Acoustic Noise Cancellation Using LMS Filter on FPGA

Madhuri Jadhav[1], Prof. Aarti Bakshi[2]
[1]PG scholar, [2] Associate Professor
[1] Electronics and telecommunication, SLRTCE,Mira road
[2]Information technology, KCCOE,Thane

Abstract **- Filtering data in real-time requires dedicated hardware to meet demanding time requirements. Programmable Digital Signal Processor (PDSP) systems are replaced by FPGA systems due to their parallel architecture, wide bandwidth, and flexible nature. For speech processing, here we present the applicability of a FPGA system. The adaptive filtering algorithms is used to estimate the signals statistics if prior information of signal is not known. Least Mean Squares (LMS) is implemented for adaption of the filter coefficients as it is one of the widely used algorithm in many signal processing environment. Due to its reliable nature and simplicity, high flexibility and high integrated FPGA technology. LMS algorithm using FPGA technology is well used in adaptive filter, compared with other algorithms it is easy to implement. It has been well regarded in the field of signal processing.**

I. INTRODUCTION

In all practical situations, the received speech waveform contains some form of noise component. Technique for optimum noise filtering for speech signals based upon the principles of least mean square (LMS) adaptive filtering has the advantage of requiring no a priori knowledge of the detailed properties of the noise signal. The technique makes use of the quasi-periodic nature of the speech, improves the perceived speech quality of a signal corrupted by acoustic noise [4]. The design tools should be carefully chosen as digital signal processing applications impose considerable constraints area, power dissipation, speed and cost. The most commonly used tools for the design of signal processing systems are: Application Specific Integrated Circuit (ASIC), Digital Signal Processors (DSP) and FPGA. The FPGA technology has achieved conspicuous development. System on a Chip (SoC) and Intellectual Property (IP) designs can be integrated and downloaded into FPGA to work with an embedded processor. The latest 65 nm technology makes the function of FPGA even more powerful. The 65nm Virtex-5 FPGAs offer unparalleled performance at speeds on average 30 percent faster and 65 percent increased capacity, while consuming 45 percent less area and reducing power consumption by 35 percent than previous generation devices .To show the performance of FPGA in digital signal processing applications, we implement an Adaptive Noise Canceller on an FPGA and use the LMS algorithm as the adaptive filtering algorithm. Adaptive Noise Canceller requires high sampling rate, thus FPGA is a good choice [6].

II. CONCEPT OF ADAPTIVE NOISE CANCELLER

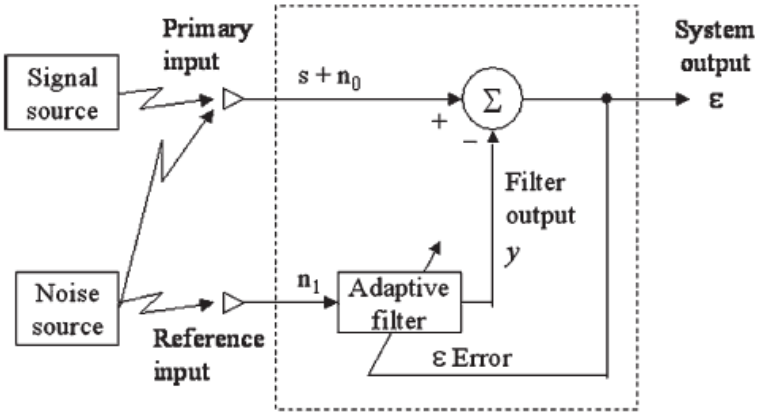

Figure 1: Adaptive Noise Cancelling Model

Fig: 1. shows the general block diagram of an adaptive noise canceller. The signal plus an uncorrelated noise n0 form the primary input to the canceller. The reference input is the noise n1, which is assumed to be uncorrelated with the signal but correlated in some unknown way with the noise n0. Adaptive filter processes noise n1 that automatically adjusts its own impulse response through a minimization algorithm such as the least mean square (LMS) algorithm that responds to an error-dependent signal. Filtered output y is a copy of n0. Filter output is subtracted from the primary input s + n0, in order to produce the system output s + n0 – y.

The objective is to adjust, as much as possible, the canceller output s + n0 − y to the signal. This is practised by feeding the system output back to the adaptive filter and adjusting the filter through an adaptive algorithm to minimize the total system output power.

Assumption: s, n0, n1, and y are statistically stationary and have zero mean. Assume that n1 is correlated with n0 and s is uncorrelated with n0. The output is

$\varepsilon = s + n0 - y$ …… (1)

Squaring (1), taking mean values, and realizing that s is uncorrelated with n0 and y yield

$E[\varepsilon^2] = E[s^2] + E[(n0 - y)^2]$ …… (2)

When the filter is adjusted so that $E[\varepsilon^2]$ is minimized, $E[(n0 - y)^2]$ is, therefore, also minimized. The filter output y is then a best least-square estimate of the primary noise n0. The smallest possible output power is

$Emin[\varepsilon^2] = E[s^2]$ …… (3)

Therefore, y = n0, and ε = s. Here, minimizing the output power causes the output signal to be absolutely free of noise.

On the other hand, when the reference input is completely uncorrelated with the primary input, the filter will turn itself off and will not increase the output noise. In this case, the output power will be

$E[\varepsilon^2] = E[(s + n0)^2] + E[(y)^2]$ …….... (4)

To implement the adaptive canceller, different solutions are available such as implementation of the adaptive controller field-programmable gate array (FPGA) programmable logic devices. This solution has several advantages: The development system works on personal computers where high processing speed is reached due to the optimized specific design as FPGAs are low-cost devices [9].

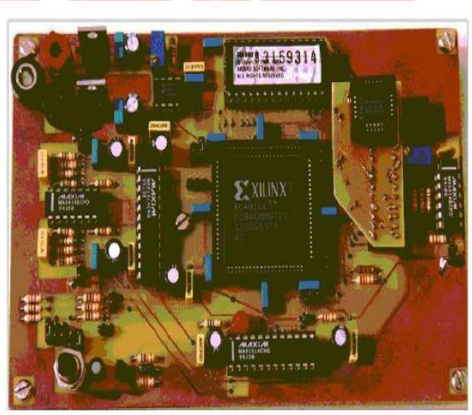

Figure 2: Adaptive Canceller prototype picture

III. METHODOLOGY

3.1 Least Mean Square algorithm

A method called Least Mean Square algorithm is used to suppress the acoustic noise by using Simulink in MATLAB. MATLAB 11a Simulink has a Data Acquisition Toolbox to cancel the acoustic noise from the original signal. We are using the Block LMS filter in order to suppress the acoustic noise from the original signal. The algorithm is derived by the following equations:

y (n) =

wi (n) * x(n-i) (1)

e (n) = d(n) – y(n) (2)

wi (n+1) = wi(n) + 2ue(n)x(n-i) (3)

In these equations, the tap inputs x (n), x (n-1)… x (n-M+1) form the elements of the reference signal x (n), where M-1 is the number of delay elements. e (n) denotes the error signal and represent the overall system output. wi (n) represent the tap weight at the nth iteration. In equation (3), the tap weights update in accordance error. u is the step-size parameter. Scaling factor u controls the convergence speed and stability of the algorithm. LMS algorithm is convergent in the mean square,If u satisfies the condition $0 < u < 2$ / tap-input power, where tap-input power

$$\sum_{k=0}^{M-1} E\left[\left|u(n-k)\right|^2\right]$$

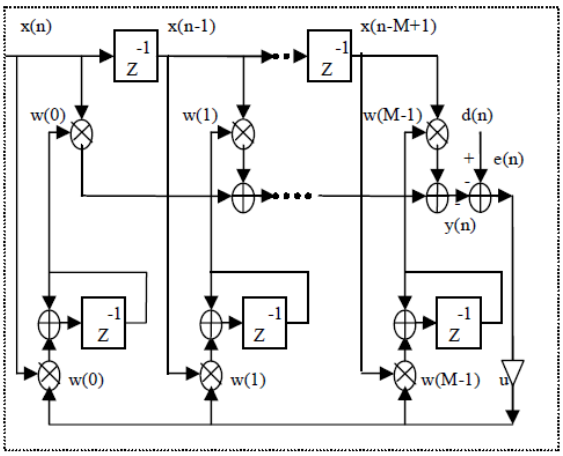

Figure 3: LMS Algorithm Flowchart

There are usually two ways to implement the LMS algorithm, hardware implementation and software implementation. The hardware implementation of the algorithm in an FPGA has good real-time ability, but requires large resources. From Fig 3, we can see that an N-tap adaptive filter requires at least 2N multipliers and 2N adders. The software implementation consumes trivial amount of resources, however, the low speed of which makes it uncommonly used [6].

Table 1: Variables used and their definition

Symbol	Definitions
W	Filter coefficients
μ	Condition for convergence
e(n)	Error signal
d(n)	Desired signal
y(n)	Filtered output
K	Filter order

3.1.1 Stability of LMS algorithm

[1] The LMS algorithm is convergent in the mean square if the step size parameter satisfy

$$0< \mu<2/\lambda max$$

[2] λmax : largest eigen values of the correlation matrix of the input data

[3] The stability test is

$$0< \mu<2/\text{input signal power.}$$

[4] If μ : large, converge rate is fast and the adaption is quick, but if μ is too large it could diverge and can lead to instability.

[5] If μ : too long to converge.

3.2 LMS Core Implementation

The LMS core is the central part of our hardware architecture. We programmed the LMS core with VHDL under the platform of Xilinx ISE 9.1i, and simulate it with ModelSim 6.1b. Validity of the LMS core is tested by using System Generator 9.1.The block diagram of LMS core consist of five basic blocks:

A. Control Block: Control block is used for controlling and arranges the timing for all the blocks. It enable the blocks separately by generating four enable signals which are given to the individual blocks. There are three input signals clk, read l and write 1 given to the control block. When clk and readl=l, all the signals are enable and becomes l. And when clk and write l= l, all the enable signals become O.

B. Delay Block: Delay Block receives the primary input signal d (n) and reference signal x (n) and produces M tap delay signals xout and dout respectively from the output. When the enable signals En_x and En_d get 1 output follows the input otherwise output produces delay signals.

C. MAC Block: MAC Block is Multiply and Accumulator Control block. This block is used to multiply M tap output reference signal (xout) with the M tap weight (w) separately and adding them all so that to produce the output y (n). Here Booth multiplier is used for multiplication purpose. When clk and reset =1, the multiplication output will become O. On the contrary, when clk=1 and reset=O then output will be product of reference signal and weight and adding them separately with the previous output filter.

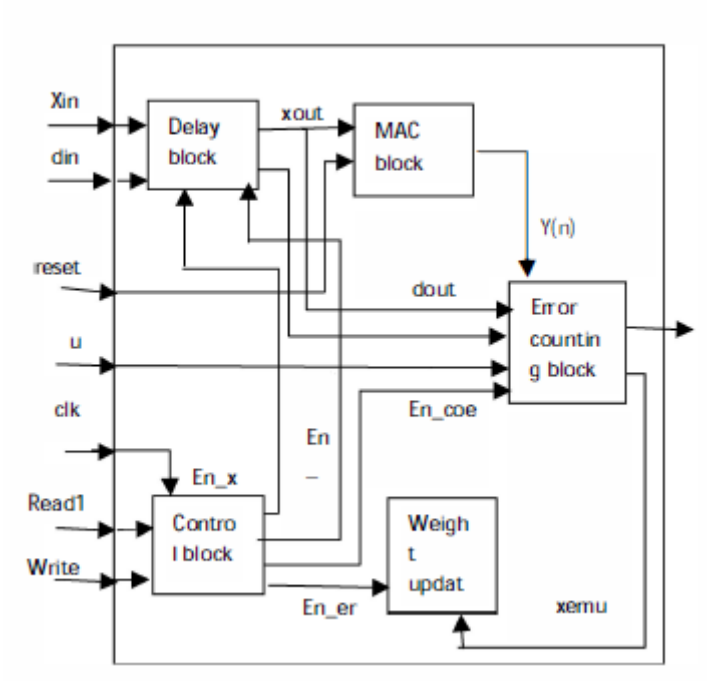

Figure 4: Proposed Block Diagram of LMS Core

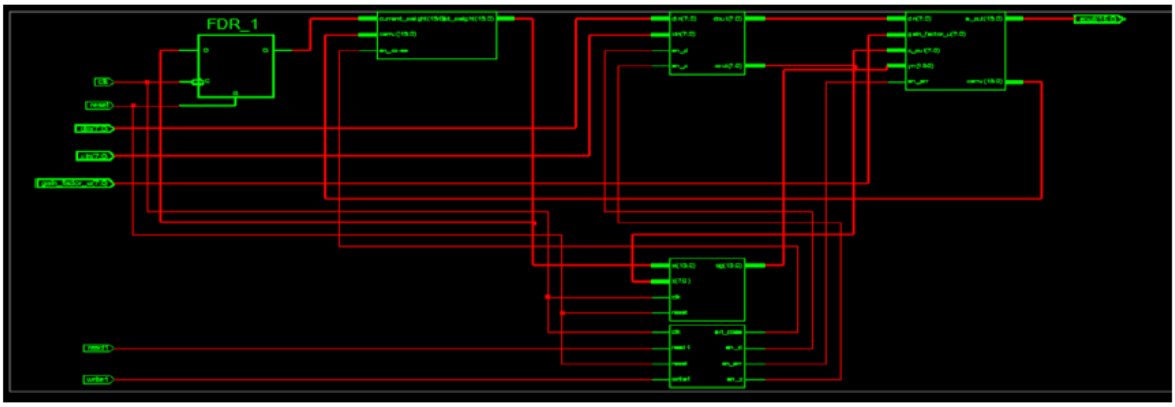

Figure 5: RTL Schematic of LMS Core Block

D. MAC Block: MAC Block is Multiply and Accumulator Control block. This block is used to multiply M tap output reference signal (xout) with the M tap weight (w) separately and adding them all so that to produce the output y (n). In this block we are use Booth multiplier for multiplication purpose. When clk and reset =1, the multiplication output becomes O. On the contrary, when clk= 1 and reset=O then it will give the product of reference signal and weight and adding them separately with the previous output filter.

E. Error Counting Block: Error counting Block gives the output of the LMS core e(n) which is the difference between MAC output y(n) and the primary input signal d(n). In order to adjust the filter weight for minimizing the error, Error counting Block produces xemu which is the feedback given to the weight updation block. Where in which the reference signal x (n), fixed step size and the enable signal En_err are giving as an input to the Error counting Block

F. Weight Update Block: Weight update block is used to update the filter weight w (n) to w (n+l) which is used for the next iteration. When enable signal En_coef =1, it will update the weight by adding the previous weight with the feedback from the error counting Block. Otherwise next weight will be zero [3].

IV. VHDL IMPLEMENTATION OF SYSTEM

The VHDL design of the system is as shown in Fig. 6. Arithmetic is modelled with Q format number representation which provides for each pipeline stage an appropriate number of guard bits for representing the integer part and avoiding overflow effects.

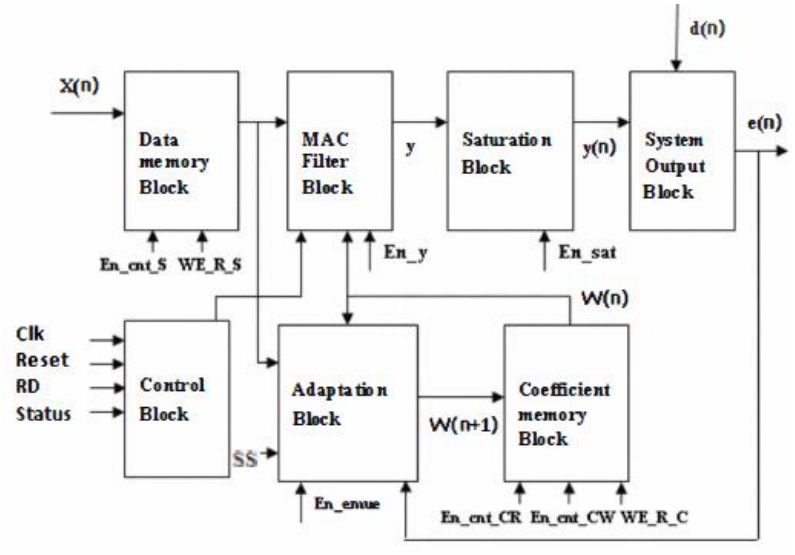

Figure 6: The VHDL design of the adaptive filter

VHDL design is classified into seven blocks as follows:

1. Data Memory block: The single port RAM is designed for storage of the audio samples. The filter is implemented as a sequential MAC unit which performs M accumulations of products during every sample period so that a resource sharing can be utilized. Since the audio sample period fs provides a large amount of available clock cycles per audio sample, no parallel structure with M multipliers and M-1 adders is necessary. For the filtering cycle this block is designed as three-stage pipeline .The input samples read from the data RAM block are multiplied with their corresponding filter coefficient taken from the dual-ported Coefficient RAM block and stored in the accumulator.

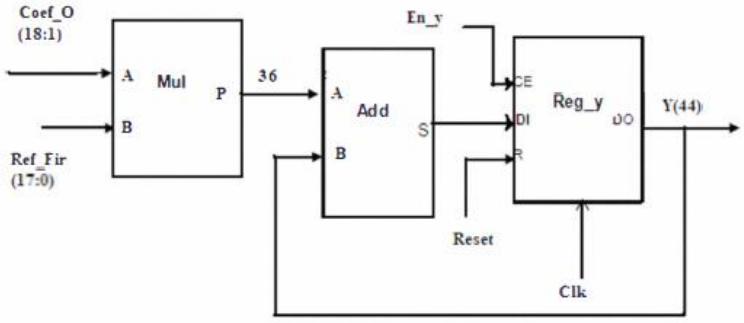

Figure 7: Block diagram of an FIR filter

The flexibility of FPGAs gives them a distinct advantage over other programmable logic devices on the market. Designs can be modified after initial implementation as FPGAs are reprogrammable and can implement any sort of logic circuit.

2. MAC Filter Block: The FIR filter design is based on the transposed direct form in order to keep the sequence which is started with each new input sample pair Err (error signal) and XN (reference signal).

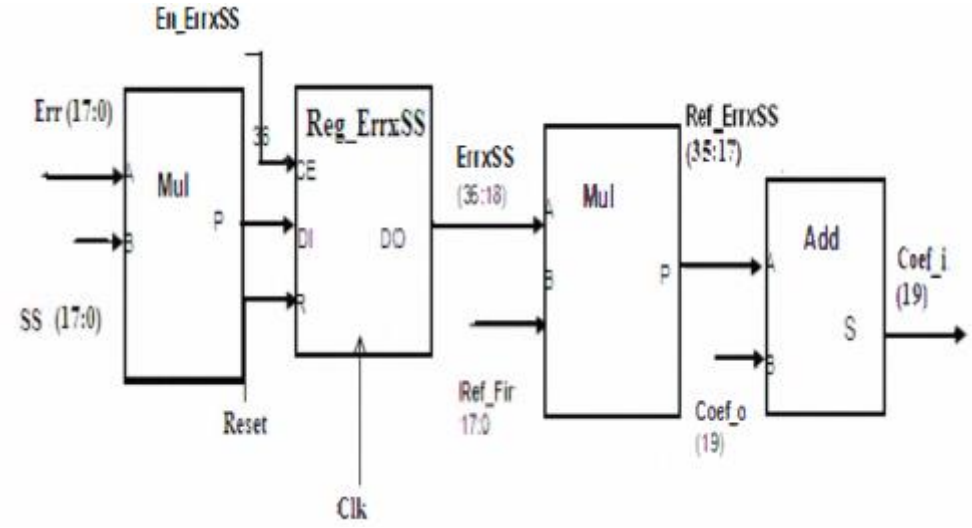

Figure 8: Coefficient adaption unit

The FSM involves the following sequence:

A. START: This is default state in which all registers clears results from the previous calculation cycle. Through an input RD, a new sample XN is stored in RAM by the signal WE S.
B. ERRXSS: performs calculation of the product of error signal Errand step size factor SS and stored in the register Reg_ErrxSS by the signal En _ErrxSS.
C. C/D. FILTER / ADAPT: There is alternating sequence of two pipeline operations runs in parallel. The filter block performs operations of updating address for reading input sample and coefficient, outputting memory and accumulating product of sample and Coefficient and saving in Register Reg_Y by the signal En _Y. Operations performed by the pipeline for the adaptation of the coefficients are updating address for writing and outputting memory, updating address for reading input sample and coefficient. Accumulation of a product from ReCFir and ErrxSS on the current coefficient and storage of the adapted coefficients. The status Cnt_st indicates the highest reading at the address is coefficient present at dual port RAM.
D. STOP: It is the last multiplication of sample and coefficient and accumulation of the result by the signal En_Y. Then read address of coefficient from memory for the transition to a next state.
E. UPDATE: The accumulation result (filter output) fed to the saturation block, for one sample period this value is hold and performs the adjustment of the RAM address counters for the next sequence.
F. RESET: When system is Reset, state becomes Reset to reset all registers and content of RAMs.

3. Saturation Block: The filter output signal is fed to the saturation block, which prevents the filter output from overflow and inverts the sign of the output signal to provide the phase shift for the compensation step.
4. System output block: Implementing equation of error signal e(n) from saturation block output y(n) and primary signal d(n) ,the Adder unit is used . This is the required system output.
5. Adaption Block algorithm: A four-stage pipeline structure designed for the adaptation of the coefficients. Product of the input sample (ReCFir), the error signal (Err) and Step size parameter (SS) is used to calculate the coefficient. A register inserted in this path splits the arithmetic chain for achieving a shorter signal delay so that a clock frequency of feLK = 50MHz can be met.
6. Coefficients Memory block: This block designed for storage of the current filter coefficients. The dual port RAM is chosen to support a parallel processing of the coefficient update block and the FIR Filter block. With two address inputs the reading address of the coefficients and the address for writing back the updated coefficients can be incremented within two interleaved clock periods.
7. Control Block: The Control path functionality is implemented as the Finite state machine (FSM). The FSM controls the processing of the two parallel pipelined data paths [7].

V. FPGA IMPLEMENTATION OF LMS ALGORITHM

5.1 Review of Number Representation

FPGA implementation, conversion from a higher level system description into a lower level implementation in terms of signal flow through hardware circuits, as bit streams. Data flow (streams of bits) can be represented either in fixed point or floating point number format and arithmetic.

5.2 Fixed Point Number Representation

Fixed point representation assigns a fixed width to integer and fraction. Example, a 32 bit number can be considered as 16 integer bit and 16 fractional bits (shown in Fig-9), which can go up to the range of 216-1 to -216 (integer part i.e., 65535 to -65536) , with a fractional range of 1/216.

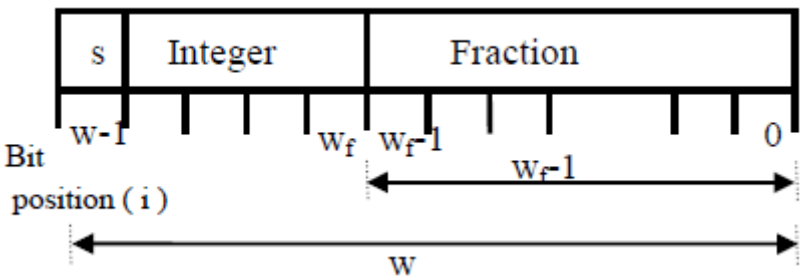

Figure 9: Fixed point number representation, where w represents total width and wf represents fraction width

5.3Floating Point Number Representation

Floating point numbers are, represented approximately to a fixed number of significant digits (the mantissa) and scaled using an exponent. 2, 10 or 16 are the scaling base normally. The typical number that can be represented exactly is of the form: Significant digits × base exponent

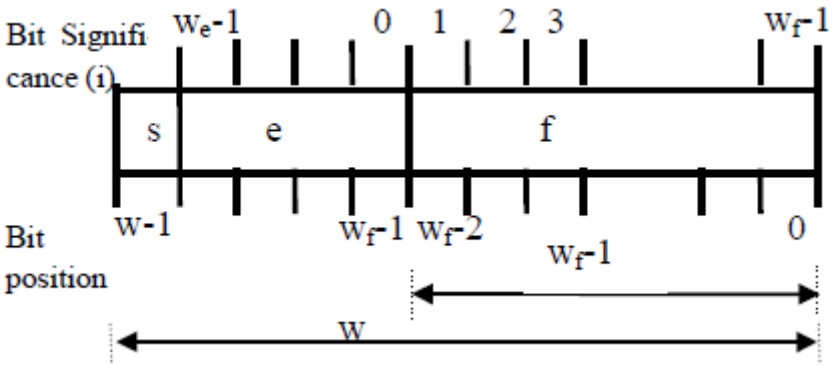

Figure 10: IEEE 754 Floating point number representation, where w represents total width, we represents exponent width and wf is the fraction width

Floating point representation with its exponent component achieves greater range compared to conventional fixed point representation. For example, contemplate floating point format number of IEEE 754 with 32 bit single precision, implemented as 23 bit mantissa (or fraction, f), 8-bit exponent(e) and sign bit(s) described in Fig-10. The 24 bit mantissa (including sign bit) can achieve a precision of 16M (224) compared to the 6K (216) of fixed point format. Remaining 8-bit exponent provides larger dynamic range (in the order of 264), as compared to fixed-point format. The LMS algorithm with a behavioural model using fixed point packages. But a structural model system design is preferred since it takes into account the efficiency with which FPGA resources can be configured and interconnected to achieve the desired functionality. Thus a structural model with FSM is developed to examine the parallel processing capabilities of FPGA.

5.4 Fixed Point Implementation

Using VHDL with fixed point packages, LMS algorithm is implemented in FPGA hardware. The implementation make use of signed fixed point (sfixed) data type with 16 integer bits and 16 bits for fraction (as shown in Fig-10). The mathematical operations such as multiplication, addition, type conversion and resize (rounding of the results) are defined in the package. Using LUT's in FPGA the adaptive filter buffer W and the input circular buffer X with a buffer size N are implemented. New input sample triggers the FSM from idle to new_data state. Data is converted to sfixed data type and saved in the input circular buffer in x_store state and acknowledged. The MAC (multiply and accumulate) operation initiated by the FIR state calculates output in N clock cycles. MAC result is rounded off to the sfixed data type. From corresponding buffers, .input sample x (n-k) and filter coefficient w (k) are read parallelly. Output of an FIR is sent out at data_out state. The error calculation and mu*err multiplication are done in new_err state. The operation of weight updating is implemented in three states. Next address is generated when new weight value is calculated (with rounding off the result) in the wt_calc state, updated weight value are stored in the wt_delay1 state and wt_delay2 state. Input and filter coefficients are read simultaneously. In order to update weight 3*N clock cycles are required. So in 9 individual states with 5+4N clock cycles the LMS algorithm can be implemented [5].

VI. CONCLUSION

The hardware implementation of adaptive filters is a challenging issue in real-time practical noise cancellation, echo cancellation, prediction and one time programmable technologies, where logic is set at the factory and no changes can be made after manufacture. In this paper, Adaptive filter uses LMS algorithm for updating the filter coefficient as it found to be the most efficient training algorithm for FPGA based adaptive filter. Therefore here we represent the strategies and hardware implementation of the LMS Core and simulate it in the VHDL code. In case of high speed architecture, for design purpose direct-form approach is Preferred.

REFERENCES

[1] S.Thilagam,Efficient Implementation of Adaptive Noise Canceller Using FPGA for Automobile Applications International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Vol. 2, Issue 12, December 2013.

[2] Muhammad Wasimuddin and Navarun Gupta, Design and Implementation of Least Mean Square Adaptive Filter on Fetal Electrocardiography, Proceedings of Zone 1 Conference of the American Society for Engineering Education (ASEE Zone 1)2014.

[3] D.B. Bhoyar , Soumita Bera , C.G. Dethe and M.M.Mushrif, FPGA Implementation of Adaptive Filter for Noise Cancellation ,Electronics and Communication Systems (ICECS), 2014 International Conference on Feb. 2014.

[4] MARVIN R. SAMBUR, Adaptive Noise Canceling for Speech Signals IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. ASSP-26, NO. 5, OCTOBER 1978.

[5] Shashikala Prakash, Renjith Kumar T.G, Subramani H,AN FPGA IMPLEMENTATION OF THE LMS ADAPTIVE FILTER FOR ACTIVE VIBRATION CONTROL, Volume: 02 Issue: 10 | Oct-2013. IJRET: International Journal of Research in Engineering and Technology eISSN: 2319-1163 pISSN: 2321-7308.

[6] Tian Lan1 and Jinlin Zhang, FPGA Implementation of an Adaptive Noise Canceller, 2008 International Symposiums on Information Processing.

[7] A. B. Diggikar and. S. S. Ardhapurkar, Design and Implementation of Adaptive filtering algorithm for Noise Cancellation in speech signal on FPGA, International Conference on Computing, Electronics and Electrical Technologies [ICCEET]2012.
[8] Shing-Tai Pan and Xu-Yu Li, An FPGA-Based Embedded Robust Speech Recognition System IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 61, NO. 9, SEPTEMBER 2012.
[9] Rafael Ramos, Antoni Mànuel-Làzaro, Joaquín Del Río, and Gerard Olivar,FPGA-Based Implementation of an Adaptive Canceller for 50/60-Hz Interference in Electrocardiography , IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 56, NO. 6, DECEMBER 2007.
[10] Chang-Min Kim, Hyung-Min Park, Taesu Kim, Yoon-Kyung Choi, and Soo-Young Lee, FPGA Implementation of ICA Algorithm for Blind Signal Separation and Adaptive Noise Canceling IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 14, NO. 5, SEPTEMBER 2003.