A Methodology and Design Environment for DSP ASIC Fixed Point Refinement

R.Cmar, L.Rijnders, P.Schaumont, S.Vernalde and I.Bolsens
IMEC, Kapeldreef 75, 3001 Leuven, Belgium

Abstract

Complex signal processing algorithms are specified in floating point precision. When their hardware implementation requires fixed point precision, type refinement is needed. The paper presents a methodology and design environment for this quantization process.

The method uses independent strategies for fixing MSB and LSB weights of fixed point signals. It enables short design cycles by combining the strengths of both analytical and simulation based methods.

1 Introduction

Modern signal processing ASICs, such as integrated cable modems and wireless multimedia terminals, are specified with algorithms in floating point precision. Often, the architectural style with which these algorithms are implemented is precision-limited, and relies on a fixed point representation. This requires a designer to translate floating point types into fixed point types, using a refinement strategy. For each refined floating point number, a fixed point characteristic (including fractional and integral wordlength, overflow and rounding behavior) must be chosen.

Traditionally, this type refinement is a manual and time consuming job. One reason for this is the enormous search space that is present due to the multitude of wordlengths. Several tools exist which support elaborate capturing of the system fixed-point specification into a behavioral description like in [4]. Still, the burden of wordlength decision is completely on the designer.

There are two kinds of approaches that increase the designers' support in quantization to a more substantial level.

- The **simulation based approach** [1] which compares the performance of the whole system with a reference model. The wordlengths are chosen heuristically while observing some error criterion. This process is repeated until wordlengths converge. A more elaborate solution which uses the technique of operator overloading and monitoring of signal characteristics was presented in [2].

  This method yields precise results but ([3]) it can lead to long simulations in the case of slow convergence, which is not acceptable for complex systems.

- Another method suggests an **analytical approach** [3]. Wordlengths are derived using source code structure, local annotations, and an (analytical) interpolation method. This method yields results very fast, but ([1]) it is a conservative approach which leads to overestimation of signal wordlengths.

Since both methods clearly have strengths and weaknesses, we propose a smooth combination of both techniques. In a combined technique, fast convergence speed is desired, while at the same time wordlength estimation should be optimal. Therefore, we developed a technique that combines simulation statistics and analysis.

The technique uses a separate strategy for fixing of the MSB and LSB weights of a fixed point signal. It was integrated and tested into a C++ based design environment [5] which is used for the design of high speed ASICs.

We introduce our technique as follows. In Section 2, it is indicated how fixed point data types are described in C++. Next, we outline the characteristics of our method by a motivational example (Section 3). Following this, Section 4 gives a detailed description of the quantization approach. Next, Section 5 elaborates on the design flow, while Section 6 comments on the obtained results. Finally, the main strengths of our approach are summarized in the conclusions.

2 Design environment

The fixed-point modeling is integrated in our design environment [5]. The environment allows simulation and synthesis of a digital hardware system using exclusively a C++ object oriented description. The constructed systems consist of several communicating processors. Each processor is described first behaviorally at high level and next at clock cycle true level. A simulation engine performs processor execution and their communication. Finally, a code generator enables translation of the cycle true C++ description to synthesizable VHDL.

2.1 Fixed-point modeling

In our design environment, fixed point data types are represented by a dedicated object type rather than a standard

float, double or int. This object type, called `sig`, can express both the fixed-point and floating-point signal behaviors.

The constructor is `sig(name, dtype)` for fixed-point signal representation, and `sig(name)` for floating-point representation.

The `dtype` is an object which carries information of the wordlength and the quantization behavior. The constructor is:

```
dtype(name, n, f, vtype, msbspec, lsbspec )
```

- `n` is the total wordlength
- `f` is the number of fractional bits
- `vtype` specifies the signal representation, two's complement (tc) or unsigned (ns)
- `msbspec` selects the MSB mode to be wrap-around (wp), saturation (st) or error (er). The latter produces an error message during simulation in case of overflow. This is an indication for the designer to increase the wordlength or to select another MSB mode.
- `lsbspec` specifies the LSB rounding mode, i.e. round-off (rd) or floor (fl).

In the following we will use the terms *LSB position* and *MSB position* referring to the absolute position with respect to the binary point. Hence $MSB = n - f$, $LSB = f$.

2.2 Fixed-point arithmetic

When working with fixed-point arithmetic, it is vital to have an efficient representation of values and simulation of operations. For this purpose, all operations are performed with floating point arithmetic. Only when assigning a signal, the quantization is performed. In the case an intermediate result needs to be quantized, a `cast` operator is available.

```
sig a("a",T1);
sig b("b",T2);
sig c("c",T3);
c = a * b;
```

In the above example the multiplication $a * b$ is a floating-point operation having as input two fixed-point values of types *T1* and *T2* respectively. During the assignment to c the floating point result is automatically casted through the specified quantization scheme defined by type *T3*.

2.3 RT description style

The notion of time in modeling component behavior gives rise to the concept of registered (`reg`) and non-registered signals (`sig`). For arrays these become `regarray` and `sigarray` respectively. The overloading of operators for these classes expresses a large scale of operations allowing our description of the algorithm to differ with ANSI-C only in the declaration. Example:

```
dtype T1("T1",8,5,ns,st,rd);
sigarray a("a", N, T1);
regarray b("b", N, T1);
sig c("c", T1);
reg d("d", T1);
...
c = a[1] * b[2];
d = c + d;
```

3 Motivational example

In order to demonstrate our fixed-point quantization refinement, let's consider the following algorithm which is a simplified symbol-spaced adaptive LMS equalizer shown in Figure 1. The behavioral C++ description that corresponds to this circuit is shown next.

```
// constructor definition
int N = 3;
sigarray c("c", N);
regarray d("d", N);
sigarray v("v", N+1);
sig x("x", T_input);
sig y("y");
sig w("w");
reg b("a");
reg s("s");

// initialization
double coef[] = { -0.11, 1.2, -0.11 };
for (i = 0; i < N; i++)
  c[i] = coef[i];

// execution
while (1) {
  d[0] = get(x);
  for (i = N-1; i > 0; i--)
     d[i] = d[i-1];
  v[0] = 0;
  for (i = 1; i <= N; i++)
     v[i] = v[i-1] + d[i-1] * c[i-1];
  w = v[N] - b * s;
  y = w > 0 ? 1 : -1;
  b = b + 0.001 * s * (w - y);
  s = y;
  put(y);
}
```

As seen in the source code, the input signal `x` is fed into the delay-line `d` and subsequently equalized by the FIR filter with constant coefficients `c`. The result stored in `v[N]` corrected by the feedback path `b*s` enters the slicer to obtain the output `y` with values 1 and -1 as the system works with the binary PAM signal. The slicer error `w-y` is used for adaptation of the single feedback coefficients `b`.

The problem to solve in this example is to determine the fixed point signal types. Current techniques would take one of the following approaches:

- exhaustive algorithm simulation with relevant input stimuli in order to obtain a good view on the quantization. This approach won't guarantee avoiding overflows for untested stimuli input and it needs many iterations for LSB determination

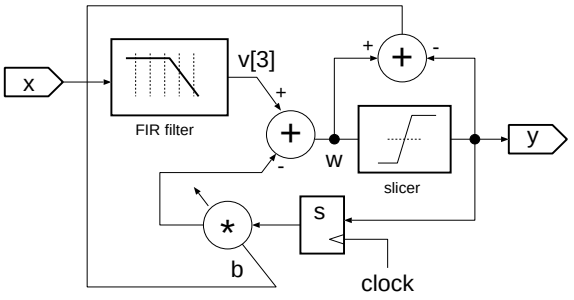

Figure 1. Processor example

- statical analysis of the source code (i.e. parsing) in order to obtain the signal flow graph structure and analysis for the worst case type propagation. This can lead to conservative results.

We will show that the efficient quantization of this algorithm can be achieved using a combined approach which has a short and safe determination process. Our combined approach is comprised of statistical-based monitoring and quasi-analytical range estimation at the MSB side. The LSB side determination is based on the novel simulation strategy which approaches an analytical error calculation. All techniques use the same description and can be applied during the same simulation run.

4 Quantization approaches

For the quantization determination we exploit the concept of the operator overloading which is also recognized in [2], [4] to be a valuable strategy. In our case the operators defined for signal objects perform several simultaneous tasks during the simulation. In addition to the fixed-point simulation two quantization determination processes are executed. The *range monitoring* technique is used to find an optimal MSB position and the *error monitoring* approach is used to determine an optimal LSB position. This is illustrated in Figure 2 and further described in the following sections.

4.1 MSB-side: Range monitoring

Range monitoring is a method which estimates a signal range by determining the minimum and maximum values that occur in a signal. Then a safe MSB position of signals can be selected to prevent unwanted overflows. Three alternative approaches are proposed.

Statistic-based

This approach allows monitoring of the range information through the overloading of the assignment operator, and keeping track of the signal range during simulation. After the simulation the stimuli-dependent MSB positions are obtained. They can however be underestimated.

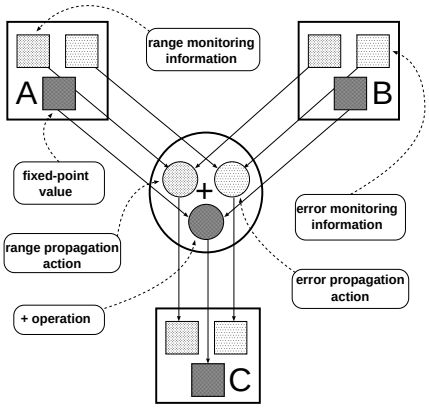

Figure 2. + operator overloading for range and error monitoring

Quasi-analytical

An additional approach that we advocate is a quasi-analytical one. Overloading of arithmetic operations here supports *range propagation*. When declaring signals with type information their range is automatically determined. Alternatively, a range can be explicitly attributed. In the following example the signal type `T1` defines the range of the signal `a` to be $(-2, 2)$, which is subsequently overridden to the fractional range $(-1.5, 1.5)$.

```
dtype T1("t1", 7,5, tc);
sig a("a", T1);
a.range(-1.5, 1.5);
```

The operators which perform the arithmetic operations will now perform also the range propagation. The table shows the range propagation (minimum-side only) for several operators:

a + b	min = a.min + b.min
a - b	min = a.min - b.max
a * b	min = MIN(a.min * b.min, a.min * b.max, a.max * b.min, a.max * b.max)
c = a	c.min = MIN(c.min, a.min)

When applied to feedback signals, range propagation can become unstable and cause *explosion* of the MSB. This can be avoided by setting the range explicitly for these feedback signals with the `range()` method or by using a saturation type definition.

We call this method quasi-analytical because it combines simulation and analysis. The worst case range estimation is typically obtained with a very short simulation, ideally it needs only one iteration. However the finding of a safe signal range requires complete *coverage of a code execution*.

Analytical

A perfect evaluation of the signal range is enabled by constructing a signal flowgraph out of the source code and analyzing the data flow using the same range propagation mechanism.

4.2 LSB-side: Error monitoring

Error monitoring is based on a combined fixed-point/floating-point simulation. The goal is to determine and minimize precision loss due to the finite wordlengths.

Each signal object contains two data members, representing the fixed-point and floating-point value. The algorithm is executed with two simultaneous calculations for each operation in the same simulation. One is performing computations with specified fixed-point behavior and another with the floating-point values. At the same time, we keep track of the difference between floating-point and fixed-point values (Δ).

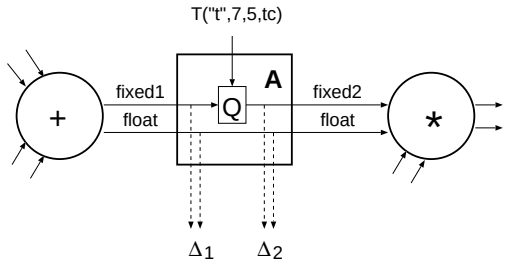

Figure 3. Derivation of error statistics for a signal (Q = quantization with defined type)

During the simulation, the error Δ from the input and already quantized signals propagates (*error propagation*) through all signals in a system. During signal assignments the difference error statistics are collected (Figure 3), both for input (consumed) difference error Δ_1 and output (produced) difference error Δ_2. These signal statistics include the mean error (μ), the standard deviation (σ) and the maximum absolute error (ϵ). These measures are directly related to the LSB position determination.

Divergence of the floating-point and the fixed-point calculation might occur in case of sensitive feedback signals. This is a consequence of the strong correlation of error values between iterations. As a result the statistics become irrelevant. To break the loop the `error()` method is introduced to define an explicit difference error Δ_2 for a signal. The following example defines this difference for the signal `a` to be a uniform random variable with $\sigma = 0.0156$ and $\mu = 0$ which corresponds to the type definition with the LSB position $f = 5$.

```
sig a("a");
a.error(0.0156);
```

Quantizing feedback signal paths still requires the final verification of the system stability and precision. This is due to effects like limit cycles.

This approach outperforms the strategy where the floating-point and the fixed-point simulation are executed in two separated runs. The advantages are that:

- the error difference statistics are effectively gathered for each signal in the system (no need for huge signal databases)
- it is guaranteed that the floating-point and fixed-point simulations take the same control decisions throughout the simulation. Such control decisions can be determined by a relational expression, which can be different for floating-point and fixed-point simulation. Therefore the relational operations are evaluated uniformly, that is, the floating-point simulation is steered by fixed-point control decisions.

5 Design flow

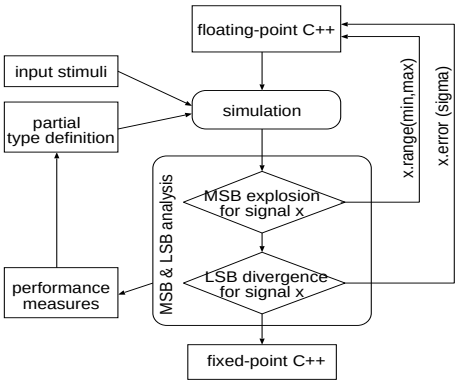

Figure 4. Design flow

Our design flow of floating-point to fixed-point refinement is shown in Figure 4. The input to our design flow is a floating-point description compliant to our C++ design model. Based on the simulation with a partial type definition (like quantization of input signals) the fixed-point description is derived. The simulation generates several quantization measures which must be properly interpreted in order to determine optimal MSB and LSB positions of all signals. This is the topic of the following sections. A reiteration might be needed in case of feedback signals. The methods *range* and *error* (which affect MSB and LSB independently unlike the type definition which affects both) were already introduced to solve possible divergence problems. Another possible iteration is needed when the performance results are not satisfactory. This is an implication of the partial type definition, which must then be revised.

5.1 Refinement rules for MSB side

The goal of this step is to decide for an MSB mode (saturated, wrap-around or error-typed) of signals and to resolve an optimal MSB position. The MSB refinement usually starts with the floating-point representation of all signals in the design except for the input signals. Signals which are already quantized are checked against overflows. Let's define a function Q (assuming 2's complement numbers) that takes the range values as parameters and returns the required MSB

$$Q(min_R, max_R) = MSB = b + 1 :$$

$$2^{b-1} < MAX(|min_R|, |max_R|) \leq 2^b$$

The two range monitoring methods, statistic-based (stat) and range-propagation (prop), can show one of the following relationships.

a) $Q(min_{prop}, max_{prop}) = Q(min_{stat}, max_{stat})$
It shows that the signal is guaranteed by both techniques not to overflow. The signal can safely be specified with MSB as obtained by simulation and the nonsaturated (error-type or wrap-around) MSB mode.

b) $Q(min_{prop}, max_{prop}) >> Q(min_{stat}, max_{stat})$
It shows that the range propagation gives very pessimistic result. Typically this is the case when an accumulation variable is concerned. It is recommended to switch the type into the saturation mode or to use the explicit saturation (*range* method) on the floating-point type.

For all signals in saturation the simulation generates also the information of guard range boundaries for safe hardware implementation of the saturation scheme.

c) $Q(min_{prop}, max_{prop}) > Q(min_{stat}, max_{stat})$
This situation gives the trade-off to choose either MSB specification based on the range propagation or the saturation mode type based on the simulation. Still it is possible that simulation didn't trigger the worst case behavior.

5.2 Refinement rules for LSB side

In order to obtain the LSB side quantization, both the LSB position and mode (round or floor) has to be determined.

The LSB refinement process should start with a fixed-point specification of input signals. The input signal quantization in DSP applications is typically known as the input comes from an AD converter and/or a SNR scenario is specified.

The signals which are to be refined for LSB should have either the floating-point or large LSB position definition. Feedback signals should be identified and set to explicit LSB behavior through applying the *error* method if they cause the floating-point/fixed-point divergence.

After simulation the μ, σ and ϵ statistics give the upper-bound for LSB specification. This means a higher LSB precision is not required because it doesn't bring any improvement (i.e. the LSB part is drowned in quantization or external noise). The rule is:

$$2^{-LSB} < \sigma * K_{em}$$

The K_{em} is the empirical constant which was found to give optimal results for the range $(1, 4)$. The smaller K_{em} is applied, the more conservative determination of LSB is obtained.

Further decrease of LSB quantization can proceed depending on the specific precision requirements (ϵ, μ, σ) of e.g. output signals.

Already quantized signals are checked for correctness of quantization. They bring different values of Δ_1 and Δ_2 (see Figure 3) which yields information on consumed precision LSB_1 and produced precision LSB_2. Generally $LSB_1 \geq LSB_2$. If $LSB_2 < LSB_1$ a precision loss due to quantization occurs. The designer must resolve whether it is intentional or not. Note that the floating-point signals pose $LSB_1 = LSB_2 = LSB$.

The feedback signals simulated with *error* method might show $LSB_2 > LSB_1$. It implies that the precision loss which might cause instability of system behavior is detected in the feedback path.

The type refinement from the round-type to floor-type specification will bring a shift of the μ measure. If such a shift is unacceptable the signal must stay round-typed, otherwise the floor-type is recommended as it leads to a cheaper hardware implementation.

6 Results

MSB determination

The goal to determine optimized MSB values for all signals of the motivational example was achieved after two iterations which is shown in Table 1.

The algorithm was evaluated with the statistic-based and quasi-analytical technique in the same simulation run which took only a fraction of a second for this example. The columns show respectively: signal name, number of accesses, minimum, maximum and msb values observed by the statistic-based approach. Next to this: minimum, maximum and msb value inferred by range propagation approach. Shown rightmost is the decided msb value obtained through applying the MSB refinement rules.

In the constructor definition we inserted `x.range(-1.5,1.5)` to initialize the range propagation. The first iteration gave satisfactory determination of all signals except for b and w. These suffered from the range propagation explosion caused by feedback effects.

For the second iteration `b.range(-0.2,0.2)` was added to help the range propagation method by specifying the realistic saturation on the signal b. Consequently b and w were successfully resolved. The boundary for the hardware saturation of the signal b was also obtained.

1st iteration								
		statistic-based			range-propagation			
name	#n	min	max	msb	min	max	msb	MSB
c_0	1	-0.11	-0.11	-2	-0.11	-0.11	-2	-2
c_1	1	1.20	1.20	2	1.20	1.20	2	2
c_2	1	-0.11	-0.11	-2	-0.11	-0.11	-2	-2
x	2000	-1.16	1.16	2	-2.00	2.00	2	2(st)
	x.range				-1.50	1.50		
d_0	2000	-1.16	1.16	2	-1.50	1.50	2	2
d_1	2000	-1.16	1.16	2	-1.50	1.50	2	2
d_2	2000	-1.16	1.16	2	-1.50	1.50	2	2
v_1	2000	-0.13	0.13	-1	-0.17	0.17	-1	-1
v_2	2000	-1.44	1.48	2	-1.97	1.97	2	2
v_3	2000	-1.37	1.39	2	-2.13	2.13	3	3
w	2000	-1.36	1.37	2	-22.0	22.0	6	?
b	2000	-0.00	0.07	-2	-19.9	19.9	6	?
y	2000	-1.00	1.00	2	-1.00	1.00	2	2
2nd iteration								
				.....				
w	2000	-1.36	1.37	2	-2.33	2.33	3	3
b	2000	-0.00	0.07	-2	-0.21	0.21	-1	-1(st)
	b.range				-0.20	0.20		

Table 1. MSB analysis

name	#n	ϵ	μ	σ	LSB
x	2000	1.5e-02	-2.9e-04	9.0e-03	6
			.....		
v_1	2000	1.7e-03	3.2e-05	1.0e-03	9
v_2	2000	2.0e-02	-3.2e-04	1.1e-02	6
v_3	2000	2.1e-02	-2.8e-04	1.1e-02	6
w	2000	2.2e-02	-2.9e-04	1.1e-02	6
b	2000	6.0e-04	-2.6e-04	1.8e-04	12
y	2000	0.0e+00	0.0e+00	0.0e+00	0

Table 2. LSB analysis

LSB determination

The determination of the worst case LSB is shown in Table 2. The columns denote respectively: signal name, number of assignments, ϵ, μ and σ statistic measures. Shown rightmost is the inferred LSB position obtained through applying the LSB refinement rule with $K_{em} = 2$. We quantized the input signal `x` with the format `<7,5,tc>` and one iteration resolved LSB positions of all signals. The following simulation with the derived quantization of all signals confirmed the stability of the system.

In order to see the impact of the LSB refinement the SQNR (signal-to-quantization-noise ratio) measure is observed. For example, SQNR of the signal w before the LSB refinement (with quantizing the input signal x only) was 39.8 dB, and after the LSB refinement (all signals quantized) 39.1 dB.

6.1 Complex example

Figure 5 shows the timing recovery loop system which contained 61 signals subject to the fixed-point refinement. A safe MSB quantization was achieved when putting 7 signals to the saturation mode. 2 feedback signals required saturation due to the MSB explosion, while the other 5 signals were the knowledge-based choice. The rest (54 signals) remained in the non-saturated mode with MSB overhead of 0.22 bits per signal compared to statistic-based results. Only 2 iterations were needed to resolve the MSB weights. In the LSB quantization there was only the mu signal inside of NCO (out of the 2 feedback signals in the system) of which the error calculation was unstable. Applying the overruling error method subsequently solved the problem. After finding a stable quantization of the mu signal one iteration successfully determined all other LSB weights.

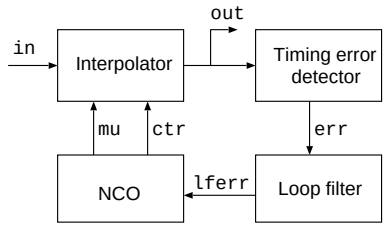

Figure 5. Timing recovery loop for PAM signals

7 Conclusions

A strategy for fixed point refinement in the hardware design of DSP algorithms was presented. We relied on C++ to embed this strategy in an object-oriented hardware design system. The fixed point refinement uses separate approaches for the MSB and LSB side of a signal, and we described the quantization rules for each of those. Our technique marries the advantages of a pure simulation based approach and a pure analysis based approach, which are the two main techniques in use at the moment. The result is a quantization strategy that allows determination of a safe quantization in a few number of iterations. The strategy has been successfully applied in a number of advanced signal processors, including a cable modem and a DECT base station signal processor.

References

[1] W.Sung, K.Kum "Simulation-Based Word-Length Optimization Method for Fixed-Point Digital Signal Processing Systems", IEEE Transactions on Signal Processing, vol.43, pp.3087-3090, Dec.1995

[2] S.Kim, K.Kum, W.Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs", Workshop on VLSI and Signal Processing '95, Osaka, pp.197-206, Nov. 1995

[3] M.Willems, V.Bursgens, H.Keding, T.Grotker, H.Meyr, "System Level Fixed-Point Design Based on an Interpolative Approach", Proc. of the DAC, Anaheim, 1997

[4] Frontier Design, *http://www.frontierd.com/artlibrary.htm*

[5] P.Schaumont, S.Vernalde, L.Rijnders, M.Engels, I.Bolsens, "A Programming Environment for the Design of Complex High Speed ASICs", Proc. of the DAC, Anaheim, 1997