Abstract—Adaptive filters have become a very useful building block in several of today’s systems. One of the most common application being Line Echo Cancellation (LEC). The ever increasing data rates used in such communication systems bring along the need for faster adaptive filtering systems that are capable of handling the echo tail generated. This poses two main requirements, i) robust and stable algorithms to ensure efficient functionality under these conditions and ii) more processing power. This paper describes the implementation of such an adaptive filter on a Xilinx Spartan 3 FPGA for use in an echo cancellation system.

I. INTRODUCTION

Echo cancellation requires digital signal processing techniques that can operate on real time data. The performance of such systems is often limited by the processing capability of the implementation. A high processing speed is therefore crucial in implementing real time systems. Flexibility to cater for the changing conditions is also required when the signals are time variant. Digital Signal Processing (DSP) based systems provide acceptable results but due to the sequential nature of the data handling by the microprocessor, they often suffer from processing capability. Dedicated hardware on the other extreme can provide fast processing but is not flexible. Reconfigurable hardware like Field Programmable Gate Array (FPGA) implementations offers a good compromise between the two extreme solutions [1-2]. An adaptive filter used for the removal of a local echo generated in a microwave point-to-point link was developed on an FPGA.

The algorithm employed for updating the filter coefficients is the Least Mean Squares (LMS) algorithm. This algorithm is used since it provides an important trade-off between complexity and convergence time. When compared to other algorithms used for implementing adaptive filters, as shown in Figure 1, the LMS algorithm is seen to perform very well in terms of the number of iterations required for convergence. Fig. 1, also shows that the Recursive Least Squares algorithm is faster in convergence than the LMS but is then very complex to implement, hence hindering system performance in terms of speed and FPGA area used.

This paper first discusses the theory behind adaptive filters as well as the LMS algorithm. In section III a description of the implementation is given while section IV displays the results obtained. Finally section V gives the conclusions made from the results obtained.

II. THEORY

A. Adaptive Filters

Conventional, non-adaptive, filters that are used for the extraction of information from an input time series, x(n), are usually linear time invariant meaning that they perform exactly the same set of linear operations on the time series x(n) to provide the output, irrespective of the value of n. In adaptive filtering this is not the case since this restriction on time invariance is removed by allowing the coefficients used in the linear filtering operation to change according to some predetermined optimization criterion. This means that adaptive filters may be applied in areas where the exact filtering operation required may not be known a-priori. Furthermore the filtering operation may be mildly non stationary [3], [4]. Figures 2(a), (b) and (c), show three different configurations in which adaptive filters are used.

Fig. 2(a): System Identification using an adaptive Filter [3].

Fig. 1: Comparison between different algorithms used for adaptive filter implementation [3].
1. Least Mean Squares;
2. Block Least Mean Squares;
3. Sampled Matrix;
4. Recursive Least Squares.
The setup shown in Fig. 2(c) is used for Echo cancellation where data that is transmitted ‘leaks’ into the receive path and hence disrupts any other data being received (in Full Duplex Systems). In the setup shown above, the output of the adaptive filter is subtracted from the output of the hybrid (Echoed data) and an error term is fed into the receiver. The coefficients of the filter change for a certain amount of time until the error term that is fed into the receiver is close to 0 [3].

The most important criteria to be considered in the design and development of Adaptive filters are stability, initial speed of convergence, consistency of convergence speed with variations in signal conditioning, ability to track time varying characteristics and robustness to additive noise [3].

B. LMS ALGORITHM

The LMS stochastic gradient algorithm is preferred to other algorithms mainly due to the fact that it removes the requirement of having explicit knowledge of the autocorrelation matrix, \( \phi_{xx} \), and the cross-correlation vector, \( \phi_{xy} \), as required in the method of steepest descent and also avoids the direct matrix inversion inherent in the Wiener equation. The LMS utilizes a time recursion method which uses an estimate of the gradient rather than the gradient itself [3].

The vector \( \hat{h}(k) \) is an estimate for the Wiener filter estimate \( h_{opt} \), the latter being the filter estimate that minimises the cosine t function. The vector \( \hat{\nabla}(k) \) is an estimate of the gradient \( \nabla(k) \) of the Mean Square Error cost function at the point where the impulse response is \( h(k) \). The variable \( \mu \) is the convergence factor of the system and for ensuring convergence this value must fall within certain bounds (to be defined later) [3].

\[
\hat{\nabla}(k) = \frac{\partial \xi}{\partial h}(h(k)) = \frac{\partial}{\partial h} \left[ \left(y(n) - h^T(k)x(n)\right)^2 \right] 
\]

The variable \( \xi \) represents the MSE cost function while \( x(n) \) and \( y(n) \) represent the system input and output respectively [3]. Rearranging Eqn. 2 yields

\[
\hat{\nabla}(k) = -2E[x(n) \ (y(n) - h^T(k)x(n))] \]

In the formation of the estimate \( \hat{\nabla}(k) \) of the actual gradient \( \nabla(k) \) the group average of Eqn. 3 is replaced by a time average over \( n \). Since \( h(k) \) changes at each data point, the time average is reduced to a single value at \( n = k + 1 \) which yields [3]:

\[
\hat{\nabla}(k) = -2\chi(k+1)e(k+1) 
\]

The variable \( e(k+1) \) is the error term from the convergence value to be used in the next iteration. From Eqn. 4 this variable can be derived as [3]:

\[
e(k+1) = y(k+1) - h^T(k)x(k+1) \]

Substituting (4) into (1) gives the LMS algorithm [3]:

\[
\hat{h}(k+1) = \hat{h}(k) + 2\mu e(k+1) \]

For coding purposes the above is adapted slightly since it would otherwise represent a timing advance. The equation used is thus:

\[
h(k+1) = h(k) + 2\mu e(k)x(k) \]

The Convergence boundary

The value of the variable \( \mu \) determines the speed of convergence of the system. If \( \mu \) is very small then the algorithm converges very slowly. A large value of \( \mu \) may lead to a faster convergence but may be less stable around the minimum value. For adaptive filtering algorithms, the value for \( \mu \) falls within a bound defined in Eqn 8 [3].

\[
0 < \mu < \frac{1}{3NE[x^2(n)]} \]

The variable \( N \) represents the number of filter coefficients. The value chosen for \( N \) represents a trade-off between the echo tail required and the data input speed. Changing this value effects the maximum value of
III. IMPLEMENTATION

The FPGA used is a 200,000-gate Xilinx Spartan-3 XC3S200 FPGA in a 256-ball thin Ball Grid Array package (XC3S200FT256). Amongst the different architectures available for implementation, post testing results showed that direct convolution provided the fastest and least area consuming algorithm. For testing purposes the initial design was carried out in a modular approach in that all the blocks were implemented as separate modules on the FPGA. This method however proves lossy in slice and flip-flop usage due to the extra components used in connecting the separate hierarchies together. This also results in increased data signal paths which carries with it increased delays and hence lowers the upper limit on the maximum possible frequency at which the system can operate. Therefore in the optimization stage of the design the separate modules were combined together into one module using separate process blocks operating either on a single global clock or separate clocks depending on the speed required.

The Finite Impulse Response (FIR) filter

The input stage of the FIR filter is connected to a system of cyclic buffers that allows the system to process data received over the link without delaying the system. Speed was of major importance due to the complex real time processing involved. Once the buffer is full it sends a signal to the filter input stage after which data, clocked at 250 MHz, enters the filter and is saved into an internal buffer containing both preceding and trailing 0's. Using a sliding window method, ten samples are output from this buffer in every clock cycle into the multiplier input ports where multiplication with the filter coefficients is carried out, using ten parallel hardware multipliers, in the next clock cycle. The ten outputs are added together to give the first filter output. This process is repeated until the window reaches the end of the buffer after which the filter ready flag is asserted indicating to the LMS block that the new coefficients are available for processing.

The FIR filter implementation uses combinations of several FPGA resources in order to obtain the throughput required from the system. The multiplication block was implemented using different architectures in an attempt to optimize computation speed and reduce area usage. Implementations used involved (i) the use of Carry-Save addition (ii) Bit pair recoding techniques based on the Booth algorithm and (iii) use of several Look Up Table (LUTs) according to the Quine Mc Cluskey algorithm. None of the above architectures performed as well as the hardware multipliers with computation time of only 1.3 ns [5]. [6] describes how the FPGA is configured for using the hardware multipliers in the Place and Route process.

The adders used were first implemented using Carry Look-ahead addition techniques. The adder thus implemented performed well in terms of speed and stability but still lacked in performance when compared to adder implementation using dedicated LUTs within the FPGA. Adder implementation using LUTs also provided the advantage of creating Multiply and Accumulate (MAC) units, these being multipliers and adders working in close synchronisation, mapped onto the same area, with intermediate intervention of the high speed RAM provided. The multiplier outputs are available to the adders on the next clock cycle and having these mapped on the same area introduces virtually no time delay due to data transmission. The RAM acts as a buffer in the case when an addition process takes longer than a multiplication process. Implementation of the MAC Units gives unrivalled performance in terms of speed and area usage.

The computations carried out on the FPGA were done using fixed point arithmetic which provides the necessary accuracy for the job at hand without requiring the area usage for floating-point implementations.

Least Mean Squares (LMS)

The filter outputs obtained from the FIR block are used by the LMS algorithm to calculate the changes to the filter coefficients, $\Delta h$, required for the next filtering process. When echo data is received from the link it is buffered and upon subtraction from the filter output values, the error term $\Delta(n)$ is obtained. This is used for obtaining the $\Delta h$ values to be added/subtracted from the current filter coefficients. Once the new coefficients are available, an h available flag is asserted informing the FIR block that the new coefficients are available for the next filtering process to initiate. This process is repeated until the error term fed into the system is negligible. The most critical part in the design of the LMS block is the learning factor whose optimum value had to be found by trial and error within the bounds specified by the algorithm. The learning factor determines how fast the algorithm converges. Setting a learning factor that is too large results in the output oscillating due to overshoot, hence convergence is never reached. On the other hand, if the learning factor is too small slow convergence speeds will result, hence increasing the risk of overflow in the input buffers.

IV. TESTING AND RESULTS

The adaptive filter algorithm was first tested using Matlab®, where the system shown in Fig. 2(b) was implemented in software. Fig. 3 shows the results obtained in this case, confirming that the algorithm used performs well in obtaining the original signal from a 'noisy' version. This approach was used to obtain an idea of the value for the learning factor to be used in practice, in this case 0.0602.
A block diagram of the final implementation is shown in Fig. 4. The Digital Phase Locked Loop (DPLL) ensures that the following stages are always running at a constant clock speed. The output denoted (a) is used by the FSK modulator. The block denoted (b) is the subtractor stage which subtracts the echo received from the link, d[n], from the filter output, y[n], to give the error term, e[n], which is then fed to the LMS block. The latter uses e[n] to generate the new set of filter coefficients. There is a continuous exchange between the LMS block and the filter ensuring that both move in lock-step. The filter was then implemented on the FPGA and the device utilisation summary as well as the timing summary are given below.

**Device utilisation summary:**

Selected Device: 3s200ft256-4

- Number of Slices:                    26  out of   1920     1.35%
- Number of Slice Flip Flops:  34  out of   3840     0.885%
- Number of 4 input LUTs:          46  out of   3840     1.2%
- Number of bonded IOBs:           6  out of    173     3.47%
- Number of GCLKs:                       3  out of      8    37.5%

**Timing Summary:**

- Speed Grade: -4
- Minimum period: 5.6 ns (Max. Frequency: 178.56 MHz)
- Minimum input arrival time before clock: 6.876ns
- Maximum output required time after clock: 5.835ns

These results show that the area and speed optimization exercise carried on the algorithms was successful and that very little further optimization is possible.

Various tests were carried out to determine the filter order to be used such that it provides the best efficiency at the data input speed concerned. From the graph shown in Fig. 5, it was concluded that a filter order of 10 gives the best results. Other system tests were performed for different convergence factor values until the value of 0.0589 yielded best convergence results. This is close to the value of 0.0602 obtained through simulation. A simulation result of the output of the adaptive filter during testing of the complete system is shown in Fig. 6.

**V. CONCLUSION**

The system performed as expected and the echo generated by the transceivers was cancelled out by the developed adaptive filter without any loss of data due to overflow. This was shown through the drastic decrease in retransmissions due to collisions that occurred in the presence of echo.

**References**


