accesses and summations. This approach is termed distributed arithmetic (DA).

III(ii) Distributed Arithmetic FIR filter architecture

Distributed Arithmetic is one of the most well-known methods of implementing FIR filters. The DA solves the computation of the inner product equation when the coefficients are pre knowledge, as happens in FIR filters.

An FIR filter of length K is described as:

$$y = \sum_{k=0}^{k-1} x[n-k]\ h_k \quad \text{..............................(1)}$$

Where h[k] is the filter coefficient and x[k] is the input data. For the convenience of analysis, x[k] =x [n - k] is used for modifying the equation (1) and we have:

$$y = \sum_{k=0}^{k-1} x[k]\ h_k \quad \text{.................................(2)}$$

In this equation, the h_k are the fixed coefficients, K is the number of filter taps and x_k are the input data words. These ones have a standard fixed-point format number, which is a two's-complement fractional representation with x_k limited in the range $-1 \leq x_k \geq 1$.

$$X_K = -X_{K0} + \sum_{n=1}^{N-1} X_{Kn}\, 2^{-n} \quad \text{...................................(3)}$$

where N is the bit number of the data, being x_{k0} the sign bit and x_{kn} the bit "n" x_k of. Equation (1) can be rewritten in this way:

$$Y = \sum_{k=0}^{k-1} h_k \left(-X_{K0} + \sum_{n=1}^{N-1} X_{Kn}\, 2^{-n}\right)$$

$$Y = -\sum_{k=0}^{k-1} h_k X_{K0} + \sum_{n=1}^{N-1} X_{Kn}\, h_k 2^{-n}$$

Using registers, memory resources and a scaling accumulator does the implementation of digital filters using this arithmetic. Original LUT-based DA implementation of a 4-tap (K=4) FIR filter is shown in Figure1. The DA architecture includes three units: the shift register unit, the DA-LUT unit, and the adder/shifter unit.

IV. SYSTEM BLOCK DIAGRAM

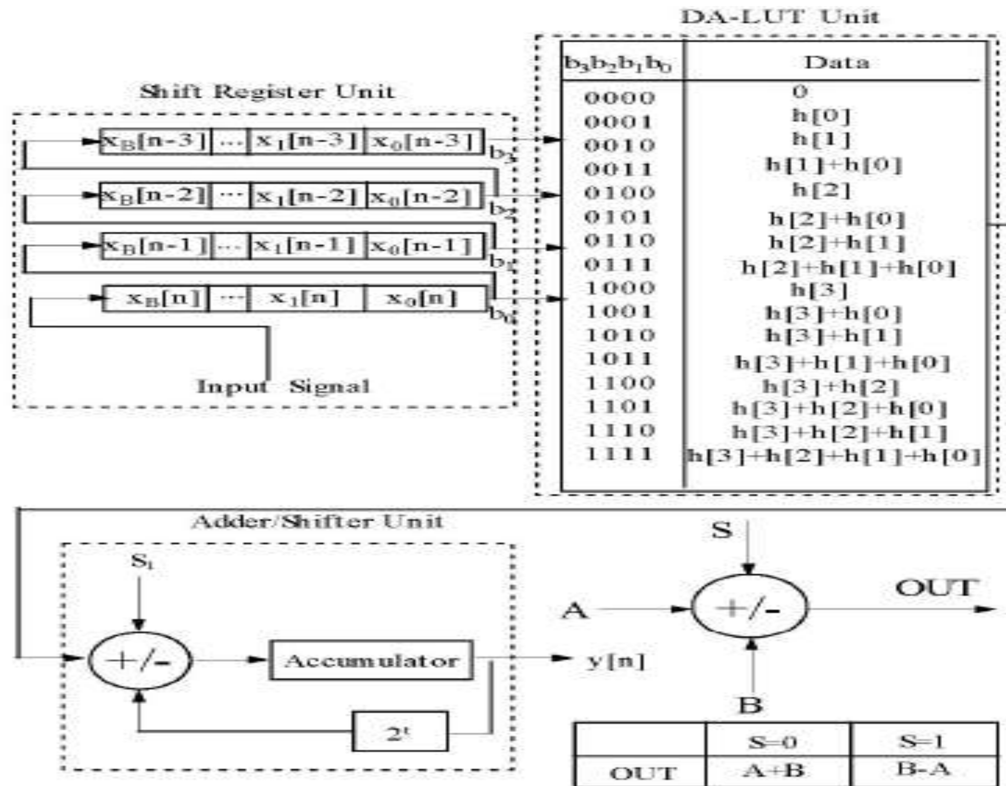


Fig2: Original LUT-based DA implementation of a 4-tap FIR filter

IV (i) Shift register

In digital circuits, a shift register is a cascade of flip flops, sharing the same clock, which has the output of anyone but the last flip-flop connected to the "data" input of the next one in the chain, resulting in a circuit that shifts by one position the one-dimensional "bit array" stored in it, shifting in the data present at its input and shifting out the last bit in the array. Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states. All flip-flops is driven by a common clock, and all are set or reset Simultaneously.

IV (ii) Storage Capacity

The storage capacity of a register is the total number of bits (1 or 0) of digital data it can retain. Each stage (flip flop) in a shift register represents one bit of storage capacity. Therefore the number of stages in a register determines its storage capacity. The serial in/serial out shift register accepts data serially – that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

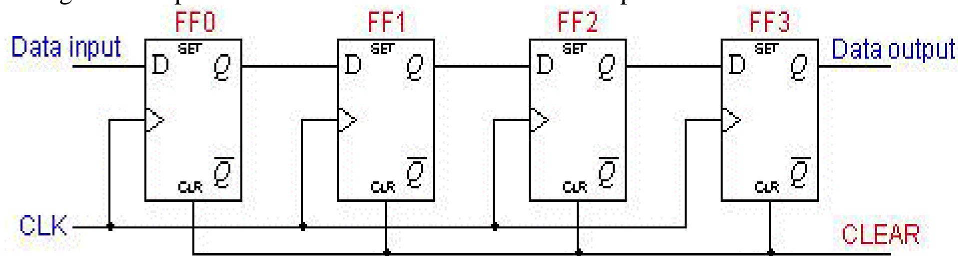


Fig 3. Four-bit shift register

A basic four-bit shift register can be constructed using four D flip-flops, as shown in Figure3. The operation of the circuit is as follows.

1. The register is First cleared, forcing all four outputs to zero.
2. The input data is then applied sequentially to the D input of the

    b. First flip-flop on the left (FF0).

3.During each clock pulse, one bit is transmitted from left to right.

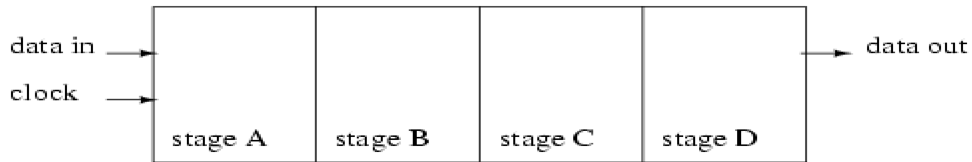


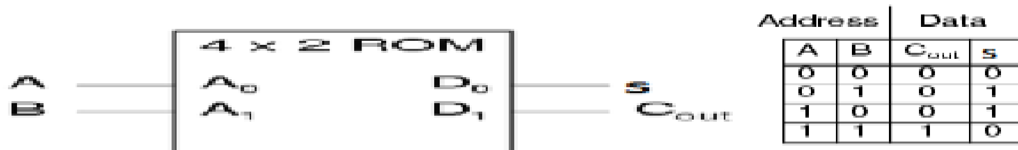
Fig 4. Serial in, Serial out Shift register with 4-stages

Above we show a block diagram of a serial-in/serial-out shift register, which is 4-stages long. Data at the input will be delayed by four clock periods from the input to the output of the shift register. Data at "data in", above, will be present at the Stage **A** output after the First clock pulse. After the second pulse stage **A** data is transfered to stage **B** output, and "data in" is transfered to stage **A** output. After the third clock, stage **C** is replaced by stage **B**; stage **B** is replaced by stage **A**; and stage A is replaced by "data in". After the fourth clock, the data originally present at "data in" is at stage **D**, "output". The "First in" data is "First out" as it is shifted from "data in" to "data out". For a K-Tap FIR filter the present input and past K-1 inputs must be available. For the FIR filter the input has been given serially bit wise. In the shift register block there are N-shift register which holds the K-inputs needs by FIR filter.

For every clock one bit of Next input will be reaching shift register module. So to accommodate the new bit x[n] shift register must be shifted right. So for every 'B' Clock cycles where as 'b is the no. of bits in each input sample, now input will be stored x[n] and old x[n] will be moved to x[n-1], x[n-1] to x[n-2]--- like that.

IV (iii) Look up table

We know that it is possible to store binary data within solid-state devices. Those storage "cells" within solid-state memory devices are easily addressed by driving the "address" lines of the device with the proper binary value(s). Suppose we had a ROM memory circuit written, or programmed, with certain data, such that the address lines of the ROM served as inputs and the data lines of the ROM served as outputs, generating the characteristic response of a particular logic function. Theoretically, we could program this ROM chip to emulate whatever logic function we wanted without having to alter any wire connections or gates. Consider the following example of a 4 x 2 bit ROM memory (a very small memory!) programmed with the functionality of a half adder



Address		Data	
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fig5. Logic Diagram of 4X2 ROM **Table 1.** Truth table of Half Adder

If this ROM has been written with the above data (representing a half-adder's truth table), driving the A and B address inputs will cause the respective memory cells in the ROM chip to be enabled, thus outputting the corresponding data as the Σ (Sum) and C_{out} bits. Unlike the half-adder circuit built of gates or relays, this device can be set up to perform any logic function at all with two inputs and two outputs, not just the half-adder function. To change the logic function, all we would need to do is write a different table of data to another ROM chip. We could even use an EPROM chip which could be re-written at will, giving the ultimate flexibility in function.

It is vitally important to recognize the significance of this principle as applied to digital circuitry. Whereas the half-adder built from gates or relays processes the input bits to arrive at a specific output, the ROM simply remembers what the outputs should be for any given combination of inputs. This is not much different from the "times tables" memorized in grade school: rather than having to calculate the product of 5 times 6 (5 + 5 + 5 + 5 + 5 + 5 = 30), school-children are taught to remember that 5 x 6 = 30, and then expected to recall this product from memory as needed. Likewise, rather than the logic function depending on the functional arrangement of hard-wired gates or relays (hardware), it depends solely on the data written into the memory (software).

Such a simple application, with definite outputs for every input, is called a look-up table, because the memory device simply "looks up" what the output(s) should to be for any given combination of inputs states.

This application of a memory device to perform logical functions is significant for several reasons:

1. Software is much easier to change than hardware.
2. Software can be archived on various kinds of memory media (disk, tape), thus providing an easy way to document and manipulate the function in a "virtual" form; hardware can only be "archived" abstractly in the form of some kind of graphical drawing.

3. Software can be copied from one memory device (such as the EPROM chip) to another, allowing the ability for one device to "learn" its function from another device.
4. Software such as the logic function example can be designed to perform functions that would be extremely difficult to emulate with discrete logic gates (or relays!).

The usefulness of a look-up table becomes more and more evident with increasing complexity of function. Suppose we wanted to build a 4-bit adder circuit using a ROM. We'd require a ROM with 8 address lines (two 4-bit numbers to be added together), plus 4 data lines (for the signed output):

In our design we are using look table to store all the different possible combination summation of filter coefficients. This look -up-table will implemented by ROM design. In the reset state we will be store the coefficient summations. Input for this LUT will be coming from the output of shift register module. For every clock cycle the LSB bits of all 'N' input samples are applied to the LUT.LUT will consider this input as on address and give the data stored in that particular address to the output. The main advantage of LUT implementation is we can avoid the multiplications.

IV(iv) Adder/Subtractor

In digital circuits, an Adder-Sub tractor is a circuit that is capable of adding or subtracting numbers (in particular, binary). Below is a circuit that does adding or subtracting depending on a control signal. However, it is possible to construct a circuit that performs both addition and subtraction at the same time.To get the FIR filter response we need to add the LUT outputs. The present LUT output will be added to the shifted (1 bit right) version of previous LUT output. Finally at B th clock cycle ('B' is the no. of bits in each sample) we have to perform subtraction. So to perform addition or subtraction we implemented adder/subtracted block which performs any me of them depending on the input select signal.

IV (v) Right Shifter

In digital circuits, a shift register is a cascade of flip flops, sharing the same clock, which has the output of anyone but the last flip-flop connected to the "data" input of the next one in the chain, resulting in a circuit that shifts by one position the one-dimensional "bit array" stored in it, shifting in the data present at its input and shifting out the last bit in the array.Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. In the right shift, 1-bit is shifted to right direction. To realize x/2 term in the FIR filter response we need to divide by 2. But to avoid multiplications we released it through a 1-bit right shifty Circuit.

V. MODULATION SIMULATION RESULTS

Matlab and Modelsime are used as the simulation platforms. We can analysis the changes between the input wave and the output wave to observe the permanence of the designed filter through Matlab , while observing the real-time implementation performance of FPGA through Modelsim. To observe the performance of the designed filter, a square signal of 1.6MHz is generated as the input by Matlab, and the sampling frequency is 32MHz. The data is changed into the two's complement form and stored in the file named fir_in.txt. And the data is called out later as the input of the FPGA filter through test bench language in Modelsim, the output data is shown in the waveform workstation and also stored into the file named fir_out.txt, which can be read by Matlab to make a contrast to analysis. In Fig.4, the waveforms are in frequency domain. They are the frequency spectrums of input signal, filter and output signal respectively. We can observe that the output wave only contains low frequency signal after the implementation of the filter. We can conclude that the filter coefficients are suitable for the test

The waveforms are in time domain. They are the input signal, output signal, and the filted signal by FPGA filter respectively. We can observe that output filted by FPGA filter is almost the same as the output simulated by Matlab, the permanence of the designed filter is perfect. The test results above show that the filter works stable, and completely meets the designed requirements. It cost 2221 logic elements and obtain 220M system speed using traditional direct arithmetic, while only cost 1110 logic elements and obtains 230M system speed using Distributed Arithmetic. Therefore, we can save almost 50 percentage of the hardware resources using Distributed Arithmetic to decrease the hardware scale. And we can obtain a higher speed if we sacrifice the hardware resources for speed using other relevant technologies

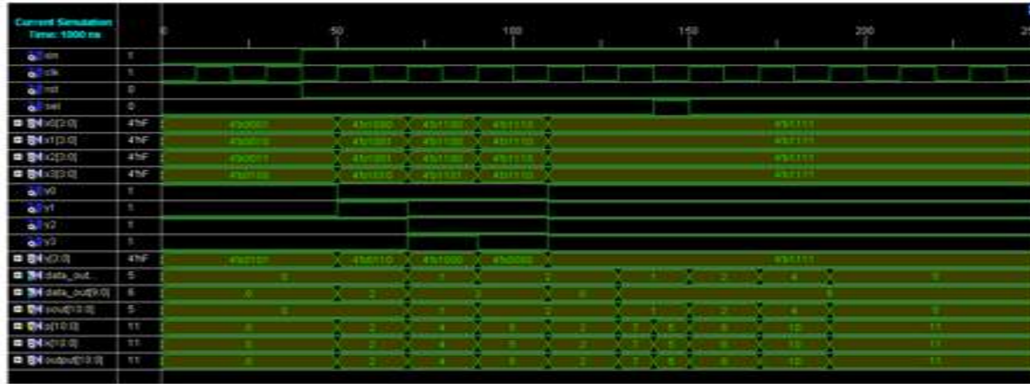


Fig.6 Modulation simulation results DA FIR FILTER

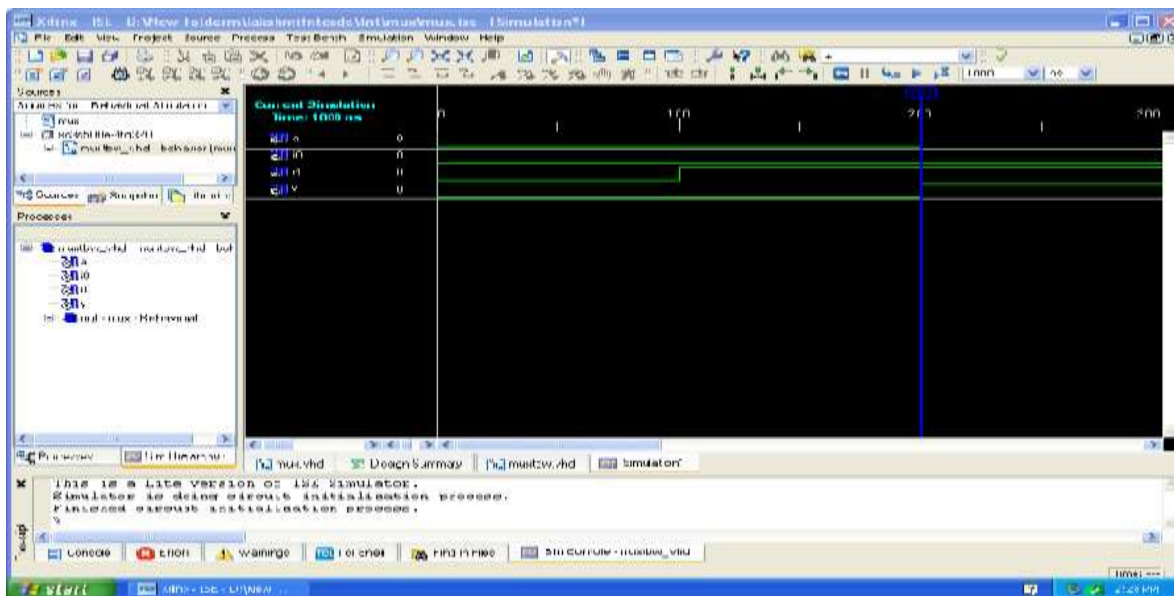


Fig.7 Synthesis Behavioral Simulation results DA FIR FILTER

VI. ADVANTAGE AND DISADVANTAGES

- High frequency resolution and accuracy.
- Fast switching between frequencies over a large bandwidth.
- Compared to LUT method, it takes less area.
- FPGA hardware can give high speed solution in comparison with software approach but not in comparison with ASIC.

VII. TOOLS AND HARD WARE

- Simulation software -Modelsim Xilinx Edition (MXE)
- Synthesis, P&R - Xilinx ISE
- On chip verification - Xilinx Chip scope
- Hardware– Xilinx Spartan 3 Family FPGA board

VIII. FPGA DESIGN AND PROGRAMMING

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL form is more suited to work with large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualization of a design.Once the design and validation process is complete, the binary file generated (also using the FPGA company's proprietary software) is used to (re)configure the FPGA.

XI. CONCLUSION

This paper presents the design and implementation based on Distributed Arithmetic, which is used to

realize a 31-order FIR low-pass filter. Distributed Arithmetic structure is used to increase the recourse usage while pipeline structure is used to increase the system speed. The test results indicate that the designed filter using Distributed Arithmetic can work stable with high speed and can save almost 50 percent hardware recourses. Meanwhile, it is very easy to transplanted the filter to other applications through modifying the order parameter or bit width and other parameters, and therefore have great practical applications in digit signal processing.

REFERENCES

[1]. Use Meyer-Baese.Digitalsignal processing with FPGA[M].Beijing:Tsinghua University Press,2006:50~51.
[2]. Tsao Y C and Choi K. Area-Efficient Parallel FIR Digital Filter Structures for Symmetric Convolutions Based on Fast FIR Algorithm [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2010,PP(99):1~5.
[3]. Chao Cheng and Keshab K Parhi. Low-Cost Parallel FIR Filter Structures With 2-Stage Parallelism[J].IEEE Transactions on Circuits and Systems I: Regular ,2007,54(2):280~290.
[4]. Tearney G J and Bouma B E. Real-Time FPGA Processing for High-Speed Optical Frequency Domain Imaging [J]. IEEE Transactions on Medical Imaging, 2009,28(9):1468~1472.
[5]. DIGITAL SIGNAL PROCESSING Principles,Algorithms,and Applications by John G.Proakis
[6]. DIGITAL SIGNAL PROCESSING by RameshBabu
[7]. Meher P K , Chandrasekaran S and Amira A. FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic [J]. IEEE Transactions on Signal Processing, 2008,56(7): 3009~3017.