

A COMPARISON OF AUTOMATIC WORD LENGTH OPTIMIZATION PROCEDURES

M.-A. Cantin¹, Y. Savaria¹, and P. Lavoie².

¹ Electrical and Computer Engineering Dept., École Polytechnique de Montréal,
P.O. Box 6079, Station centre-ville, Montréal, Quebec, Canada, H3C 3A7

³ Defence Research Establishment Ottawa, Dept of National Defence, Ottawa, Ontario, Canada, K1A 0Z4

ABSTRACT

This paper presents a comparison of word length determination procedures. It is realized using an automated testbed that exploits a C/C++ fixed-point simulation utility to model the impact of finite word length on overall accuracy. Word length determination procedures find a combination of optimum bit resolutions by computing dissimilarities between fixed-point and floating-point simulation results. The comparison helps to select a procedure that minimizes these dissimilarities and finds an optimal combination of word lengths that meet user specified objectives, in a minimum number of iterations and hardware cost. This comparison was applied on various DSP algorithms.

1. INTRODUCTION

Digital signal processing (DSP) algorithms are often expressed in floating point operations or in 32 or 64-bit fixed-point word length for convenience, since these data formats match most existing commercial off-the-shelf processors. Yet in spite of a very significant growth of the performance of available processors, the requirements of many real-time applications, in terms of pure performance or low power operation, command the use of specialized hardware. Since cost, power dissipation of hardware, and pure performance of the system are highly dependent on the bit resolution and the use of floating-point operators, optimizing word lengths is important. In some complex design, 50% of the design time is spent on word length determination [1]. Moreover, word lengths must be carefully selected to preserve algorithm stability [2]. For these reasons, an automatic word length determination method was recently presented by the authors [3]. Our method leveraged a C/C++ fixed-point simulation utility [4] to model the impact of finite word lengths on overall accuracy. The method finds a combination of minimum word length by computing dissimilarities between fixed-point and floating-point simulation results.

The next section presents related work on word-length determination methods. It is followed in Section 3 by a brief description of a word length determination tool that we implemented and used to compare the minimization procedures. Section 4 briefly reviews nine automatic word length optimization procedures applied on various DSP algorithms. The methodology and results produced by our comparative study are presented in Section 5, and our main conclusions are summarized in Section 6.

2. RELATED WORK

In the last 10 years, several techniques have been proposed to translate floating-point formulations into fixed-point formulations, especially for specific DSP applications [2,5]. In

general, a first step in translating floating-point formulations into fixed-point formulations requires evaluating the dynamic range or Integer Word Length (IWL) of each operand O_i , for $i=0, \dots, I-1$, where I is the number of operands for which we want to determine the bit resolution. Secondly, to reduce hardware requirements, data truncation must be applied at various points. Indeed, many operators naturally expand the Word length (WL) of the results. Generally, a Fractional Word Length (FWL) must also be determined, where

$$WL_i = IWL_i + FWL_i + S_i$$

with $S_i=0$ for unsigned and $S_i=1$ for two's complement representation of O_i .

Three methods to determine the FWL were found in the literature. The first method evaluates the FWL on the basis of a data flow graph (DFG) analysis. FRIDGE[1] and CoCentric of Synopsys [6] propagate through the DFG the rule that no information is lost. They called this technique interpolation. This rule is best illustrated by an example: for $a = b + c$, no information is lost if $FWL(a) = \max(FWL(b), FWL(c))$. By contrast, Wadekar[7] and Yasuura & al.[8] perform a numerical error analysis on the DFG of the analysed algorithm. Determining the FWL using a DFG is fast, but the method overestimates the FWL . Furthermore, the DFG analysis requires a fixed-point specification of the input signals, and become very complex to manage when the algorithm includes data-dependencies.

Cmar & al. [9] propose to determine the FWL_i with a method that is partly analytical and partly simulation based. The method computes the mean $\mu_i(n)$ and the standard deviation $\sigma_i(n)$ of the dissimilarities between fixed-point and floating-point simulations. Since the C description of the DSP algorithm needs major modifications to apply this analytical/simulation based method, the bit resolution analysis is performed only on one operand instead of on combinations of operands.

Finally, the FWL can be determined by simulation-based methods. Mechanical procedures were proposed by the authors [3]. Manual procedures and guidelines to determine FWL using a simulation based approach are also proposed by Sung & Kum[10], Han & al.[11], Choi and Burleson[12], and Fiore and Lee [13]. These procedures are briefly reviewed and compared in this paper. Our automatic word length determination tool [3] that implements all these minimization procedures is briefly described in the next section.

3. AN AUTOMATIC WORD LENGTH DETERMINATION TOOL

The tool is built on a fixed-point simulation utility developed by W. Sung & al. [4]. SystemC [14] could also be used. The utility developed by W. Sung & al. converts a floating-point program

written in C or C++ into a fixed-point equivalent description. The WL , IWL , sign, overflow handling scheme, and quantization mode of individual operands and constants can be assigned in the floating-point program.

The fixed-point arithmetic operations, instead of floating-point arithmetic, are conducted automatically due to the operator overloading capability of the C++ language. Except for the name of the main function, declaration of the operands belonging to a fixed-point class type, and adding a fixed-point header file, no other part of the original program is changed during the conversion process.

The automatic word length determination tool executes six steps, as shown in Figure 1.

1- Test bench generation

Test benches representing the conditions of operation must be generated. They must stimulate all operands O_i , $i=0, 1, \dots, (I-1)$, in the DSP algorithm and data paths of interest.

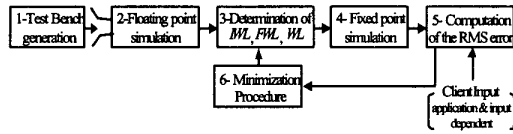


Figure 1. Automatic word length determination tool

2- Floating-point simulation

A floating-point simulation of the DSP algorithm is performed. The minimum, O_i^{\min} , maximum, O_i^{\max} , sign flag, S_i , mean $\mu_i(n)$, and standard deviation $\sigma_i(n)$ are extracted for each operand O_i . Also, results produced by the floating-point simulation for each test bench generated in Step 1 are stored. These simulations are considered as the reference model.

3- Determination of the operand format for the fixed-point simulation tool

In Step 3, IWL_i , WL_i and the operand type are determined from the O_i^{\min} , O_i^{\max} , S_i , $\mu_i(n)$ and $\sigma_i(n)$ extracted in Step 2. Finally, FWL of operand i , is initialized by the minimization procedure.

4- Fixed-point simulation

In this step, a fixed-point simulation for each test bench is performed. Results are stored.

5- Errors computation

Floating-point and fixed-point simulation results stored in Steps 2 and 4 above are used to compute various errors ξ_e for $e=0, \dots, E$, where E is the number of error computations. The user specifies the error computations as well as the number of error computations. They are used to guide the search towards an optimized solution.

6- Minimization procedure

A minimization procedure tries to find the combination of word lengths $\{WL_i\}$ that minimizes overall cost while meeting the specifications of the application. Based on the error computations ξ_e of step 6, and respectively the user specifications, S_e , a metric C is computed. The proposed metric measurement is defined as follow:

$$C = \frac{\sum_{e=0}^{E-1} (S_e - \xi_e) \mu_e}{\max \left(\sum_{e=0}^{E-1} d_e, 1 \right)} H + \frac{1}{H} \prod_{e=0}^{E-1} (d_e - 1)^j \quad \begin{cases} d_e = 0 & S_e \geq \xi_e \\ d_e = 1 & S_e < \xi_e \end{cases}$$

with the hardware cost H defined as:

$$H = \sum_{i=0}^{I-1} \gamma_i WL_i$$

where WL_i is the word length of the i th operand and γ_i is the corresponding hardware cost per bit defined in the range $[0,1]$. Each γ_i is fixed by the user. Depending on the metric value, the minimization procedure modifies the most promising WL_i , looking for a global optimum solution.

4. MINIMIZATION PROCEDURES

In this work, nine minimization procedures are compared:

(1) A *Min + b bit* procedure, proceeds in three execution phases.

The first phase finds, for each operand O_i , $i=0, 1, \dots, (N-1)$, the minimum bit resolution that satisfies the specifications when all other operands O_j , $j=0, 1, \dots, (N-1)$, $j \neq i$, have an arbitrarily long word length (or floating-point format). The second phase sets the resolution of each operand to the value found in the first phase. This combination of WL_i is called in this paper the Minimum Word Length (MWL) combination. The third phase is an iterative competition between the operands to gain b bits. In the basic form of the procedure $b=1$. In general, all combinations of b bits are simultaneously but temporarily distributed over all operands. The bit assignment combination that reduces and minimizes the error is retained. The value of b is gradually increased to $b=2, 3, \dots$ bits as required. Once a successful step is taken, and as long as the error specifications are not met, b is reset to one bit and this last phase is repeated. This procedure, restricted to the case where $b=1$, was called a *Sequential Search* in [11]. The procedure may get trapped in local minima when the resolution is increased by only one bit at a time. Thus, only the *Min + b bit* procedure is retained.

(2) A *Max - 1 bit* procedure starts with the maximum bit resolution allowed by the fixed-point simulation tool for all operands, which is 32 bits in our case. Then the operands compete to lose some of their bits as follows. One bit is temporarily removed from each operand O_i , $i=0, 1, \dots, (N-1)$, while all other operands O_j , $j=0, 1, \dots, (N-1)$, $j \neq i$, remain unchanged. The operand O_i for which the error is minimized wins the right to lose a bit. The process is repeated for another bit, as long as the error specifications are not met.

(3) An *Evolutionary* procedure starts with all operands having floating-point precision. The bit resolution of a first operand, say O_i , is set to 0, and is gradually increased until the system meets the specifications of the application. The value of WL_i is then fixed with one more bit, while a second operand is analyzed in the same way. The user determines the order in which the operands are processed. This step is repeated until all WL_i are fixed. Finally, the operands compete to lose a bit as with the *Max - 1 bit* procedure, as long as the system specifications are not met.

(4) An *Hybrid procedure* combines the *Min + b* bit procedure followed by the *Max - 1* bit procedure.

(5) The *Heuristic procedure* proposed in [10], increases all WL_i by one bit from the *MWL* until the error specifications of the system are met. At that point, as in the *Max - 1* bit procedure, the operands compete to loose their bits as follows. The operand O_i that produces the largest cost reduction wins the right to loose a bit. The process is repeated for another bit, as long as the error specifications are not met.

(6) The *Exhaustive procedure*, also proposed in [10], starts an exhaustive search from the *MWL*. The exhaustive search temporarily distributes b bits over all operands. For all cases, if the system specifications are not met, b is increased to $b=2, 3, \dots$ bits, until one assignment combination met the system specifications.

(7) A procedure called in this paper *Simulated Annealing* is proposed in [13]. Each O_i starts with a certain word length such that the overall solution meets the system specifications. Some WL_i are increased in order to allow reducing other $WL_j, j \neq i$, which reduces the total system hardware cost. The user determines how many times this step is repeated.

(8) The *Preplanned procedure* proposed in [11] proceeds in four execution phases. The first phase computes the system performance for each operand $O_i, i=0, 1, \dots, (N-1)$, when WL_i is set sequentially to every bit resolution, $WL_i=0, 1, \dots, WL^{MAX}$, and when all other operands $O_j, j=0, 1, \dots, (N-1), j \neq i$, have an arbitrarily long word length (or floatingpoint format). During this first phase, $WL^{max}-1$ a sensitivity parameter, ξ_i , is computed for each operand O_i , where:

$$\xi_i = f(WL_i + 1) - f(WL_i)$$

and $f(\cdot)$ is an objective function. The second phase constructs a global priority list in decreasing order of sensitivity. The third phase computes and sets all the O_i to the *MWL* bit resolution. The last phase is an iterative competition. The width of the operand O_i with the largest sensitivity is increased by one bit. As long as the error specifications of the system are not met, this iterative competition is repeated.

(9) The *Branch and Bound procedure* proposed in [12] proceeds in 3 execution phases. The first phase finds a minimum uniform-word length combination, called the upper bound solution. The second phase computes the *MWL* combination. The third phase explores the search space between the upper bound and the *MWL* combination and keeps the best solution found.

5. RESULTS

In order to compare these nine optimization procedures, the word lengths were determined for 12 DSP algorithms. The set of considered algorithms includes elementary operations, FIR filters, IIR filter, an adaptive filter, the CORDIC algorithm, the IDCT algorithm, a frequency estimation algorithm and a Neural Network algorithm. Since the operands word lengths determine the hardware architecture of the DSP algorithm implementation, the considered operands are carefully selected. For example, in

the FIR filters, the word lengths of both coefficients and data-paths are analyzed.

Some manual procedures were modified and adapted for the framework of our tool. In the *Simulated Annealing* procedure, the O_i s must start with values that already meet the system specifications. The *Min + b* procedure was selected to find this initial starting point. For the *Preplanned* procedure, only the required system performances, and then the sensitivity performances are computed instead of computing all system performances. The upper bound solution of the *Branch and Bound* procedure is computed using a binary search algorithm to reduce the number of iterations. Finally for the *Simulated Annealing* procedure, the best solution found in $10 \times I$ annealing phases, is kept as the final solution (recall that I is the number of considered operands).

Word lengths are found for these 12 DSP algorithms, TB1 to TB12, for $K=100$ different system specifications, using each minimization procedure. Results are reported in Table 1. For each procedure, two results are reported. The first result, $\overline{\Delta WL}_i$, compares each WL_i to the combination that constitutes the best-obtained result (assuming all $\gamma_i = 1$), which is called WL_i^{opt} . The reported results are normalized by averaging the observed differences over all operands. This is obtained by summing differences for each operand and dividing by the number of operands, where I_n is the number of operands considered with test bench TB_n. $\overline{\Delta WL}_i$ is computed as follows:

$$\overline{\Delta WL}_i = \left(\frac{1}{100} \cdot \sum_{k=1}^{100} \frac{1}{I_n} \sum_{i=0}^{I_n-1} WL_{ik} - WL_{ik}^{opt} \right)$$

The second result, $\overline{\Delta N}$, also reported in table 1, compares the number of iterations to obtain a solution, N , with the procedure that obtains a solution with the fewest iterations N^{opt} (generally, it does not correspond to the procedure producing WL_i^{opt}).

As $\overline{\Delta WL}_i$, the average $\overline{\Delta N}$ is obtained by summing differences over all iterations and dividing by the number of operands I_n . We focus on the number of iterations, since the processing time for each iteration is dominated by the time required to characterize the performance of a configuration with a simulation of enough relevant test cases. $\overline{\Delta N}$ is computed as follows:

$$\overline{\Delta N} = \frac{1}{100 \cdot I_n} \cdot \sum_{k=1}^{100} \left(N_k - N_k^{opt} \right)$$

If a procedure produces $\overline{\Delta WL}_i = 0$ and $\overline{\Delta N} = 0$ over some domain, then it dominates all other procedures. For TB1 and TB2, all procedures obtained the optimal word lengths. However, no procedure produced the optimal word length for TB1 to TB12. Note that a difference that may appear small in the $\overline{\Delta WL}_i$ results, for example 1.65 for the *Min + b* procedure applied on TB11, can produce a maximum difference as large as 38 bits in total operand widths, when comparing a solution to the optimal solution for a specific system specification. Since the processing time is dominated by the number of iterations required to find a solution, a

Table 1. Results of the comparative study

Test Bench	I_n	Min + b bit		Max - 1 bit		Evolutive		Hybrid		Exhaustive		Heuristic		Simulated Annealing		Preplanned		Branch and Bound	
		$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$	$\overline{\Delta WL}_i$	$\overline{\Delta N}$
TB1	3	0.00	2.2	0.00	70.6	0.00	1.3	0.00	3.2	0.00	3.2	0.00	0.2	0.00	20.2	0.00	1.9	0.00	2.1
TB2	3	0.00	0.0	0.00	72.5	0.00	4.0	0.00	1.0	0.00	1.0	0.00	0.0	0.00	20.0	0.00	0.0	0.00	2.7
TB3	5	0.61	2.2	0.72	56.0	0.72	0.5	0.61	3.2	0.61	3.2	0.00	1.0	0.61	52.2	0.61	1.7	0.72	2.0
TB4	7	0.19	0.2	0.68	37.2	0.11	3.4	0.04	1.6	0.41	2.1	0.04	1.6	0.19	98.2	0.23	0.2	0.34	52.1
TB5	5	0.48	2.0	0.67	63.5	0.54	0.7	0.48	3.0	0.48	3.6	0.00	0.7	0.48	52.0	0.56	1.7	0.54	5.2
TB6	4	0.39	0.0	0.00	97.5	0.00	5.0	0.00	2.3	0.00	2.3	0.39	0.0	0.39	40.0	0.34	0.0	0.00	3.6
TB7	6	0.63	0.6	0.07	134.0	0.07	8.2	0.04	5.1	0.76	7.5	0.07	0.2	0.42	60.7	0.90	0.2	0.20	2.4
TB8	4	0.12	0.2	0.06	84.0	0.05	4.5	0.00	1.6	0.31	2.5	0.03	2.3	0.06	98.8	0.18	0.2	0.18	4.8
TB9	15	0.40	0.0	0.00	105.9	0.00	5.5	0.00	2.6	0.00	0.0	0.40	2.6	0.40	450.0	0.38	0.0	0.18	40.25
TB10	3	1.01	0.4	0.52	75.6	0.25	5.5	0.22	3.7	1.01	4.1	0.22	2.2	0.98	18.4	1.30	0.1	0.83	4.2
TB11	36	1.65	1.5	0.45	172.9	0.18	12.4	0.21	12.5	1.75	15.5	0.18	2.5	1.53	2593	3.07	1.2	1.99	2964.9
TB12	3	0.02	0.2	0.42	67.3	0.04	2.3	0.01	1.3	0.11	0.7	0.23	1.72	0.02	18.2	0.05	0.0	0.03	7.6

small difference on $\overline{\Delta N}$ is not very significant when a small number of operands are processed.

However, in some applications, up to 1000 operands are processed [1], a difference of $\overline{\Delta N}=15.5$ reflects to $15.5 \times 1000 \times 0.75s = 3.23$ additional hours of processing time (750ms were required to performed one fixed-point simulation of relevant test cases).

The *Hybrid* procedure always reaches a solution equivalent or better than the *Min + b bit* procedure, which is reflected by solutions with lower hardware costs. This observation leads to two situations: (1) An optimal solution can be reached with less hardware cost than the *MWL* combination. This first counterintuitive result was observed several times and will be investigated in the future. (2) Finding a solution with the *Min + b* procedure does not ensure that all operands have their minimum word length. Therefore, for both situations, the *Hybrid* procedure takes advantage of using the *Max - 1 bit* procedure. Thus, the *Hybrid* procedure takes more iterations than the *Min + b*.

Procedures such as *Max - 1 bit* and *Evolutive* start from a solution that already meets the system error specifications, and then try to find a better solution. This type of method can be trapped in a local optimum instead a global optimum. The *Heuristic* procedure produced optimal solutions with a relative small number of iterations. The *Simulated Annealing* procedure always produced the same solutions as the *Min + b* procedure, and therefore it does not bring any advantage for our test benches.

The *Preplanned* procedure is the one that required the smallest number of iterations to find a solution. However, the solutions it produces are not always optimum in terms of hardware cost. Moreover, the *Preplanned* procedure, as the *Min + b* and the *Branch and Bound* procedures, does not consider solutions that require less hardware than the *MWL* combination that were found feasible in some cases. The *Branch and Bound*, *Exhaustive* and *Max - 1 bit* procedures take the largest number of iterations to find a solution. They may become prohibitive when a problem with a large number of operands is analyzed.

6. CONCLUSION

A first synthesis and comparison of 9 procedures, given in the literature to optimize word lengths of DSP algorithms using a

simulation based approach, was presented. Based on the 12 analyzed DSP algorithms, from the standpoints of the number of iterations and resulting implementation costs, fast searches often find optimum word length combinations. The comparative study demonstrated that procedures must consider global solutions instead of local solutions. It was also found that optimal solutions could use less hardware than the *MWL* combination. Furthermore, a procedure that independently increases by one bit the word length of each operand may fail to reach optimal solutions.

7. REFERENCES

- [1] H. Keding, M. Willems, M. Coors and H. Meyr, "FRIDGE: a fixed-point design and simulation environment", Design, Automation and Test in Europe, pp. 429 - 435, 1998.
- [2] M-A. Cantin, Y. Blaquiere, Y. Sarvaria, P. Lavoie, and E. Granger, "Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm", IEEE Int. Symposium on Circuits and Systems, Vol. 3, pp: 141-144, 2000.
- [3] M-A. Cantin, Y. Savaria, D. Prodanos, P. Lavoie, "An automatic word length determination method", IEEE Int. Symposium on Sydney, Australia, May 2001, Volume: 5, Page(s): 53-56.
- [4] S. Kim, K. Kum and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs", IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, vol. 45, no. 11, pp 1455 -64, 1998.
- [5] Yli-Kaakinen, J. and Saramaki, T., "An efficient algorithm for the design of lattice wave digital filters with short coefficient wordlength.", Int. Symposium on Circuits and Systems, vol.3, pp. 443-8, May 1999.
- [6] Press Release: Synopsys Accelerates System-Level C-Based DSP Design With CoCentric Fixed-Point Designer Tool. Synopsys Inc., April 10, 2000.
- [7] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun, "Variable Size Analysis and Validation of Computation Quality," Proc. of Workshop on High-Level Design Validation and Test (HLDVT'00), pp.95-100, Nov. 2000.
- [8] Suhrud Ashok Wadekar, "Accuracy Sensitive Word-Length Selection for Algorithm Optimization", International Conference on Computer Design: VLSI in Computers and Processors, pp 54-61, 1998.
- [9] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement" Design, Automation and Test in Europe Conference and Exhibition, pp 271-6, 1999.
- [10] W. Sung; K. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems", IEEE Trans. on Signal Processing, vol.43, no.12, pp 3087-90, 1995.
- [11] Kyungtae Han; Iksu Eo; Kyungsu Kim; Hanjin Cho, "Numerical word-length optimization for CDMA demodulator", IEEE Int. Symposium on Circuits and Systems, pp 290-3 vol. 4, 2001
- [12] H. Choi, W. P. Burlison, "Search-based wordlength optimization for VLSI/DSP synthesis", VLSI Signal Processing VII, pp. 198-207, 1994.
- [13] P.D. Fiore, Li Lee, "Closed-form and real-time wordlength adaptation", Proc. of the IEEE Int. Conference on Acoustics, Speech, and Signal Processing, vol. 4, pp 1897-900, 1999.
- [14] M. Speitel, B. Niemann, "SystemC design language for development of ASICs and systems" Elektronik, vol.50, no.13, pp. 78-83, 2001