NLMS Algorithm: Theory, Implementation & FPGA Design

With LMS Comparison and MATLAB/Vivado Integration

Project Done By:

V Kumar

Vipin Kumar Mishra

Project Mentor:

Rituparna Choudhury Mam

Assistant Professor, IIIT Bangalore

International Institute of Information Technology Bangalore

November 27, 2025

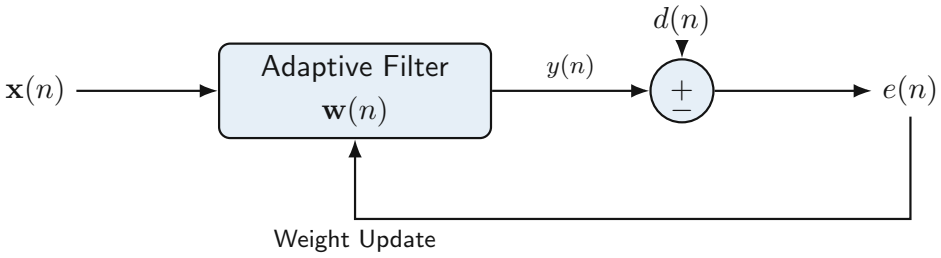
Outline

1. Adaptive Filtering Basics
2. LMS Algorithm & Limitations
3. Need for NLMS
4. NLMS Derivation & Formulas
5. LMS vs NLMS Comparison
6. NLMS Algorithm & Flowchart
7. MATLAB Data Generation for FPGA
8. Vivado Testbench Flow
9. NLMS on FPGA (Custom RTL, No IP)
10. NLMS on FPGA (Using IP / HLS)
11. Results, Discussion & Conclusion

Adaptive Filtering Basics

- Input vector:
$$\mathbf{x}(n) = [x(n), x(n-1), \ldots, x(n-L+1)]^T$$
- Weight vector:
$$\mathbf{w}(n) = [w_0(n), w_1(n), \ldots, w_{L-1}(n)]^T$$
- Filter output:
$$y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$
- Error signal:
$$e(n) = d(n) - y(n)$$
- Goal: adapt $\mathbf{w}(n)$ to minimize mean-square error $\mathbb{E}[e^2(n)]$.

Adaptive Filter Block Diagram



LMS Algorithm: Formulation

- Cost function:

$$J(n) = \frac{1}{2} e^2(n)$$

- Stochastic gradient descent update:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\, e(n)\mathbf{x}(n)$$

- Convergence condition:

$$0 < \mu < \frac{2}{\lambda_{\max}},$$

where $\lambda_{\max}$ is largest eigenvalue of $R = \mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)]$.

Limitations of LMS

- Fixed step-size μ:
  - If μ is large $\Rightarrow$ possible divergence.
  - If μ is small $\Rightarrow$ slow convergence.
- Behavior heavily depends on $\|\mathbf{x}(n)\|^2$:
  - Large input power $\Rightarrow$ unstable updates.
  - Small input power $\Rightarrow$ tiny updates.
- Needs retuning of μ when input statistics change.
- Not robust for real-time FPGA systems where inputs can vary widely.

Need for NLMS

- Idea: make the effective step-size depend on the input power.
- Define effective step-size:

$$\mu_{\text{eff}}(n) = \frac{\mu}{\|\mathbf{x}(n)\|^2}$$

- This leads to:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu_{\text{eff}}(n)\, e(n)\mathbf{x}(n)$$

- Benefits:
  - Robust to input power changes.
  - Faster and more stable convergence.
  - Well suited for fixed-point FPGA implementation.

NLMS Derivation – Step 1

Optimization viewpoint:

- Find new weights $\mathbf{w}(n+1)$ as close as possible to old weights $\mathbf{w}(n)$
- Subject to satisfying the current input/output relation.

Optimization problem:

$$\mathbf{w}(n+1) = \arg\min_{\mathbf{w}} \|\mathbf{w} - \mathbf{w}(n)\|^2$$

subject to

$$d(n) - \mathbf{w}^T\mathbf{x}(n) = 0.$$

Use Lagrange multiplier λ:

$$J(\mathbf{w}) = \|\mathbf{w} - \mathbf{w}(n)\|^2 + \lambda\big(d(n) - \mathbf{w}^T\mathbf{x}(n)\big).$$

NLMS Derivation – Step 2

Take gradient w.r.t. $\mathbf{w}$ and set to zero:

$$\frac{\partial J}{\partial \mathbf{w}} = 2(\mathbf{w} - \mathbf{w}(n)) - \lambda \mathbf{x}(n) = 0.$$

Hence,

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\lambda}{2}\,\mathbf{x}(n).$$

Apply constraint $d(n) - \mathbf{w}^T(n+1)\mathbf{x}(n) = 0$:

$$d(n) - \left(\mathbf{w}(n) + \tfrac{\lambda}{2}\mathbf{x}(n)\right)^T \mathbf{x}(n) = 0.$$

Using $e(n) = d(n) - \mathbf{w}^T(n)\mathbf{x}(n)$, we obtain:

$$\lambda = \frac{2e(n)}{\|\mathbf{x}(n)\|^2}.$$

Final NLMS Update Equation

Substitute λ back:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{e(n)\mathbf{x}(n)}{\|\mathbf{x}(n)\|^2}.$$

Practical NLMS:

$$\boxed{\mathbf{w}(n+1) = \mathbf{w}(n) + \tilde{\mu}\frac{e(n)\mathbf{x}(n)}{\|\mathbf{x}(n)\|^2 + \delta}}$$

- $\tilde{\mu}$: normalized step-size $(0 < \tilde{\mu} < 2)$
- δ: small regularization constant to avoid division-by-zero

LMS vs NLMS: Conceptual Difference

LMS

- Update:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\, e(n)\mathbf{x}(n)$$

- Fixed μ
- Sensitive to signal power
- Convergence behavior changes if input statistics change

NLMS

- Update:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \tilde{\mu}\frac{e(n)\mathbf{x}(n)}{\|\mathbf{x}(n)\|^2 + \delta}$$

- Effective step-size depends on $\|\mathbf{x}(n)\|^2$
- Stable for a wide range of signals
- Faster convergence in practice

LMS vs NLMS: Compact Summary Comparison

LMS

Advantages

- Simple architecture
- Low computation
- Minimal hardware

Disadvantages

- Step-size sensitive
- Slow convergence
- Performance varies with input power

Applications

- Basic ANC
- System ID

NLMS

Advantages

- Auto-normalized step-size
- Fast, stable convergence
- Robust to signal variations

Disadvantages

- Needs divider hardware
- More quantization-sensitive
- Slightly higher complexity

Applications

- High-performance ANC/AEC
- Adaptive equalization

NLMS Algorithm – Step-by-Step

1. Form the input vector:

$$\mathbf{x}(n) = [x(n), x(n-1), \ldots, x(n-L+1)]^T$$

2. Compute filter output:

$$y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$

3. Compute error:

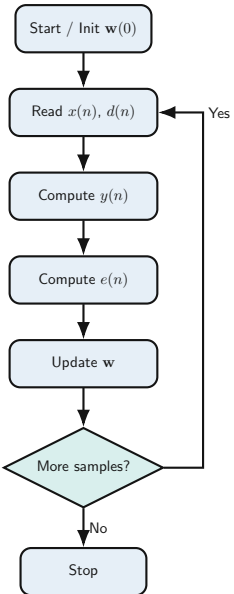
$$e(n) = d(n) - y(n)$$

4. Update weights:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \tilde{\mu}\frac{e(n)\mathbf{x}(n)}{\|\mathbf{x}(n)\|^2 + \delta}$$

5. Repeat for all samples n.

NLMS Flowchart



MATLAB: Generating Input and Desired Signals

To test NLMS on FPGA, we generate $x(n)$ and $d(n)$ in MATLAB.

Example:

```
N = 2000;
x = randn(1, N);          % Input signal
h = fir1(31, 0.4);        % Unknown system
d = filter(h, 1, x);      % Desired signal (noiseless)

data = [x.' d.'];         % Column-wise
writematrix(data, 'nlms_input.csv');
```

- $x(n)$: excitation signal.
- $d(n)$: output of unknown system (to be identified).
- We later convert these to fixed-point for FPGA.

MATLAB: Fixed-Point Conversion for FPGA

For FPGA implementation, use fixed-point format (e.g. Q1.15):

```
x_fx = fi(x, 1, 16, 15);   % signed, 16-bit, 15 fractional bits
d_fx = fi(d, 1, 16, 15);

data_fx = [x_fx.int16 d_fx.int16];
writematrix(data_fx, 'nlms_fixed.txt');
```

Key points:

- Q1.15 range is approximately $[-1, 1)$.
- Scaling is chosen so that no overflow occurs.
- File `nlms_fixed.txt` is read by Vivado testbench.

Vivado: Testbench Flow

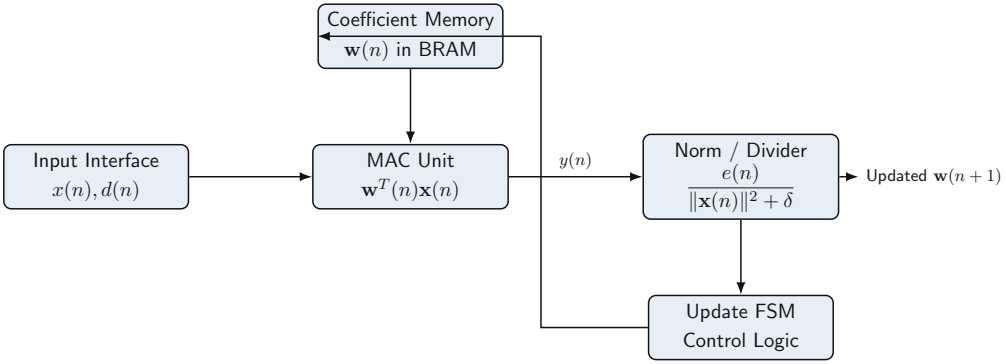
1. Add `nlms_fixed.txt` as a simulation data file.
2. Testbench reads one line per clock cycle:
   - First column $\rightarrow x(n)$
   - Second column $\rightarrow d(n)$
3. Apply $x(n)$, $d(n)$ to NLMS DUT (Design Under Test).
4. Capture outputs:
   - Filter output $y(n)$
   - Error $e(n)$
5. Optionally write $y(n)$ and $e(n)$ to text files.
6. Compare FPGA outputs with MATLAB results (for verification).

NLMS on FPGA (No IP) – Custom RTL

Custom RTL approach:

- All modules written manually in Verilog/VHDL:
  - MAC unit for $\mathbf{w}^T(n)\mathbf{x}(n)$
  - Norm computation for $\|\mathbf{x}(n)\|^2$
  - Division/reciprocal block
  - Coefficient memory (BRAM)
  - FSM for update sequencing
- Pros:
  - Full control of architecture and timing
  - Can be highly optimized for your filter order
- Cons:
  - Longer development time
  - More complex to debug

Custom RTL NLMS Block Diagram



NLMS on FPGA (With IP) – IP/HLS Based

Instead of fully custom RTL, we can use Xilinx IP cores:

- DSP48 MAC IP for multiplication and accumulation.
- Divider / Reciprocal IP for normalization.
- BRAM controller IP for coefficient storage.
- HLS-generated NLMS core:
  - Written in C/C++
  - Synthesized to RTL using Vivado HLS / Vitis HLS

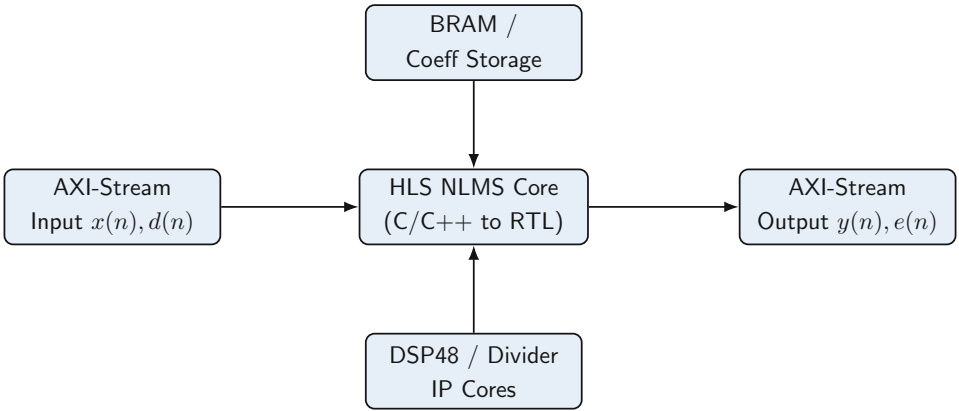
Pros:

- Rapid development
- Automated pipelining and resource sharing

Cons:

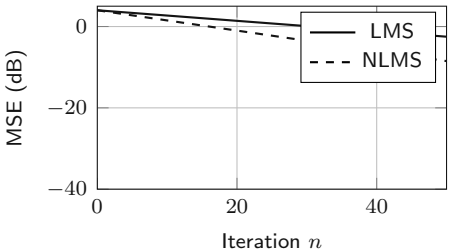
- Less fine-grained control
- Potentially higher resource usage than hand-optimized RTL

IP-Based NLMS Block Diagram



Conceptual Convergence: LMS vs NLMS

- Mean-Square Error (MSE) vs iteration.
- NLMS converges faster due to normalized step-size.
- LMS may require very small μ to stay stable, which slows learning.



Resource and Timing – Conceptual Comparison

Metric	Custom RTL	IP/HLS Design
Design control	Very high	Medium
Development time	Long	Short
DSP slice usage	Optimized	Slightly higher
BRAM usage	Tunable	Higher (for generic cores)
Max frequency	Can be very high	Depends on IP settings
Portability	HDL-specific	Easier to retarget

These trends help decide which architecture to use for a given FPGA project.

Overall Design Flow Summary

1. **Theory:** Understand LMS and need for NLMS.
2. **Derivation:** Obtain NLMS update formula.
3. **Simulation in MATLAB:**
   - Generate $x(n)$ and $d(n)$.
   - Verify NLMS convergence in floating-point.
4. **Fixed-Point Conversion:**
   - Convert signals and coefficients to Q1.15.
5. **FPGA Implementation:**
   - Option 1: Custom RTL NLMS.
   - Option 2: IP/HLS-based NLMS.
6. **Verification:**
   - Compare FPGA outputs with MATLAB.

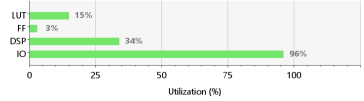
Resource Utilization Comparison (With IP vs Without IP)

Resource Utilization Results

Without IP

Summary

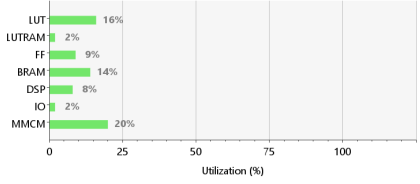
Resource	Utilization	Available	Utilization %
LUT	3026	20800	14.55
FF	1308	41600	3.14
DSP	31	90	34.44
IO	102	106	96.23



With IP

Summary

Resource	Utilization	Available	Utilization %
LUT	3331	20800	16.01
LUTRAM	171	9600	1.78
FF	3925	41600	9.44
BRAM	7	50	14.00
DSP	7	90	7.78
IO	2	106	1.89
MMCM	1	5	20.00



Timing Comparison (With IP vs Without IP)

Timing / Frequency / Slack Results

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.790 ns	Worst Hold Slack (WHS):	0.161 ns	Worst Pulse Width Slack (WPWS):	74.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2358	Total Number of Endpoints:	2358	Total Number of Endpoints:	1309

All user specified timing constraints are met.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.390 ns	Worst Hold Slack (WHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	7623	Total Number of Endpoints:	7623	Total Number of Endpoints:	3899

All user specified timing constraints are met.

Power Consumption Comparison (With IP vs Without IP)

Power Analysis Results

Without IP

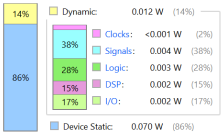
Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.082 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (11.9 W)
Ambient Temperature:	25.0 °C
Effective θJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



With IP

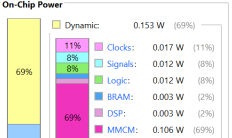
Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.223 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**26.1°C**
Thermal Margin:	58.9°C (11.7 W)
Ambient Temperature:	25.0 °C
Effective θJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

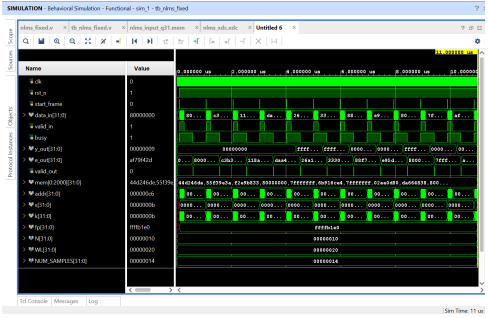
On-Chip Power



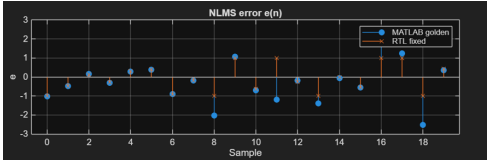
Simulation Waveform, RTL vs MATLAB Reference

Vivado Simulation RTL vs MATLAB Expected Output

Vivado Simulation



MATLAB Expected Output



Conclusion:

- NLMS solves the main limitations of LMS by normalizing the update with input power.
- FPGA implementation can be done using:
  - Custom RTL (no IP)
  - IP-based / HLS approach
- MATLAB and Vivado together form a complete verification flow.

Future Work:

- Implement variable step-size NLMS.
- Explore fixed-point optimization for lower word-length.
- Extend design to multi-channel adaptive filtering.

Questions?