A Proficient Design of Hybrid Synchronous and Asynchronous Digital FIR Filter using FPGA

Paulchamybalaiah[1] and Dr.Ila Vennila[2]

[1]Assistant Professor
Department Of ECE
Hindusthan Institute of Technology
Coimbatore-32

[2]Assistant Professor
Department Of EEE
PSG College of Technology
Coimbatore-32

Abstract

In this paper, a hybrid synchronous and asynchronous digital FIR filter is designed and implemented in FPGA using VHDL. The digital FIR filter of high throughput, low latency operating at above 1.3 GHz was designed. An adaptive high capacity pipelined was introduced in the hybrid synchronous asynchronous design of the filter. The degree of the pipelining is dynamically variable depending upon the input. Concurrent execution of software or program can be achieved in FPGA through parallel processing. The designed digital FIR filter is simulated using ModelSim and implemented using Xilinx. The simulation results are presented for different order such as 3, 6 and 15. The FIR filter designed is synthesized in Xilinx 9.1i and the device utilization report is presented for filter of order 3, 6 and 15.

Keywords: FPGA, Asynchronous pipeline, dynamic logic, FIR Filter

1. Introduction

Basically the filters are designed by using finite number of samples of impulse response which is termed as finite impulse response filters. It is a non- recursive, discrete-time filter. The output depends only on present and previous inputs. It is to remove unwanted parts of the signal such as random noise and also to extract useful parts of the signals such as the components lying within a certain frequency range [1], [2]. FIR filters are inherently stable due to the fact that all the poles are located at the origin and thus are located within the unit circle. FIR filters require no feedback means that any rounding errors are compounded by some iteration. They can be designed to be linear phase by making the coefficient sequence symmetric , linear phase or phase change proportional to frequency , corresponds to equal delay at all frequencies.

In signal processing, the function of a filter is to measure unwanted parts of the signal such as random noise and to extract useful parts of the signal, such as the components lying within a certain frequency range [3]. Digital filter uses a digital processor to perform numerical calculation on sampled value of the signal. The processor may be a general purpose computer such as PC or a specialized DSP (digital signal processor) chip.

The types of the filter are as follows:

- Low pass filter: They leave to pass the low frequencies.
- High pass filter: They leave to pass the high frequencies and they strongly attenuate the low ones.
- Band pass filter: They leave to pass the mean frequencies and they attenuate the high

Ones and the low ones.

2. Overview of Digital Implementation of FIR

2.1 Digital Implementation of FIR Filter using DSP

Distributed Arithmetic has been used to implement a bit-serial scheme of a general asymmetric version of an FIR filter, taking optimal advantage of the 4-input LUT-based structure of FPGAs and a highly area-efficient multiplier-less FIR filter is designed. To implement DSP functions in Field FPGAs, which offer a balanced solution in comparison with traditional devices? Although ASICs and DSP chips have been the traditional solution for high performance applications, now the technology and the market are imposing new rules. On one hand, high development costs and time-to-market factors associated with ASICs can be prohibitive for certain applications and, on the other hand, programmable DSP processors can be unable to reach a desired performance due to their sequential-execution architecture. The research Community has put great effort in designing efficient architectures for

DSP functions such as FIR filters, which are extensively used in multiple applications in telecommunications, wireless or satellite communications, video and audio processing, biomedical signal processing and many others. Traditionally, the design methods were mainly focused in multiplier-based architectures to implement the multiply-and- Accumulate (MAC) blocks that constitute the central piece in FIR filters and several DSP functions: But careful analysis shows that multiplier-based filter implementations may become highly expensive [2], [4].

2.2 Digital Implementation of FIR Filter using FPGA

FPGAs offer a very attractive solution that balance high flexibility, time-to-market, cost and performance. This issue has been partially solved with the new generation of low- cost FPGAs that have embedded DSP blocks. However, if the final product will reside on an ASIC for instance, the problem is still present. To resolve this issue, several multipliers-less schemes were proposed. Basically, these methods can be classified in two categories according to how they manipulate the filter coefficients for the multiply operation. The first type of multiplier-less technique is the conversion-based approach, in which the coefficients are transformed to other numeric representations whose hardware implementation or manipulation is more efficient than the traditional binary representation. Example of such techniques is the Canonic Sign Digit method, in which coefficients are represented by a combination of powers of two in such a way that multiplication can be simply implemented with adder/subtractions and shifters, and the Dempster-Mcleod method, which similarly involves the representation of filter coefficients with powers of two but in this case arranging partial results in cascade to introduce further savings in the usage of adders. The second type of multiplier-less method involves the use of memories (RAMs, ROMs) or LUTs to store pre-computed values of coefficient operations. These are called memory-based methods. Examples of them are found in the Constant Coefficient Multiplier method and the very-well known DA method. DA appeared as a very efficient solution especially suited for LUT-based FPGA architectures. This technique is a multiplier-less architecture that is based on an efficient partition of the function in partial terms using 2's complement binary representation of data. The partial terms can be pre-computed and stored in LUTs. The flexibility of this algorithm on FPGAs permits everything from bit-serial implementations to pipelined or full-parallel versions of the scheme, which can greatly improve the design performance. The main problem with DA is that the requirement of memory/LUT capacity increases exponentially with the order of the filter, given that DA implementations need $2K$ - words (K being the number of taps of the filter). That constitutes a first obstacle for FIR filters of high order. A flexible architecture that gradually replaces LUT requirements with multiplexer/adder pairs was introduced. An asymmetric FIR filter architecture using the bit-serial LUT-based DA technique is presented. For this implementation, we use a scheme that takes advantage of the 4-input LUTs in FPGAs, and rearranges the input sequence to implement a modified version of the shifter/accumulator stage. We show that our modified version is superior in terms of area to previous LUT-less DA architectures [5], [7].

3. Problem Formulation

In the existing method, fixed order filter is used. The filter is a ten-tap six bit FIR filter Partial sums are pre-computed and stored in a LUT, indexed by the input data values. The signed-digit offset binary notation is used in which the symbols “0” and “1” stand for negative and positive co-efficient of powers of 2.The Figure1.Shows that Existing Fixed Mode Filter.

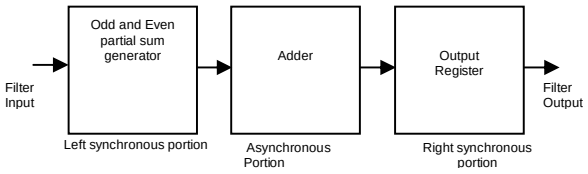

Fig. 1 Existing fixed tap Mixed Mode Filter

4. Proposed Methodology

In this method, variable order filter is proposed. Fig.2.Shows the Programmable tap fixed mode Filter. We can change the filter order to any number if purpose the 3^{rd} order, 5^{th} order, 15^{th} order can be designed. Concurrent execution of software or program can be achieved in FPGA through adaptive high capacities pipelined which performed parallel processing. With this methodology we aim to design a digital FIR filter operating at above 1.3GHZ.

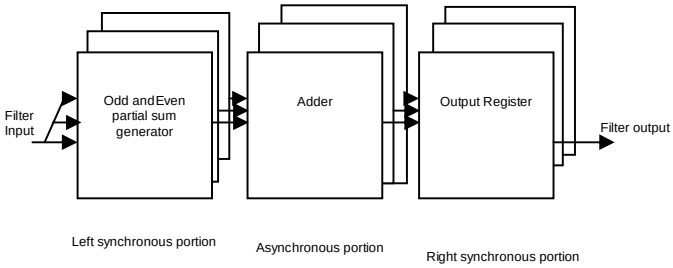

Fig. 2 Proposed Programmable tap fixed mode Filter

5. Design Concepts of Digital FIR Filter

Fir filter are commonly designed in DSP and FPGA platforms. Therefore, the basic design concepts using DSP and FPGA are discussed below.

5.1 Design Concepts of Digital FIR Filter using DSP

A design method for FIR digital filter based on DSP processor with fixed point series in which the coefficient of filter is obtained and verified with the DSP measuring system. The digital filter's all functionalities met design expectations. Filtering plays a significant role in digital signal processing. Digital filtering is a basic calculation method for language and graphics treatment, mode recognition, and spectrum analysis. This method has many advantages over an analogue filter, such as broad design amplitude, precision guarantee, and accurate linear phase position; and prevention of voltage shifting, temperature migration, and noise. Since its response to unit impulse is in limited long sequence, FIR filter is always stable. In addition to those advantages, digital filtering using DSP chip is flexible, convenient to change the filter's parameters, and easy to modify its specificity. The methodologies for high-level synthesis of dedicated DSP architectures using the COMET design system is in use. The system is tuned to the synthesis of DSP ASICs from behavioral specifications written in VHDL. COMET is capable of generating more efficient architectures using innovative scheduling and resource allocation algorithms which exploit the cluster information and maximize the parallel tasks. With these transformations, major improvements are achieved with fewer registers and interconnections; an industrial quality design is then derived in both FIR and elliptic filter examples. Filter banks are often used in signal and image processing applications for dividing a signal into frequency bands and reconstructing the signal from the individual bands. Quadrature Mirror Filter is one particular application using the sub-band coding technique, have not able advantages for image compression / restoration compared with the Discrete Cosine Transform. Silicon compilation has become essential to automate the VLSI design of DSP system as chips increase in size and complexity. High-level synthesis, an important front end task from an algorithmic behavioral specification, has received a lot of attention in both the academic and Industrial environments. Generally, the input description is converted into a Data Flow Graph and all synthesis tasks work from this Data Flow Graph. Behavioral synthesis is a complex task composed primarily of two interacting subtasks: scheduling and allocation. A great deal of progress has been made on the theory of high-level synthesis and promising results [2, 4].

5.2 Design Concepts of Digital FIR Filter using FPGA

Implementation of the filter requires considerably less resources than the previousdesign using DSP. This requires about half the resources in terms of configurable blocks, lookup tables. The saving in the adder chain is not so high, since most of the adder tree size is dictated by the coefficients size, not by the samples size. The lookup tables must be writable. This increases its complexity, especially in terms of routing resources. The mixer multiplier must be implemented using hard multipliers, not lookup tables. A single large lookup table to hold sine/cosine values is still needed. Especially for Altera FPGAs, this is a large advantage, as these chips have smaller RAM blocks, but also one or two large RAMs. Re-tuning the band is relatively slow [5, 11]. The filter has no capability for frequency hopping. This is not a requirement, and tap reloading is in any case faster than for a full1024 tap filter. Some intelligence is needed in the control processor to recalculate filter taps from the low pass prototype, but this is within the capabilities of any current microprocessor. We use ModelSim Tool to determine filter coefficients, and designed a 16-orderconstant coefficient FIR filter by VHDL language [3, 9], simulate filters, the results meet performance requirements. As the word indicates, a filter separates a desired signal from unwanted disturbances. When we want to remove a disturbance such as noise from an audio signal, we design an appropriate filter that passes only the desires signal. But only in a few cases can we remove the disturbance completely and recover the desired signal; most of the time we have to settle for a compromise, most of the disturbance is rejected, most of the signal is recovered. The first candidate in filter is a linear filter. The main reason for this choice is that we have a good understanding of how a linear system operates. It is only when a linear design fails or it yields unsatisfactory results that we look for other solutions, such as nonlinear or, adaptive techniques, for example. Digital filters include infinite impulse response (UR) digital filter and finite impulse response (FIR) digital filter. As the FIR system have a lot of good features, such as only zeros, the system stability, operation speed quickly, linear phase characteristics and design flexibility, so that FIR has been widely used in the digital audio, image processing, data transmission, biomedical and other areas. FIR filter has a variety of ways to achieve, with the processing of modem electronic technology, taking use of field programmable gate array FPGA for digital signal processing technology has made rapid development, FPGA with high integration, high speed and reliability advantages, FIR filter implementation using FPGA is becoming a trend. The algorithm is proposed for the design of low complexity

linear phase finite impulse response (FIR) filters with optimum discrete coefficients. The proposed algorithm, based on mixed integer linear programming, efficiently traverses the discrete coefficient solutions and searches for the optimum one that results in an implementation using minimum number of adders. During the searching process, discrete coefficients are dynamically synthesized based on a continuously updated sub expression space and, most essentially, a monitoring mechanism is introduced to enable the algorithm's awareness of optimality. Benchmark examples have shown that the proposed algorithm can, in most cases, produce the optimum designs using minimum number of adders for the given specifications. The proposed algorithm can be simply extended for the optimum design with the maximum adder depth constraint. Linear phase finite impulse response (FIR) filters are widely used in digital signal applications such as speech coding, image processing, MultiMate systems, etc. Although the stability and linear phase is guaranteed, the complexity and power consumption of the linear phase FIR filter are usually much higher than that of the infinite impulse response (IIR) filter which meets the same magnitude response specifications. Therefore, many efforts have been dedicated to the design of low complexity and low-power linear phase FIR filters. A conventional filter structure, called transposed direct form, in which the input signal is first multiplied by the constant filter coefficients and then goes into the delay elements. This operation is often referred to as multiple constants multiplication problem. The constant multipliers can be realized using multiplier less techniques where the general multipliers are replaced by a network of shifts and adders. The adders can be further classified into structural adders and multiplier block adders. Structural Adders are used to add the temporarily stored values. An efficient semi definite programming method for the design of a class of linear phase finite impulse response filter banks whose filters have optimal frequency selectivity for a prescribed regularity order is proposed. The design problem is formulated as the minimization of the least square error subject to peak error constraints and regularity constraints. By using the linear matrix inequality characterization of the trigonometric semi-infinite constraints, it can then be exactly cast as a Semi definite programming problem with a small number of variables and, hence, can be solved efficiently. Finally, the image coding performance of the filter bank is presented. The filter has found important applications in image processing, speech processing, communications, and the construction of wavelet bases. The filter bank design is commonly formulated as a highly nonlinear optimization problem because of the perfect reconstruction condition. As a result, high complexity algorithms are required to obtain a good solution, and the globally optimal solution is not guaranteed. To reduce the computational complexity of the design, finding filter bank structures that structurally satisfy perfect reconstruction is of great interest. Lifting structures are very attractive for the construction and implementation of filter and wavelets because the perfect reconstruction property can be structurally imposed offers a filter bank with low implementation complexity. However, there are certain restrictions on the frequency responses.

6. Digital FIR Filter Architecture

The architecture of the FIR filter is shown in figure.3.The filter is a ten-tap six-bit FIR filter using the distributed arithmetic architecture. Six bit Slices, stacked on top of each other. It consists of three portions namely [3].

6.1 Left Synchronous Portion

Receives data from the environment and processes it into partial sums Asynchronous portion: Ads the partial sums to compute the final result.

6.2 Right Synchronous Portion

Right Synchronous Portion synchronizes the result to the clock and produces it as an output for the environment. Data inputs enter from the left, and are processed by the filter as they flow to the right. The filter can be divided into three portions, from Left to right. The leftmost portion is clocked, from the input side to the domino latches. The middle portion, from the XOR gates to the end of the carry look ahead adder, is asynchronous. Finally, the rightmost portion, consisting of an output latch, is again clocked. The architecture of the filter is best understood by following the flow of data from left to right [4]. As the stream of data enters the filter, it first passes through a shift register, which stores the most recent input values that are needed to compute the filter output. In particular, for a *p*-tap filter, for each bit, there is a *p*-place shift register that stores the most recent history for that bit. These stored input values are then multiplied by their respective filter weights. The multiplication is accomplished very efficiently by fetching precompiled results from a lookup table.

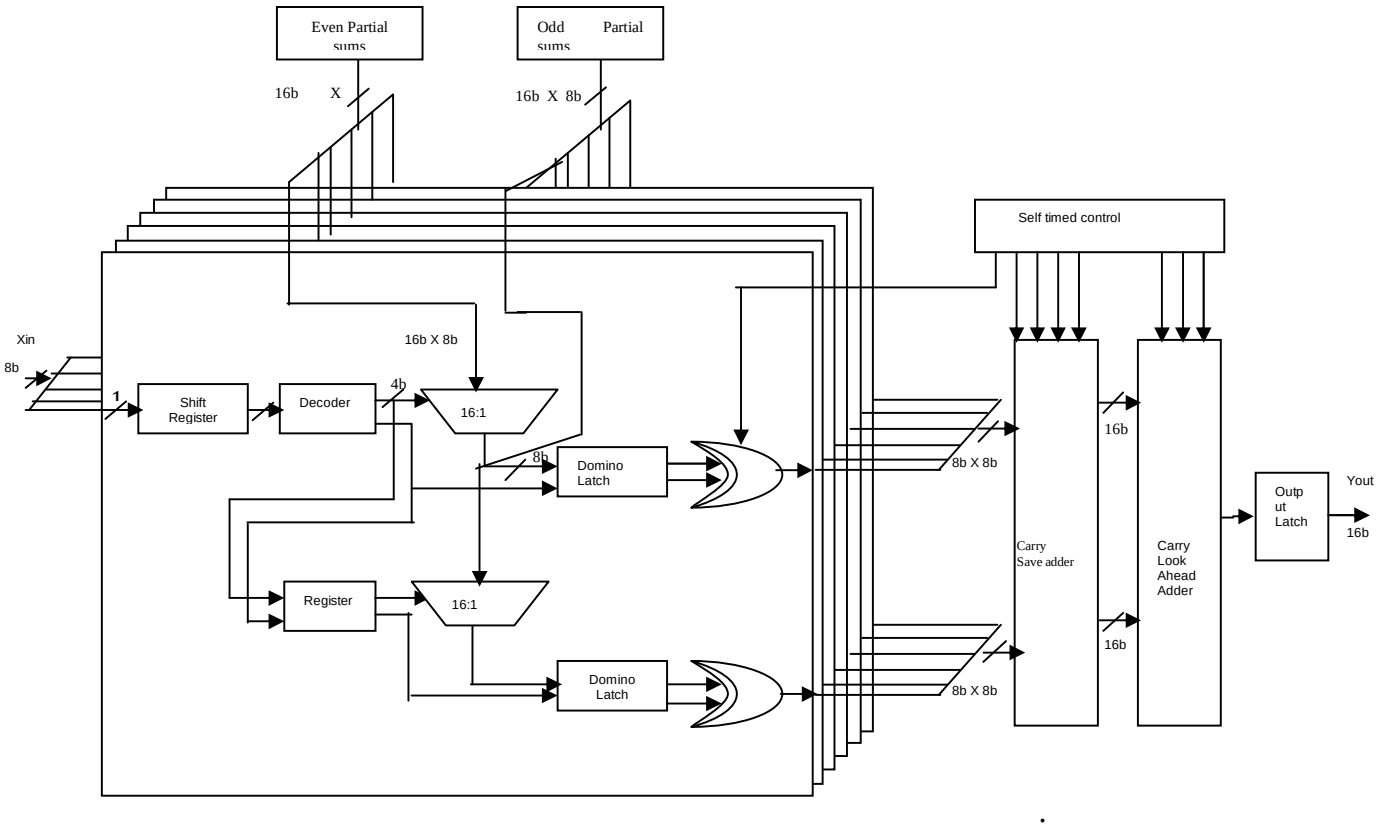

Fig. 3 FIR Filter Architecture

The entire multiplication process is bit-sliced, with one slice for each bit of the input data. The result of the multiplications is a set of partial sums which are fed to the asynchronous portion of the filter pipeline for addition [5]. In the figure1, the lookup table is composed of two banks of registers containing the precompiled result scaled even and odd partial sums and two output multiplexors.

6.3 Asynchronous Portion

It is a nine-stage pipeline that adds all of the partial sums together, and produces the result. Finally, this result is latched by a clocked latch and output to the right environment.

7. FILTER IMPLEMENTATION

The FIR filter implementation is now considered in more detail. The synchronous and asynchronous portions of the chip are discussed separately, followed by a discussion of the interface between the two domains [6].

7.1 Synchronous Portion

The synchronous portion of the filter consists of two parts, one at the input side of the filter, and the other at the output side.

7.2 Synchronous Input Portion

This part receives the input to the filter. The input stream consists of data values which are six bits wide [5]. A 10-slot shift register at the input side of the filter stores the 10 most recent data values. These stored input values are needed to compute the current filter output, which is a weighted sum of these values. The multiplication of inputs

by their respective filter weights is accomplished very efficiently by pre-computing all possible products and storing them into a lookup table. The entire multiplication is bit-sliced, with one slice for each of the six bits in the input data. Therefore, within each bit slice, there are 10 input bits which together forma 10-bit address for accessing the lookup table [3].
The size of the lookup table is reduced by employing two techniques is used: the 10-bit address is divided into two 5-bit addresses, one composed of only the even-index bits, and the other composed of the odd-index bits. Each of these two addresses has a distinct lookup table associated with it [6]. To understand the filter operation with a partitioned lookup table, consider a simulation of partial sum lookup. The 10-bit pattern (after passing through the decoder unit) is used to generate separate groups of even and odd-indexed bits. In particular [6], only the five even bits are used; they are forked to the even multiplexor as its select bits, and also to a clocked register where, after one clock cycle delay, they become the odd-index select bits to the bottom multiplexor, for the next clock cycle. Appropriate entries in the event and odd lookup tables are then selected and sent to the domino latches.

A signed-digit offset binary notation is used to represent table entries and addresses, which enable the separation of the sign-bit from each address, further shortening the addresses to 4-bit words.[4] As a result, the table size is dramatically reduced: two tables with only 16 (= 24) entries each are needed, as opposed to one table with 1024 (= 210) entries. The lookup tables are implemented using registers and multiplexors. Each table has 16 registers, each of which can store an 8-bit entry, per bit slice. Each of the tables has a 16:1 multiplexor at its output, controlled by the 4-bit address word.2 The odd-index address word is generated front he even-index address word by delaying it by one clock cycle[5]. The result of the multiplication is a set of products, called partial sums, that is sent to the asynchronous pipeline for addition, through the synchronous-asynchronous interface [2].

7.3 Synchronous Output Portion

The right synchronous portion simply consists of a master slave latch that receives the final result from the asynchronous pipeline and makes it available as the filter output.

7.4 Asynchronous Portion:

The asynchronous portion of the filter consists of a pipeline that lies between the synchronous input and output portions, the function of this asynchronous pipeline is to take the partial sums generated byte synchronous input portion, add them up to produce the final filter result, and send it to the synchronous output portion. The pipeline was designed using the high-capacity pipeline style the asynchronous data path uses dynamic logic, and consists of nine stages [1], [2]. The first stage is a layer of XOR gates that restores the correct sign to the partial sums. The next five stages correspond to five layers of carry save adders The last three stages implement a carry look ahead adder Since both true and complement values of the data bits are needed to compute the XOR and addition functions, the entire data path was implemented in dual-rail.

The data path is quite wide at the input to the first stage: 216 wires (= (8 data bits + 1 sign bit) (even and odd) ·6 (bit slices) ·2 (wires/bit)). The output of the last stage is a 15-bit result represented using 30 wires. Interestingly, since the filter has a very fine-grain data path, no explicit matched delays are required [6]. The delay of each function block is matched by the completion generator's A*C* element itself, through appropriate device sizing. The self-timed control of a high-capacity pipeline needs a slight modification to handle the wide data path of the filter. In particular, buffers must be inserted in order to amplify the control signals which are broadcast to the entire width of the data path [4]. Two different versions of the control were designed, one more robust and the other faster.
The two versions differ in the placement of the amplifying buffers. In the first version, the buffers amplify data path as well as the completion generator. This version is very robust to variations in buffer delays because the completion signals are delayed by the same amount as the data path [3], [4]. However, the buffers are on the critical path, thus increasing the pipeline cycle time. In the second version, the completion generators use control signals that are tapped off from before the buffers. As a result, the buffer delays are taken off of the critical path, resulting in a shorter cycle time. However, each stage's function block now lags behind its completion generator by an amount equal to the buffer delay. Consequently, for the pipeline to function correctly, all the stages throughout the pipeline are required to have comparable buffer delays

7.5 Synchronous-Asynchronous Interfaces

The interface between the asynchronous and the synchronous portions of the chip must mediate certain differences in data representation and control sequencing. In particular, the asynchronous data path uses dual-rail dynamic logic, whereas the synchronous portions of the chip use single-rail static logic [5], [6]. Moreover, the asynchronous pipeline communicates by means of local

handshakes (using *req*'s and *ack*'s) at each end, whereas the synchronous portion uses global clocking.

8. Internal Modules

The internal modules used for the design of the digital FIR filter are Adder, Multiplexer and Shift and Add Multiplication. The respective internal modules are explained in details as follows:

8.1Adder

An Adder is a digital circuit that performs addition of number in modern computer adders reside in the arithmetic logic unit where other operation are performed. Adder can be constructed for much numerical representation such as binary coded decimal. The most common adders operate on binary number.

8.2 Multiplexer

It is a device that performs multiplexing. It selects one of many analog or digital input signals and forwards the selected input to a single line. An electronic multiplexer makes it possible for several signal to share one device. The multiplexer units are used to select the appropriate output from the shift and add unit

8.3 Shift-and-Add Multiplication

Shift-and-add method adds the multiplicand *X* to itself *Y* times, where *Y* denotes the Multiplier. In the binary multiplication, 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 (1 × multiplicand) are placed in the proper positions

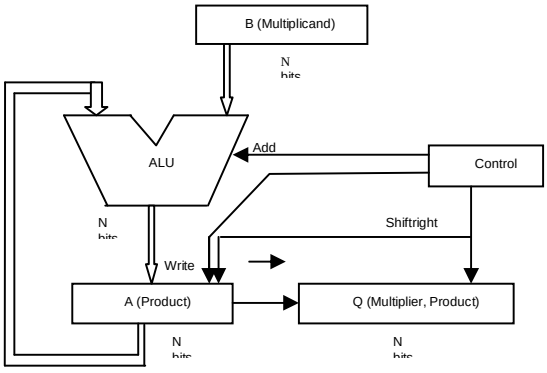

Fig. 4 Final Version of Shift and Add Multiplication Circuit.

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 the figure. 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-bitmultiplicand register with the multiplicand placed in the right half of the register and with 0in the left half.

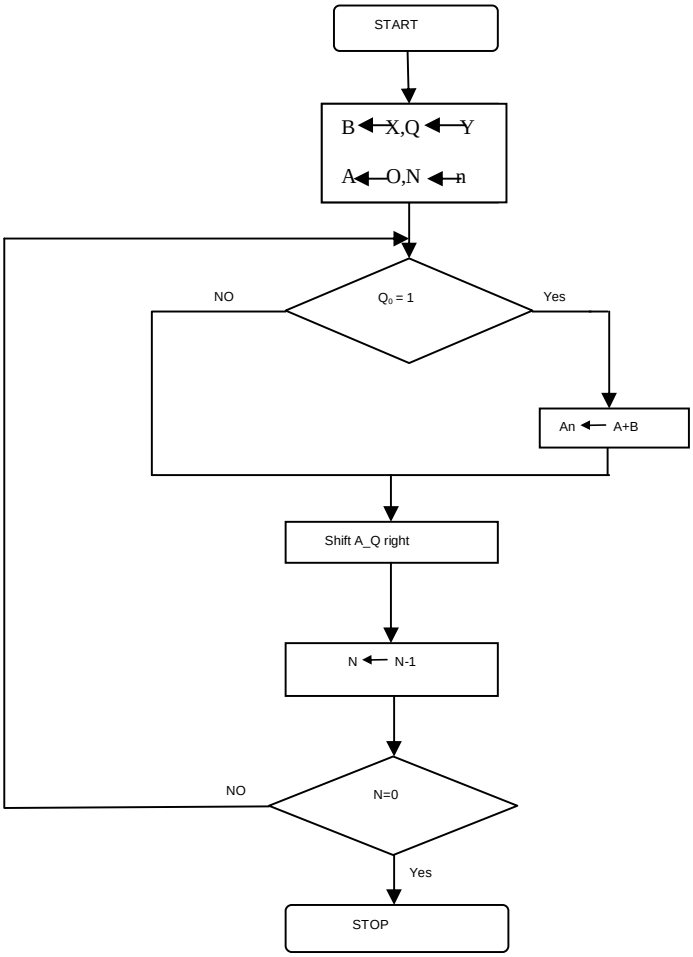

Fig. 5 Flowchart of the Final version algorithm of the Shift and Add Multiplication

The Fig.5. 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. The right shift of the multiplier prepares the next bit of the multiplier to examine in the next iteration.

9. FPGA Design flow

The flow chart of a typical FPGA design flow is shown in Fig.6. The design flow and FPGA design methodologies are reviewed by eminent researchers. The design specifications are written first to describe the functionality, interface and overall architecture of the digital circuit to be designed with short development time [5]. A behavioral description is then created to analyze the design in terms of functionality to meet the performance, compliance to standards and other high-level issues. Behavioral descriptions can be written with HDLs and it is converted to RTL description in an HDL. The designer has to describe the data flow to implement the desired digital circuit using any market standard simulator. The functionality of the intended application is verified in the functional verification and tested using different set of stimulus. The Logic synthesis tools convert the RTL description to a technology independent gate-level net list, which is a description of the circuit in terms of gates and connections between them. Behavioral synthesis tools have begun to emerge recently. These tools can create RTL descriptions from a behavioral or algorithmic description of the circuit..

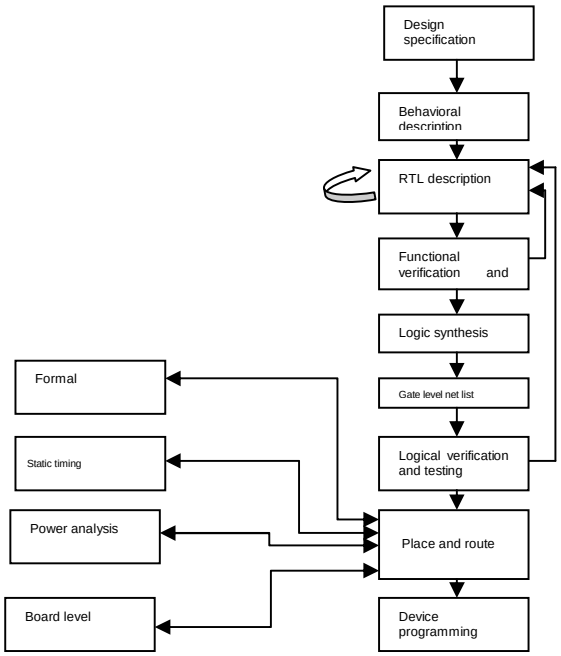

Fig. 6 Flow chart of FPGA Design Flow

Formal verification is widely used to verify traditional standard cell ASIC designs. The term timing analysis is used to refer to two methods called Static Timing Analyses (STA) and the timing simulation. By running formal verification in conjunction with STA, it is confirmed that the post-route net list is the same as the RTL design in functionality. STA is one of the techniques available to verify the timing of a digital design. The STA is static since the analysis of the design is carried out statistically and does not depend upon the data values being applied at the input pins. An alternate approach used to verify the timing is the timing simulation which can verify the functionality as well as the timing of the design where a stimulus is applied on input signals, resulting behavior is observed and verified, then time is advanced with new input stimulus applied, and the behavior is observed and verified and so on. Thus, timing analysis simply refers to the analysis of the design for timing issues. In power analysis, power consumption of the implemented digital circuit is calculated to satisfy the power requirement specifications. After meeting all the specifications, the PROM file is generated to download into FPGA/CPLD using JTAG Cable. The Signal Integrity (SI) stage addresses two concerns in the design aspects; they are timing and the quality of the signal. The goal of signal integrity analysis is to ensure reliable high-speed data transmission. In a digital system, a signal is transmitted from one component to another in the form of logic '1' or '0', which is actually at certain reference voltage levels. The receiving component needs to sample the data in order to obtain the binary coded information. Any delay of the data or distortion of the data will result in a failure of the data transmission. The SI check plays an important role in high speed FPGA design in order to satisfy the quality of the signal at the far end of the board route, as well as the propagation delay. Major FPGA vendors provide ISE comprising simulator, synthesizer and implementation tools and third party support is provided for simulation, synthesis, power analysis depending on the design requirement.

10. RESULTS & DISCUSSIONS

In this paper the internal modules such as adder, multiplexer and shift and add multiplication is used as the basic modules. First, the basic modules are simulated and the results are presented. Then designed digital FIR filter is simulated for different orders and frequencies and the results are presented.

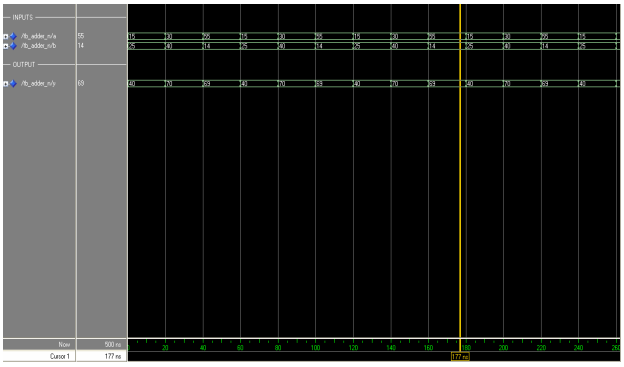

Fig. 7 Simulation Result of Adder

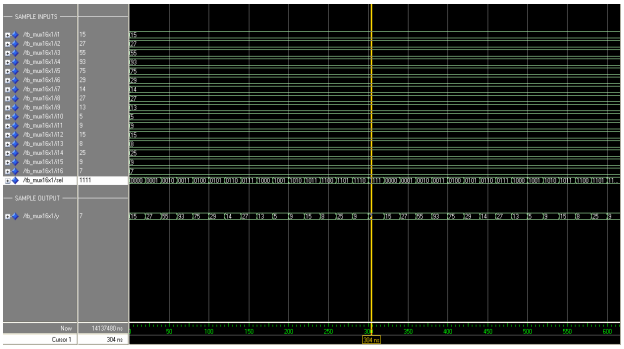

Fig. 8 Simulation Result of Multiplexer

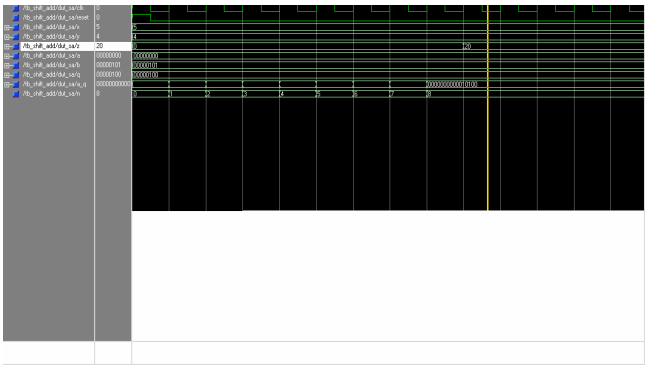

Fig. 9 Simulation Result of Shift and Add multiplication

10.1Simulation results of digital FIR filter with different orders

The simulated results of digital FIR filter of different orders such as order 3, order 6 and order 15 are presented below with their order, cut off frequencies and sampling frequencies respectively. As the order of the filter increases, the performance of the filter increases. The magnitude of the output signal decreases when the frequency of the input signal increases. The response of the band stop filter is clearly visible when the order of the filter increases

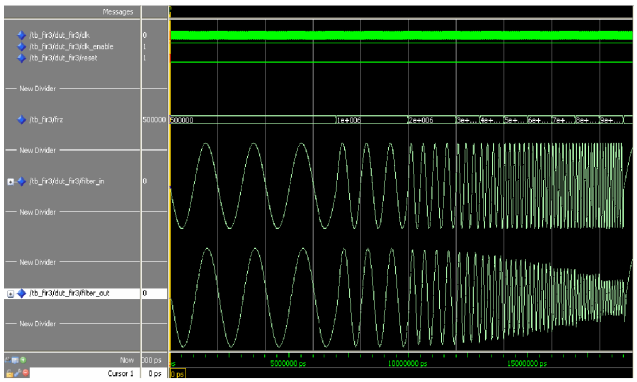

Fig. 10 FIR Filter output of Order3, cut off frequency 4MHZ, sampling rate 50MHZ

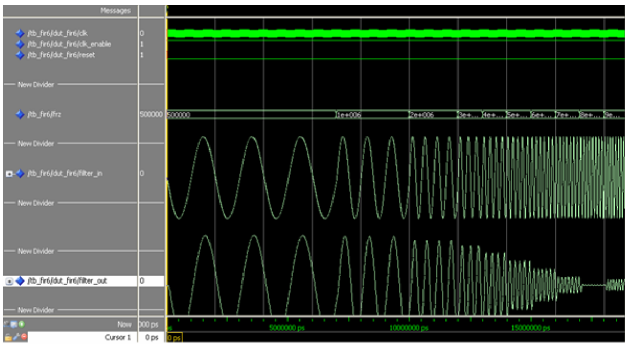

Fig 11 FIR Filter output of order 6, cutoff frequency 4MHz, sampling rate 50MHz

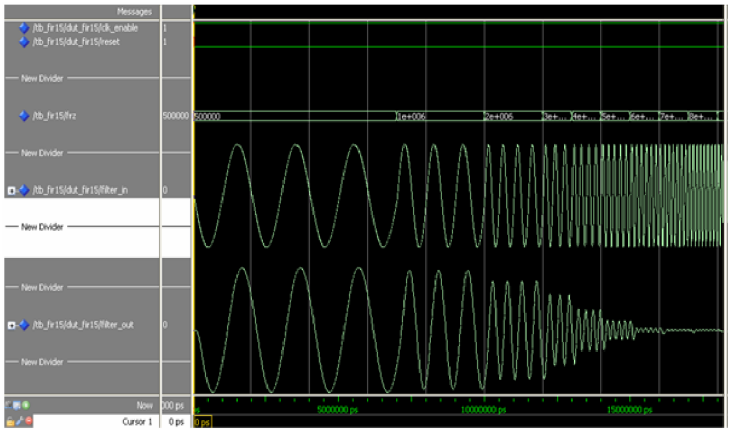

Fig. 12 FIR Filter output of order 15, cutoff frequency 4MHz, sampling rate 50MHz

10.2 Synthesis Report

The FIR filter designed is synthesized using Xilinx 9.1i and the device utilization report is presented for the order 3, 6 and 15 in Table 1 to Table 3 respectively.

Table 1: Synthesis report of FIR filter of order 3

Logic utilization	Used	Available	Utilization
Number of Slices	64	3584	1%
Number of Slice Flip Flops	84	7168	1%
Number of 4 input LUTs	67	7168	0%
Number of bonded IOBs	27	141	19%
Number of GCLKs	1	8	12%

Table 2: Synthesis report of FIR filter of order 6

Logic Utilization	Used	Available	Utilization
Number of Slices	113	3584	3%
Number of Slice Flip Flops	121	7168	1%
Number of 4 input LUTs	125	7168	1%
Number of bonded IOBs	29	141	20%

Table 3: Synthesis report of FIR filter of order 15

Logic Utilization	Used	Available	Utilization
Number of Slices	219	3584	6%
Number of Slice Flip Flops	198	7168	2%
Number of 4 input LUTs	365	7168	5%
Number of 4 input LUTs	30	141	21%
Number of GCLKs	1	8	12%

11. Conclusion

A hybrid synchronous and asynchronous digital FIR filter has been designed and Implemented in FPGA using VHDL. The digital FIR filter of high throughput, low latency operating at above 1.3 GHz has been designed. An adaptive high capacity pipelined was introduced in the hybrid synchronous asynchronous design of the filter. The degree of the pipelining is dynamically variable depending upon the input. Concurrent execution of software or program can be achieved in FPGA through parallel processing. The designed digital FIR filter is simulated using ModelSim and synthesized using Xilinx. The simulated results of digital FIR filter of different orders. As the order of the filter increases, the performance of the filter increases. The magnitude of the output signal decreases when the frequency of the input signal increases. The response of the band stop filter is clearly visible when the order of the filter increases. The FIR filter designed is synthesized in Xilinx 9.1i and the device utilization report is presented for filter of order 3, 6 and 15 respectively.

12. References

[1]OppenheimA.V. and SchaferR. W., discrete-Time Signal processing. Upper Saddle River, NJ: Prentice hall, 1989.
[2]Ching-Tang Chang, Kenneth Rose and Robert A. Walker, “High-Level DSP Synthesis Using the COMET Design System” IEEE
Trans. On digital signal processing systems, vol. 8, no. 6, pp. 408-411, 2010.
[3]Douglas. L. Perry, VHDL: Programming by examples. New York: McGraw-Hill, 2002.
[4]Guo Gaizhi, Zhang Pengju, Yu Zongzuo, Wang Hailong, “Design and Implementation of FIR Digital Wave Filter Based on DSP”
IEEE Trans. on digital signal processing systems, vol.489-491, no. 978, pp. 978-989, 2010.
[5] Odriguez-AndinaJ. J. R., MooreM. J., andValdesM. D., “Features, design tools, and application domains of FPGAs” IEEE Trans.
Ind. Electron. vol. 54, no. 4, pp. 1810-1823, 2007.
[6] Tierno.J, Rylyakov.A.S, Rylov, Singh.S, Ampadu.P, Nowak’s, Immediato.M, and Gowda.S, A 1.3 GSamples10 tap full rate
Variable latency self-timed FIR filter with clocked interfaces in Proc. Int. Solid State Circuit Conference, San Francisco, CA, pp-444.
Feb. 2002
[7] Montek Singh, Jose Tierno.A, Alexander Rylyakov, Sergey Rylov, and Steven Nowick .M, Fellow, IEEE. “An Adaptively Pipelined
Mixed Synchronous Asynchronous Digital FIR Filter Chip Operating At 1.3 GHz”. IEEE Transactions on very large scale integration
(VLSI) systems, vol. 1.8, no.7, July 2010.
[8]Singh.M, Tierno.J.A, Rylyakov.A, Rylov.S and Nowick.S.M, “An adaptively- Pipelined mixed synchronous-asynchronous digital
FIR filters chip operating at 1.3GHz, in proc. IEEE Int. Symp. Asynchronous Circuits and System, Manchester, U.K, pp.84-95,
pr.2002.
[9]Peter Ashenden.J, “VHDL Tutorial,” Ashenden Designs Pty. Ltd., Elsevier Science, USA, 2004.
[10] Ramesh Babu, “Digital Signal Processing”,-TATA McGraw Hill, 2007.
[11] Xilinx Corporation “Xilinx Spartan-3E FPGA family: Complete data sheet,” 2007. [Online]. Available: http://www.xilinx.com.