2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Quantization Methodology based on Value Range Analysis

Shigetaka Nata, Siemens EDA Japan (*shigetaka.nata@siemens.com*)

Petri Solanti, Siemens EDA (petri.solanti@siemens.com)

***Abstract*—Algorithm developers are usually using double precision data types to be able to focus on the mathematical functionality of the algorithm. When this algorithm is implemented as a hardware module, the data accuracy must be reduced to minimum number of bits that still fulfills the system performance requirements. The process of converting the floating-point algorithm to bit-level optimized model is complicated and requires special knowledge. This paper introduces a simple, easy, and robust quantization methodology based on value range analysis.**

***Keywords*—quantization; algorithm; floating point; fixed point; high level synthesis; ac datatypes**

I. Introduction

Quantization is important design phase and there are many definitions for it. In the electronics context it means constraining a signal from a large set of values to a limited set of values. Some philosopher described it as the art of reducing data accuracy without losing algorithm performance.

Algorithm developers like to use double precision floating-point numbers. It allows them to focus on the mathematical solution without having to think about peak values or minimum values or any other irrelevant aspects. When the algorithm works correctly in floating-point domain, it can be analyzed and converted to a fixed-point model with optimal word lengths.

Quantization process itself is a thoroughly studied domain. There are lots of academic studies available, but most of them are not suitable for hardware design projects. Therefore, a simpler approach is needed. The simplest quantization methodology is simulation based dynamic analysis. Another option is static mathematical analysis. Usually both methodologies are needed to reach the best results.

Reducing the set of available values causes some unavoidable side effects. The most important one is the quantization noise that is the difference between the floating-point value and the quantized value. The diagrams in Fig.1 show the difference between floating-point representation that has almost unlimited number of values, and 5-bit fixed-point representation having only 32 possible values. Quantization noise is not nicely behaving signal like white noise, because its' spectrum is non-linear and depends on many different parameters.

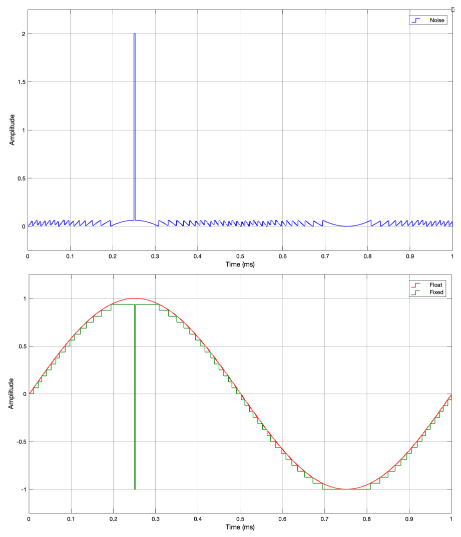


Figure 1 – Quantization noise

Another nasty phenomenon is the overflow behavior of the integer and fixed-point data types. When the value to be assigned to the variable or signal exceeds the maximum value that can be represented with this data type, the sign of the value is inverted, thus the value switches from positive to negative or vice versa from one sample to another. In feedback structures like IIR filter this transforms the filter to an oscillator. This can be avoided by using saturation function of the data type.

A less known side effect is quantization offset. If the accuracy is reduced by truncating the least significant bits without rounding, the error is always into the same direction. This can be clearly seen in the upper diagram. In accumulating structures like integrators this causes a crawling effect. This problem can be avoided by using a correct rounding scheme in the critical places of the design.

II. Quantization Process

Figure 2 shows the quantization process, which is an iterative process that can use either simulation based or analytical method.

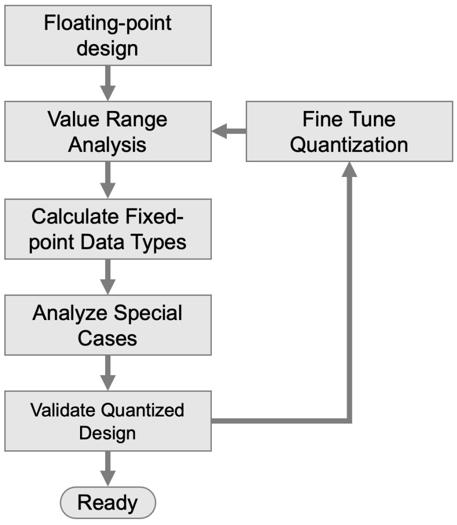


Figure 2 – Quantization Process

A. *Simulation Based*

- Simulation based quantization begins with the floating-point design.
- In the first step, the design is simulated with instrumented wide range data types that collect the value data.
- The initial fixed-point bit widths are calculated based on the analysis data.
- Some special cases like constants and coefficients that are initialized only once require manual analysis.
- When all fixed-point types are declared, the quantized design is validated with simulation to ensure that the system performance fulfills the requirements.
- Usually, the initial quantization is not optimal. Running value range analysis with the initial fixed-point types gives additional information about the quantization quality.
- The optimal quantization can be reached after 3-4 iterations. Sometimes a few additional iterations are needed.

B. *Analytical Method*

- Analytical quantization method can be used for certain types of designs.
- The analysis begins with input and output data type analysis.
- When the input types are fixed, the data path word lengths can be calculated based on the arithmetic operations.
- For validation and fine tuning the same process steps used in the dynamic analysis apply.

Quantization process has three phases that use the process described above, which is (1) pre-quantization phase, (2) coarse quantization phase and (3) bit utilization analysis phase.

(1) In the pre-quantization phase, the inputs and outputs are quantized. Usually a digital system has bit-width limitations by A-to-D converter or interconnect width. This limits the stepsize and causes the quantization effects even if the rest of the design is still using floating-point types. These quantized values propagate throughout the system carrying the step size from operation to operation. These step sizes are detected by the value range analysis and used for fixed-point type definition.

(2) In the coarse quantization phase, all internal bit widths are analyzed using value range analysis. The analysis results provide a reliable number of integer bits and a range of fractional bits. Based on this information, the internal fixed-point data types are calculated, and variable declarations modified accordingly.

(3) In the bit utilization analysis, the fully quantized design is simulated and analyzed for overflows, underflows and possible unused bits.

Principally, many designs can be analyzed manually without bigger effort. For example, an FIR-filter is easy to analyze without running a single simulation. Yet, the value ranges are not always predictable. In feedback architectures like IIR-filter, the signal level differences may be up to 10.000x and dependent on several parameters like signal frequency or cumulative offset. In the example design the input and output value range is +/-2, but the feedback accumulator reaches values below -10.000. Therefore, a value range analysis using floating-point or wide fixed-point type, for example 64-bits total word length and 32 integer bits is necessary.

III. DYNAMIC VALUE RANGE ANALYSIS

Dynamic value range analysis collects data value metrics during the simulation. The values can be stored into a vector and post processed later or the statistics can be collected at every value assignment, stored into the data object and printed out at the end of the simulation. This requires instrumentation of the model but gives reliable data for all nodes in the complete design. The QoR depends heavily on the stimulus quality.

Static analysis relies on the value ranges of the inputs. The maximum and minimum values after the arithmetic operations and the number of required bits can be calculated. Addition and subtraction increment the number of integer bits by one and multiplication output has the sum of the integer bits of the operands and the total number of bits is the sum of the word lengths of the operands. At the model output, the number of decimal bits limits the visibility of the internal decimal bits that reduces the need for internal bits. Thus, this method is not suitable for a complex design. Static analysis is error prone with feedback systems such as IIR filter, but it works well for simple feed-forward systems such as FIR filter.

To determine the required number of integer and fractional bits in the fixed-point type we need the followings:

- Maximum absolute value of the signal assigned to a variable → Number of integer bits required to handle the value without overflow
- Minimum non-zero absolute value → Minimum value to set a fraction bit
- Minimum non-zero difference between two samples → Minimum value that toggles a bit
- Signed ness of the signal →Adds one bit to the integer part

Please refer to the Figure 3. for each of relevant metrics. These metrics enable almost lossless implementation of the algorithm. To determine the final number of bits, a careful analysis of noise figure and saturation needs is required.

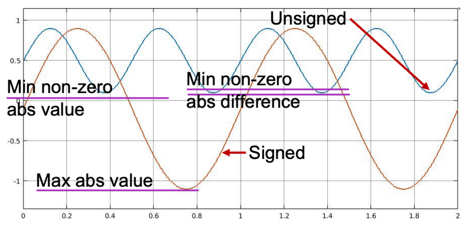


Figure 3. Relevant Metrics for number determination

Simulation-based Value Range Analysis begins with floating-point or wide range fixed-point types that have enough bits to handle the required headroom and accuracy of each node in the design. A 64-bit fixed-point type is a good starting point and if it is not enough, 128-bit fixed-point can handle almost anything. Then follows the steps below:

1. The first step is to quantize all inputs and outputs of the design. In a large hierarchical design this should be done to each block individually. This quantization doesn't need fixed-point data types, but the number of possible values can be limited by using a pseudo-quantization method.
2. The algorithm performance is degraded due to the loss of precision in the block interfaces. The interface bit widths should not be driven to minimum, because the block internal quantization adds some more noise to the signal and there must be enough tolerance to handle that.
3. When the I/O quantization is done, all variables in the design must be instrumented. This is easy to do with ac_fixed class that has built-in instrumentation. Simulation of the instrumented design outputs the value range metrics for all variables of ac_fixed type with the given stimulus.
4. The pre-quantization limits the number of possible values inside the design. For example, if un-quantized input signal results after an addition a minimum difference that requires 12 fractional bits, an input signal quantized to 6 fractional bits results a minimum difference that requires 6 fractional bits.
5. Execute simulation of instrumented model and analyze max and min value of the simulation data.

Where Pseudo quantization is a method to limit the number of possible values of the variable by quantizing the value and assigning it back to a floating-point or wide fixed-point variable. The number of value levels of this variable carries are just the quantized values. For example, converting a value to 6-bit fixed-point type, reduces the possible number of value levels to 64, whereas a double-precision floating-point type has 2 to power of 53 possible levels scalable by 11 bits exponent. Pseudo-quantization generates all fixed-point dirty effects of the quantized design without need to change anything in the algorithm. Saturation and rounding can be used in the pseudo-quantization.

IV. Static Range Value Analysis

Static value range analysis calculates the output bit widths of the arithmetic operations based on the input data types, essentially assign max and min values to operands and calculating output values through the signal chain.

Let's look at the example - 32 tap FIR filter with Input<16,2, true>, Coeff<16,0, true>, Output<16,2, true>.

The first step is to analyze, if the 2 integer bits can handle all possible outputs without risk of overflow. The worst-case output can be only as high as the sum of the absolute values of the coefficients multiplied by the maximum input value. In this case, the sum of absolute values of the coefficients is 1.3657 and maximum absolute input value is 2.0, so the maximum possible output value is 2.7314 and it is a signed value requiring 3 integer bits. The output data type with only 2 integer bits must be a saturating type to avoid overflow in worst case.

The accumulator type must be able to handle the worst-case maximum value and the number of fractional bits must ensure the accurate 14 fractional bits to the output. A 16-by-16-bit fixed-point multiplier has 32 output bits with 2 integer bits with the given inputs. The output is reduced to 14 fractional bits, but because 32 multiplier outputs are accumulated, the uniform distributed error of the loss-of-precision at every multiplier output must be taken in account.

This can be calculated by multiplying the rounding error to 14 fractional bits by number of accumulations, which is 32. Base 2 logarithm of this value is -18,002 rounded down to -19, resulting 19 fractional bits. 18 fractional bits can be enough, but it must be validated by simulation. Thus, the resulting accumulator width is 22 bits with 19 fractional bits.

V. Handling Special Cases

Some values need closer analysis. For example, constants and configuration settings may be instantiated only once, thus, the minimum value is the same as the maximum value and it is not possible to calculate the minimum difference between two consecutive samples. Yet especially the filter coefficient accuracy has a significant impact to the system functionality.

Also, the values that are close to the next integer or between 0.5 and 1 are problematic because Log2 returns a small number of fractional bits although many bits are needed. And for the Power of 2 integer values require special attention because Log2 returns one bit less than needed to represent the value.

For example, constant values can be analyzed with the static approach. Integer value is easy to calculate, but the number of fractional bits is more complicated.

Case1	Value	Log2	Signed	Bits	Total bits
Const value	1.99996	0.999971	false	1	1 integer bit
Min value	0.00004	-14.60964	d.c.	15	15 fractional bits

Case2	Value	Log2	Signed	Bits	Total bits
Const value	-0.51	-0,97143	true	0 + sign	1 integer bits
Min value	0.01	-6,643856	d.c.	7	7 fractional bits

Figure 4 Special Case Examples

In the Case1 in Figure 4, calculating the base 2 logarithm for the fractional part returns 6 fractional bits, which is probably too few. Subtracting the fractional part from the closest integer value, in this case one, gives a minimum value that returns 15 fractional bits. The resulting fixed-point type has 16 bits with 15 fractional bits, and it is unsigned. But as seen in the Case2 in Figure 4, sometimes one subtraction is not enough, but the constant value subtraction must be iterated multiple times, which is to continue subtracting 2^{-N} (0.5, 0.25, …) until a value small enough is reached.

VI. AC DATATYPE AND VALUE RANGE ANALYSIS

The Algorithmic C datatypes (AC Datatypes) include a numerical set of datatypes and an interface datatype for modeling channels in communicating processes in C++. The numerical datatypes provide an easy way to model static bit-precision with minimal runtime overhead. The numerical datatypes were developed to provide a basis for writing bit-accurate algorithms to be synthesized into hardware. Operators and methods on both the integer and fixed-point types are clearly and consistently defined so that they have well defined simulation and synthesis semantics. They enable algorithm, system, and hardware designers to precisely model bit-true behavior in C++ specifications while accelerating simulation speeds by 10-200x faster versus alternate bit-accurate integer and fixed-point datatypes.

AC Datatypes package can be found at https://hlslibs.org and downloadable from GitHub as well. The package is licensed under Apache 2.0 license. And AC Datatype Value Range Analysis feature is built in its ac_fixed datatype and it allows simple and easy analysis in quantization process. With the compiler flag "-DAC_FIXED_VRA" specified, the multiple metrics analysis can be done during simulation execution, such as minimum and maximum values, signed-ness, minimum absolute fractional value, activation count (number of assignments to the variable), overflow count, and minimum quantization error.

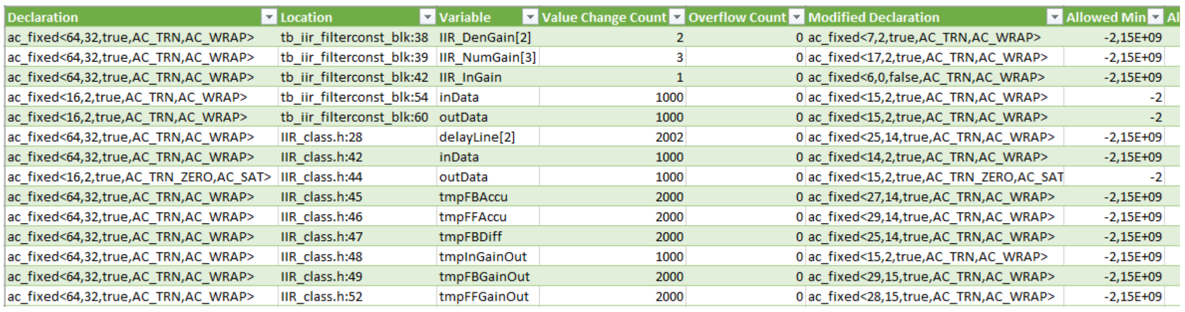
At the end of the simulation run, Value Range Analysis prints out a report for each fixed-point variable that was assigned at least once during the simulation, shown in Figure 5.

```
------------------------------------------------------------
DECLARATION: ac_fixed<16,2,true,AC_TRN,AC_WRAP>
  Available Value Range  (min -2 : max 1.99994)
  Observed Value Range   (min -1.71369 : max 1.71387 : min_frac 0.000210514)
  Signed                 = true
  Fractional bits        = 13..18
  Value Change Count     = 1000
  Overflow Count         = 0
  Smallest Quant Err     = 0.000001
  RMS Quant Err          = 0.000000
  Modified declaration   = ac_fixed<15,2,true,AC_TRN,AC_WRAP>
  CALL STACK:
    tb_iir_filter.cpp:54 'main'
 /home/projects/Quantization/src_IIR_demo/tb_iir_filter.cpp:54
------------------------------------------------------------
```

Figure 5. ac_fixed Value Range Analysis printed report

The report contains the available and observed minimum and maximum values, signed-ness, value change count and overflow count. It also gives a range of fractional bits. The first number is a number of fractional bits calculated from the minimum non-zero absolute value. The second number is iterated by subtracting the power of 2 value of the first fractional bit count from the minimum absolute value and calculating the base 2 logarithm of the remainder. The iteration is continued until the remainder is less than the epsilon value, which is by default 10 to -6. The purpose of the fractional bit range is to give the minimum number of fractional bits that works safely and the maximum number of bits that still make sense. The report proposes also a new fixed-point declaration to the variable.

In the large designs the report log becomes long and difficult to analyze. But the report can also be dumped into a .csv file that can be analyzed with any spreadsheet tool. The csv log feature must be activated by setting the environment variable $AC_FIXED_VRA_OPTS to '-f<filename>'. Then csv file can be seen as Figure 6.



Declaration	Location	Variable	Value Change Count	Overflow Count	Modified Declaration	Allowed Min	Al
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	tb_iir_filterconst_blk:38	IIR_DenGain[2]	2	0	ac_fixed<7,2,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	tb_iir_filterconst_blk:39	IIR_NumGain[3]	3	0	ac_fixed<17,2,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	tb_iir_filterconst_blk:42	IIR_InGain	1	0	ac_fixed<6,0,false,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<16,2,true,AC_TRN,AC_WRAP>	tb_iir_filterconst_blk:54	inData	1000	0	ac_fixed<15,2,true,AC_TRN,AC_WRAP>	-2	
ac_fixed<16,2,true,AC_TRN,AC_WRAP>	tb_iir_filterconst_blk:60	outData	1000	0	ac_fixed<15,2,true,AC_TRN,AC_WRAP>	-2	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:28	delayLine[2]	2002	0	ac_fixed<25,14,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:42	inData	1000	0	ac_fixed<14,2,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<16,2,true,AC_TRN_ZERO,AC_SAT>	IIR_class.h:44	outData	1000	0	ac_fixed<15,2,true,AC_TRN_ZERO,AC_SAT	-2	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:45	tmpFBAccu	2000	0	ac_fixed<27,14,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:46	tmpFFAccu	2000	0	ac_fixed<29,14,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:47	tmpFBDiff	2000	0	ac_fixed<25,14,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:48	tmpInGainOut	1000	0	ac_fixed<15,2,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:49	tmpFBGainOut	2000	0	ac_fixed<29,15,true,AC_TRN,AC_WRAP>	-2,15E+09	
ac_fixed<64,32,true,AC_TRN,AC_WRAP>	IIR_class.h:52	tmpFFGainOut	2000	0	ac_fixed<28,15,true,AC_TRN,AC_WRAP>	-2,15E+09	

Figure 6. CSV Report generated from Value Range Analysis

In the csv log, the most important data is collected into the first 6 columns to make the analysis easier. The fractional bit range is not included, because it is not needed that frequently.

VII. VALIDATING QUANTIZED DESIGN

Simulation is the only reasonable tool for validating the quantized design against the original floating-point golden model. System-level performance requirements like Signal-to-Noise Ratio or Bit-Error-Rate can be determined by thorough simulations.

Verification of the quantized design should take place in multiple steps:

- The first verification should be done for the pseudo-quantized model. If the performance doesn't meet the requirements, the number of moving parts is still small and it is easier to debug the model.
- The second analysis should be done with fully quantized model.
- For further block-level optimization, a block-level performance analysis can be used. It helps to focus on one block at the time, when all other blocks are pseudo-quantized.

Automatic regression is recommended to detect possible performance issues as early as possible.

VIII. SUMMARY

Quantization is a crucial part of hardware design process, which optimizes design area and power consumption while managing noise level, offset and overflow behavior of the algorithm implementation.

Value Range Analysis based quantization methodology with static analysis of special cases is a simple and robust way to analyze the needed fixed-point word lengths for any types of digital designs. To make this methodology easier, we embedded the mechanism of the Value Range Analysis feature into the one of the AC Datatypes which is ac_fixed. This feature makes quantization easy in terms of a) Automating instrumentation of all fixed-point variables, b) Capturing quantization information during simulation run and proposing fixed-point datatype for each variable individually, and c) Analyzing of required number of integer bits and a range for number of fractional bits.

Fixed-point effects must still be analyzed with the original testbench to ensure the algorithmic performance of the fixed-point design. And again AC Datatypes with Value Range Analysis feature is available through Apache 2.0 license.

References

[1] HLS LIBS website - https://hlslibs.org

[2] HLS Libs GitHub site - https://github.com/hlslibs

[3] “Algorithmic C Datatypes” presented by Kunal Bindal, Siemens EDA to SystemC Datatypes Working Group, at SystemC Evolution Day, 2017 - https://workspace.accellera.org/document/dl/10940