

Matrix math packages user's guide

By David Bishop (dbishop@vhdl.org)

A Matrix is a way of looking at a group of associated numbers. Most standard math functions can work on Matrices. These packages are targeted to people who are familiar with “matlab” matrix manipulation. It is designed to perform basic matrix functions for the standard VHDL types.

The VHDL matrix math packages can be downloaded at:

http://www.vhdl.org/fphdl/real_matrix_pkg.zip

In the ZIP archive you will find the following files:

- “real_matrix_pkg.vhdl” and “real_matrix_pkg_body.vhdl” (for types “REAL” and “INTEGER”)
- “complex_matrix_pkg.vhdl”, and “complex_matrix_pkg_body.vhdl” (for types COMPLEX and COMPLEX_POLAR)
- “fixed_matrix_pkg.vhdl” and “fixed_matrix_pkg_body.vhdl” for types UNSIGNED, SIGNED, UFIXED and SFIXED.
- “test_real_matrix.vhdl”, “test_complex_matrix.vhdl”, and “test_fixed_matrix.vhdl” which test the functionality of these packages.
- “compile.mti” – A compile script.
- “compile_93.mti” – Compile script for VHDL-93 (to 2002 rules for shared variables)

These packages have been designed for use in VHDL-2008. However, a compatibility version of the packages is provided that works for VHDL-1993. The VHDL-1993 versions of the packages have an “_93” at the end of their file names. These packages were designed to be compiled into the IEEE_PROPOSED library.

Dependencies:

- “real_matrix_pkg” is dependant only on IEEE.MATH_REAL.
- “complex_matrix_pkg” is dependent on “real_matrix_pkg”, IEEE.MATH_REAL and IEEE.MATH_COMPLEX.
- “fixed_matrix_pkg” is dependent on the VHDL-2008 “numeric_std” and “fixed_pkg”, and “real_matrix_pkg”. The VHDL-1993 version of “fixed_matrix_pkg” is dependent on the “IEEE_PROPOSED” library which can be downloaded at <http://www.vhdl.org/fphdl/vhdl2008c.zip>. It is also dependant on the “real_matrix_pkg”

Overview

The “real_matrix_pkg” package defines the types “real_matrix” and “integer_matrix”. The VHDL-1993 version defines the types “real_vector” and “integer_vector” (which are normally predefined in VHDL-2008). These types are defined as follows:

```

    type real_matrix is array (NATURAL range <>, NATURAL
range <>) of REAL;
    type integer_matrix is array (NATURAL range <>, NATURAL
range <>) of INTEGER;

```

Usage model:

```

use ieee.real_matrix_pkg.all;    -- ieee_proposed    for
compatibility version
....
    signal a: real_matrix (0 to 3, 0 to 1);
    signal b: real_matrix (0 to 1, 0 to 2);
    signal c: real_matrix (0 to 3, 0 to 2);
begin
....
c <= a * b; -- Matrix multiply

```

As a concession to C, all matrices are assumed to be in column, row format, and starting at index “0”. Thus for the matrix:

```

Z := ((1.0, 2.0, 3.0),
      (4.0, 5.0, 6.0),
      (7.0, 8.0, 9.0));
Z (0,2) = 7.0

```

Note that this notation is different from the Matlab notation. In that language the first column is column “1”, not “0”.

All matrices are assumed to be of the “X to Y” range. If a “downto” range is used when defining a matrix or a vector then that matrix will be reversed when it is operated on.

A vector is assumed to be a matrix with one row. All functions which read in vectors treat them in this manner. If you want a matrix with one column, then you need to define them as follows:

```

signal z : real_matrix (0 to 3, 0 to 0);

```

One of the wonderful things about Matlab is the ability to pull a slice out of a vector or a matrix. In Matlab you can say “(i:j)” or “(:,j)” or “(r,i:j)”. Due to the strong typing in VHDL we do not have that ability, so we have several functions which are designed to do this:

a) Taking a matrix apart - For this you need the “SubMatrix” command

```

Result := submatrix (arg, x, y, rows, columns)

```

where : arg – Input matrix

 x – “x” or row location to start from

 y – “y” or column location to start from

 rows – number of rows in result

columns – number of columns in result

Examples:

Take the matrix:

```
variable A : real_matrix (0 to 3, 0 to 3);
A := ((1.0, 2.0, 3.0, 4.0),
      (5.0, 6.0, 7.0, 8.0),
      (9.0, 10.0, 11.0, 12.0),
      (13.0, 14.0, 15.0, 16.0));
```

```
B := submatrix (A, 1,1,2,2);
```

Would return a 2x2 matrix (real_matrix (0 to 1, 0 to 1)) starting at location (1,1) in the input matrix A, or:

```
B := ((6.0, 7.0), (10.0, 11.0));
```

If “rows” is a “1” then the vector version can be used:

```
variable BV : real_vector (0 to 2);
```

```
BV := submatrix (A, 1,0, 1, 3);
```

Would return 1 row, 3 columns (real_vector (0 to 2)) starting at location (1,0) or:

```
BV := (5.0, 6.0, 7.0);
```

b) Putting a matrix back together:

BuildMatrix (arg, result, x, y)

where: arg – submatrix to put in the “result” matrix (input)

result – Final matrix (inout)

x – “x” or row location to start from

y – “y” or column location to start from

Examples:

```
variable A : real_matrix (0 to 3, 0 to 3);
```

```
variable B : real_matrix (0 to 1, 0 to 1)
```

```
B := ((7.0, 2.0), (3.0, 4.0));
```

```
A := ones (A'length(1), A'length(2));
```

```
buildmatrix (B, A, 1, 1);
```

Will result in:

```
((1.0, 1.0, 1.0, 1.0),
 (1.0, 7.0, 2.0, 1.0),
 (1.0, 3.0, 4.0, 1.0),
 (1.0, 1.0, 1.0, 1.0));
```

There are special versions for vectors:

BuildMatrix (arg, result, x, y) – where “arg” is a vector

InsertColumn (arg, result, x, y)

Where: arg – real_vector

Result – final matrix (inout)

x – “x” or row location to start from

y – “y” or column location to start from

Example:

```

variable A : real_matrix (0 to 3, 0 to 3);
variable BV, CV : real_vector (0 to 3)
BV := (5.0, 6.0, 7.0, 8.0);
CV := (10.0, 11.0, 12.0, 13.0);
A := ones (A'length(1), A'length(2));
-- Put Vector BV in Matrix A at 2,0 along the "X" (row)
axis
BuildMatrix (BV, A, 2, 0);
-- Put Vector CV in Matrix A at 0,2 along the "Y" (column)
axis
InsertColumn (CV, A, 0, 2);

```

Will result in:

```

((1.0, 1.0, 10.0, 1.0),
 (1.0, 1.0, 11.0, 1.0),
 (5.0, 6.0, 12.0, 8.0),
 (1.0, 1.0, 13.0, 1.0));

```

A vector and a matrix with 1 row are considered to be equivalent. Thus:

```

constant A : real_matrix (0 to 0, 0 to 5) := (others =>
1.0);
constant C : real_vector (0 to 5) := (others => 1.0);
A = C would be True (Assuming C to be a row).

```

The “reshape” function can be use to convert a vector or a matrix to one of any shape desired. For instance:

```

variable M : integer_matrix (0 to 2, 0 to 2);
variable N : integer_vector (0 to 8);
...
begin
  N := (1, 2, 3, 4, 5, 6, 7, 8, 9);
  M := reshape (N, M'length(1), M'length(2));

```

This will result in the following integer matrix:

```

M := ((1, 2, 3),
      (4, 5, 6),
      (7, 8, 9));

```

Index:**Operators:**

“*” - Matrix multiply, overloaded for the following:

real_matrix * real_matrix return real_matrix – Number of Columns in left matrix must match number of rows in right matrix. Returns a matrix which is (left row length by right column length)

`real_matrix * real_vector` return `real_matrix` – Matrix must have 1 column, number of rows must match length of vector. Returns a matrix which is square (vector length by vector length)

`real_vector * real_matrix` return `real_vector` - Matrix must have same number of rows as length of vector. Returned vector will be number of columns in Matrix.

`real_matrix * real` return `real_matrix`

`real * real_matrix` return `real_matrix`

`real_vector * real` return `real_vector`

`real * real_vector` return `real_vector`

“+” – Matrix addition, overloaded for the following:

`real_matrix + real_matrix` return `real_matrix` – Dimensions must match

`real_vector + real_vector` return `real_vector` – Dimensions must match

“-“ – Matrix subtraction, overloaded as follows:

`real_matrix - real_matrix` return `real_matrix` – Dimensions must match

`real_vector - real_vector` return `real_vector` – Dimensions must match

unary minus (`real_matrix`)

“/” – Matrix division

`real_matrix / real_matrix` return `real_matrix` (= `real_matrix * inv(real_matrix)`),

Matrix must be square for this function to work.

`real_matrix / real` return `real_matrix`

`real_vector / real` return `real_vector`

“**”

`real_matrix ** integer` return `real_matrix` – This function is recursive. `Arg**-1 = inv(arg)`. Matrix must be square for this function to work.

“=”

`real_vector = real_matrix` – True if the matrix has one row and equal to the vector

`real_matrix = real_vector`

`real_vector /= real_matrix`

`real_matrix /= real_vector`

“abs” – return the absolute value (`real_matrix` or `real_vector`)

Functions:

Times – Similar to matlab “.*” function (element by element multiply, same as `real_matrix * real`)

Rdivide – Similar to matlab ./ function (element by element divide)

Mrdivide – Similar to matlab mrdivide function (`l * inv(r)`)

Mldivide – Similar to matlab mldivide function (`inv(l)* r`)

Pow – Similar to matlab “.^” function, (element by element `l**r`)

Sqrt – element by element square root function

Exp – element by element exp function

Log – element by element natural log function

Trace – Sum the diagonal of a matrix

Sum (vector) – returns the arithmetic sum of the input vector

Sum (matrix, dim) – returns the sum of a matrix along a given dimension

Dim = 1, sum along the Y axis,

Dim = 2, sum along the X axis
Prod(vector) – returns the arithmetic multiplication of the input vector
Prod (matrix,dim) - returns the arithmetic multiplication of the input along a given dimension
 Dim = 1, multiply along the Y axis,
 Dim = 2, multiply along the X axis
Dot – returns the dot product of two vectors
Cross – returns the cross product of two vectors (or matrices)
Kron – returns the Kronecker product of two matrices
Det – returns the determinant of a matrix
Inv – Inverts a matrix
Linsolve (matrix, vector) – Solves a linear equation
Normalize (matrix, rval) – Normalizes a matrix to the value “rval” (which defaults to 1.0)
Polyval – Evaluates a polynomial

Isempty – returns true if the matrix or vector is null
Transpose – Transposes a matrix
Repmat (val, rows, columns) – Creates a matrix by replicating a single value
Zeros (rows, columns) – returns a matrix of zeros
Ones (rows, columns) – returns a matrix of ones.
eye (rows, columns) – returns an identity matrix
Rand (rows, columns) – returns a matrix of random numbers
Cat (dim, l, r) – Concatenates two matrices along dimension “dim”
Horzcat (l, r) – Concatenates two matrices horizontally
Vertcat (l, r) – Concatenates two matrices vertically
Flipdim (arg, dim) – Flips a matrix along a given dimension
Fliplr – Flip a matrix left to right
Flipup – flip a matrix top to bottom
Rot90 (arg, dim) – rotates a matrix 90 degrees (or more depending on “dim”)
Reshape (arg, rows, columns) – reads a matrix and creates a new one of a different dimensions, can read in a matrix or a vector and return a matrix or a vector.
Size – returns the size of a matrix
Isvector – returns true if the matrix has only one dimension
Isscalar – returns true if there is only one element in this matrix
Numel – returns the number of elements in a matrix
Diag (arg: real_matrix) – returns a vector which is the diagonal of a matrix
Diag (arg: real_vector) – returns a matrix which as the argument as its diagonal.
Blkdiag (arg: real_vector) – returns the block diagonal of a vector.
blockdiag(arg: real_matrix, rep : positive) – Replicates matrix “arg” along the diagonal of the resultant matrix “rep” times.
Repmat (arg, rows, columns) – replicates the “arg” matrix rows*columns times
Tril – returns the lower triangle of a matrix
Triu – returns the upper triangle of a matrix

submatrix (arg, x, y, rows, columns) return real_matrix – Please see above for details
submatrix (arg, x, y, rows, columns) return real_vector

buildmatrix (arg, result, x, y)

buildmatrix (arg, result, x, y) – where “arg” is assumed to be a vector

InsertColumn (arg, result, x, y) – where “arg” is assumed to be a vector

Exclude (arg, row, column) – Return a matrix with the designated row and column removed.

All of the above functions are replicated for types “integer_matrix” and “integer_vector” with the exception of “rdivide”, “/”, “mldivide”, “sqrt”, “log”, “exp”, “inv”, “linsolve”, “normalize”, and “rand”. For these functions, overloads which return “real_matrix” have been created.

Functions which mix real and integer (vector and matrices) are also defined in these packages. The output of these functions is always a real matrix or vector. Thus you can divide a real_matrix by an integer_matrix, with the result being a real_matrix. The following conversion functions are defined:

To_integer (real_matrix) – converts a real_matrix into an integer_matrix with the same dimensions.

To_integer (real_vector) – converts a real_vector into an integer_vector with the same dimensions

To_real (integer_matrix) – converts an integer_matrix into a real_matrix with the same dimensions.

To_real (integer_vector) – converts an integer_vector into a real_vector with the same dimensions.

Textio Functions:

To_string (arg: real_matrix) – returns a string (with LF at the end of every row) delimited by spaces.

Read (L: line; VALUE: real_matrix) – Reads a string which may contain several lines and reads them into matrix “VALUE”. Punctuation and LF are ignored.

Read (L: line; VALUE: real_matrix; GOOD: boolean) – Reads a string which may contain several lines and reads them into matrix “VALUE”. Punctuation and LF are ignored. A Boolean “good” is returned to tell you if the matrix is valid.

Write (L: line, VALUE: real_matrix) – Writes matrix “VALUE” into line “L”. The matrix is punctuated with “(”, “”, “)” and “LF” to delimit columns and rows.

print_matrix (arg : real_matrix; index : Boolean := false);

If index is “false” then the size of the matrix is printed out on the first line, followed by the values of the matrix (one row/line).

If index is “true” then the index of every element is printed before that element, and the matrix size is not printed.

print_vector (arg : real_vector; index : Boolean := false);

If index is “true” then the index of every element is printed before that element.

All of these functions are replicated for “integer_matrix”.

The following functions are also defined in this area (and should be removed when VHDL-2008 supports these)

To_string (arg : real_vector) - returns a string, delimited by spaces

Read (L: line; VALUE: real_vector) – Reads a string into vector “VALUE”.

Punctuation is ignored.

Read (L: line; VALUE: real_vector; GOOD: boolean) – Reads a string into vector “VALUE”. Punctuation is ignored. A Boolean “good” is returned to tell you if the matrix is valid.

Write (L: line, VALUE: real_vector) – Writes vector “VALUE” into line “L”. The matrix is punctuated with “(“, “”, “)” to delimit elements in the array.

These functions are also replicated for “integer_vector”.

Complex_Matrix_pkg package:

This package depends on the IEEE “math_complex” package, as well as “complex_matrix_pkg” (and “math_real”). In the “complex_matrix_pkg” package you will find the following types:

```
type complex_vector is array (NATURAL range <>) of
complex;
type complex_matrix is array (NATURAL range <>, NATURAL
range <>) of complex;
type complex_polar_vector is array (NATURAL range <>) of
complex_polar;
type complex_polar_matrix is array (NATURAL range <>,
NATURAL range <>) of complex_polar;
```

CMPLX(arg) – Converts a real_matrix (or vector) to a complex_matrix (or vector) with a 0 complex portion

CMPLX(X,Y) – Converts the real_matrix (or vector) X into the real portion of the complex_matrix result, and the real_matrix “Y” into the complex part.

COMPLEX_TO_POLAR – Converts a complex_matrix (or vector) to a complex_polar result

POLAR_TO_COMPLEX – converts a complex_polar_matrix (or vector) to a complex_matrix (or vector)

CONJ – returns the complex conjugate of the input

Operators

All of the functions defined for “real_matrix” are defined for “complex_matrix” and “complex_polar_matrix” with the exception of “pow” and “poly” (because there is no generic “**” function in math_complex), and “rand”. These operators are also overloaded for mixing any combination of complex or complex_polar with real.

Exceptions:

Abs(complex_matrix) – returns a real_matrix (complex_vector, complex_polar_matrix, and complex_polar_vector are also valid input types).

Ctranspose(complex_matrix) – returns the complex conjugate of the input matrix. (complex_polar_matrix is also a valid input type).

Textio Functions:

There are no textio functions in “math_complex”, so “to_string”, “read” and “write” are defined in complex_matrix_pkg for the types “complex”, “complex_polar”, “complex_vector”, “complex_polar_vector”, “complex_matrix” and “complex_polar_matrix”.

Fixed_Matrix_pkg package:

This package depends on the IEEE “numeric_std” and “fixed_pkg” packages. The VHDL-93 version is dependent on the “IEEE_PROPOSED” library which can be downloaded at <http://www.vhdl.org/fphdl/vhdl2008c.zip>.

This package was designed to make use of package generics. Also, in VHDL-2008 it is possible to have an unconstrained array of unconstrained arrays. However at this time those options are not available, so we use constants instead. These constants define how large the array types are in the matrices.

```
constant ufixed_matrix_high    : INTEGER := 15;
constant ufixed_matrix_low     : INTEGER := -16;
constant sfixed_matrix_high    : INTEGER := 15;
constant sfixed_matrix_low     : INTEGER := -16;
constant unsigned_matrix_high  : NATURAL := 15;
constant signed_matrix_high    : NATURAL := 15;
```

These constants are used to define the sizes of the elements of the matrices and vectors as follows:

```
subtype ufixedr is UNRESOLVED_ufixed (ufixed_matrix_high
downto ufixed_matrix_low);
subtype sfixedr is UNRESOLVED_sfixed (sfixed_matrix_high
downto sfixed_matrix_low);
subtype unsignedr is UNSIGNED (unsigned_matrix_high
downto 0);
subtype signedr is SIGNED (signed_matrix_high downto 0);

-- Define the base types
type ufixed_matrix is array (NATURAL range <>, NATURAL
range <>) of ufixedr;
```

```

type ufixed_vector is array (NATURAL range <>) of
ufixedr;
type sfixed_matrix is array (NATURAL range <>, NATURAL
range <>) of sfixedr;
type sfixed_vector is array (NATURAL range <>) of
sfixedr;
type unsigned_matrix is array (NATURAL range <>, NATURAL
range <>) of unsignedr;
type unsigned_vector is array (NATURAL range <>) of
unsignedr;
type signed_matrix is array (NATURAL range <>, NATURAL
range <>) of signedr;
type signed_vector is array (NATURAL range <>) of
signedr;

```

All of the functions which are described in the “real_matrix_pkg” documentation are defined for the types “sfixed_matrix” and “sfixed_vector” with the exception of the “rand” function.

For the ufixed_matrix and ufixed_vector types the functions which involve matrix inversion (Matrix/Matrix, mrdivide, mldivide, and inv) are not defined. Also “polyval” and “linsolve” are undefined. Copies of these functions which accept ufixed_matrix and return sfixed_matrix are defined.

For the unsigned_matrix, unsigned_vector, signed_matrix, and signed_vector types any functions which perform a “divide” are not defined (normalize, any “/” operators and inv). Overloads which return “sfixed_matrix” exist for the “signed” types. However for unsigned types you will need to convert to “signed” or “ufixed” before performing any of these functions.

Conversion functions are defined as follows:

To_ufixed – Can have “unsigned_matrix”, “real_matrix”, “integer_matrix” (and the _vector versions) as input.

To_sfixed – Can have “signed_matrix”, “real_matrix”, “integer_matrix”, “ufixed_matrix”, “unsigned_matrix” (and the _vector versions) as input.

To_unsigned – Can have “ufixed_matrix”, “integer_matrix” (and the _vector versions) as input.

To_signed – Can have “sfixed_matrix”, “integer_matrix”, “unsigned_matrix”, “ufixed_matrix” (and the _vector versions) as input.

To_real – Can have “ufixed_matrix”, “sfixed_matrix” (and the _vector versions) as input.

To_integer – can have “unsigned_matrix”, “signed_matrix” (and the _vector versions) as input.