

An Embedded Adaptive Filtering System on FPGA

Chang Choo, Prasannalakshmi Padmanabhan, Susmita Mutsuddy

Department of Electrical Engineering
San Jose State University
San Jose, CA 95198-0084, USA
(408)924-3980
cychoo@email.sjsu.edu

ABSTRACT

The NLMS adaptive FIR filter is an essential part of many DSP systems including echo cancellers, channel equalizers and noise cancellers. In this paper, we describe an embedded NLMS adaptive filtering system consisting of a flexible LMS core and Xilinx MicroBlaze soft processor, both of which are implemented on a Xilinx Spartan-3 FPGA. The LMS adaptive filter core, which features parameterizable bit-width and tap-length, has been coded in Verilog-HDL, simulated with ModelSim and synthesized and implemented using Xilinx ISE tools. The normalization algorithm is coded in C and runs on the MicroBlaze processor. This embedded system has been implemented in a Xilinx 'Spartan-3 Starter Kit' board using Xilinx EDK toolset. This system works at a clock frequency of 50MHz and uses about 90% of XC3S200 Spartan-3 FPGA Slice resources.

INTRODUCTION

Normalized Least-Mean-Square (NLMS) adaptive FIR filters is the main component of many DSP and communication systems such as echo cancellers, channel equalizers, and noise cancellers. Traditionally, such adaptive filters are implemented in Digital Signal Processors (DSPs). Sometimes, they are implemented in ASICs, when performance is the key requirement. However, many high-performance DSP systems, including NLMS adaptive filters, may be implemented using Field Programmable Gate Arrays (FPGAs) due to some of their attractive advantages. Such advantages include flexibility and programmability, but most of all, availability of tens to hundreds of hardware multipliers available on a chip. Moreover, FPGAs have become more flexible and scalable as they come

with a processor embedded in them. For example, Xilinx Virtex-II Pro device has a Power PC processor embedded in it. There is also an option of embedding a soft processor core, such as MicroBlaze, in various Xilinx FPGAs that do not have the embedded hardcore processor.

MicroBlaze, a 32-bit soft core processor from Xilinx, is a reduced instruction set computer (RISC) and is optimized for the Virtex and Spartan-3-based FPGA implementation. It has three available bus connections: Local Memory Bus (LMB), On-Chip Peripheral Bus (OPB) and Fast Simplex Link (FSL). The FSL interface is an efficient way to communicate to a high-speed hardware due its high bandwidth. Unlike most other RISC processors available in the market (that usually have 2 dynamic inputs and 1 output) it has 16 FSL interface buses, which enables it to be configured with up to 8 inputs and 8 outputs [17].

In MicroBlaze architecture, the instruction and data accesses are done in separate address spaces, each of which is capable of handling up to 4 GB of instructions and data memory. The data and memory interfaces of MicroBlaze are 32 bits wide. It has both on-chip block memory and off-chip memory options available. This processor has the capability to reach up to of 125 Dhrystone MIPS. The MicroBlaze with a standard set of peripherals is included in the Embedded Development Kit (EDK) software tool from Xilinx. The EDK kit includes a complete set of GNU-based software tools with a compiler, assembler, debugger, and linker and also supports various Xilinx FPGA families including Spartan-3 series [9].

In this paper, we describe an embedded NLMS adaptive filter system consisting of a flexible LMS core and Xilinx MicroBlaze soft processor,

both of which are integrated on a Xilinx Spartan-3 FPGA. Thus, the resulting design may be called an embedded DSP system, which is particularly useful for adaptive FIR filter with data bit width and tap length parameterizable.

This paper is organized as follows. In the next section, the architecture of the embedded adaptive filtering system, including the MicroBlaze and the LMS accelerator core and their interface, is briefly described. Software part of the system will also be described. After presenting synthesis results in the following section, we conclude this paper with some concluding remarks.

ARCHITECTURE

The NLMS system consists of a software component, which is the normalization algorithm written in C, and a hardware component consisting of the MicroBlaze processor sub-system and the LMS co-processor core.

The hardware component of the NLMS system architecture, as implemented in the Spartan-3 Starter board (see Figure 1), consists of the following blocks:

MicroBlaze processor sub-system, which consists of a MicroBlaze processor core running the software, controller hardware for interfacing the processor with various peripherals, and a couple of FSL buses to integrate the LMS core with MicroBlaze LMS co-processor core which implements the adaptive FIR filter using the LMS algorithm.

Figure 2 shows the hardware component of the NLMS system as described above.

A. MicroBlaze Processor

This sub-system consists of a MicroBlaze processor core, which runs the normalization software. The software image is stored in the on-board 1MB SRAM, which is interfaced to MicroBlaze by the SRAM controller block. XMD (Xilinx Microprocessor Debugger) core in conjunction with a hardware debug module (OPB_MDM) is used to load the external SRAM with the software image. The SRAM controller is a Xilinx IP core, which controls the data transfer between the memory and the processor via an IBM core-connect standard On-chip Peripheral Bus (OPB). The UART controller is a Xilinx core called as OPB UART lite, which

interfaces the processor with the RS-232 port on the board. RS-232 port of the Spartan-3 board is configured as the system's standard input/output and the results are made available on a hyper-terminal connected to the RS-232 port of the host PC. An 8KB internal Block RAM is used optionally as the cache memory for the external SRAM for temporary code and data storage. This Block RAM also stores a small boot-loop program that is initialized with the bitstream to make the processor wait until the software image is loaded into the external SRAM.

B. LMS Co-Processor Core

The LMS co-processor core is the parameterizable adaptive FIR filter designed using the Least Mean Square (LMS) algorithm. The architecture of the LMS core is described in Figure 3. There is also the FSL-LMS interface controller, which is a state machine that interfaces the LMS core with the FSL buses by which the LMS core is integrated into the MicroBlaze-based NLMS system.

The LMS core mainly consists of data memory, coefficient memory, respective memory controllers, FIR module, and LMS module. All the input vectors from MicroBlaze are first checked for the possibility of an illegal fraction. In particular, if incoming data is 100...0 (illegal number: -1), the number 1 is added to make it 100...01 (the least negative fractional number). This module calculates the error and updates the filter coefficients according to the Least Mean Square (LMS) algorithm.

The error output is the difference between the desired signal $d(n)$ and the filtered output $y(n)$.

$$e(n) = d(n) - y(n)$$

The subtractor output is also checked for overflow/underflow and corrected by similar saturation logic.

The filter coefficients are updated based on the LMS algorithm as follows:

$$\mathbf{w}_k(n+1) = \mathbf{w}_k(n) + \mu(n) e(n) \mathbf{x}(n-k)$$

where k is the position in the filter delay-line

$\mathbf{w}_k(n)$ are the current coefficients

$\mathbf{w}_k(n+1)$ are the updated coefficients

$\mu(n)$ is the convergence factor (normalized) provided by the software running on MicroBlaze

$x(n-k)$ is the delay-line samples provided by the data RAM controller

$\mu(n) e(n) x(n-k)$ is the correction factor

The product $e(n)x(n-k)$ is calculated first and rounded to 16-bits. Then, the multiplication with $\mu(n)$ can be replaced with a shifter because the step-size is usually a power-of-2. Then, this correction factor is added to the current coefficients and the updated coefficients are sent to the coefficient RAM controller to be written to the RAM.

The LMS core is architected so that the coefficient update for previous sample is pipelined with the FIR filtering of current sample. Because of this optimization, for a 64-tap filter for example, the latency of error signal is 66 clock cycles instead of 130.

C. FSL Interface

The FSL bus, 32-bit wide and interfaced directly to MicroBlaze, is dedicated unidirectional point-to-point data streaming interface. This interface, with no arbitration, can be used to send data in and out of the custom hardware. The same FSL link can be used for both data and instruction. A separate bit in the bus indicates whether the information is data or instruction. The performance of the FSL interface can reach up to 300 MB/sec and the throughput depends on the target device itself. The FSL bus has one master and one slave. Currently, the FSL bus only supports synchronous communication. Unlike the LMB and OPB, which are memory-mapped, the FSL interfaces are accessed through simple get and put assembly instructions (also available in blocking and non blocking options) [17].

Figure 4 shows the FSL interface with the available signals. The MicroBlaze with the FSL interface can dramatically enhance the speed and performance of the whole implemented system by employing the most time consuming software in hardware. This makes the design flexible and change at the last moment of the project is permissible making the whole development process less costly. Thus, our whole embedded system consists of the target FPGA containing MicroBlaze with peripheral controllers and LMS custom hardware core, one FSL bus system, SRAM to store instructions and data, and two OPB (On-chip Peripheral Bus) peripherals which are UART lite and MicroBlaze Debug Module.

D. FSL-LMS Interface Controller

The sub-system also consists of a couple of FSL buses to integrate the LMS co-processor with the MicroBlaze system. FSL is a uni-directional point-to-point communication channel bus used to perform fast communication between any two-design elements on the FPGA when implementing an interface to the FSL bus [10]. Up to 8 master and slave FSL interfaces are available on the MicroBlaze. The interfaces are used to transfer data in 2 clock cycles to and from the register file on the processor to hardware running on the FPGA [10]. The FSL bus implements a uni-directional point-to-point FIFO-based communication. It provides mechanism for unshared and non-arbitrated communication mechanism. This can be used for fast transfer of data words between master and slave implementing the FSL interface [10].

In our system, one FSL bus is used to transfer the input signal $x(n)$, desired signal $d(n)$, and normalized convergence step-size $\mu(n)$ from MicroBlaze (Master) to the LMS core (Slave) and another FSL bus to transfer the error signal $e(n)$ from the LMS core (Master) to MicroBlaze (Slave).

E. Software

The software component of the NLMS system consists of a C application written and compiled for MicroBlaze. The normalization algorithm for NLMS is implemented in C. The software reads the data samples, $x(n)$ and $d(n)$ from data memory as stored as include header files and computes the value of the convergence step-size $\mu(n)$ for each sample by normalizing it with the input signal power.

The division for the normalization is done by calling the function '*div*', which performs fixed-point division. Refer to appendix # for the division algorithm. The data samples, $x(n)$ and $d(n)$, and the normalized step-size $\mu(n)$ are then passed to the FSL-LMS Interface Controller logic, and in turn to the LMS adaptive filter core, through a FSL bus. This is done by calling the following driver function with the above inputs one by one.

```
//Blocking Data Write to Local Link no. Id  
microblaze_bwrite_datafsl (val, id)
```

FSL peripherals may be created as a Master or a Slave to the FSL bus. A peripheral connected to the master ports of the FSL bus pushes data and

control signals onto the FSL. All peripherals that act as a master to the FSL bus should create a bus interface of the type MASTER for the bus standard FSL in the Microprocessor Peripheral Description (MPD) file. A peripheral connected to the slave ports of the FSL bus reads and pops data and control signals from the FSL. All peripherals that are a slave to the FSL bus should create a bus interface of the type SLAVE for the bus standard FSL in the MPD file. The put and get instructions of MicroBlaze can be used to transfer the contents of a MicroBlaze register onto the FSL bus and vice-versa and the `microblaze_bread_datafsl` and `microblaze_bwrite_datafsl` driver functions use the 'put' and 'get' assembly instructions. The `bread` and `bwrite` functions use the blocking version of get and put instructions, which makes the processor wait until all the values are written to and all the read from the FSL buses. Therefore, after writing the above 3 samples into the FSL bus, MicroBlaze waits until it receives the error sample $e(n)$ from the LMS adaptive filter core through the second FSL bus. Data is read from the FSL bus using the following driver function.

```
// Blocking Data Read to Local Link no. Id
microblaze_bread_datafsl (val, id)
```

After receiving $e(n)$, it is sent out to the hyper terminal connected to the RS-232 port configured with the same baud rate as the UART lite controller when the system was created.

IMPLEMENTATION

To implement the system, the LMS adaptive filter core and FSL-LMS interface controller core was first coded in Verilog-HDL. Simultaneously, a thorough testbench was written in Verilog-HDL to verify the LMS adaptive filter core. The testbench provides the input test vectors to the core at random intervals simulating the actual processor-based system. Using this testbench, RTL simulations were done using ModelSim called from the Xilinx ISE environment. The values of $y(n)$ and $e(n)$ obtained from ModelSim were then compared with the ones obtained from the MATLAB bit-accurate model with $\mu = 0.25$ in both the cases. The RTL was checked and corrected until the values matched. Once the golden RTL model was available, it was run through the Xilinx XST synthesis tool from ISE, which generate an ngc netlist. Gate-level simulations, using ModelSim, were done on this netlist as a sanity check.

Once the golden LMS core was available, the MicroBlaze processor sub-system, including the processor core, surrounding peripheral controllers, and FSL buses, were generated using EDK. Then, the LMS core and the FSL-LMS interface controllers were incorporated into the embedded system. Because EDK tool supports user logic written only in VHDL, a VHDL wrapper for LMS core was added to the 'pcores' directory, which is the repository for the user-logic files in the EDK project.

The hardware component of the system was then synthesized using the EDK-XST flow, with the LMS core treated as a black box. After synthesis, the ngc netlist of the LMS core was added to the 'implementation' directory in the EDK project, along with the ngo files for the CoreGen components. The LMS core was synthesized with the 'Add I/O buffers' options in ISE turned off. The 'implementation' directory contains the synthesized ngc and other files for the MicroBlaze system.

Then, the system was run through the implementation step, where the components are fit into the target FPGA, placed, and routed. The result of the implementation is an ngd file, which was then run through the 'Generate bitstream' step, which creates a 'system.bit' bit file. This bit file is passed through 'Update bitstream' step, which adds a small bootloop image to system.bit. This bootloop code is stored in Block RAM and enables MicroBlaze to wait until the software image and data are stored in the external SRAM and are ready. The final bit file to be downloaded to the FPGA is called 'download.bit'.

In parallel, the C code for the normalization algorithm was written and compiled for MicroBlaze using the EDK-GNU flow. The output of the compiler is an 'elf' file. A hyper terminal session was opened and configured with the baud rate and other parameters same as the MicroBlaze system.

The final bit file was then downloaded to the FPGA on the Spartan-3 starter kit board using EDK-iMPACT flow. After successful download, an XMD (hardware debugger) session was opened through JTAG and connected to the MicroBlaze system. Then, the 'elf' file was downloaded to the external SRAM and kicked off using this XMD session. We have written a small C code to make the execution of the

NLMS code wait until the user presses one of the push buttons available on the board. Therefore, after a push button was pressed, the software kicks off the LMS adaptive filter core and the $e(n)$ values appear on the hyper terminal. These values were stored in a file and plotted using MATLAB and then compared with the MATLAB model output for convergence.

For debugging, we used Xilinx ChipScope tool [12]. We generated an ICON core and an ILA core in EDK along with the MicroBlaze system and implemented the design with these ChipScope cores. The bitstream was then downloaded to the FPGA and then the ChipScope Analyzer tool was opened and connected to the required internal signals and triggered accordingly.

The ChipScope Pro Analyzer tool interfaces directly to the ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 cores (collectively called the ChipScope Pro cores). One can configure the device, choose triggers, set up the console, and view the results of the capture on the fly. The data views and triggers may be manipulated in many ways, providing an easy and intuitive interface to determine the functionality of the design [12].

Because ChipScope Analyzer and XMD use the same JTAG port, it was not possible to use both of them at the same time. Therefore, just during debugging, the code and data for MicroBlaze was stored in the internal Block RAM instead of external SRAM, which required a XMD session to download code and data.

SYNTHESIS RESULTS

This section describes the implementation results of the NLMS embedded system in terms of area occupied in the Spartan-3 FPGA chip and the clock speed achieved.

Area occupied in XC3S200 FPGA device was:

- LMS Core: 280 slices out of 1920 (14%)
- MicroBlaze System: 1383 slices out of 1920 (72%)
- Total FPGA Utilization: slices – 86%, LUTs – 53%
- Block RAMs – 3 (2 for LMS core memory and 1 for MicroBlaze cache)

- Embedded Multipliers – 2 (one each for the FIR engine and the LMS engine in the LMS adaptive filter core)

Clock Speed achieved was:

- LMS Core: 120MHz
- MicroBlaze System: 50MHz

Table 1 below shows the comparison of a 64-tap adaptive filter implementation in MicroBlaze in Spartan-3 and Virtex-4 FPGAs with the TI C55 DSP currently available in the market. This table compares the above implementations in terms of number of MAC units available versus utilized, clock speed achieved and achievable, latency of $e(n)$ for a 64-tap adaptive FIR filter based on LMS algorithm, and a rough estimate of the price of the chip.

Processor	Device	Number of MAC units	Clock speed (MHz)	Latency for a 64-tap adaptive filter	Price (Volume production)
MicroBlaze	Spartan-3 Starter Board (XC3S200 FPGA)	2	50	66 cycles = 1320 ns	Under \$5
		12	50	12 cycles = 240 ns	
	Virtex-4 SX	128	200	1 cycle = 5ns	Around \$60
DSP	TMS320VC5502	2	200	64 cycles = 320 ns	Around \$10

Table 1: Performance comparison of FPGA and DSP implementation of 64-tap LMS adaptive filter

The first row of the above table shows the performance achieved in our implementation, with MicroBlaze in a Spartan-3 XC3S200 FPGA using only 2 out of 12 multipliers available in this chip. The clock speed achieved is 50MHz and the latency of every $e(n)$ output is 66 clock cycles. The second row projects the estimated performance if our architecture is extended to use all the available multipliers, with 6 multipliers used by the FIR engine and 6 by the LMS engine in parallel. Assuming the clock speed remains the same (similar pipelining stages as the 2-multiplier design), the latency can be reduced to 12 cycles. Comparing this with a comparable TI DSP processor in the market, TMS320VC5502, which has 2 MAC units in

total, it can be seen that even though the DSP processor runs at a higher speed than the FPGA, it is not possible to parallelize the FIR and LMS operations due to the architectural constraint. Therefore, the best case latency of $e(n)$ in the case of the DSP is estimated to be 64 cycle, which is much more than the Spartan-3 implementation utilizing all the 12 multipliers in parallel. The Spartan-3 solution also appears more attractive than the DSP because of its lower cost as advertised.

If performance rather than cost is the key factor in this implementation, the best solution is to use a bigger FPGA with much more embedded multipliers. The third row shows a high-performance Virtex-4 SX FPGA, which has many DSP specific features, including 128 embedded MAC blocks. This makes the filter implementation 'truly' parallel with the latency of $e(n)$ being just 1 cycle.

VERIFICATION

For verification and demonstration of the implemented embedded system, input samples, $x(n)$, were generated by digitizing human voice speech with echo (generated for the echo cancellation application) conforming to the ITU-T recommendation G.168. The voice speech was sampled at 8 KHz/sec and the duration of the speech was about half a second. The MATLAB models, developed using the same logic as the hardware, were used to verify the results of the hardware. The same input signal $x(n)$ and desired signal $d(n)$ were applied to the MATLAB model and also to implemented embedded system and the estimated error signals $e(n)$ generated were compared.

Figure 5 below shows an example of $e(n)$'s from MATLAB model and from the embedded system.

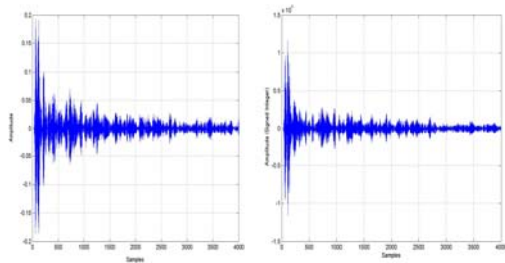


Figure 5: Estimated error signal ($e(n)$) from MATLAB bit-accurate model (left hand side) and from the Spartan 3 FPGA board (right

hand side) in LMS algorithm (Input signal $x(n)$ of 4000 samples)

CONCLUSIONS

We designed a Least Mean Square (LMS) adaptive filter as a flexible core in Verilog-HDL. The LMS core, which has parameterizable bit-width and tap-length has been simulated with ModelSim and implemented using Xilinx ISE (ver. 6.3i) tools. This core acts like a hardware accelerator or co-processor to the MicroBlaze processor core, which runs the normalization algorithm in C. We have implemented the Normalized LMS (NLMS) adaptive filter as an embedded system in Xilinx Spartan-3 Starter Kit Board using EDK (ver. 6.3i) toolset. This system works at a clock frequency of 50MHz and uses 86% (LMS core: 14%, MicroBlaze and surrounding logic: 72%) of XC3S200 Spartan-3 FPGA Slice resources. The embedded system has been verified with actual voice as test vectors and the results from the hardware are found to match the results from the MATLAB bit-accurate model.

Various DSP applications that use NLMS adaptive FIR filters, like Echo Cancellers, Channel Equalizers, and Noise Cancellers, can be added as a software layer on top of our system, without any hardware modifications.

Our project may be extended by one or more of the following ways to support various applications or to further enhance the performance.

- The LMS architecture can be extended to support multiple parallel MAC units by utilizing all the 12 embedded multipliers in the Spartan-3 FPGA. This may be done to fully demonstrate the 'parallelism' advantage of FPGA implementations.
- Amount of pipelining in the LMS core may be increased to further improve speed.
- The software may be further enhanced so that it has the ability to initialize the filter coefficients. This feature may be needed for applications requiring faster convergence.

ACKNOWLEDGMENTS

We deeply thank Xilinx University Program for their generous support of Xilinx EDA tools, hardware, and training to our Electrical

Engineering Department, San Jose State University. Xilinx also hired one of the authors.

BIBLIOGRAPHY/REFERENCES

- [1] Simon Haykin, *Adaptive Filter Theory*, 4th ed., Pearson Education Pte. Ltd., Singapore, 2003.
- [2] Monson H. Hayes, *Statistical Digital Signal Processing and Modeling*, John Wiley and Sons, Inc., NY, 2002.
- [3] Uwe Meyer – Bäse, *Digital Signal Processing with Field Programmable Gate Arrays*, Springer Private Ltd., India, 2001.
- [4] *Spartan-3 Starter Kit Board User Guide*, Xilinx, Inc. UG130 (v1.1), May 2005. Available: <http://direct.xilinx.com/bvdocs/userguide/s/ug130.pdf>
- [5] *Embedded System Tools Reference Manual*, Xilinx, Inc. UG111 (v3.0), Aug 2004. Available: http://www.xilinx.com/ise/embedded/est_rm.pdf
- [6] *MicroBlaze Processor Reference Guide*, Xilinx, Inc. UG081 (v4.0), Aug 2004. Available: http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf
- [7] *Spartan-3 FPGA Family: Complete Data Sheet*, Xilinx, Inc. DS099, Aug 2005. Available: <http://www.xilinx.com/bvdocs/publications/ds099.pdf>
- [8] *Platform Studio User Guide*, Xilinx, Inc. UG113, Aug 2004. Available: http://www.xilinx.com/ise/embedded/edk6_2docs/platform_studio_ug.pdf
- [9] Hans-Peter Rosinger, “Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel”, XAPP529, May 2004. Available: <http://www.xilinx.com/bvdocs/appnotes/xapp529.pdf>
- [10] *Fast Simplex Link (FSL) Bus (v2.00a)*, Xilinx, Inc. DS449, Aug 2004. Available: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/FSL_V20.pdf
- [11] Dr. Chang Y.Choo, “Hardware Accelerator for High-density VoIP Echo Canceller”, GSPx and International Signal Processing Conference, March 31 – April3, 2003, Dallas Texas.
- [12] *ChipScope Pro Software and Cores User Guide*, Xilinx, Inc. UG029, Oct 2004. Available: http://www.xilinx.com/ise/verification/chipscope_pro_sw_cores_6_3i_ug029.pdf
- [13] *ChipScope ILA (Integrated Logic Analyzer)*, Xilinx, Inc. DS284, Jul 2004. Available: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/chipscope_ila.pdf
- [14] <http://www.mathworks.com>
- [15] *Xilinx ISE 6 Software Manuals and Help – HTML Collection*, Xilinx, Inc. <http://toolbox.xilinx.com/docsan/xilinx6/books/manuals.htm>
- [16] MicroBlaze Architecture, Xilinx, Inc. Available: http://www.xilinx.com/ipcenter/processor_central/microblaze/architecture.htm
- [17] Mathew Ouellette & Dr. Dan Connors, “Analysis of Hardware Acceleration in Reconfigurable Embedded Systems,” Available: <http://rogue.colorado.edu/drac/papers/raw05-hybrid.pdf>
- [18] Vinay K. Ingle & John G. Proakis, *Digital Signal Processing using MATLAB*, Brooks/Cole Publishing Company, CA, 2000.
- [19] UcheChukwu Ofoegbu , “Effect of NLMS-Based Noise Cancellation in the Enhancement of Voiced Speech Detection”, December 2004. Available: http://astro.temple.edu/~uchel/EE%20648/Final_report.pdf
- [20] EE 254 lecture note of Dr. Essam Marouf, Department of Electrical Engineering, San Jose State University, Spring 2005
- [21] Kiran Kumar Shetty, “A Novel Algorithm for Uplink Interference Suppression Using Smart Antennas in Mobile Communications.” Electronic Thesis & Dissertations of Florida State University, Feb 2004. Available: http://etd.lib.fsu.edu/theses/available/etd-04092004-143712/unrestricted/Ch_6lms.pdf

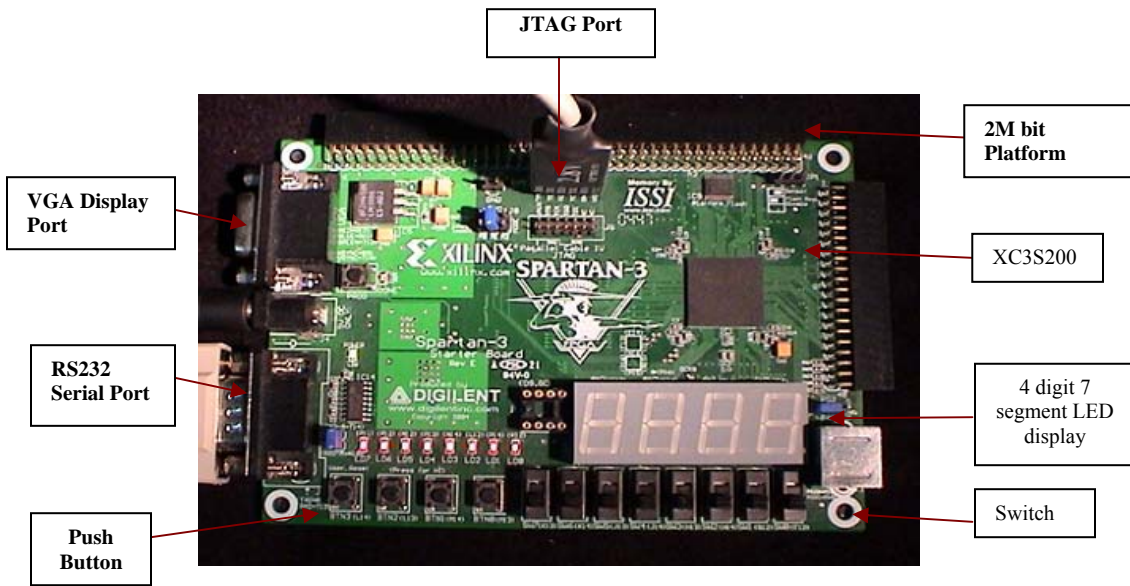


Figure 1: Top view of the Spartan 3 Starter Board

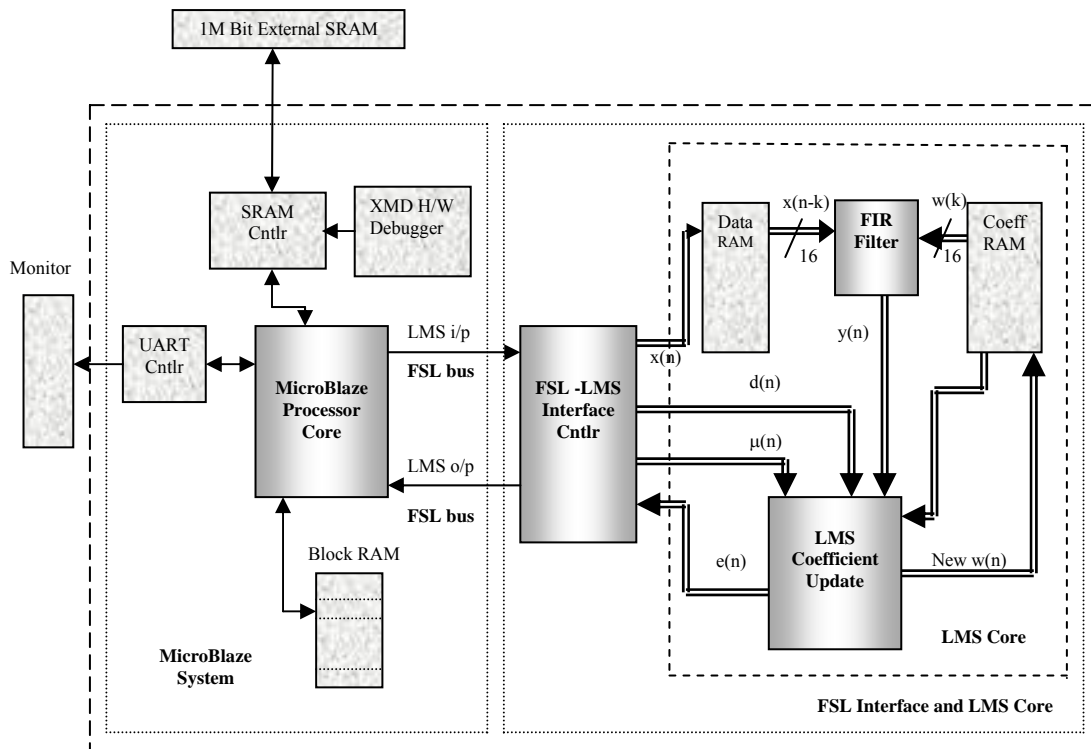


Figure 2: Embedded NLMS System Architecture

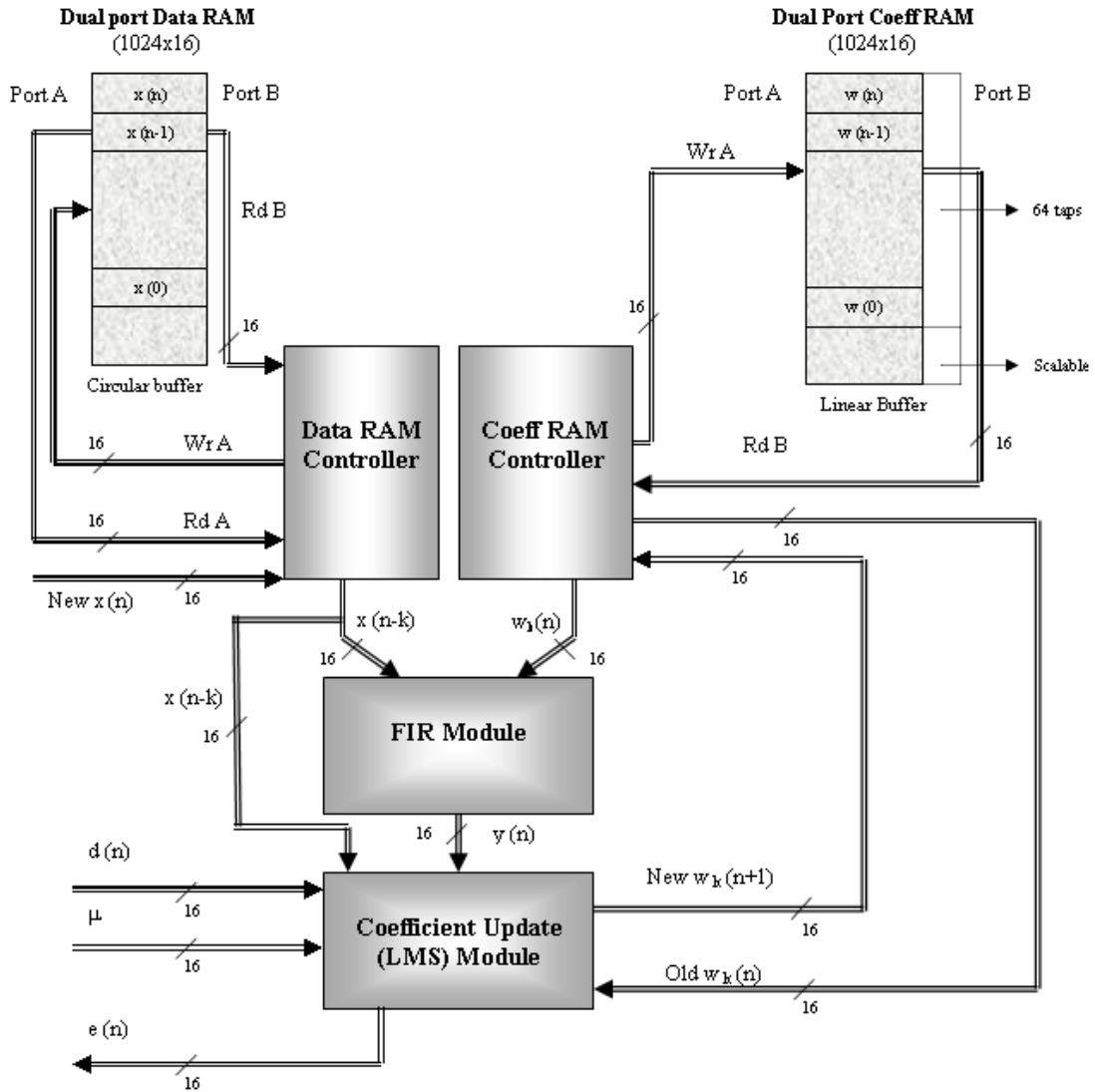


Figure 3: Architecture of the LMS Adaptive Filter Core

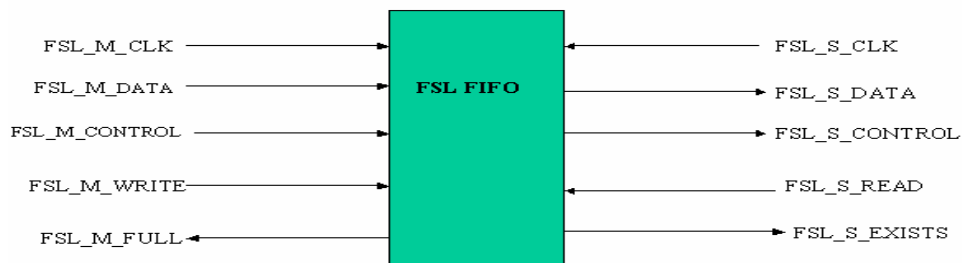


Figure 4: FSL Interface [9]