FPGA Implementation of Adaptive Filter and its Performance Analysis

J. Malarmannan, S. Malarvizhi
Department of ECE, SRM University
Chennai, India
arunmalar88@gmail.com, malarvizhi.g@ktr.srmuniv.ac.in

***Abstract*- Adaptive filters are used in various real-time applications such as echo cancellation, noise cancellation, system identification and prediction. Field -programmable gate arrays (FPGAs) are also used most widely for applications where timing requirements are strict. Thus implementation of filter in real-time is needed. The objective of this paper is to design and implement an Adaptive filter which is robust to impulsive noise using hardware description language (HDL) design. The design implementation and its performance analysis are presented. The targeted FPGA is Altera CycloneIV. The obtained design results in superior performance, greater data sample frequency and less logic occupation.**

***Keywords*-** *Adaptive filters, Application Specific Integrated Circuits (ASIC), Field-programmable gate array (FPGA), Hardware description language(HDL).*

I. INTRODUCTION

Adaptive filters are digital filters capable of self- adjustment. These filters can change with accordance to their input signals. An adaptive filter is used in applications that require differing filter characteristics in response to variable signal conditions. Adaptive filters are typically used when noise occurs in the same band as the signal or when the noise band is unknown or varies over time. The adaptive filter requires two inputs: the input signal and a noise or reference input. An adaptive filter has the ability to update its coefficients. New coefficients are sent to the filter from a coefficient generator. The coefficient generator is an adaptive algorithm [1] that modifies the coefficients in response to an incoming signal. In most applications the goal of the coefficient generator is to match the filter coefficients to the noise so that the adaptive filter can subtract the noise from the signal. Since the noise signal varies, the coefficients must vary to match it, hence the name adaptive filter. The digital filter is typically a special type of finite impulse response (FIR) filter, but it can be an infinite impulse response (IIR) or other type of filter. On the other hand, Application Specific Integrated Circuits (ASIC) could provide the solution that meets all the constraints. However, they lack the flexibility existing in a DSP processor or Field Programmable Gate Array (FPGA) and leave no room for re-configurability of their circuitry. Hence the use of FPGAs is increasing. Hence, FPGAs are the target hardware here for the implementation of Adaptive Algorithm.

Adaptive filters have uses in a number of applications including noise cancellation, linear prediction, adaptive signal enhancement, and adaptive control which are variedly applied in Acoustic environment modelling for Sonar applications [2]. Noise cancellation is also an important field where adaptive filtering is used [3].Adaptive modelling plays a significant role in control systems and signal processing. It is widely used in control tasks, especially in cases when system structure is known, but its parameters are poorly defined. Impulsive noise suppression in speech is very important for clear voice communications. For signal processing, impulsive noise suppression is carried out using nonlinear M-filters, namely, median and myriad filters. And the algorithm used here is a new Adaptive algorithm that is applicable for noise cancellation.

II. GENERAL DESCRIPTION OF FPGA BASED SIGNAL PROCESSING

Most digital signal processing done today uses a specialized microprocessor, called a digital signal processor, capable of very high speed multiplication. This traditional method of signal processing is bandwidth limited. There occur a fixed number of operations that the processor can perform on a sample before the next sample arrives. FPGA-based digital signal processing is based on hardware logic and does not suffer from any of the software based processor performance problems [4]. FPGAs allow applications to run in parallel so that a 128 tap filter can run as fast as a 10 tap filter. Applications can also be pipelined in an FPGA, so that filtering, correlation, and many other applications can all run simultaneously. In an FPGA, most of the application is working mostly when timing requirements are strict. An FPGA can offer 10 to 1000 times the performance of the most advanced digital signal processor at similar or even lower costs.

Nowadays, the use of FPGAs is increasing. They are the prototyping hardware devices, combining the main advantages of ASIC and DSP processors, since they provide both a programmable and a dedicated hardware solution. It is usual to use an FPGA as the prototyping device due to factors such as time and cost. Moreover, an FPGA is more efficient in power consumption, an advantage for battery-operated systems, and, for the same application, requires less clock system speed compared to a DSP or a general-purpose processor,

offering better electromagnetic compatibility properties. Thus, for a wide range of applications, FPGA implementation might be the best option. However, in the case of low sampling frequency requirements or no low power consumption needs, some other devices could be more suitable. In this paper, FPGAs are the target hardware used. FPGAs are ideally suited for the implementation of adaptive filters. However, there are several issues to be addressed. Calculation of adaptive algorithm in software simulations are normally carried out with floating point precision. Another concern is the filter tap itself. Various techniques havebeen devised to efficiently calculate the convolution operation when the filters coefficients are fixed in advance. For an adaptive filter whose coefficients change over time, these methods will not work or need to be modified significantly.

III.ADAPTIVE ALGORITHM

Adaptive algorithm includes calculation of values like e(n), β and μ. The Update Algorithm [1] is divided into four logic steps. The Adaptive algorithm is fed back to the Adaptive system as shown in Fig. 1 and the response output is y(n).

Step 1) Calculation of y(n) . Here L=9.

$y(n) = \sum_{k=0}^{L-1} x(n-k) \,.\, w_n(k) = X^t_n . W_n$ **(1)**

where y(n) = Convoluted filter output.

L = Convolution Length.

$x(n-k)$= Delay line samples by the input of the filter.

$w_n(k)$ = Weight Signal

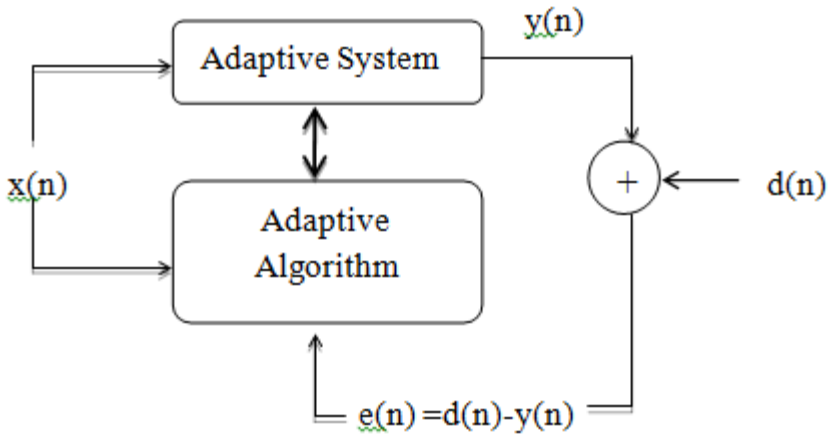


Fig 1.Adaptive Filter Scheme

Step 2) Calculation of error e(n). The Estimation error is the difference between desired input and the adaptive system output y(n).

$e(n) = d(n) - y(n)$. **(2)**

where e(n) = Filter error signal.

d(n) = Filter desired signal.

Step 3)Calculation of β by finding theVariance(σ^2)[5] with low computational cost.

$\sigma^2(n) = 0.95.\sigma^2(n-1) + e^2(n)/n$, if $n = 0, \sigma^2(n) = 1$. **(3)**

where $\sigma^2(n)$ = Variance of the filter.

β = Cost function of the filter.

Step 4) Calculation of weight update coefficient $w_{n+1}(k)$. The filter coefficients are updated based on the following equation where $w_n(k)$ and $w_{n+1}(k)$ are the current and updated co-efficients, μ is the learning rate, and $x(n-k)$ is the delay-line samples provided by the input of the filter:

$$w_{n+1}(k) = \begin{cases} w_n(k) + \mu . \text{Sign}[e(n)].x(n-k), & \text{if } |e(n)| > \sigma^2 \\ w_n(k) + \mu.[(2/\sigma^2) - |e(n)|/\sigma^2)].e(n).x(n-k), & \text{if } |e(n)| \le \sigma^2 \end{cases} \quad \textbf{(4)}$$

where μ = Learning rate of the filter.

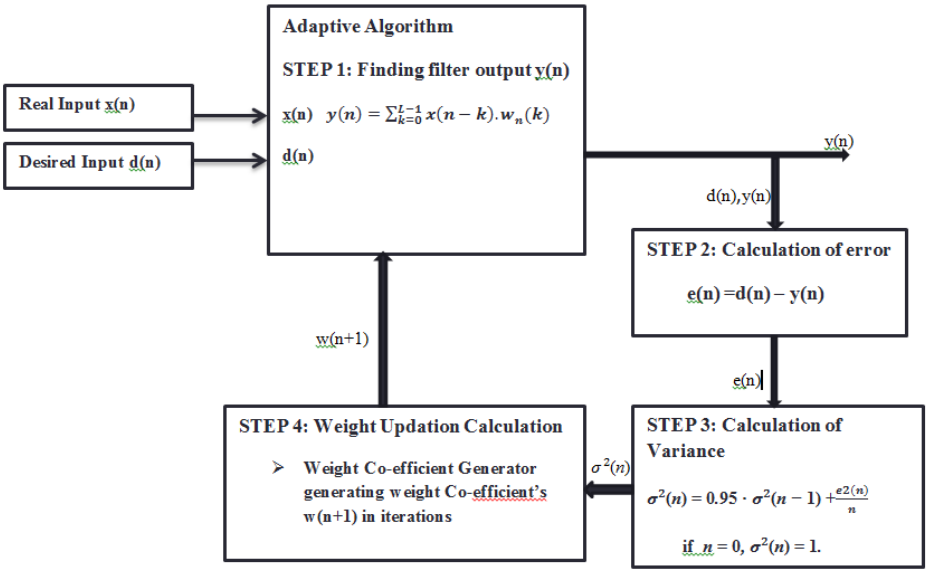


Fig 2. Architecture of Adaptive filter Scheme.

In Fig 2, the architecture of Adaptive algorithm is shown where the calculation of convoluted output y(n), error e(n), Variance and weight updation are done in a loop form where the weights are given as feedback to the filter itself. The desired d(n) signal is to be retrieved back by removing the impulse noise. The input x(n) is a real time audio signal. Noise is added additionally and the error is computed. After the weight updation, the filter output is processed again in iterations to get back the desired signal.

IV. FPGA IMPLEMENTATION OF THE FILTER

In order to perform all the required calculations for each new data sample (iteration), we defined a finite state machine (FSM) shown in Fig. 3. The dataflow of the FSM comprises of six different states which calculates the adaptive algorithm, The FSM, when in State0 called as idle state waits for a new data sample. Once the data is received, the FSM increments the

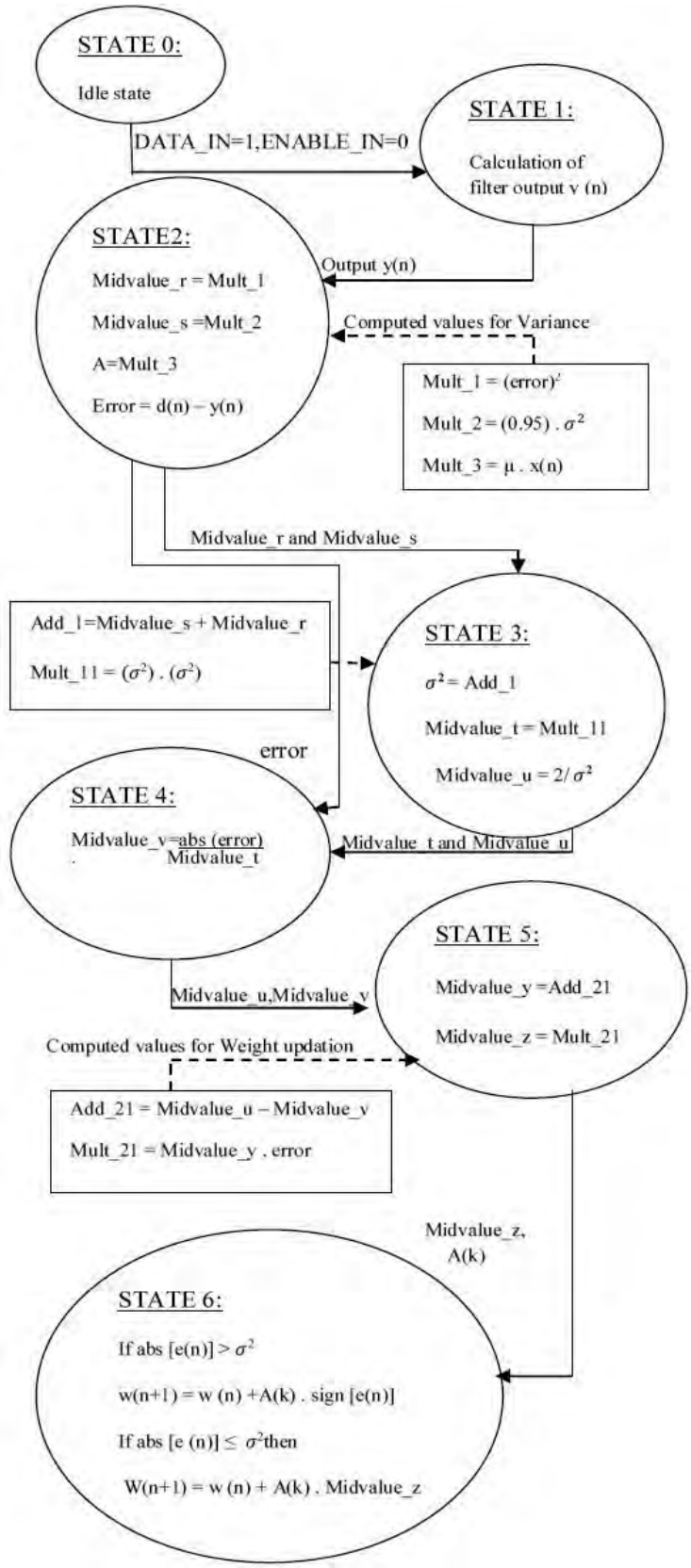


Fig 3. FSM flow of the design

state for every new clock cycle in order to perform different calculations at each state. Thus, most of values in these calculations are intermediate values (Midvalue_∗ signals). After six clock cycles, all the calculation process is completed, including the output value and coefficient's update, being ready for a new data sample. For the FSM to work properly, the signal DATA_IN must be high during all the states. There also exists an

ENABLE_IN signal preventing the FSM to start a new calculation for the same input data, only allowing a new calculation when DATA_IN goes low and then high, i.e., new input data are received. The intermediate values used here are Midvalue_r, Midvalue_s, Midvalue_t, Midvalue_u, Midvalue_v, Midvalue_y and Midvalue_z. All the calculations are distributed among the states, obtaining partial results to optimize logic resources. The dataflow for FSM is shown in Fig 3 which includes six different states are as follows.

In STATE 1 the filter output signal y(n) is calculated. This state makes use of nine multiplications, eight additions, and one FIFO register to store the new sampled input and the L − 1 previous inputs (input vector).In STATE 2, calculation of Midvalue_r, Midvalue_s, error, and the product [μ · x(n)] is done. Midvalue_r is the result of a division, then it will be ready for the next state (S3), and the product [μ · x(n)] is stored in a second FIFO register called updatevector. Here, two multiplications, one division, and one addition are required. This logic resource is reused iteratively for every clock cycle thereby reducing the area consumption. In STATE 3, the variance (sigmasquare), Midvalue_t, and Midvalue_u values are obtained, requiring a single multiplication, an addition, and a division. This state also undergoes reusability of an adder and a multiplier. STATE 4 computes one division for the calculation of Midvalue_v. In STATE 5, Midvalue_y and Midvalue_z are calculated using a multiplication and an addition. In STATE 6, the coefficient's update algorithm is performed for all the k^{th} coefficients, using all the intermediate values obtained from previous states by Midvalue_$*$ signals.

V.HARDWARE OPTIMIZATION

Hardware optimization is a criterion which results in better performance analysis. In this filter implementation the computations done are addition, multiplication and division. Here, in each state, maximum three multiplications and one addition is performed. No more than one state will be executed in the same time since the implementation is done using finite state machine (FSM). So, instead of using separate hardware's for each state we reuse same multipliers and adders. This will give better hardware efficiency without compromising the performance. There is an increase in area consumption of the filter which does not reuse the hardware's such as adder and multiplier for adaptive algorithm implementation which is tabulated in Table 1 with a maximum of 2% Logic elements and 5% multiplier elements of the total FPGA memory which undergoes iterative process for weight updation.

Hardware's	**Area Utilization**
Total Logic Elements	2,461(2%)
Total Registers	672
9-bit Multiplier Units	25(5%)

Table 1. Area Utilization of filter

VI.SIMULATION RESULTS

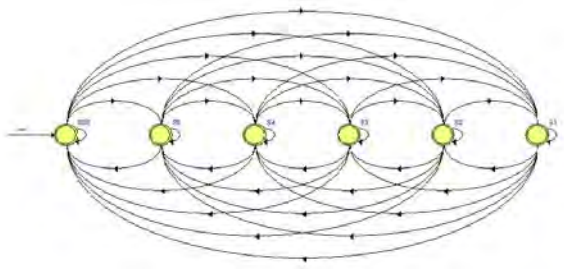


Fig 4. Flow of State machine

The FSM flow simulation is shown in Fig. 4, which involves flow of data according to DATA_IN valves at different iterations involving at six clock cycles.

Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	8.41 MHz	8.41 MHz	clk	

Fig 5.F_{max} report of filter design without reusability of hardware

The simulation result of filter without and with reusability of hardware is shown in Fig 6 and Fig 7 respectively. The clk, rst and enable signals are made high and later reset is set to 0. Now, the data inputs of the filters are given. Now the simulation is done and the filter output is acquired for every clock cycle. At the end of sixth clock cycle, the iteration is completed. Also, the weight gets updated and at the end of sixth clock cycle, it becomes 0. The design which is done in HDL is synthesized using Altera QuartusII tool. The logic occupation of the design is viewed after synthesis. The adder and the multiplier units are reused again and again for every iteration, thereby increase in area occurs. Hence this is avoided by using the same adder and multiplier units for each iteration thereby reducing the logic occupation. The overall multiplier units consumed is 5% and that of reusing hardware is 2% which is shown in Fig 8 and Fig 9 respectively. The maximum operating frequency of the design is 81.7MHz. Table 2 highlights the logic occupation of the filter in Altera CycloneIV FPGA target hardware. Since the logic elements are reused here for every iterations, the area utilisation is greatly reduced.

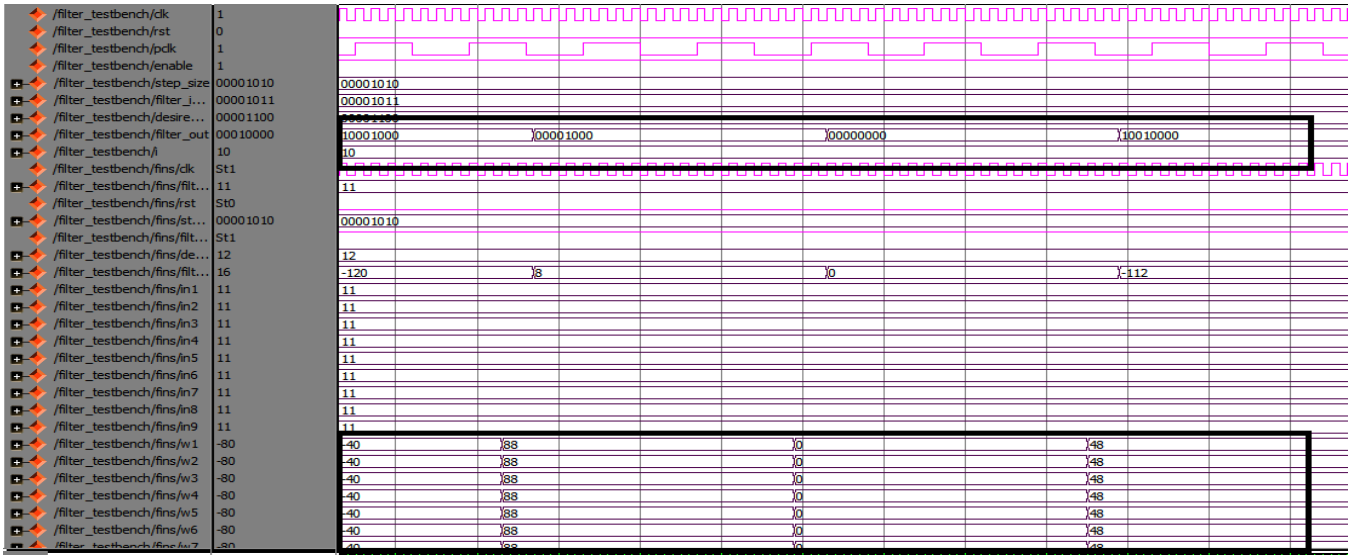


Fig 6. Simulated Output of Normal filter

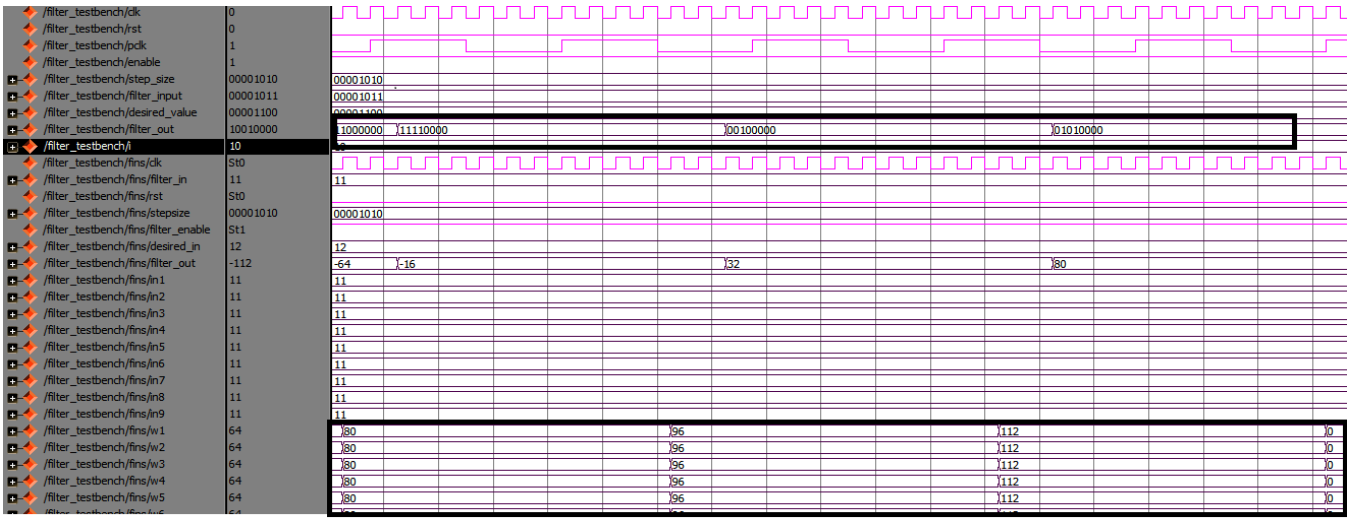
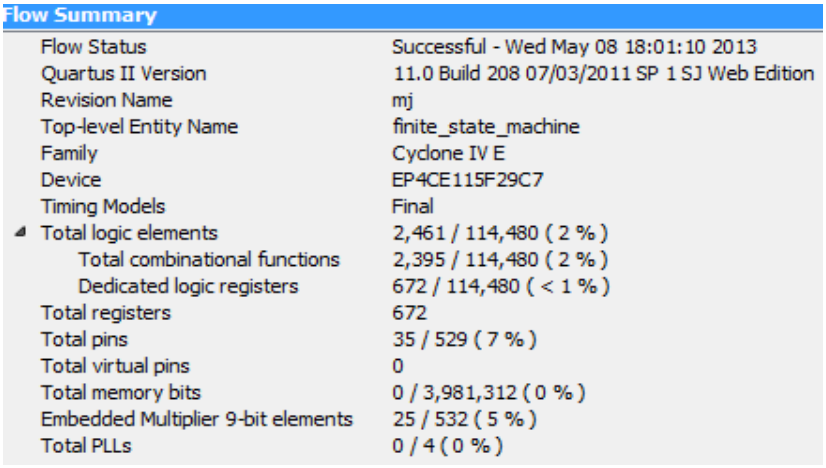
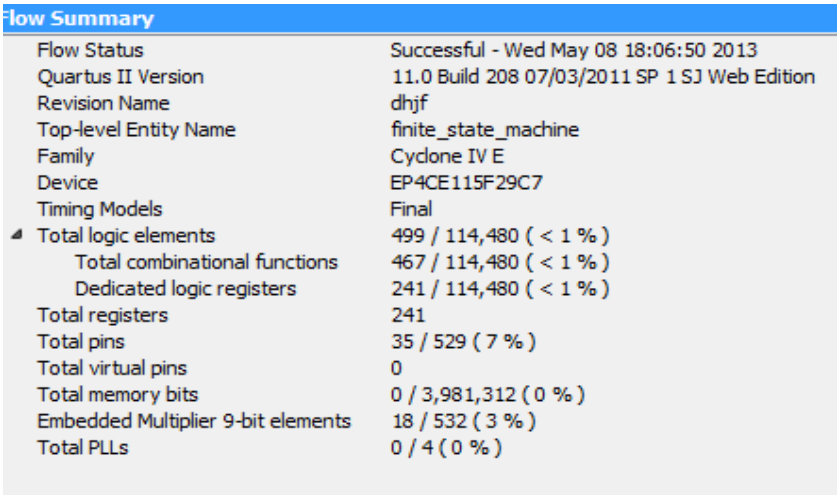


Fig 7. Simulated Output of filter with reusability of hardware



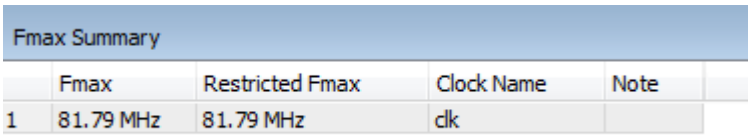
Flow Summary	
Flow Status	Successful - Wed May 08 18:01:10 2013
Quartus II Version	11.0 Build 208 07/03/2011 SP 1 SJ Web Edition
Revision Name	mj
Top-level Entity Name	finite_state_machine
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	2,461 / 114,480 (2 %)
Total combinational functions	2,395 / 114,480 (2 %)
Dedicated logic registers	672 / 114,480 (< 1 %)
Total registers	672
Total pins	35 / 529 (7 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	25 / 532 (5 %)
Total PLLs	0 / 4 (0 %)

Fig 8. Area utilization of Normal filter



Flow Summary	
Flow Status	Successful - Wed May 08 18:06:50 2013
Quartus II Version	11.0 Build 208 07/03/2011 SP 1 SJ Web Edition
Revision Name	dhjf
Top-level Entity Name	finite_state_machine
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	499 / 114,480 (< 1 %)
Total combinational functions	467 / 114,480 (< 1 %)
Dedicated logic registers	241 / 114,480 (< 1 %)
Total registers	241
Total pins	35 / 529 (7 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	18 / 532 (3 %)
Total PLLs	0 / 4 (0 %)

Fig 9. Area utilization of filter withreusability of hardware



Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	81.79 MHz	81.79 MHz	clk	

Fig 10.F_{max} report of filter design with reusability of hardware

From the Simulated results the Weight updation generator generates Weight coefficients for every six clock cycles in the FSM unit. By reusability of hardware's the area occupation is greatly reduced to 3% than the filter design which does not reuse hardware. Hence the area and the power consumption greatly reduces thereby increasing the performance and speed which is shown in Fig. 10.The data sample frequency is greatly increased thereby increasing the performance.

Filter type	**Area Utilization**	**Sample Frequency (f_{max})**
Filter without reusability of hardware	5%	8.41MHz
Filter with reusability of hardware	2%	81.79MHz

Table 2. Comparison table for performance analysis of filter

VII. FPGA IMPLEMENTATION

Fig 11.FPGA Implementation of the filter design

Implementation of Adaptive filter is done in Cyclone-IV target hardware as shown in Fig 11. An audio signal, which is a real-time input is being processed along with the addition of noise and filtered output is retrieved back after updation of weight co-efficient. Higher speed of system clock is achieved with six clock cycles which offers greater data sample frequency. The logic occupation which includes mainly the Logic elements, multipliers and registers occupy only about 5% of the total area in the hardware. The maximum frequency obtained is high thereby increasing the performance. The real time input which is the audio signal is added with noise and the desired or the original signal is retrieved back by self-adjusting.

VIII. CONCLUSION

The implementation of a useful adaptive filter is done using FPGA devices. Furthermore, the FPGA implementation allows the usage in hardware platforms where low power consumption is required. Here we reduce area consumption and also power consumption by reusing same hardware for various stage of computations s.In contrast to other devices such as DSP processors, a high data sample rate is achieved by increasing the frequency of operation, thus throughput is increased.

REFERENCES

[1] Alfredo Rosado-Muñoz, Manuel Bataller-Mompeán, Emilio Soria-Olivas, Claudio Scarante, and Juan F. Guerrero-Martínez, "FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches," Signal Process., vol. 58, no. 3, March 2011.

[2] E. S. Nejevenko and A. A. Sotnikov, "Adaptive modelings for hydroacoustic signal processing." Pattern Recognit. Image Anal., vol. 16, no. 1, pp. 5–8, Jan. 2006.

[3] J. Benesty, T. Gänsler, D. R. Morgan, M. M. Sondhi, and S. L. Gay, Advancesin Network and Acoustic Echo Cancellation. Berlin, Germany: Springer-Verlag, 2001.

[4] M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "An FPGA-based implementation for median filter meeting the realtime requirements of automated visual inspection systems," in Proc. 10th Mediterr. Conf. Control Autom.—MED, Lisbon, Portugal, 2002.

[5] E. Soria, J. D. Martín, A. J. Serrano, J. Calpe, and J. Chambers, "Steadystate and tracking analysis of a robust adaptive filter with low computational cost," Signal Process., vol. 87, no. 1, pp. 210–215, Jan. 2007.

[6] D. Zhang and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," IEEE Trans. Ind. Electron., vol. 55, no. 2, pp. 551–561, Feb. 2008.

[7] J. G. Proakis, Digital Communications. New York: McGraw-Hill, 2008.

ACKNOWLEDGEMENT

J. Malarmannan completed his M.Tech degree in VLSI Design from SRM University, Kattankulathur,Chennai in 2013. He did his B.E. degree in in the stream of Electronics & Communication Engineering, Karunya University, Coimbatore. His interests include FPGA based design and Digital designs.His work is related to DSP based FPGA designs and Digital design.

S.Malarvizhi received Ph.D degree in wireless communication from the College of Engineering,Gunidy,Anna University, Chennai -25 in 2006. She is currently a professor and Head of Electronics and Communication Engineering Department , SRM University. Her work is related to wireless communication and VLSI structures.