Mentor®
A Siemens Business

Algorithmic C (AC) Math Library

Sine/Cosine Lookup Table ac_sincos_lut()

May 2018

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777. Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210 Website:
www.mentor.com
SupportNet: supportnet.mentor.com/

Introduction

The ac_sincos_lut function is designed to provide sine and cosine values using a lookup table(LUT) mechanism. The default lookup table is populated with 512 evenly-spaced entries for both the sine and cosine functions and has been shown to have very nice accuracy and hardware representation.

For details of ac_sincos_lut implementation, as well as how to change the table size, refer to $MGC_HOME/shared/pdfdocs/ac_math_ref.pdf.

The Sine/Cosine Lookup Table Toolkit

The ac_sincos_lut toolkit is accessed in Catapult by picking Examples->Math->AC Sine/Cosine. The example uses fixed-point input and output data types with representative choices made as to data width and precision. The example provides a C++ testbench which configures and calls the ac_sincos_lut function for a number of datapoints, comparing results with the C++ math functions cos() and sin().

This document is accessed by selecting "AC Sine/Cosine Function" in the Documentation section.

Invoke Catapult in a clean directory, and select "Run fixed-point example (ac_fixed)", then click on "Export Files". The following files are observed:

- ReadMe.txt: Describes the toolkit briefly.
- ac_sincos.pdf: This document.
- ac_sincos_lut_tb.h: Header file for the testbench function which calls ac_sincos_lut - "project". Also defines data types for input and output signals. Data types are defined using a mechanism to allow compile-time override of default values.
- ac_sincos_lut_tb.cpp: C++ file which implements the "project" function which calls ac_sincos_lut. Additionally implements a testbench to exercise ac_sincos_lut, and check for correctness and accuracy against C++ cos() and sin() functions.
- ac_sincos_lut_tb_ac_fixed.tcl: Script to run Catapult to synthesize the ac_sincos_lut function using ac_fixed input and output data types. SCVerify is used to build and execute the testbench to exercise the ac_sincos_lut function. A Simulink S-Function is also produced by this script.
- ac_sincos.mdl: A Simulink model pre-configured to exercise the ac_sincos_lut function in Simulink and compare accuracy with the Simulink trigonometric function.
- lutgensincos.cpp: A C++ file which can be used to generate a lookup-table with sizes that are different than the default.

The ac_sincos_lut_tb.h defines input and output datatypes, and compile-time macros to change default bit widths and signedness. Data definitions are:

```
typedef ac_fixed<Wi, Ii, Si, AC_RND, AC_SAT>          input_type;
typedef ac_fixed<Wout, Iout, Sout, AC_RND, AC_SAT>    output_ri_type;
```

```
typedef ac_complex<output_ri_type>                    output_type;
```

The constant variables `Wi`, `Li`, and `Si` define input bit width, integer bits, and signedness respectively. As described in ac_math_ref.pdf, the input domain need be no more than 0 to 1.0; and 512 table entries are used. Hence default width is 12 bits, with 0 integer bits, and unsigned. These constants can be changed using macros `Wivalue`, `Iivalue`, and `Sivalue`.

The constant variables `Wout`, `IOut`, and `Sout` define output width, integer bits, and signedness respectively. The range of sine and cosine is -1.0 to 1.0, and chosen defaults are 24 bits wide, 2 integer bits, and signed. These constants can be changed using macros `Woutvalue`, `Ioutvalue`, and `Soutvalue`.

An example for how to set these definitions for a catapult run, narrowing output width, would be:

```
options set Input/CompilerFlags {-DWoutvalue=20}
solution options set Input/CompilerFlags {-DWoutvalue=20}
```

Note that the output ac_complex type returns both sine and cosine. Cosine is the real value, sine is the imaginary value.

Running the Lookup Table Sine/Cosine Example

After exporting files as discussed above, type "source ac_sincos_lut_tb_ac_fixed.tcl" in Catapult. Alternatively, clicking on "Launch Project" will have the same effect. The function "project" in ac_sincos_lut_tb.cpp is synthesized. The Catapult transcript shows the following:

- Synthesis output: The top-level function "project" calls the ac_sincos_lut function, with ac_fixed input and output data widths as described above.
- SCVerify testbench compilation and run: this output is discussed below in this document.
- Simulink S-Function compilation: The Matlab MEX compiler is used to compile the synthesized design into a Simulink-compatible S-Function. More discussion of this is below.

SCVerify Behavior and Output

In the ac_sincos_lut_tb.cpp file a testbench to excersise ac_sincos_lut is provided. This testbench is implemented in CCS_MAIN, and "sweeps" from some minimum value to a maximum value, incrementing by some step size. At each step, the C++ sin() and cos() functions are called, and compared to ac_sincos_lut output. A percentage error is calculated, and if this error percentage is over some value, an error is flagged.

In this example, we sweep input from 0 to 1 by increments of one "Quantum". A Quantum is based on precision of the fraction size of the input variable. For this testcase the input is a fixed point with 12 bits width using 12 fraction bits; thus a Quantum is 1/(2^12) or 2.44141e-4. This is 4096 sample points by default as the test is set up.

At each datapoint, a comparison is made to the C++ math functions cos() and sin(). A difference percentage is calculated, and if the maximum difference is over one percent, an error is flagged. These parameters are specified in the ac_sincos_lut_tb_ac_fixed.tcl file by this line:

```
flow package option set /SCVerify/INVOKE_ARGS {0 1 - 1}
```

The first two arguments are min and max values to be applied. Third parameter is increment size where '-' means to use a Quantum from input specification, and fourth argument is allowed difference threshold as a percentage. Difference percentage is calculated for each output using the following equation:

$$d(x,y) = \frac{abs(x-y)}{\max(abs(x), abs(y))} \times 100$$

At the end of the SCVerify run, these lines are printed:

```
# ============================================
# Simulating design
# cd ../..;
./Catapult/project.v1/scverify/orig_cxx_osci/scverify_top 0 1 - 1
# ==================== ac_sincos_lut Test ======================
# lower_limit      = 0
# upper_limit      = 1
# step             = 0.000244141
# allowed_error    = 1
#
# Testbench finished
# Cosine: max_error =0.00126403  with sample =0.751221
expected=0.00766983  actual=0.00766993
# Sine: max_error =0.00126403  with sample =0.501221  expected=-
0.00766983  actual=-0.00766993
# ============================================
```

SCVerify output shows simulation parameters followed by test results. We see for cosine that maximum difference was 0.001% using input value 0.751221. Maximum sine difference is 0.001% using input value 0.751221.

A plot_values.csv file is optionally produced which is suitable to be read into Excel for further analysis if desired. This file has data for each datapoint simulated, and enabled by specifying this compile-time flag:

```
-DPLOT_RECIPROCAL_FILE_WRITE
```

In computing difference percentage, note that an odd thing happens when either value is exactly 0 (0 is legal value for sin() and cos()). If either value is 0, and the other is any non-zero value, then the above expression will compute 100% error differential. If both values are 0, then division by 0 is attempted which is also bad. Because of this, the error computation special-cases these scenarios to avoid pessimism. If both values are zero, then zero error is used. If only one value is zero, then the error is an extrapolation based on history of computations of the previous two datapoints - else 0 is used.

Simulink Simulation and Accuracy Visualization

As discussed above, default accuracy for the ac_sincos_lut function is very nice. However, if different parameters are used, accuracy is likely to be affected. A Simulink setup, or harness, is provided to allow the user to modify ac_sincos_lut parameters, then measure accuracy. This section shows how to use Simulink to measure accuracy for the default toolkit setup, as well as how to modify bit widths and measure the impact on accuracy. In order to do either of these, it is required that the user have Simulink licensing and installation of R2017b or later. The MATLABROOT environment variable should be set to that installation.

Default Toolkit Model Measurements

In Catapult, after source-ing ac_sincos_lut_tb_ac_fixed.tcl, issue this command:

```
flow run /Matlab/launch_matlab ac_sincos.mdl
```

After some time, Matlab/Simulink will be seen. This command invokes matlab in the ./simulink directory, then opens the ac_sincos.mdl system model. A pre-configured Simulink model is opened. This setup will simulate the ac_sincos_lut function for 10 seconds of time, using a simple ramp function with step size .0024 seconds; there will be 4096 samples in all. The ramp is configured to sweep from 0.0 to 1.0 over the 10 seconds. The ac_sincos_lut function is embedded in the S-Function.

The Simulink schematic looks something like Figure 1:

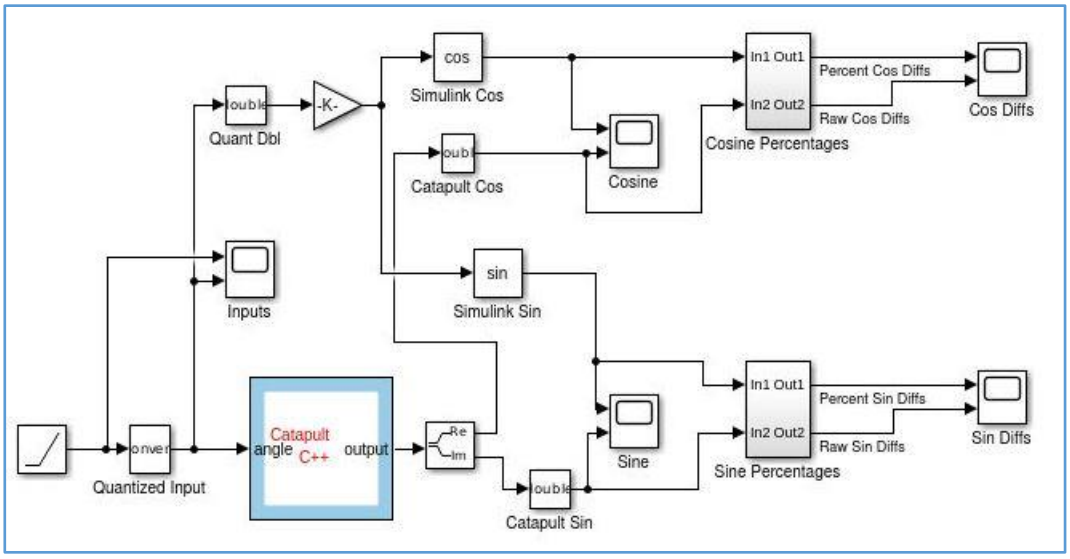

Figure 1: Simulink Schematic for ac_sincos_lut Measurements

It is seen in Figure 1 that the Simulink ramp source drives both the catapult S-Function for ac_sincos_lut and the Simulink trigonometric function configured as both sine and cosine. Scopes are used to view ramp input ('Inputs'), cosine outputs ('Cosine'), sine outputs ('Sine'), and differences between the two models for sine and cosine comparisons ('Cos Diffs' and 'Sin Diffs').

The percent difference between two model outputs is computed by matlab at each timestep as:

$$d(x,y) = \frac{abs(x-y)}{\max(abs(x), abs(y))} \times 100$$

Simulink sine/cosine is taken as "x", and the corresponding catapult approximated ac_sincos_lut value for sine/cosine is "y". Difference is plotted as a percentage, as well as raw value. This computation is in each of the 2 sub-systems, with special processing to handle zero values.

In Simulink, choose Simulation->Run. The observed waveforms will look something like Figure 2:

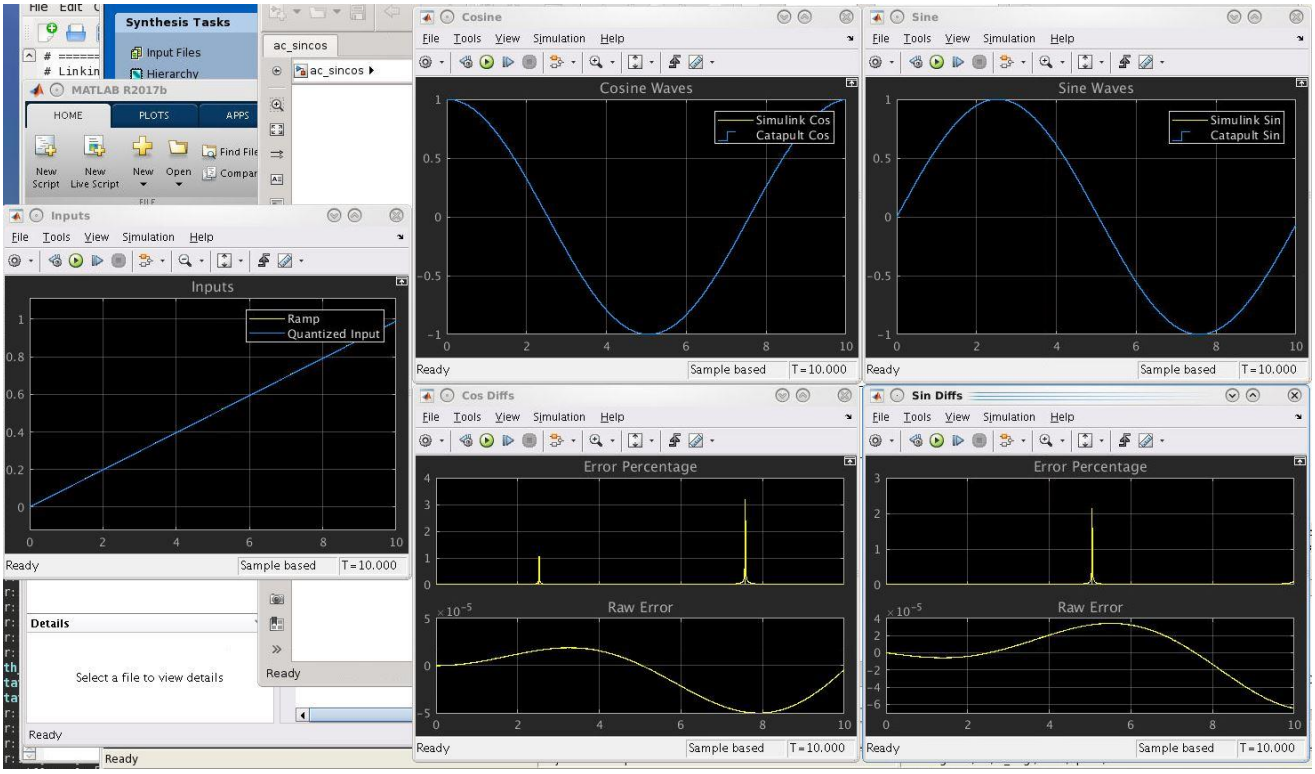

Figure 2: Waveforms After ac_sincos_lut Simulation

It is seen that "Error Percentage" for cosine looks to vary between 0% to 3% and for sine between 0% and 2%. Simulink can provide more accurate measurement and analysis. In each of the Diffs scopes, choose Tools->Measurements->Signal Statistics. The measured data is shown in Figure 3:

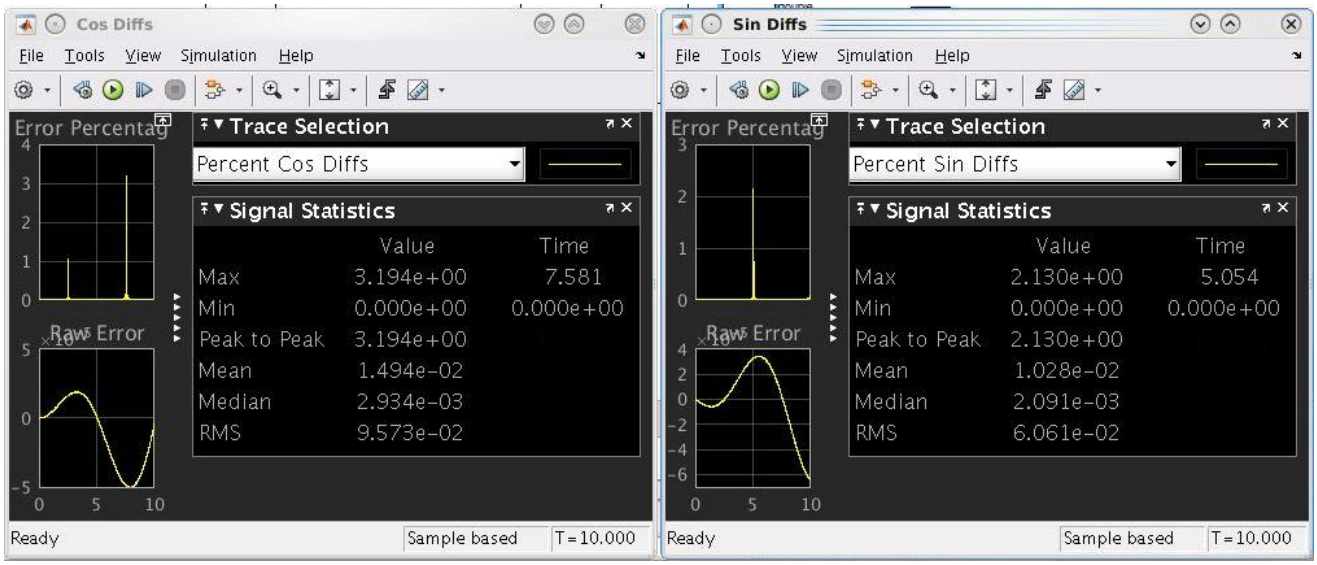

Figure 3: Difference measurements for ac_sincos_lut vs Simulink trigonometric model

While the maximum %difference is around 3% for cosine and 2% for sine, it is seen that the mean difference is around 0.015% for cosine and 0.010% for sine. Looking more closely at the cosine wave at its time of maximum % difference (7.581s) is shown in Figure 4 (note scaling on the Y-Axis):

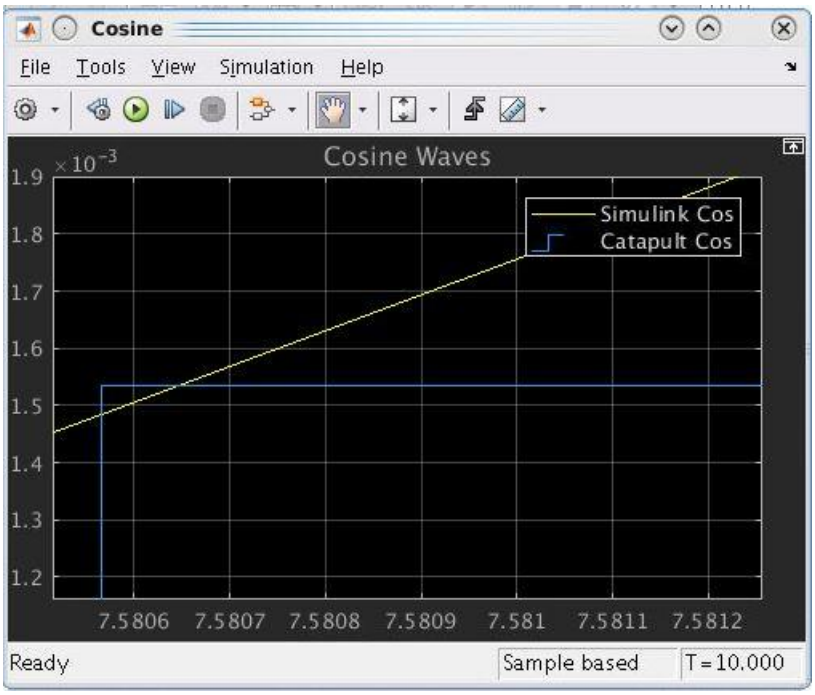

Figure 4: Maximum cosine differences at t=7.581s

In looking at the difference, we see that the actual difference in cosine value is about 0.0002. We conclude that the relatively high difference percentage difference is due to two factors:

1. Quantization differences between the Simulink trig function which uses a double as input while ac_sincos_lut use a 12-bit fixed point input. This is amplified due to the relatively rapid rate of change of cosine around angle 0.
2. This sample is very close to cosine value 0, and the difference function (described earlier) is known to be very sensitive to small values in the denominator.

Thus we conclude that the mean difference reported by Simulink is most useful in understanding differences.

Exit both Simulink and Catapult.

Modified Model and Accuracy Measures

Now assume that different parameters are desired for QoR reasons, and accuracy needs to be verified.

We will experiment with reduced precision by reducing both input and output data precision. As described earlier, these parameters are specified in the ac_sincos_lut_tb.h

header file, and can be set using compile-time C++ macros. We will reduce input width to 9 and output width to 18.

Now edit the ac_sincos_lut_tb_ac_fixed.tcl script and change these lines from this:

```
options set Input/CompilerFlags {-DTEST_SINCOS_AC_FIXED}
solution options set Input/CompilerFlags {-DTEST_SINCOS_AC_FIXED}
```

to this:

```
options set Input/CompilerFlags {-DTEST_SINCOS_AC_FIXED -
DWivalue=9 -DWoutvalue=18}
solution options set Input/CompilerFlags {-DTEST_SINCOS_AC_FIXED -
DWivalue=9 -DWoutvalue=18}
```

Invoke Catapult and rebuild the project:

```
  source ac_sincos_lut_tb_ac_fixed.tcl
```

Bring up Simulink on the rebuilt S-Function:

```
flow run /Matlab/launch_matlab ac_sincos.mdl
```

Since we changed input fixed point parameters, we need to re-configure the "Quantized Input" type-conversion model in Simulink. As shown in Figure 5, change fixed-point width to 9, with 9 fraction bits:

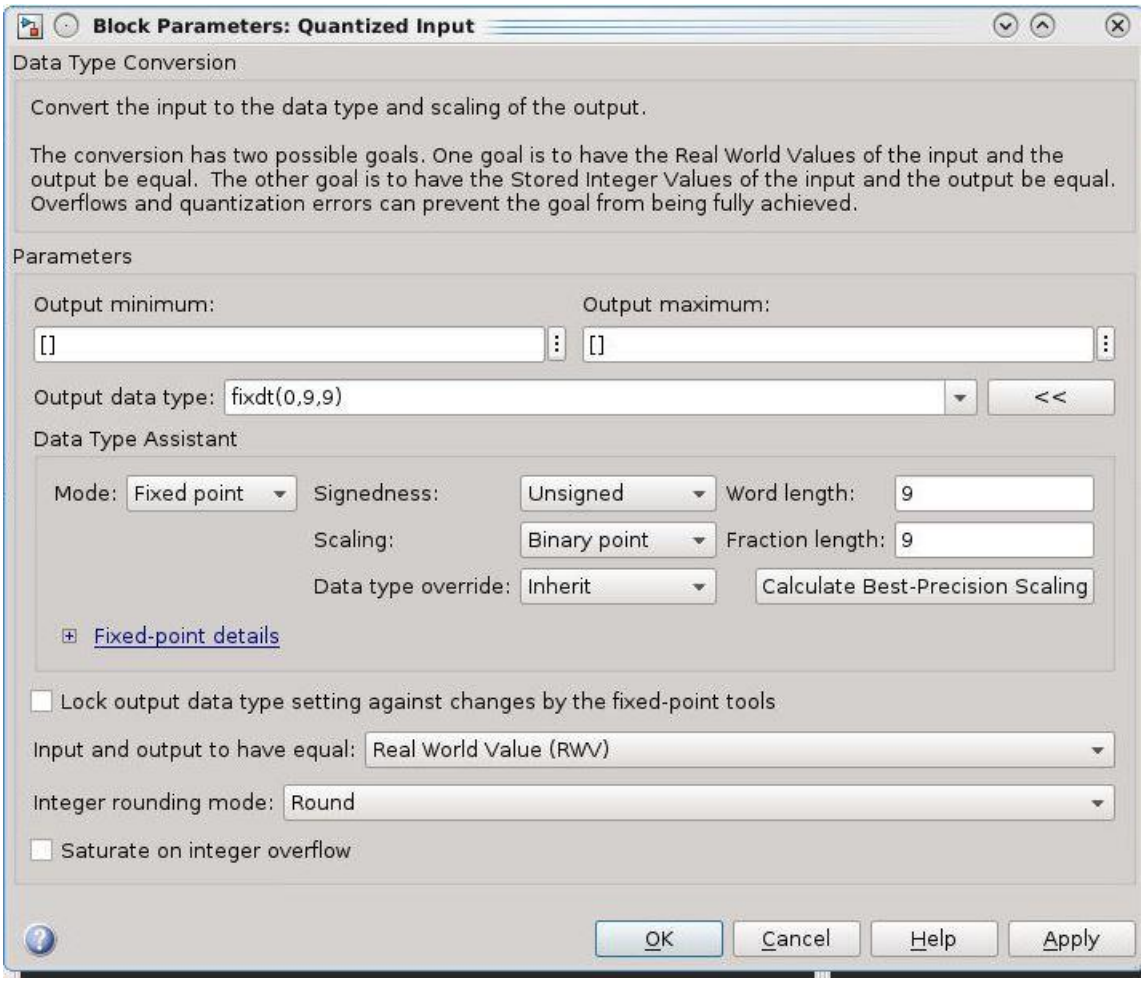

Figure 5: Edit "Quantized Input" type conversion model

Save the edits and re-run the simulation. Differences are shown in Figure 6:

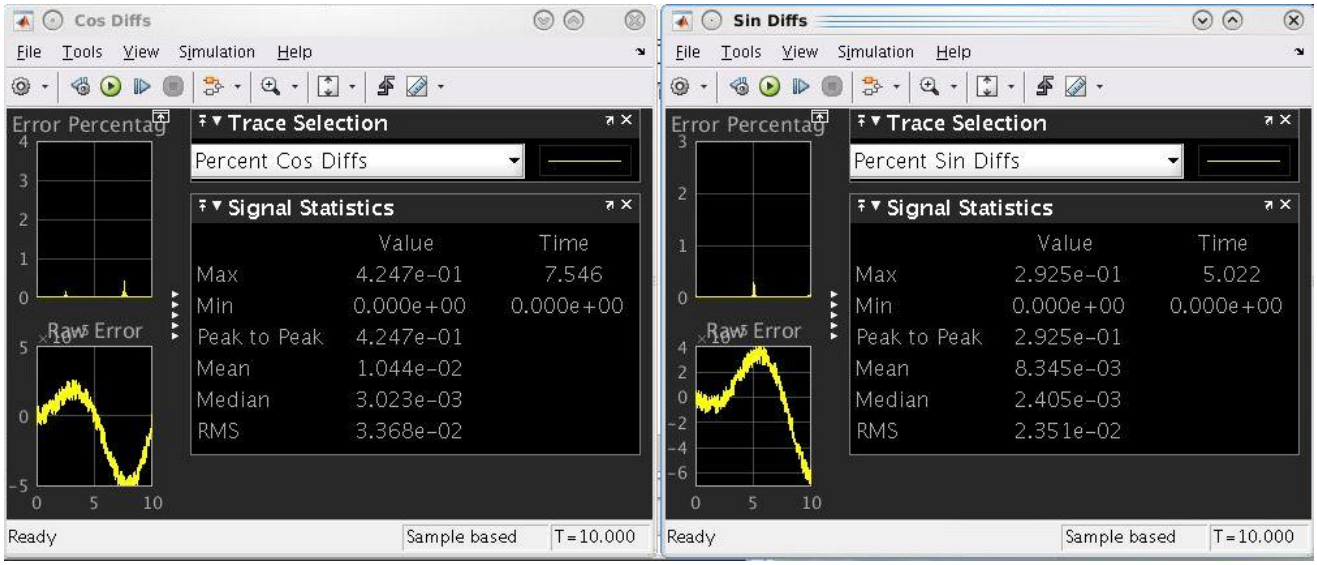

Figure 6: Difference Measurements for ac_sincos_lut after accuracy changes

We see that due to slightly different quantization of sample-time input values, maximum difference percentages are actually better, leading to also slight improvement in mean percentage differences.