SIMULATION OF DSP ALGORITHMS ON FIXED POINT ARCHITECTURES

K.B. Cullen, G.C.M. Silvestre and N.J. Hurley

Department of Computer Science
University College Dublin – Ireland
Email: {keith, guenole, neil}@ihl.ucd.ie

ABSTRACT

This paper presents software tools to simulate DSP algorithms on a wide variety of fixed point architectures including microprocessors, DSPs and FPGA devices. Existing solutions for evaluating the signal quality in fixed point algorithms are either unable to deal with non-linear systems, fail to consider the architectural details of the target device or do not produce a real output that can be used in subjective testing. Using example non-linear algorithms it is shown that architectural details must considered when evaluating numerical performance.

1. INTRODUCTION

Fixed point DSP devices are preferred over floating point devices in systems that are constrained by complexity, cost and power consumption such as mobile phones, personal digital assistants and wearable computing devices. In general a fixed point algorithm implemented on one of these devices starts life as a high level floating point simulation model. Converting the simulation model to fixed point arithmetic and then porting it to a target device is a very time consuming and difficult process. DSP devices have very different instruction sets so an implementation on one device cannot be ported easily to another device if it fails to achieve sufficient quality. Choosing a target device with an abundance of resources will exceed the constraints of any low power, low cost system. For these reasons it is necessary to evaluate a fixed point DSP algorithm to determine whether implementation on a particular device is feasible, to select an appropriate target device, to locate stages in the system where extended precision routines are necessary or to choose an ideal architecture for an FPGA or ASIC implementation.

There are analytical methods available to estimate the mean and variance for the fixed point error in *linear* DSP algorithms and these have been successfully applied to the analysis of FFTs, DCTs, FIR filters ...etc [1] [2]. However most real systems contain non-linear modules. Other analytical methods such as affine arithmetic can be used to evaluate non-linear algorithms but the fixed point error is estimated in terms of upper and lower bounds which convey even less information than mean and variance. Analytical methods are not suitable for evaluating perceptual systems where signal quality can only be measured using subjective tests involving an actual output signal. The simulation tools currently available [3][4][5] can be used to measure fixed point error directly. These tools require interaction with the user to determine binary point locations and do not take the architectural details of the target device into consideration.

The software tools presented in this paper were designed to address all of these problems. They operate in three phases. In the *conversion* phase a C/C++ floating point algorithm is automatically transformed into a fixed point simulation model. During the *optimization* phase the number of scaling operations required to implemented the algorithm is minimized without reducing the signal quality. Finally, in the *simulation* phase, which is the focus of this paper, the simulation model produces a bit-exact output for a user specified fixed point architecture.

The rest of this paper is organized as follows: section 2 presents the simulation tools, section 3 describes the user interface and section 4 presents an evaluation using minimax polynomials and the MPEG psychoacoustic model II.

2. FIXED POINT SIMULATION TOOLS

The software tools are implemented as a C++ class hierarchy. The lowest level of the hierarchy, the `Integer` class, deals with 2's compliment binary pattern operations. Arbitrary precision arithmetic is used to overcome the limitations of the host machine running the simulation so that objects of any size can be created, added, multiplied ...etc. Each overloaded operator determines the correct word length to use for the output so that no data is lost. This means that in an arithmetic expression the word lengths of the intermediate results get larger and larger as more operations are performed until an assignment operator is encountered. The next layer is the fixed point class, `Fixed`, which is a composite of the `Integer` class. It stores binary point information and implements the scaling operations required in fixed point arithmetic due to differences in binary point locations. These two layers conform to the SystemC standard [3]. The

third layer in the hierarchy, the `ArchFixed` class, deals with the architectural details of the target device. Instead of allowing the word lengths of intermediate results to get larger and larger the data is processed by a configurable architecture that constrains the word lengths of intermediate results according to the size of the data-bus, accumulator, multiplier input ...etc and applies rounding and limiting at the appropriate stages. The architecture can be configured to emulate various existing and conceptual devices simply by changing parameter values.

2.1. Configurable Fixed Point Architecture Model

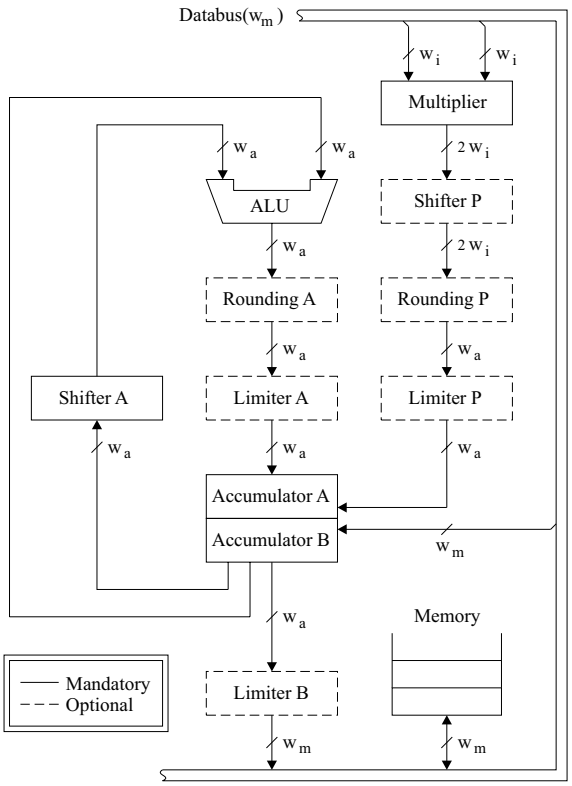

Fig. 1. Configurable Fixed Point Architecture Model. At points where a reduction in word length may occur, such as the connection between the w_m-bit databus and the w_i-bit multiplier input, the least significant bits are discarded.

The architecture model, shown in Fig. 1, was developed by comparing the major DSP and microprocessor devices to identify the important architectural differences. It consists of mandatory and optional components and is governed by three word length parameters w_m, w_a and w_i which are the memory, accumulator and multiplier input word lengths respectively. The parameter values and optional components for some example platforms are shown in Table 1.

The architecture model is only concerned with numerical details. There will be some structural differences between the model and the target devices it can emulate. For example, in DSP devices the output of the multiplier is normally connected to the input of the ALU but this arrangement does not affect the numerical result. Another example is the two accumulators which ensure that the scaling operations required in fixed point arithmetic are performed on w_a-bit values at all times. This is important if its necessary to shift both operands before an ALU operation. The TMS320C5x and DSP56xxx have two accumulators, integer processors can use any memory location as a second accumulator since the memory and accumulator word lengths are the same size on these devices and the ADSP-BF5xx devices have data registers that can serve as accumulators for scaling operations.

Table 1. Parameter values and optional components - Rounding A (RA), Limiter A (LA), Limiter B (LB), Shifter P (SP), Rounding P (RP) and Limiter P (LP) for some example platforms.

	w_m	w_a	w_i	RA	LA	LB	SP	RP	LP
TMS320C5x	16	32	16		*				
ADSP-BF5xx	16	40	16	*	*				
DSP56xxx	24	56	24	*		*			
ARM9	32	32	16						
ARM9 †	32	64	32	*	*				
ARM9 ‡	32	32	32				*	*	*
FPGA *	32	32	32				*		

† with routines for double precision arithmetic, rounding and limiting

‡ with double precision multiplication routine including rounding and limiting and single precision addition

* this is just one possible FPGA architecture

The architecture model represents the combined software/hardware operations used to carry out fixed point operations. For instance integer processors like the ARM9 need extra code to adapt the hardware to fixed point processing. On these devices the multiplier doesn't calculate the higher order bits in the product. To prevent overflow in fixed point multiplication word length reduction is required on the multiplier inputs. On a real device this involves reading a value into the accumulator, right shifting and writing the reduced value back to memory so it can be read back in. In the architecture model this is represented by the connection between the w_m-bits of the databus and the w_i-bits at each input to the multiplier. The architecture model can also represent software routines on the target device for double precision multiplication, addition ...etc, since these algorithms simply emulate more powerful devices.

The architecture model is embedded into the overloaded operators of the `ArchFixed` class using objects of type `Fixed` to represent registers and busses. The multiplication operator, for example, reduces the two inputs to w_i-bits, performs

the multiplication, shifts the product according to the binary point locations of the inputs and output, performs rounding and limiting if enabled and returns the w_a-bit result.

2.2. Maintaining Binary Point Information

With operator overloading the expression `z = x * y + w` will pass the variables `x` and `y` to the multiplication operator described above. To complete this operation the binary point locations for the inputs and the output are required however the multiplication operator has no knowledge of the output variable. To solve this problem a new system for maintaining binary point information was developed. Each variable in the algorithm is given a unique identifier constructed from its name and the function its defined in. This identifier is used to access a centrally stored table of binary point locations. The identifier for the output of any operator is determined from the identifiers of the inputs and the operator involved. If `x`, `y` and `w` have the identifiers `"f:x"`, `"f:y"` and `"f:w"` respectively then the binary point location for the product is stored in the binary point table under the identifier `"(f:x)*(f:y)"` and the output of the addition has the identifer `"((f:x)*(f:y)) + (f:w)"`. The overloaded operators can find the binary point location for the output variable simply by looking at the input variables. The table is stored in a text file so that binary point information can be entered manually by the user or by an automated conversion process. This system is described in more detail in [6].

3. USER INTERFACE

Starting with a C/C++ floating point algorithm the first step in creating a simulation model is to run the source code formatter. It changes variable definitions from floating point to `ArchFixed` and gives each variable its unique identifier. No structural changes are made to the algorithm so the source code remains human readable. During the conversion phase, which is presented in [6], the simulation model is executed using sample input signals and the binary point locations are automatically adjusted to find optimum values. During the simulation phase the architecture model uses the binary point locations, which now remain constant, to determine the correct scaling (shift) operations to use at every stage in the algorithm.

The word length parameters and the status of the optional components in the configurable architecture are set using function calls. At any point in the source code these values can be changed to simulate an algorithm that uses double precision arithmetic in specific parts or to simulate an algorithm partitioned across multiple devices.

4. EVALUATION

Simulation models for some example non-linear DSP algorithms were created to illustrate the capabilities of the tools. These algorithms were implemented on an ARM922T processor and an XCV1000 FPGA to confirm that the simulation results are bit-exact.

4.1. Minimax Polynomial

Minimax polynomials are often used to approximate functions like cosine, tangent, logarithm ...etc. These polynomials have the property that the maximum value of the approximation error is minimized. For example the 3rd degree minimax polynomial approximation to $\cos(x), 0 \leq x < \pi/2$ is the expression `y = ((c3 * x + c2) * x + c1) * x + c0` where the coefficients, `cn`, can be obtained using the Remez-exchange algorithm.

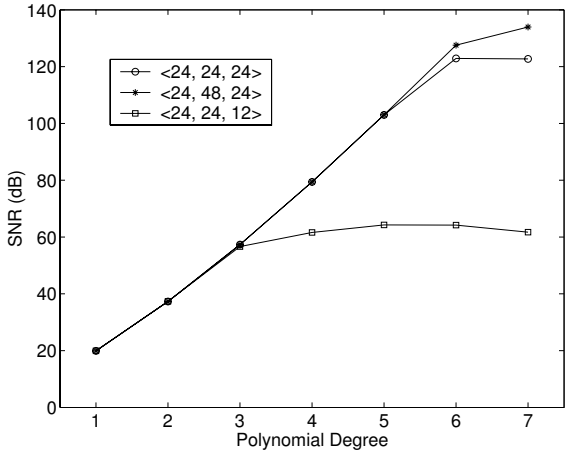

Fig. 2. Signal-to-Noise Ratio (dB) vs polynomial degree for a fixed point minimax approximation of $\cos(x), 0 \leq x < \pi/2$ with different parameter values $< w_m, w_a, w_i >$.

Fig. 2 shows the Signal-to-Noise Ratio (SNR) for the minimax cosine approximation with polynomial degree on the abscissa and different values assigned to the parameters $< w_m, w_a, w_i >$. Each polynomial degree is implemented as a separate algorithm. The curves shown represent a 24-bit FPGA implementation ($< 24, 24, 24 >$), a 24-bit DSP device implementation ($< 24, 48, 24 >$) and a 24-bit integer microprocessor implementation ($< 24, 24, 12 >$). For the 7th degree polynomial the difference in SNR between the DSP and FPGA implementations is 11.26 dB. The difference between the DSP and integer microprocessor versions is 72.24 dB. All of the curves reach a maximum value of SNR that cannot be exceeded. This is due to the interaction between approximation and fixed point error which is discussed in [7]. This example illustrates the point that even

for an algorithm with a small number of operations architectural details have a significant effect on fixed point error.

4.2. MPEG psychoacoustic Model II

The MPEG psychoacoustic model II is used in both MP3 and AAC. Its objective is to compute a masking threshold for every frame of input audio data (typically 256 to 2048 samples) which describes the minimum level of noise that will be inaudible to human listeners as a function of frequency. It was chosen as a case study to evaluate the tools because it is a non-linear algorithm with large dynamic range and is very difficult to implement in fixed point arithmetic. This algorithm was implemented exactly as described in the MPEG-2 AAC specification [8].

Table 2. Signal-to-Noise Ratios (dB) for the masking curves produced by different devices when compared to the floating point implementation.

Configuration	SNR (dB)
TMS320C5x	-7.06
TMS320C5x †	11.57
ADSP-BF5xx	-6.30
DSP56xxx	11.55
ARM9	-7.03
ARM9 †	11.57

† with double precision arithmetic

Using real audio data the masking curves produced by the fixed point and floating point versions were compared using signal-to-noise ratio. The device configurations and results are given in Table 2. The 32-bit ARM9 produces very poor quality which is comparable to that of the 16-bit TMS320C5x and ADSP-BF5xx devices. The 24-bit DSP56xxx produces much better results. Even with double precision arithmetic the TMS320C5x and ARM9 cannot improve on the DSP56xxx. Since a masking threshold is an approximation of human auditory perception the output of these simulations can also be used in subjective tests involving human listeners.

4.3. MPEG-2 AAC Encoder

The tools were used to create a simulation model of an independently developed MPEG-2 AAC stereo encoder [9]. The algorithm was then ported to a 32-bit ARM9 processor and part of the system was ported to an FPGA device [10]. The simulation model was used to determine the ideal architecture for the FPGA version and to establish the need for double precision arithmetic in the ARM version.

5. CONCLUSION

This paper has presented software tools to evaluate the numerical performance of fixed point DSP algorithms. The tools are unique in their ability to emulate specific fixed point architectures. The examples given in this paper have shown that the accuracy of a DSP algorithm implementation is not obvious from the single word length (16, 24, 32) used to categorize fixed point devices. It has also been shown that in some situations an increase in device complexity does not lead to an increase in signal quality. The advantage with using these tools is that several different implementations of a DSP algorithm can be explored quickly using a single simulation model so that the complexity, cost and power consumption of the final system are kept to a minimum.

6. REFERENCES

[1] Casper W. Barnes, Boi N. Tran and Shu H. Leung, "On the Statistics of Fixed-Point Roundoff Error," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Jun. 1985.

[2] Il Dong Yun and Sang Uk Lee, "On the Fixed-Point-Error Analysis of Several Fast DCT Algorithms," in *IEEE Transactions on Circuits and Systems for Video Technology*, Feb. 1993.

[3] The Open SystemC Initiative, "SystemC Version 2.0 User's guide," *http://www.systemc.org*, 2002.

[4] Seehyun Kim, Ki-Il Kum and Wonyong Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Nov. 1998.

[5] Markus Willems, Volker Bürsgens, Holger Keding, Thorsten Grötker and Heinrich Meyr, "System Level Fixed-Point Design Based on an Interpolative Approach," in *Proc. 34th Design Automation Conference*, Jun. 1997.

[6] K.B. Cullen, G.C.M. Silvestre and N.J. Hurley, "Simulation Tools for Fixed Point DSP Algorithms and Architectures," in *Proc. International Conference on Signal Processing*, Dec. 2004.

[7] K.B. Cullen, A. Guérin, N.J. Hurley and G.C.M Silvestre, "Evaluation of Fixed Point Elementary Functions for FPGA Audio Perceptual Coding," in *Proc. Irish Signals and Systems Conference*, Jul. 2003.

[8] ISO/IEC, 13818-7, "Information technology – Generic coding of moving pictures and associated audio – Part 7: Advanced audio coding (AAC)," 1997.

[9] K.B. Cullen, N.J. Hurley, G.C.M Silvestre, "Scalable Architecture for MPEG-2 AAC Encoders," in *Proc. Irish Signals and Systems Conference*, Jun. 2002.

[10] A. Guérin, K.B. Cullen, N.J. Hurley and G.C.M Silvestre, "FPGA Implementation of the MPEG-2 AAC Filter Bank," in *Proc. Irish Signals and Systems Conference*, Jul. 2004.