Algorithmic C (AC) Math Library

Piece-wise Linear Reciprocal ac_reciprocal_pwl()

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_reciprocal_pwl function is designed to provide a quick approximation of the reciprocal of real and complex numbers using a Piecewise Linear (PWL) implementation with 7 points/6 segments, optimized to give a high-accuracy implementation. Many hardware division operations are calculated indirectly by first obtaining the value of the reciprocal of the denominator, and then multiplying that with the numerator. The calculation of the reciprocal is generally done using PWL approximation, which is faster and requires lesser area than actual division hardware.

Implementation and usage details of the ac_reciprocal_pwl function can be found at $MGC_HOME/shared/pdfdocs/ac_math_ref.pdf. This document presents the corresponding Catapult toolkit which provides example usages of the function.

The Piece-wise Reciprocal Toolkit

The ac_reciprocal_pwl toolkit is accessed in Catapult by picking Examples->Math->AC Reciprocal Piece-wise Linear. Four example usages are provided, each using a different input and output data type.

- Run fixed-point example uses ac_fixed data type.
- Run floating-point example used ac_float data type.
- Run complex fixed-point uses ac_complex<ac_fixed> data type.
- Run complex floating point uses ac_complex<ac_float> data type.

In each case, representative choices are made as to data width and precision. Each example provides a C++ testbench which configures and calls the ac_reciprocal_pwl function for a number of datapoints, comparing results with native C implementation.

This document is accessed by selecting "Piece-wise Linear Reciprocal Function" in the Documentation section.

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

- ReadMe.txt: Describes the toolkit briefly.
- ac_reciprocal_pwl.pdf: This document.
- ac_reciprocal_pwl_tb.h: Keader file for the testbench function which calls ac_reciprocal - "project". Data types are defined depending on the setting of certain compile-time definitions.
- ac_reciprocal_pwl_tb.cpp: C++ file which implements the project function which calls ac_reciprocal_pwl. Additionally implements a testbench to exercise ac_reciprocal_pwl, and check for correctness and accuracy.
- ac_reciprocal_pwl_tb_ac_fixed.tcl: Script to run Catapult to synthesize the ac_reciprocal_pwl function using ac_fixed data types.
- ac_reciprocal_pwl_tb_ac_float.tcl: Script to run Catapult to synthesize the ac_reciprocal_pwl function using ac_float data types.

- ac_reciprocal_pwl_tb_ac_complex_ac_fixed.tcl: Script to run Catapult to synthesize the ac_reciprocal_pwl function using ac_complex data types, using ac_fixed for both real and imaginary components.
- ac_reciprocal_pwl_tb_ac_complex_ac_float.tcl: Script to run Catapult to synthesize the ac_reciprocal_pwl function using ac_complex data types, using ac_float for both real and imaginary components.
- ac_reciprocal.mdl: A Simulink model pre-configured to exercise the ac_reciprocal_pwl function in Simulink and compare accuracy with Simulink built-in functions.

In the ac_reciprocal_pwl_tb.h header file, input and output data types are defined for each of the 4 example usages, and enabled as shown here:

```
//These macros are defined from the appropriate version of the
.tcl file.
#if defined(TEST_RECIPROCAL_AC_FIXED)
typedef ac_fixed<W_INT_FIXED, I_INT_FIXED, S_BOOL_FIXED, AC_RND,
AC_SAT> input_type;
typedef ac_fixed<64, 32, S_BOOL_FIXED, AC_RND, AC_SAT>
output_type;
#elif defined(TEST_RECIPROCAL_AC_COMPLEX_AC_FIXED)
typedef ac_complex<ac_fixed<W_INT_FIXED, I_INT_FIXED,
S_BOOL_FIXED, AC_RND, AC_SAT> > input_type;
typedef ac_complex<ac_fixed<64, 32, true, AC_RND, AC_SAT> >
output_type;
#elif defined(TEST_RECIPROCAL_AC_FLOAT)
typedef ac_float<W_INT_FLOAT, I_INT_FLOAT, E_INT_FLOAT, AC_RND>
input_type;
typedef ac_float<64, 32, 10, AC_RND> output_type;
#elif defined(TEST_RECIPROCAL_AC_COMPLEX_AC_FLOAT)
typedef ac_complex<ac_float<W_INT_FLOAT, I_INT_FLOAT,
E_INT_FLOAT, AC_RND> > input_type;
typedef ac_complex<ac_float<64, 32, 10, AC_RND> > output_type;
#endif
```

The four tests are defined by specifying exactly one of these variables to be used to compile the test:

- TEST_RECIPROCAL_AC_FIXED: Run the test with ac_fixed data types for input and output.
- TEST_RECIPROCAL_AC_COMPLEX_AC_FIXED: Run the test with ac_complex input and output. Real and imaginary components are ac_fixed.
- TEST_RECIPROCAL_AC_FLOAT: Run the test with ac_float data types for input and output.
- TEST_RECIPROCAL_AC_COMPLEX_AC_FLOAT: Run the test with ac_complex input and output. Real and imaginary components are ac_fixed.

In addition, this toolkit allows the user to modify fixed and floating point widths and precision for the input data by setting these variables before compile:

- W_INT_FIXED: Fixed point total width. Default is 32.
- I_INT_FIXED: Fixed point Integer width. Default is16.
- S_BOOL_FIXED: Fixed point signedness. Default is true.
- W_INT_FLOAT: Floating point mantissa width. Default is 19.
- I_INT_FLOAT: Floating point mantissa integer size. Default is 9.
- E_INT_FLOAT: Floating point exponent width. Default is10.

For more information on ac datatypes see:
`$MGC_HOME/shared/pdfdocs/ac_datatypes_ref.h`

An example for how to set these definitions for a catapult run would be:

```
options set Input/CompilerFlags {-DTEST_RECIPROCAL_AC_FIXED -
DW_INT_FIXED=64 -DI_INT_FIXED=32}
solution options set Input/CompilerFlags {-
DTEST_RECIPROCAL_AC_FIXED -DW_INT_FIXED=64 -DI_INT_FIXED=32}
```

Running The Fixed-Point Reciprocal Example

After exporting files as discussed above, type "source ac_reciprocal_pwl_tb_ac_fixed.tcl". Alternatively, clicking on "Launch Project" will have the same effect. The function "project" in ac_reciprocal_pwl_tb.cpp is synthesized using macro `TEST_RECIPROCAL_AC_FIXED` defined. Catapult transcript shows the following:

- Synthesis output: The top-level function "project" calls the ac_reciprocal_pwl function, with ac_fixed input and output data widths as shown 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_reciprocal_pwl_tb.cpp file a testbench for ac_reciprocal_pwl is provided. This testbench "sweeps" input from -15 to 15 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 32 bits width using 16 fraction bits; thus a Quantum is 1/(2^16) or 1.52588e-05. This is about 1.9M sample points by default as the test is set up.

At each datapoint, a comparison is made to native C++ implementation (i.e. for input x, expected output is 1/x). A difference percentage is calculated, and if the difference is over a one percent, an error is flagged. These parameters are specified in the ac_reciprocal_pwl_tb_ac_fixed.tcl file by this line:

```
flow package option set /SCVerify/INVOKE_ARGS {-15 15 - 1.00}
```

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.

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

```
# ============================================
# Simulating design
# cd ../..;
./Catapult/project.v1/scverify/orig_cxx_osci/scverify_top -15 15 -
1.00
# ============ reciprocal_pwl test ==================
# lower_limit    = -15
# upper_limit    = 15
# step           = 1.52588e-05
# allowed_error  = 1
#
# Testbench finished
# max_error =0.322046  with sample =-8.66145  expected=-0.115454
actual=-0.115827
# ===================================================
```

SCVerify output shows simulation parameters followed by test results. We see that maximum difference was 0.32% using input value -8.66.

Difference percentage is calculated for each output using the following equation:

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

A plot_values.csv file is optionally produced which is suitable to be read into Excel for further analysis if desired. This is not default but is enabled by specifying this compile-time flag:

```
-DPLOT_RECIPROCAL_FILE_WRITE
```

Simulink Simulation and Accuracy Visualization

As discussed above, default accuracy for ac_reciprocal_pwl is very nice. However, if different parameters are used, accuracy is likely to be affected. In addition to the SCVerify setup discussed above, a Simulink setup is provided to allow the user to modify ac_reciprocal_pwl parameters, then measure accuracy. This section shows how to measure accuracy for the default toolkit setup, as well as how to modify parameters and measure the impact on accuracy in Simulink. 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

After running the ac_reciprocal_pwl_tb_ac_fixed.tcl as shown above, a Simulink S-Function is left in the ./Simulink directory(sfun_project.mexa64). More on the Catapult integration with Matlab and Simulink can be seen at Examples->Methodology->Matlab Flow toolkit documentation.

The testbench in this example instantiates the ac_reciprocal_pwl function using input signed fixed point with 32 bits width, and 16 fraction bits. Output width is signed fixed point with 64 bits width, and 32 fraction bits.

In Catapult, issue this command:

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

Alternatively, cd into the Simulink directory. Then invoke matlab, and type "open('ac_reciprocal.mdl')" command. A pre-configured Simulink model is opened which simulates the ac_reciprocal_pwl function for 10 seconds of time, using a simple ramp function with step size 0.001 seconds. The ramp is configured to sweep from -15 to 15 over the 10 seconds. The Simulink schematic looks something like Figure 1:

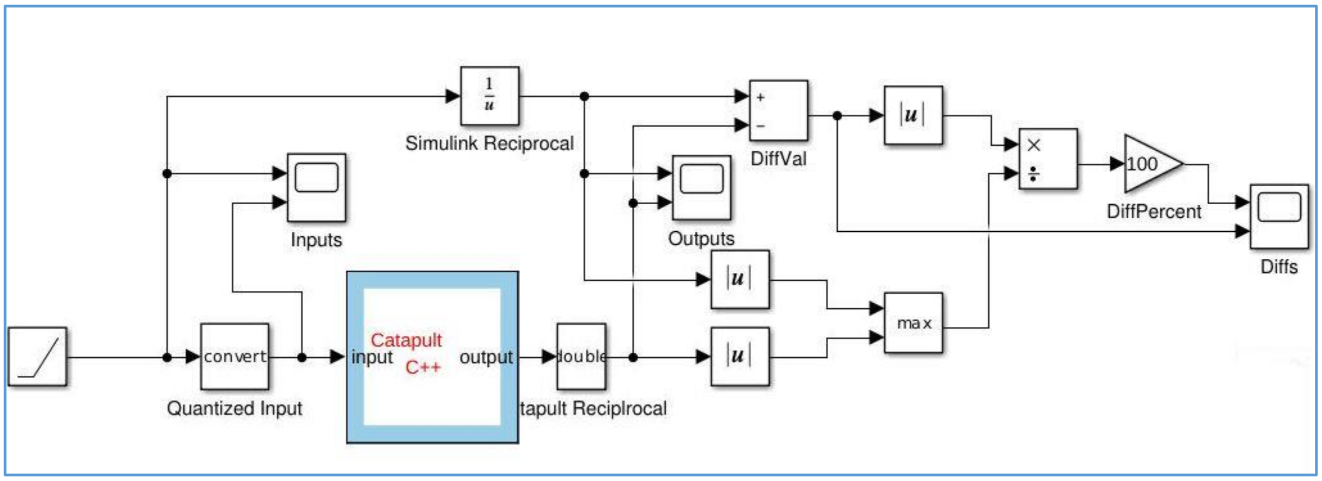


Figure 1: Simulink Schematic for ac_reciprocal_pwl Measurements

It is seen that the Simulink ramp source drives both the catapult S-Function for ac_reciprocal_pwl and the Simulink math function configured as reciprocal model. Scopes are used to view ramp input ('Inputs'), model outputs ('Outputs'), and differences between the models ('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 is taken as "x", and the approximated ac_reciprocal_pwl function is "y". Difference is plotted as a percentage, as well as raw value.

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

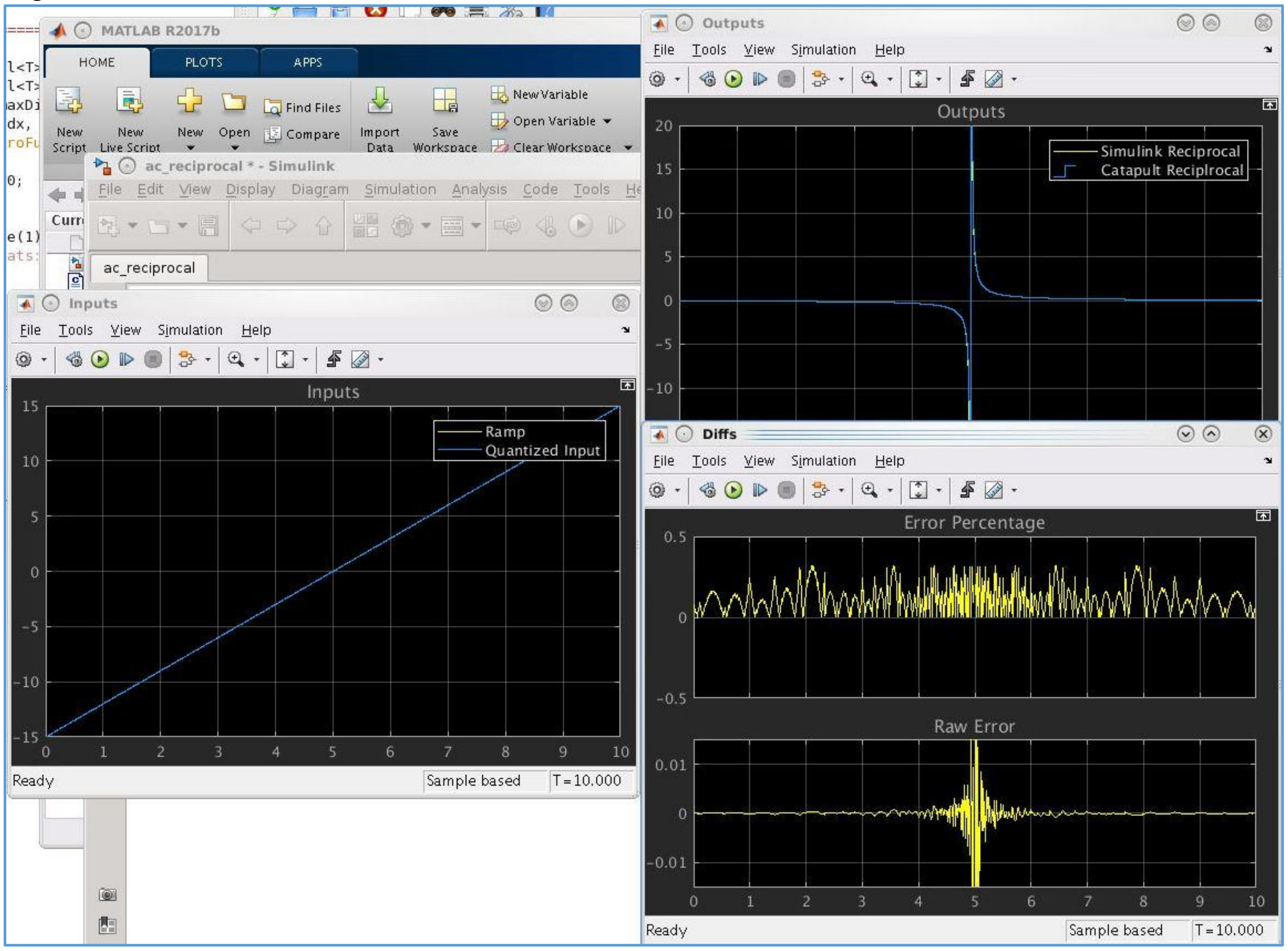
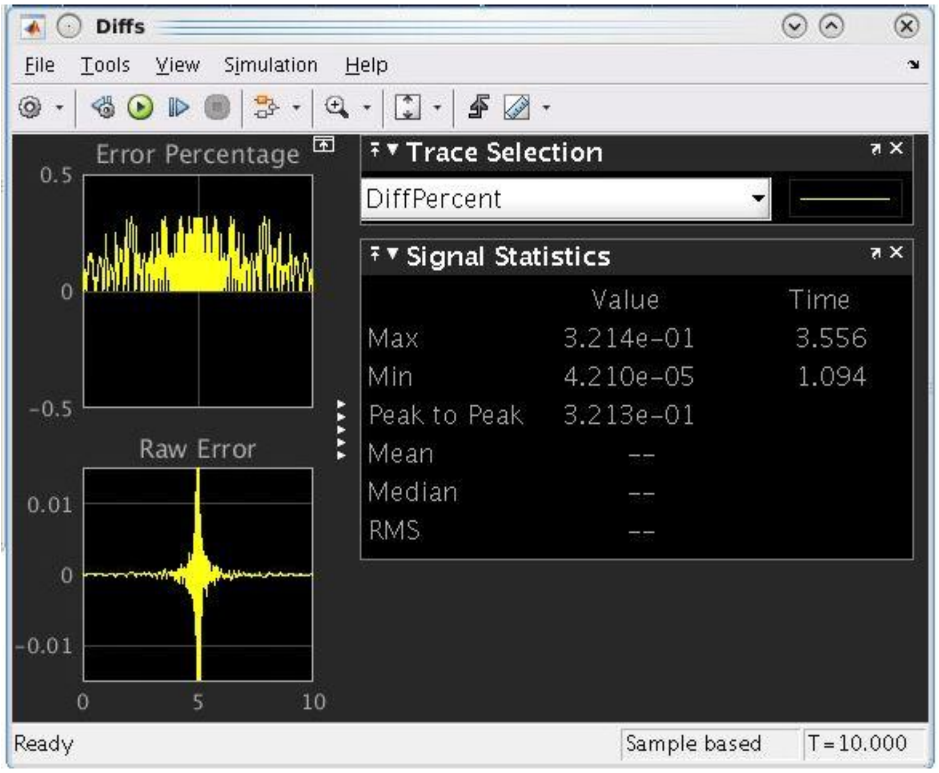


Figure 2: Waveforms After ac_reciprocal_pwl Simulation

It is seen that "Error Percentage" looks to vary between 0.0% to -0.3%. Simulink can provide more accurate measurements. In the Diffs scope, choose Tools->Measurements->Signal Statistics. The measured data is shown in Figure 3:



	Value	Time
Max	3.214e-01	3.556
Min	4.210e-05	1.094
Peak to Peak	3.213e-01	
Mean	--	
Median	--	
RMS	--	

Figure 3: Difference Measurements for ac_reciprocal_pwl vs Simulink reciprocal model

The default toolkit usage has maximum difference 0.32% which compares to the SCVerify testbench value 0.32%.

A Note on Differences Between SCVerify and Simulink Sampling

Note that the SCVerify testbench sweeps input values, while Simulink sweeps through time. In this case, the SCVerify setup has approximately 1.9M sample points which include all values between +/-15 which can be represented by a signed fixed-point value with 32 bits width using 16 fraction bits. In the Simulink setup 10000 points are simulated, some of which may not be exactly on a Quantum point; thus rounding is involved, and some Quantums are not simulated. The Simulink function sees the ramp output which is a double, while the Catapult model sees the quantized (perhaps rounded) fixed point value. This rounding and fewer sampling points can lead to slightly different error percentage in Simulink than in SCVerify.

In addition, Simulink and SCVerify can have slightly different ways to deal with very large (infinite) and very small (zero) numbers. In the SCVerify testbench when the input value is exactly 0, the sample point is basically ignored, This is because of the way infinite is handled when the C++ reciprocal value (1/0 infinite) is compared to Catapult output which saturates the output fixed point to 2.147e9.

As a way to see these effects - not illustrated here - the following experiments can be performed in Simulink.

1. Effects of non-quantized (important) sample points: In Simulink, change the "Fixed Step Size" in Simulation Parameters to .0001. Issue the "clear mex" matlab

command to reset the model, then re-simulate. Now the error percentage should be around 1.5% at time 5.0 instead of 0.3% at time 3.6. This is basically because a Simulink sample point was seen in the range between 0 and one Quantum. The Simulink function sees this value, but the Quantization conversion rounds this value before the PWL function can process the value. Thus a relatively larger error percentage is seen.

2. Effects of simulink quantization and saturation: Now change the schematic to drive the Simulink reciprocal function from the Quantized input rather than ramp output which is a double. Now the Simulink function will see only values seen by the Catapult model exactly. Re-simulate (after "clear mex") and now the error percentage is 100%. This is because the Simulink model saturates its output to a value based on its (now) fixed point input definition. Recall that the input fixed point is 32 bits wide, while output is 64 bits wide. Thus saturated values are much different leading to a large error percentage. This can be seen in the Outputs scope by using Tools->Measurements->Signal Statistics.
3. Effects of processing infinite double values: Now add a conversion of the Quantized input back to a double, before driving the Simulink reciprocal. Re-simulate. Now the error percentage is 0.32% again although a slightly different time 21. This is because the Simulink function "saturates" the output as a double which is infinite. This can be seen in the Outputs scope. This infinite value is basically ignored for that sample point as it propagates to downstream math processing. This special handling of infinite is also happens in the SCVerify testbench run and error computation at that data point.

All this points out that much care should be taken when simulating and comparing accuracy around points of high rate of change generally, and discontinuous points specifically.

Exit out of Simulink and matlab, without saving edits, before continuing.

Modified Model and Accuracy Measures

Now assume that some specialized usage chooses different parameters based on specific needs, and accuracy impacts need to be verified. Since we already exported files (above), we can simply edit the parameters in source files, then rebuild the Simulink S-Function.

We will experiment with reduced accuracy by reducing output data width which is specified in the ac_reciprocal_pwl_tb.h header file. We will reduce output width from 64 bits to 24, using 16 fraction bits instead of 32 bits. Note that this choice only has 8 integer bits (signed) which means output values less than 1/127 (.00787) will have increasing difference. But we want to see how bad it is.

In ac_reciprocal_pwl_tb.h, change this line:

```
typedef ac_fixed<64, 32, S_BOOL_FIXED, AC_RND, AC_SAT>
output_type;
```

To this:

```
typedef ac_fixed<24, 8, S_BOOL_FIXED, AC_RND, AC_SAT> output_type;
```

Then rebuild the project by:

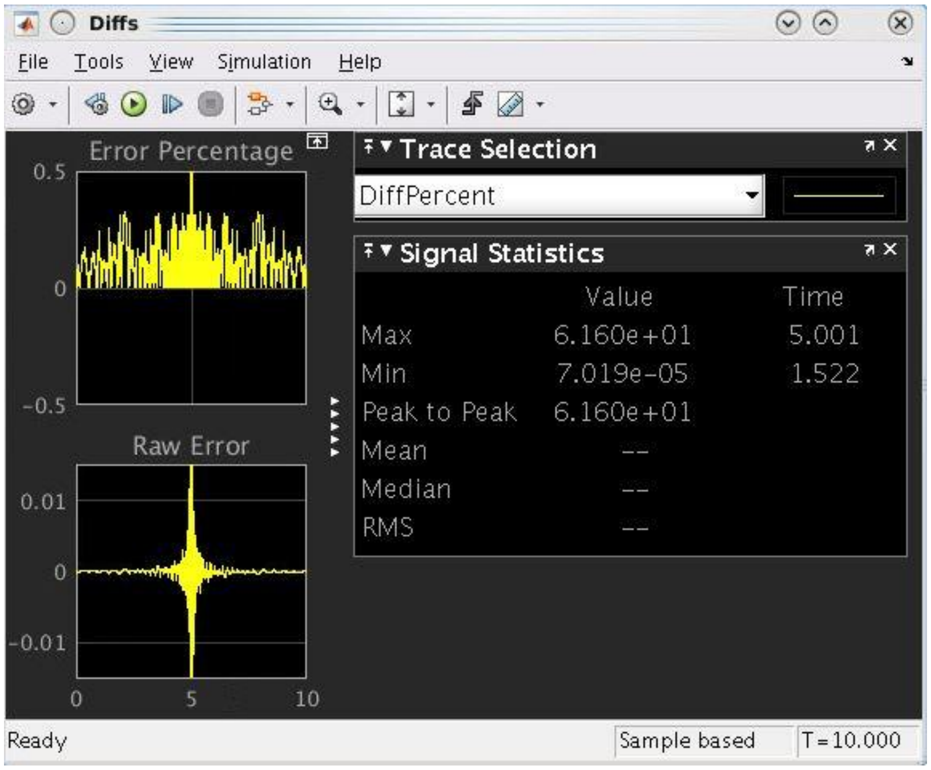
```
 source ac_reciprocal_pwl_tb_ac_fixed.tcl
```

Notice that the SCVerify run reports that difference is now more than 1%.

```
# max_error =99.8047  with sample =1.52588e-05  expected=65536
actual=128
```

This 99% max error% is with the small input value 1.52588e-05 which we saw earlier is one Quantum above 0.0. As discussed previously, this difference is as expected due to the limited number of integer bits in the new fixed point output representation.

After running Simulink again the Diffs scope is shown in Figure 4:



```
              Value        Time
Max           6.160e+01    5.001
Min           7.019e-05    1.522
Peak to Peak  6.160e+01
Mean          --
Median        --
RMS           --
```

Figure 4: Difference Measurements for ac_reciprocal_pwl after accuracy changes

In Simulink we now "see" that error is not too much and generally looks about the same except around input value 0. As discussed earlier, it we decrease fixed step size from .001 to .0001 we might get more alignment with SCVerify. In fact if that experiment is performed we now see 96% error at time 5.00. Depending on the application, for example if very small input values are not allowed, this might be a good accuracy tradeoff with hardware cost.