LMS Adaptive Filter

February 2012 Reference Design RD1031

Introduction

Adaptive algorithms have become a mainstay in DSP. They are used in wide ranging applications including wireless channel estimation, radar guidance systems, acoustic echo cancellations and many others.

An adaptive algorithm is used to estimate a time varying signal. There are many adaptive algorithms like Recursive Least Square (RLS), Kalman filter, etc. but the most commonly used is the Least Mean Square (LMS) algorithm.

LMS is a simple but powerful algorithm and can be implemented to take advantage of the Lattice FPGA architecture.

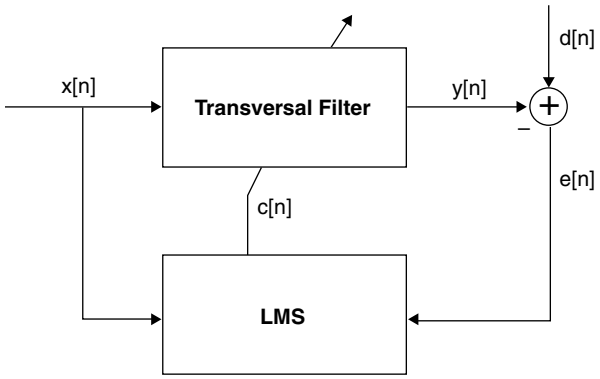
LMS Overview

The LMS algorithm was developed by Windrow and Hoff in 1959. The algorithm uses a gradient descent to estimate a time varying signal. The gradient descent method finds a minimum, if it exists, by taking steps in the direction negative of the gradient. It does so by adjusting the filter coefficients so as to minimize the error.

The gradient is the del operator (partial derivative) and is applied to find the divergence of a function, which is the error with respect to the nth coefficient in this case. The LMS algorithm approaches the minimum of a function to minimize error by taking the negative gradient of the function.

A LMS algorithm can be implemented as shown in Figure 1.

Figure 1. LMS Implementation Using FIR Filter



The desired signal d(n) is tracked by adjusting the filter coefficients c(n). The input reference signal x(n) is a known signal that is fed to the FIR filter. The difference between d(n) and y(n) is the error e(n). The error e(n) is then fed to the LMS algorithm to compute the filter coefficients c(n+1) to iteratively minimize the error.

The following is the LMS equation to compute the FIR coefficients:

$$c(n+1) = c(n) + \mu * e(n) * x(n) \quad (1)$$

The convergence time of the LMS algorithm depends on the step size μ. If μ is small, then it may take a long convergence time and this may defeat the purpose of using an LMS filter. However if μ is too large, the algorithm may never converge. The value of μ should be scientifically computed based on the effects the environment will have on d(n).

Functional Implementation on a Lattice FPGA

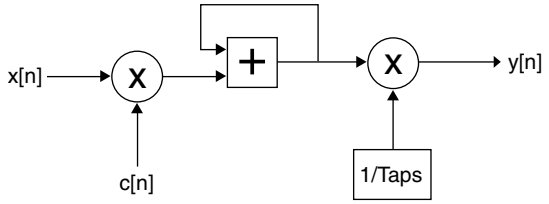
The LMS reference design has the following two main functional blocks:

1. FIR Filter
2. LMS Algorithm

FIR Filter

The FIR filter is implemented serially using a multiplier and an adder with a feedback as shown in the high level schematic in Figure 2. The FIR result is normalized to minimize saturation.

Figure 2. FIR Implementation

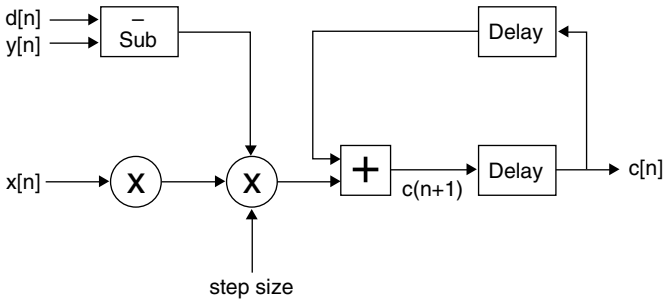


The LMS algorithm iteratively updates the coefficient and feeds it to the FIR filter. The FIR filter than uses this coefficient c(n) along with the input reference signal x(n) to generate the output y(n). The output y(n) is then subtracted from the desired signal d(n) to generate an error, which is used by the LMS algorithm to compute the next set of coefficients.

LMS Algorithm

The LMS algorithm is implemented as shown in Figure 3. The coefficients are calculated according to equation (1). The delay is necessary in the design to separate the current coefficients from the next set of coefficients. The delay size is nearly the same as the tap size. The delay block in Figure 3 is implemented using EBRs to minimize slice utilization, since implementation of one delay of 24-bit data will take about 12 slices.

Figure 3. LMS Algorithm Implementation



Reference Design Features

The LMS reference design can be configured to meet user specifications. This can be achieved by setting the following user-configurable parameters:

- Input data bit width
- Output data bit width
- Binary point
- Tap size
- Step size

All data, except the Tap size, is signed binary point. The input and output data has the same binary point. Step size data is also user configurable using the Lattice gateway-in block in the Simulink® environment. All Lattice input gateway-in blocks should be configured to match the data bit-width and binary point information entered in the LMS reference design block. Users can select various data bit widths to minimize fixed-point effects like saturation, truncation, etc.

Since the FIR filter is implemented serially, the Tap size will dictate the number of clock cycles (cc) required to generate the FIR output. For example, a 128 TAP Fir will take 128 cc to produce each result. This implies that the input sampling rate should be selected as follows:

m = Clock speed of the reference design / (Tap size * Input sampling rate)
m should be Š 1

Therefore, the clock speed of the reference design should be at least Tap size * Input sampling rate. Using the following specifications as an example.

Input sampling rate = 8KHz
Tap size = 256

The reference design should at least be clocked at 256 x 8 = 2.048 MHz for m =1. A higher value for m will result in better estimation.

LMS Filter Application Example

A LMS filter, as previously mentioned, is used to track a time varying signal. The d(n) signal is compared to FIR output y(n), and the error is used by the LMS algorithm to compute the next set of FIR coefficients. The coefficients of the FIR filter estimates the channel response, and so the tap size should be selected such as to model the environment.

One of the many applications of LMS is acoustic echo cancellation. In the following example an echo is generated by delaying the reference signal. The LMS reference design is then used to estimate the delayed signal.

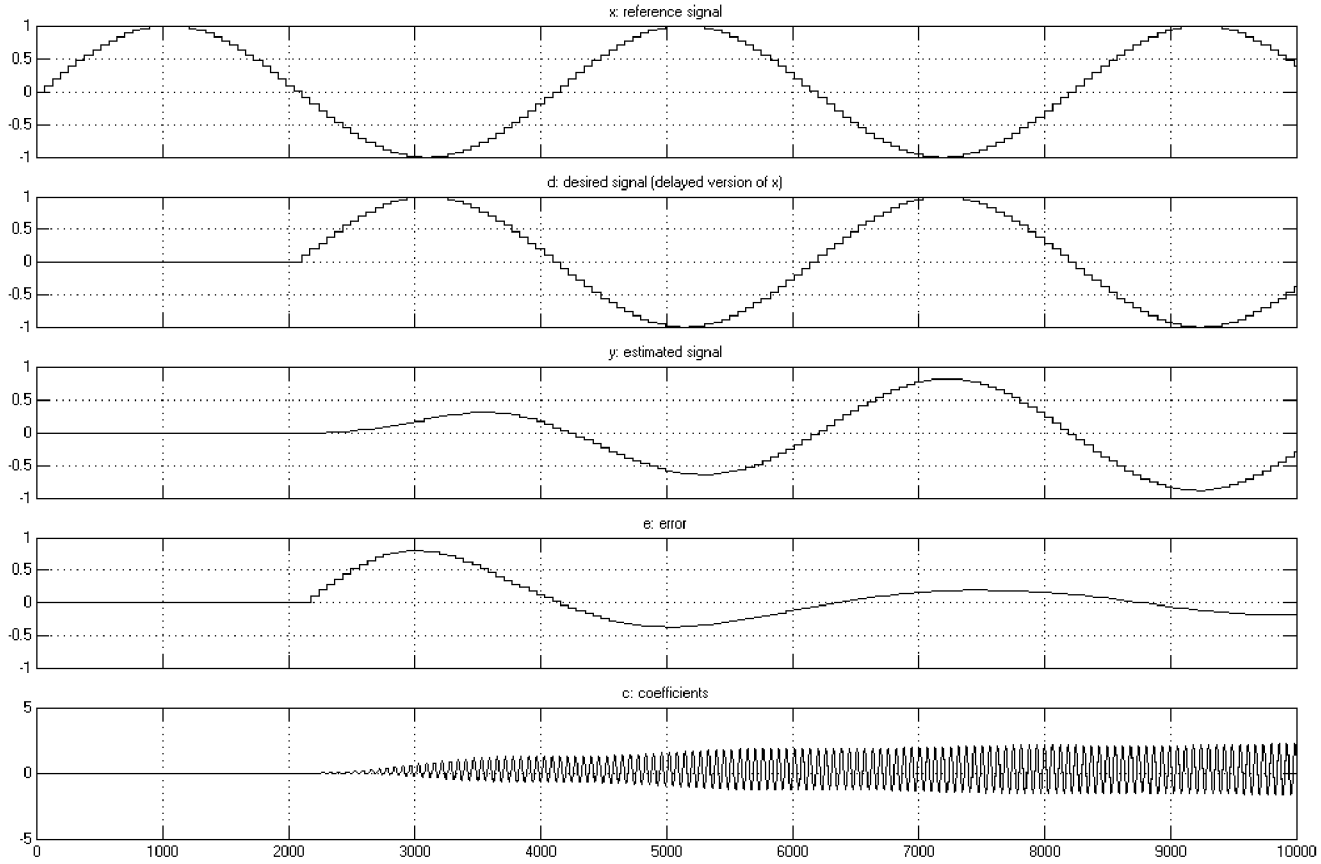
The reference design is configured with the following parameter values:

- Input data bit width: 16
- Output data bit width: 24
- Binary point: 13
- Tap size: 64
- Step size: initial simulation with $\mu = 0.2$ and then with $\mu = 1.0$

The sampling rate is selected such that the value of m = 1.

The MATLAB® simulation results are shown in Figure 4 for $\mu = 0.2$. As shown in Figure 4, it takes more than 4,000 clock cycles for the estimated signal y(n) to closely track the desired signal d(n). This is confirmed by error converging (approaching) to a near zero value after that many clock cycles. The changing (adapting) coefficients values are also shown as they estimate the channel response in real time.

Figure 4. LMS Algorithm MATLAB Simulation with μ = 0.2

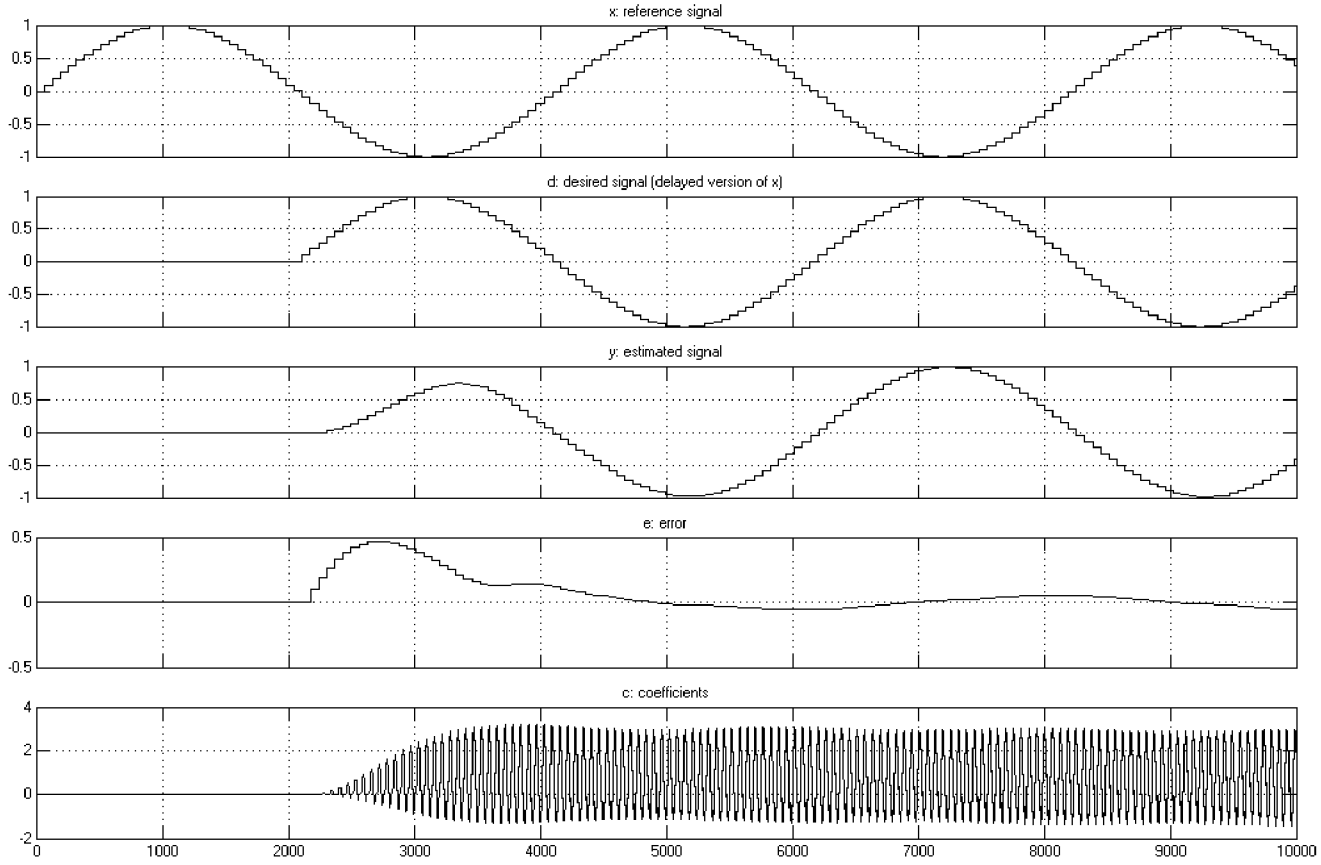


The MATLAB simulation results are again shown in Figure 5 for μ = 1.0. The increase in step size results in a significant improvement in terms of clock cycles needed for an error to converge. This is confirmed in Figure 5, as it now takes about 2,500 clock cycles for the estimated signal y(n) to closely track the desired signal d(n).

The step size, though, cannot be arbitrarily increased. A high value can eventually cause the LMS algorithm to diverge and this causes the error to oscillate with high amplitude. The step size, therefore, has to be scientifically computed based on a number of criteria such as the sampling rate, channel properties, signal properties, etc.

In this particular example, the frequency of the input sample is constant. Therefore, only the change in the value of the input sample mattered. As shown in Figures 4 and 5, the input sample changes by +/- 0.1 every few clock cycles. This change must be adapted by varying the coefficients. The step size was experimentally varied to obtain faster convergence.

Figure 5. LMS Algorithm MATLAB Simulation with μ = 1.0



Resource Utilization

The LMS reference design can be targeted to any Lattice FPGA. The resource utilization shown below is based on the LatticeECP™33 (speed grade -5) device. The following is the resource utilization of the LMS reference design for the specifications mentioned in the echo cancellation example:

Slices: 204
EBRs: 3

sysDSP™ Components:

Mult 36 x 36: 3
Mult 18 x 18: 1
f_{MAX}: 76 MHz

f_{MAX} can be further increased by increasing the pipeline stages. The EBRs are used to implement the delays and to store the input sample. The memory the delay and the input sample it will consume is as follows:

Delay: Tap size * output data bit width
Input Sample: Tap size * input data bit width

Design Implementation Advantages Using Lattice FPGAs

The LatticeECP2™ and LatticeECP2M™ FPGA families have been specifically developed for DSP applications. These devices have embedded sysDSP blocks that can be easily configured to implement a multitude of arithmetic

functions such as multiply and multiply-accumulate. The devices also have embedded memory in the form of EBR blocks.

This reference design takes advantage of the capabilities provided by the LatticeECP2/M devices. The multipliers were implemented using sysDSP blocks, which greatly boosted the performance without taking any LUTs. The delays, as mentioned previously, were implemented using EBRs. This makes it possible to implement an application that requires a Tap size of > 1K. This cannot be done if the delay is implemented using slices.

Design Platform

This reference design was developed using the Lattice ispLEVER® 6.0 and MATLAB 7.1 environments. Users will need MATLAB and Simulink along with ispLEVER software to simulate the design and to generate the HDL.

Pin Description

The following table lists the I/O signals for this reference design.

Signal Name	Active	I/O	Description
xin	—	I	Reference signal
din	—	I	Desired signal
stepsz	—	I	Step size
reset	high	I	Reset
y	—	O	Estimated signal
err	—	O	Error
c	—	O	Coefficients

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
December 2006	01.0	Initial release.
February 2012	01.1	Updated document with new corporate logo.