Arithmetic Libraries User's Guide

Stefan Zwicky, Jeffrey Stähli, Christian Senning, Benjamin Weber

December 5, 2016

1 Overview

The arithmetic libraries are intended to make the VHDL-designer's life easier. Basic arithmetic operations both on real and complex numbers are provided as VHDL-functions together with the corresponding bit-true MATLAB functions.

The files under consideration are the following:

- `vhdl/VHDLTools/VHDLTools.vhd`
- `vhdl/RealARITH/RealARITH.vhd`
- `vhdl/ComplexARITH/ComplexARITH.vhd`
- `matlab/VHDLTools/*.m`
- `matlab/RealARITH/model/Real*.m`
- `matlab/ComplexARITH/model/Complex*.m`

The package `VHDLTools.vhd` contains auxiliary functions such as minimum(.), max(.) which are directly available in MATLAB. The functions of VHDLTools are used in RealARITH. ComplexARITH is based on the RealARITH library.

The functions in Table 1 are provided both in MATLAB and VHDL. There are some additional functions in the MATLAB directories, but they do not have a VHDL counterpart. Table 2 shows the functions that are only provided in the VHDL library.

2 Usage

2.1 Allowed Data Types

The allowed data types in VHDL are unsigned and signed for the RealARITH functions and signed for the ComplexARITH functions. An exception is the shift argument in the arithmetic shift operations, which must be of type unsigned for ASR and ASL and of type signed or a constant integer for AS (shift left for positive values, shift right for negative values). In the RealARITH library, not all possible combinations of signed and unsigned inputs/outputs are supported (please refer to the VHDL package `RealARITH.vhd`).

Description	RealARITH		ComplexARITH	
get word width (incl. sign bit(s))	RealWIDTH		ComplexWIDTH	
merge imaginary and real part	-		ComplexMERGE	
changing the fixed-point representation	RealRESIZE	●	ComplexRESIZE	●
absolute value	RealABS	●	-	
negation	RealNEG	●	ComplexNEG	●
complex conjugate	-		ComplexCONJ	●
addition	RealADD	●	ComplexADD	●
subtraction	RealSUB	●	ComplexSUB	●
arithmetic shift left	RealASL	●	ComplexASL	●
arithmetic shift right	RealASR	●	ComplexASR	●
arithmetic shift (left or right)	RealAS	●	ComplexAS	●
multiplication	RealMULT	●	ComplexMULT	●
division	RealDIV	●	-	

Table 1: Arithmetic functions provided in VHDL as well as in MATLAB. The dots indicate the support of data logging in MATLAB

get real part of signal	GetREAL
get imaginary part of signal	GetIMAG
multiply a complex number with a real one	ComplexRealMULT
division of a complex number by a real one	ComplexRealDIV

Table 2: Functions provided only in VHDL.

2.2 Fixed-point Representation

The fixed-point word length is specified through a record constant of type FixP in VHDL and a cell array in MATLAB respectively. It contains the following three items:

1. WIDTH_INT is the number of bits for the integer part (excluding the sign bit)

2. WIDTH_FRAC is the number of bits for the fractional part

3. SIG_TYPE is either 'u' for unsigned signals or 's' for signed signals

The SIG_TYPE item is of type character in MATLAB and of an enumerated type (u,s) in VHDL. In MATLAB, the three cell array items must be defined always in the indicated order.

Even though it contradicts the definition, both WIDTH_INT and WIDTH_FRAC can be negative. A negative value for WIDTH_INT indicates that the comma position is more to the "left" than the MSB of the word, basically reducing WIDTH_FRAC. However, the condition

$$\text{WIDTH\_INT} + \text{WIDTH\_FRAC} \geq 1$$

must always hold.

In VHDL, complex numbers are represented by real and imaginary part, both having the same fixed-point representation. Real and imaginary part are concatenated to a single signed value.

```
Complex_D(N downto 0) <= Imag_D & Real_D;
```

`Comple_D` must always be of type signed.

In MATLAB, complex numbers are represented by "normal" complex variables of type double. The fixed-point configuration is contained in the value of the variable, by restricting it to the maximal/minimal value according to WIDTH_INT and by reducing its accuracy according to WIDTH_FRAC.

When changing the fixed-point representation of a value (either directly with the resize function or through any other function that calls the resize function), one must specify the behavior of the circuit if the number of bits is not sufficient to represent that value. There are two possibilities for both the most significant and the least significant part of the word.

- Most significant bits:

  Wrp: Wraps the value if the value is out of range. (WARNING: The old argument **Clp** should no longer be used)

  Sat: Saturate to the largest/smallest representable value (or allowed value, refer to section 2.4).

- Least significant bits:

  Trc: Truncate the least significant bits without considering their content.

  Rnd: Round according to the content of the bits that are cut off.

Combining these possibilities results in four quantization possibilities:

$$\text{QuantType} := (\text{WrpTrc}, \text{WrpRnd}, \text{SatTrc}, \text{SatRnd})$$

2.3 Function Call

In the following section, some examples of function calls are listed

VHDL The fixed-point representation of signal A_D is denoted as A_FixP or directly specified through e.g. (2,5,s). The first value in the brackets corresponds to WIDTH_INT, the second to WIDTH_FRAC and the third to SIG_TYPE. In VHDL, the FixP records of both input and output must be provided as arguments.

- `Out_D <= RealRESIZE(InA_D,InA_FixP,Out_FixP,WrpTrc);`
- `Out_D <= ComplexRESIZE(InA_D,(4,3,u),(3,2,u),WrpTrc);`

Functions with two input signals require the FixP constants for both inputs and the output:

- `Out_D <= RealADD(InA_D,InB_D,InA_FixP,InB_FixP,Out_FixP,WrpRnd);`
- `Out_D <= ComplexMERGE(Real_D,Imag_D,Real_FixP,Imag_FixP,Out_FixP,SatTrc);`

Exceptions are the arithmetic shift operations, where it is assumed that the shift argument is integer (i.e. WIDTH_INT is always 0).

- `Out_D <= RealASL(InA_D,Shift_D,InA_FixP,Out_FixP,WrpRnd);`
- `Out_D <= ComplexAS(InA_D,CONST,InA_FixP,Out_FixP,SatRnd);`

An additional argument is required for the complex multiplication. One can choose between a fast architecture with four real-valued multipliers and a slow architecture with only three real-valued multipliers, but with an additional adder stage.

- `Out_D <= ComplexMULT(InA_D,InB_D,InA_FixP,InB_FixP,Out_FixP,WrpRnd,fast);`
- `Out_D <= ComplexMULT(InA_D,InB_D,InA_FixP,InB_FixP,Out_FixP,WrpRnd,slow);`

Another extra argument is required for RealADDSUB. It selects between addition ('0') and subtraction ('1').

- `Out_D <= ComplexADDSUB(InA_D,InB_D,'1',(2,3,s),InB_FixP,Out_FixP,WrpTrc);`
- `Out_D <= ComplexADDSUB(InA_D,InB_D,'0',InA_FixP,InB_FixP,(5,8,s),WrpRnd);`

The functions GetREAL/GetIMAG and RealWIDTH/ComplexWIDTH are special cases. They require only one input argument:

- `Real_D <= GetREAL(Complex_D); Imag_D <= GetIMAG(Complex_D);`
- `WW_REAL <= RealWIDTH(InA_FixP); WW_COMPLEX <= ComplexWIDTH(InB_FixP);`

Additionally, there is a simplified version of ComplexMERGE. It assumes that `Real_D`, `Imag_D` and `Out_D` all have the same FixP representation and just concatenates imaginary and real part.

- `Out_D <= ComplexMERGE(Real_D,Imag_D);`

MATLAB Since the fixed-point representation of a Matlab variable is inherently given by the accuracy of its content, the corresponding Matlab functions require only the FixP cell array of the return value.

- `Out = RealRESIZE(InA,{-2,10,'s'},'SatTrc');`
- `Out = ComplexMULT(InA,InB,Out_FixP,'WrpRnd');`

The only exception to this rule is the `RealDIV` function. It can be called in two ways:

- `Out = RealDIV(inA,inB,FixP,QType);`
- `Out = RealDIV(inA,inB,inA_FixP,inB_FixP,out_FixP,QType);`

Only the latter call fully reflects the corresponding VHDL implementation.

2.4 The Weird Number

The most negative number in a signed fixed-point configuration (i.e. -1 for FixP = (0,N)) is sometimes called the weird number because there is no positive counterpart. In order to have a symmetric dynamic range, the too small values are saturated to the next larger number which has a positive counterpart (i.e. 11111 is saturated to 1001 instead of 1000). However, if the values are not saturated, it can happen (intentionally or unintentionally) that the return value is exactly the weird number.

2.5 Data Logging in MATLAB

The `RealRESIZE` function in Matlab is able to keep track of specific data items. At the moment, it calculates the complementary/inverse cumulative distribution function (ccdf/icdf) of its input values.

Compilation: The `RealRESIZE` function supporting data logging in Matlab is implemented as a Mex function written in C. To compile this code for your machine, run `make` in the

- `asc_matlab/Common/RealARITH/model/`

folder.

The `ArithLibStatistics` variable: To include the data logging ability in your simulations, you must provide a global variable called `ArithLibStatistics` of type `struct` at the beginning of our code. Every function supporting data logging (see Table 1) will dynamically add a field to the `ArithLibStatistics` struct. The `ArithLibStatistics` variable can be interpreted as a one-dimensional array of structs. It is arranged as shown in Figure 1.

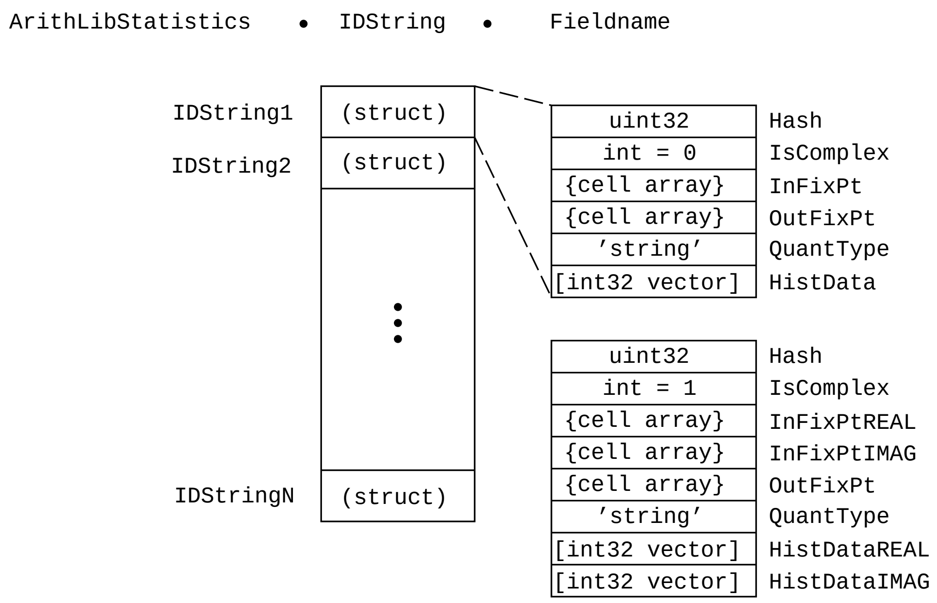

Figure 1: The `ArithLibStatistics` variable (1-D array of structs). Note that the sub-structs are differently organized for real-valued and complex-valued arithmetic operations.

The Identifier String: In order to distinguish individual arithmetic operations, you need to supply the functions with a unique identifier string. If you leave out this string in the function call, `RealRESIZE` operates as usual and will not log data from this particular function. When a function is called multiple times with the same identifier string, the data is superimposed on the existing data. Complex-valued operations are internally split up into real and imaginary parts and the identifier string is labeled accordingly by appending `(REAL)` or `(IMAG)` to the given string. The identifier string is always the last argument in the function call.

Using the `ArithLibStatistics` variable: The data can be accessed using this scheme:

- `ArithLibStatistics.IDString` for a struct.
- `ArithLibStatistics.IDString.FieldName` for the a specific field in the struct.

Refer to Figure 1 for valid fieldnames.

A function `generateHistPlots(m,n)` which creates histogram-like plots of the complementary cumulative distribution functions is provided. The arguments distribute the subplots in a `m`×`n` arrangement on the figure window. This display function also creates new figures when the total number of subplots exceeds the number `m`·`n`.

A simple code example to illustrate the mentioned points is given below.

```
% Global Log Variable
global ArithLibStatistics
ArithLibStatistics = struct; %initialize empty 1x1 struct

% Do some calc
Add01Out = RealADD(InA,InB,{4,2,'s'},'SatRnd','Add01');
...

% Graphical display
generateHistPlots(3,2); % (m,n) one figure holds m x n subplots
```

The complementary cumulative distribution function plot: Figure 2 shows an example of such a plot. It basically shows the "total usage" of the individual bits. One can see, that the addition of two integer bits and one fractional bit to the existing configuration at the output would give optimum results. When overflows occur, a number designating the percentage of overflows with respect to the total number of values is indicated for both positive and negative input values.

3 Outlook

Features that might be added in the future:

- Defining a default value (e.g. WrpTrc) for QuantType. Replicate all functions without QuantType input.
- Introducing a possibility to omit some FixP arguments (e.g. assume InA_FixP = InB_FixP = Oup_FixP if only one FixP argument is provided)
- Introducing a `ComplexDIV`.
- Introducing a `ComplexANGLE` by means of a cordic.
- Introducing a `ComplexABS` by means of a cordic.
- ...

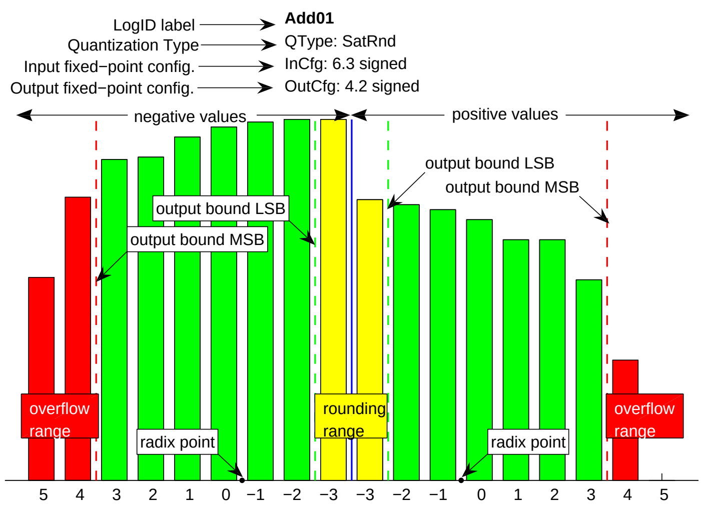

Figure 2: Example of a plot. Positive numbers on the x-axis denote integer bits, negative ones label fractional bits.