CLOSED-FORM AND REAL-TIME WORDLENGTH ADAPTATION

Paul D. Fiore, Li Lee
Sanders, A Lockheed Martin Company, Nashua, NH
and Massachusetts Institute of Technology, Cambridge MA
pfiore@sanders.com, llee@mit.edu

ABSTRACT

FPGA and configurable computing-based DSP algorithms have demonstrated significant performance improvements over software implementations. This has caused recent renewed interest in developing or mapping DSP algorithms to custom hardware. An algorithm will be successfully mapped if the intermediate wordlengths can be reduced to maintain reasonable hardware size. In this paper, we consider linear hardware cost functions, for which we can derive closed-form expressions for the reduced wordlengths. We then apply these results to an adaptive LMS filter, where we adapt not only the tap weights, but also the wordlengths as a function of the data in real-time.

1. INTRODUCTION

Recent advances in field programmable gate array (FPGA) [1, 2] and configurable computing technology [3, 4] have caused renewed interest in custom digital signal processing (DSP) hardware designs. For many DSP problems, reduced precision arithmetic will maintain acceptable system performance. FPGA-based DSP computers can be programmed to bit-level granularity, allowing for a more efficient implementation than is possible for a conventional programmable computer [5]. A mapping of an algorithm to an FPGA architecture will be successful if the designer can limit wordlength growth without sacrificing algorithm performance.

In order to reduce hardware requirements, one must round or truncate data at various points throughout an algorithm. Many operations such as multiplication or accumulation can greatly expand the wordlength of the data. If these operations are cascaded, then without some form of wordlength reduction, excessive hardware growth results. Wordlength reduction introduces noise into the data stream, so the designer must balance the need for an efficient implementation with output quality.

In order to perform this tradeoff, we treat the special case of a linear hardware cost metric (which can be applied to adders and multipliers). We model the effect of truncation as the injection of additive white uniform noise, which leads to a convenient performance metric. The algorithm wordlengths are determined by minimizing one of these two metrics subject to constraints on the other. In our previous work [5] we developed an ad hoc method for this optimization; in this paper, we derive closed-form expressions for the optimal wordlengths, which would be suitable for any FPGA or ASIC technology. Additionally, we then show that these expressions can be efficiently calculated in hardware, leading to wordlengths that change in real-time as a function of the data. We show that this result can be directly applied to several adaptive FIR filter architectures.

2. CLOSED-FORM SOLUTION

We consider two possible formulations. In the first, the hardware cost is minimized subject to constraints on the output noise variance. In the second, the output noise variance is minimized subject to a hardware constraint. In both cases, the unknown variable is the vector of wordlengths $\mathbf{b} = [b_1, b_2, ..., b_N]^T$ to be assigned at the desired locations in the algorithm. The hardware cost function $C(\mathbf{b})$ and the output noise variance $V(\mathbf{b})$ takes the form

$$C(\mathbf{b}) = \sum_{i=1}^{N} c_i b_i, \quad V(\mathbf{b}) = \sum_{i=1}^{N} \alpha_i 2^{-2b_i}. \tag{1}$$

The form of $V(\mathbf{b})$ results from a linearization of the algorithm that is being mapped about an operating point. Essentially, the α_i are scaled versions of the maximum squared slope of the transfer function from the point of rounding to the algorithm output. The 2^{-2b_i} term results from assuming uniform additive noise [6]. Notice that $C(\mathbf{b})$ is a linear function of $\mathbf{b}$, and $V(\mathbf{b})$ is a convex function of $\mathbf{b}$. We will assume that the cost coefficients c_i, and the variance injection scale factors α_i are known.

2.1. Minimize Cost Subject to Variance Constraints

In this section we consider the problem of minimizing the hardware cost subject to a constraint on the maximum allowable noise variance:

$$\begin{gathered} \min C(\mathbf{b}), \\ \text{subject to} \quad V(\mathbf{b}) \leq A \text{ and } \mathbf{b} \geq \mathbf{b_{min}} \geq \mathbf{0}. \end{gathered} \tag{2}$$

Notice that we have not imposed integrality on the b_i's, and that this is the optimization of a linear function over a convex set.

To obtain a closed-form expression, we first relax the minimum wordlength constraints in (2). Using a Lagrange multiplier, we minimize

$$J = \sum_{i=1}^{N} c_i b_i + \lambda \sum_{i=1}^{N} \alpha_i 2^{-2b_i}. \tag{3}$$

Taking the derivative and setting equal to zero yields

$$\frac{\partial J}{\partial b_k} = c_k + \lambda \alpha_k 2^{-2b_k} \ln(2)(-2) = 0, \tag{4}$$

which gives

$$\lambda = \frac{c_k}{\alpha_k 2 \ln(2)} 2^{2b_k}, \quad k = 1, \ldots, N. \tag{5}$$

We can thus equate the right-hand side of (5) for different values of k, specifically

$$\frac{c_1}{\alpha_1}2^{2b_1} = \frac{c_k}{\alpha_k}2^{2b_k}. \tag{6}$$

Rearrangement gives

$$\left(\frac{c_k}{c_1}\right)\left(\frac{\alpha_1}{\alpha_k}\right) = 2^{2(b_1-b_k)}. \tag{7}$$

Taking logarithms and rearranging finally gives

$$b_k = b_1 + \frac{1}{2}\log_2\left(\frac{c_1}{\alpha_1}\frac{\alpha_k}{c_k}\right) \quad k = 1,\ldots,N. \tag{8}$$

Thus, (8) allows us to find all the wordlengths from knowledge of b_1. We can actually derive b_1 as well as the remaining b_k by substituting (8) into (2):

$$\begin{aligned} A &= \textstyle\sum_{i=1}^{N} \alpha_i 2^{-2b_i} \\ &= \textstyle\sum_{i=1}^{N} \alpha_i 2^{-2\left(b_1+\frac{1}{2}\log_2\frac{c_1}{\alpha_1}\frac{\alpha_i}{c_i}\right)} \\ &= 2^{-2b_1}\frac{\alpha_1}{c_1}\textstyle\sum_{i=1}^{N} c_i. \end{aligned} \tag{9}$$

Solving for b_1,

$$b_1 = \frac{1}{2}\log_2\left(\frac{\alpha_1}{c_1}\frac{1}{A}\sum_{i=1}^{N} c_i\right). \tag{10}$$

From (8) we therefore have

$$b_k = \frac{1}{2}\log_2\left(\frac{\alpha_1}{c_1}\frac{1}{A}\sum_{i=1}^{N} c_i\right) + \frac{1}{2}\log_2\left(\frac{c_1}{\alpha_1}\frac{\alpha_k}{c_k}\right). \tag{11}$$

Finally, we simplify this expression as

$$b_k = \frac{1}{2}\log_2\left(\frac{\alpha_k}{c_k}\right) + \frac{1}{2}\log_2\left(\frac{\sum c_i}{A}\right). \tag{12}$$

It is important to note that (12) separates the wordlength formula into a variable portion (e.g. changes with k), and a portion that is common to all b_k.

A shortcoming of (12) is that the wordlengths may violate the minimum wordlength constraints of (2). We resolve this by an iterative approach in which those b_k's that violate the constraint are fixed to be $b_{min,k}$, and then (12) is applied again to those variables which are not already assigned. Another problem is that the b_k's are not necessarily integral. This is easily remedied by taking the ceiling of the expressions.

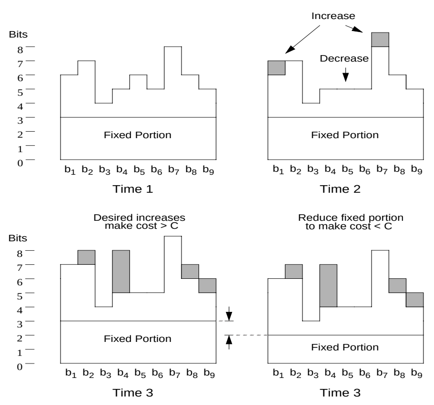
2.2. Minimize Variance Subject to Cost Constraints

Now we consider the problem of minimizing the roundoff noise variance subject to a constraint on the maximum allowed hardware cost:

$$\begin{gathered} \min V(\mathbf{b}), \\ \text{subject to } \; C(\mathbf{b}) = C \text{ and } \mathbf{b} \geq \mathbf{b_{min}} \geq \mathbf{0}. \end{gathered} \tag{13}$$

Notice that again we have not imposed integrality on the b_i's, and that this is the optimization of a convex function over a linear space, for which well-defined optimality conditions exist.

Relaxing the minimum wordlength constraint of (13) and using a Lagrange multiplier, we obtain exactly the same expression as in (8). We now perform manipulations similar to those used to derive (12), obtaining



Increase
Decrease
Bits
8
7
6
5
4
3
2
1
0
Fixed Portion
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
Time 1
Time 2
Desired increases make cost > C
Reduce fixed portion to make cost < C
Time 3

Figure 1. Adaptively varying wordlengths.

$$b_k = \frac{1}{2}\log_2\left(\frac{\alpha_k}{c_k}\right) + \frac{C + \frac{1}{2}\sum c_i \log_2 \frac{c_i}{\alpha_i}}{\sum c_i}. \tag{14}$$

We again ensure that the wordlength allocations do not violate the minimum wordlength constraints of (13), and also ensure integrality by taking the floor of the results.

3. REAL-TIME WORDLENGTH ADAPTATION

Comparing (14) to (12), we see that the components that vary with k are identical. We can take advantage of this for an adaptive computing implementation, where the wordlengths are to be calculated as a function of the data in real-time.

For example, suppose we are operating in a regime where the costs c_i, the variance injection parameters α_i and the desired variance A are such that reasonable wordlengths result. In this case, the total cost will be less than C. The design method will produce b_k's that minimize the total hardware cost, allowing perhaps other hardware algorithms to share the hardware computing resources, or perhaps allowing for a lower power utilization. If only the α_k's are allowed to vary, then the common term in (12) remains constant and can be precalculated.

If the data becomes worse in some way, causing the c_k and α_k's to be less favorable, the total cost may exceed C. In this case we could use (14) to calculate the wordlengths. However, because the k-varying portions of (12) and (14) are identical, the common term in (14) must act to uniformly reduce the wordlengths so that the total cost is less than C. We thus do not need to calculate that common term; we merely reduce each wordlength by the same amount until the cost target is met. In this regime, we will not be able to meet the noise variance target A, but we will come as close as we can given the constraint on total available hardware.

Figure 1 illustrates this idea. At time 1 in the figure, certain wordlengths are allocated. At time 2, we increase some of the wordlengths according to (12). At time 3, we desire to further increase some wordlengths, but this would cause the total hardware cost to exceed the bound C. At this point, we uniformly decrease the common portion for all wordlengths until the total cost bound is met. If the data improves so that we are substantially below our cost boundary, then we uniformly add bits back to the common portion to improve our noise variance.

A simpler, approximate mechanization of this approach is to use two thresholds. We try to keep the filter wordlengths "between" the two thresholds. For example, we may require that maximum wordlength is between the thresholds

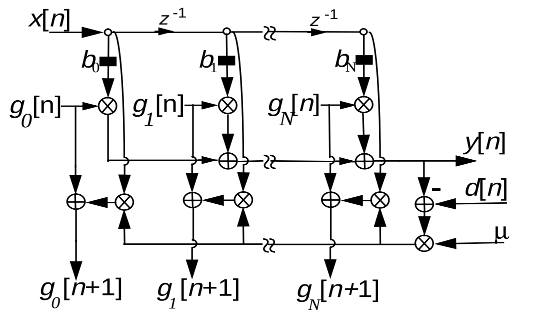


Figure 2. LMS adaptive FIR filter.

(or perhaps some minimum number of b_k's are between the thresholds). If the maximum wordlength crosses the upper threshold from below, then we reduce the common portion of the wordlength for each b_k. If the data improves so that the b_k's are reduced individually, then eventually the maximum wordlength will drop below the lower threshold from above. At this point, we increase the common portion of the wordlength for each b_k. In each of these cases, we act to bring the maximum wordlength back to between the two thresholds.

Now examine the k-varying portion of the wordlength calculation:

$$\frac{1}{2}\log_2\left(\frac{\alpha_k}{c_k}\right) = \frac{1}{2}\log_2 \alpha_k - \frac{1}{2}\log_2 c_k. \qquad (15)$$

We can approximate the calculation of the varying portion by locating the lead-bit location of α_k and c_k, and then scaling and subtracting these numbers. The cost of these operations in terms of hardware will generally be small compared to the cost of implementing the total algorithm.

4. CONFIGURABLE COMPUTING ADAPTIVE FILTERS

We have implemented real-time wordlength adaptation in the problem of adaptive equalization of an unknown linear dispersive channel. The least-mean-square (LMS) algorithm is used to update a finite impulse response (FIR) filter, and the wordlengths of inputs to each of the filter taps are dynamically varied as the filter taps change.

The LMS algorithm, described in great detail in [7], gradually changes the taps of an FIR filter to minimize the error between the actual ($y[n]$) and the desired ($d[n]$) output signals. Let $x[n]$ denote the input at time n, and $g_k[n]$ be the value of the k-th filter tap at time n. The filter output $y[n]$ is computed as

$$y[n] = \sum_{k=0}^{N} g_k[n]x[n-k], \qquad (16)$$

and the filter taps are adjusted according to

$$g_k[n+1] = g_k[n] + \mu x[n-k](d[n]-y[n]), \qquad (17)$$

where μ is an appropriately chosen stepsize parameter.

We implemented the LMS algorithm with wordlength adaptation using the structure shown in Figure 2. In the figure, the input multiplying $g_k[n]$ is truncated to b_k bits prior to multiplication, and b_k is re-calculated according to (12) or (14) as the filter tap $g_k[n]$ is updated. The variance injection scale factors α_k are therefore simply $g_k[n]^2/12$.

While there are many ways of implementing LMS in hardware, we have chosen to use a simple bit-serial implementation, which is highly area-efficient. A block diagram of a bit-serial multiplier is shown in Figure 3. At each clock cycle, 1 bit of the input is gated with a corresponding filter coefficient value, and the result is added to the accumulator. The total number of clock cycles needed to generate one output value is therefore the sum of the number of bits allocated to the inputs ($\sum b_i$). Hence, by setting all of the cost coefficients c_i to be 1 we can either minimize the calculation time subject to variance constraint or minimize the variance subject to time cost constraints.

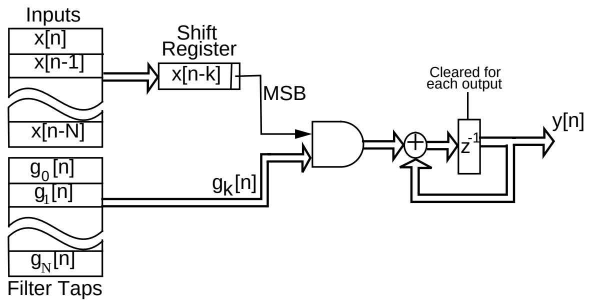


Figure 3. Bit-serial implementation of an FIR filter.

5. EXPERIMENTS

In this section we show the results of implementing wordlength adaptation on the bit-serial adaptive LMS filter. Our experiments are based on those presented in Section 9.13 of [7]. Specifically, the system is a simple adaptive equalizer with input

$$x[n] = h[n] * a[n] + v[n], \qquad (18)$$

where $a[n]$ is a real single-bit signal ($a[n] = \pm 1$), $h[n]$ is the impulse response of the channel:

$$h[n] = \begin{cases} \frac{1}{2}\left[1+\cos\left(\frac{2\pi}{2.7}(n-2)\right)\right], & n = 1,2,3 \\ 0, & \text{otherwise,} \end{cases} \qquad (19)$$

and $v[n]$ is Gaussian white noise with zero-mean and variance 0.001. $g_k[n]$ has 11 non-zero taps, and the desired signal $d[n]$ is a delayed version of $a[n]$. The implementation used a stepsize μ of 0.0625. Unless otherwise noted, all fixed-point implementations used 9-bits as the default wordlength.

For brevity, we show the results for wordlength adaptation only for the case of minimizing variance subject to cost constraints. This is directly applicable to a bit-serial implementation, where the cost constraint corresponds to the amount of time permitted for each output sample. With $c_k = 1$ and $\alpha_k = g_k[n]^2/12$, we note that the k-varying portion of (14) is $\log_2 g_k[n]$, which can be approximated using a lead-bit detector. We further assume that the filter tap values do not change significantly from iteration to iteration, since the step size μ is relatively small. Under these assumptions, we let

$$b_k = \log_2 g_k[n] + q[n], \qquad (20)$$

where $q[n]$ represents the portion common to all b_k. Instead of a full calculation, $q[n]$ is gradually adjusted over time so that the cost constraint is met. For example, if the total cost from the previous cycle was less than the constraint, then $q[n]$ is incremented ($q[n] = q[n-1]+1$). Even though the approximation no longer guarantees that the cost constraint is always met, we will show in Figure 5 that it is met on average.

To illustrate the relative performance of wordlength adaptation, we first show in Figure 4 the ensemble averaged

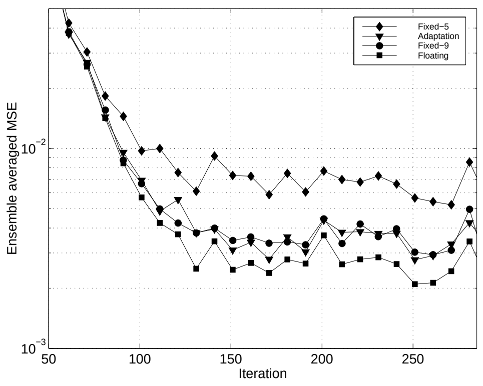


Figure 4. Comparison of LMS performance for four implementations.

mean-squared-error as a function of time for four implementations of the LMS algorithm: floating-point, $b_k = 9$ fixed-point, $b_k = 5$ fixed-point, and finally, real-time wordlength adaptation with the time constraint $\sum b_k \leq 55$.

From the figure, the floating-point implementation gives the lowest average MSE, and the 9-bit fixed-point implementation comes very close to it. Cutting back to 5 bits at the inputs gives rise to significantly higher MSE. However, by using wordlength adaptation, we were able to match the performance of the 9-bit implementation using only an average of 5 bits at the input.

Figure 5 shows the sum of the bit allocations at each LMS iteration for different constraint levels of 33, 55, or 77 total computing cycles per output. While the constraint is not met at every iteration, the average time per output sample does meet the constraints. More sophiscated methods of setting $q[n]$ in (20) can of course reduce the amount of variance in the total number of allocated bits in each cycle.

To further illustrate that the wordlength adaptation process does indeed allocate reasonable wordlengths, Figure 6a shows b_k averaged over time during the last 100 iterations for varying hardware constraints. In each case, the bit allocations were constrained to use at least 2 bits. Figure 6b shows the impulse response to which the adaptive filters converged. By comparing Figures 6a and 6b, we see that more bits are allocated to those inputs multiplying taps with large absolute values, just as we expected.

6. DISCUSSION AND FUTURE DIRECTIONS

In this paper we were concerned with optimizing both the wordlengths and performance of hardware-based algorithms. By restricting the cost function to the simple but important linear case, we were able to derive closed-form expressions for the wordlengths. Additionally, the form of the expressions admitted an efficient hardware implementation to allow the real-time calculation of the wordlengths. We applied this to a simple bit-serial implementation of an LMS adaptive filter.

Higher speed, parallel implementations of the LMS filter are also of great interest. We are currently investigating a fully parallel implementation, where truncating the input data results not in a hardware savings, but rather in a power savings, due to a reduction in the number of logic transitions.

The approach outlined here is more generally applicable than to just linear filtering. We are currently examining optimal wordlength allocation for more general cases of nonlinear algorithms and nonlinear cost functions.

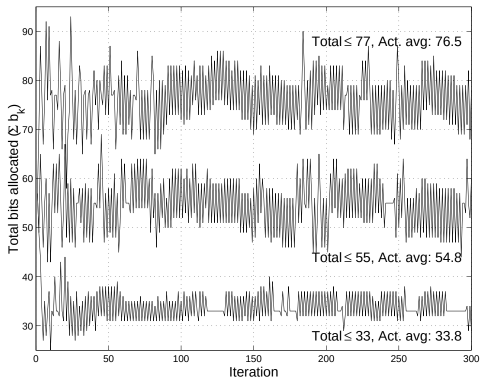


Figure 5. Total bits allocated using approximate wordlength adaptation procedure in each iteration for constraints of 33, 55, and 77 total bits.

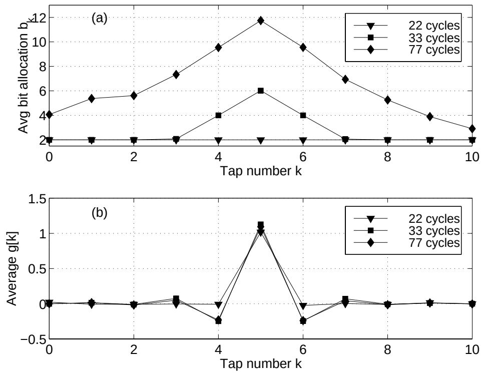


Figure 6. (a) Average number of bits allocated to each input position during the last 100 iterations of LMS processing. (b) Average tap weights during the last 100 iterations.

ACKNOWLEDGEMENT

This work was supported by the Defense Advanced Projects Research Agency (DARPA) and the United States Air Force Research Laboratory (AFRL) under Contract No. F33615-97-C-1174. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or AFRL.

REFERENCES

[1] Scott Hauck. The roles of FPGA's in reprogrammable systems. *Proceedings of the IEEE*, 86(4), April 1998.

[2] Xilinx Corporation. The programmable logic data book. *http://www.xilinx.com*, 1998.

[3] Paul D. Fiore, Cory Myers, John M. Smith, and Eric Pauer. Rapid implementation of mathematical and DSP algorithms in configurable computing devices. In *Proc. Configurable Computing: Technology and Applications, part of SPIE Intl. Symposium on Voice, Video and Data Comm.*, November 1998.

[4] W. H. Mangione-Smith et al. Seeking solutions in configurable computing. *Computer*, December 1997.

[5] Paul D. Fiore. Low complexity implementation of a polyphase filter bank. *Digital Signal Processing, A Review Journal*, 8(2):126–135, April 1998.

[6] Alan V. Oppenheim. *Applications of Digital Signal Processing.* Prentice-Hall, 1978.

[7] Simon Haykin. *Adaptive Filter Theory.* Prentice-Hall, 1991.