

Proposal how to handle 2-dimensional interpolation in Table-Driven Modeling

Purpose of the material

The purpose of the material is to provide a proposal concerning handling of real-valued two-dimensional interpolation in VHDL-AMS based on a solution for real-valued one-dimensional interpolation. The solution shall be in accordance with the Verilog-AMS approach. Problems of an extension to multi-dimensional interpolation are discussed.

1. Starting point

The Verilog-AMS LRM defines the following procedure for multi-dimensional interpolation (see [1], pp. 224-225 and 230):

A typical example will help to explain the process. A user may wish to create a data based model of some function $f(x,y)$ over some range of x and y and use that data as the basis of a behavioral model described in Verilog-AMS.

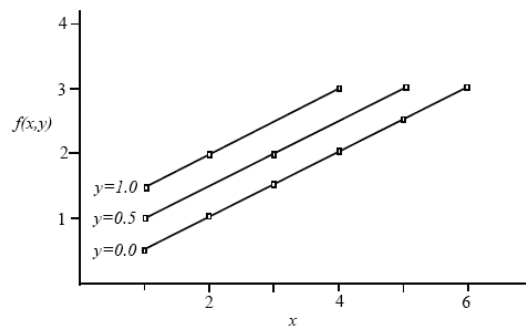


Figure 9-1: Samples on isolines

When describing the sampled set, x and y are called independent variables and $f(x,y)$ is called the dependent variable. The sampling scheme also introduces the concept of an innermost and outermost dimension. In this example, x is the fastest changing or innermost dimension associated with the sampled function $f(x,y)$ and y is the slowest changing or outermost dimension.

Understanding that the underlying multidimensional function is sampled on a set of isolines, we can now describe a simple recursive process to interpolate, extrapolate, or perform lookup on this sampled function. Using the above example let us assume the user requests a value for the lookup pair $(x1,y1)$. We first look through the set of isolines in y and find the pair that bracket $y1$. Now for each isoline in y we find the two points that bracket $x1$ and interpolate each isoline to find $f(x1,yh)$ and $f(x1,yl)$. Having thus generated an isoline in y for the point $x1$ in x , we may interpolate this isoline to find the value $f(x1,y1)$. If the lookup point falls off the end of any given isoline then we extrapolate its value on that isoline.

As a consequence of this algorithm, the interpolation and extrapolation schemes always operate in a single dimension analogous to how the data was originally generated, so the interpolation and extrapolation schemes used may be specified on a per dimension basis.

[...]

The data source may also be specified as an array.

```

module example_tb(a, b);
    electrical a, b;
    inout a, b;
    real y[0:11], x[0:11], f_xy[0:11];
    analog begin
        @(initial_step) begin
            // y=0.0 isoline
            y[0] =0.0;  x[0] =1.0;  f_xy[0] =0.5;

```

```

        y[1] =0.0;  x[1] =2.0;  f_xy[1] =1.0;
        y[2] =0.0;  x[2] =3.0;  f_xy[2] =1.5;
        y[3] =0.0;  x[3] =4.0;  f_xy[3] =2.0;
        y[4] =0.0;  x[4] =5.0;  f_xy[4] =2.5;
        y[5] =0.0;  x[5] =6.0;  f_xy[5] =3.0;
        // y=0.5 isoline
        y[6] =0.5;  x[6] =1.0;  f_xy[6] =1.0;
        y[7] =0.5;  x[7] =3.0;  f_xy[7] =2.0;
        y[8] =0.5;  x[8] =5.0;  f_xy[8] =3.0;
        // y=1.0 isoline
        y[9] =1.0;  x[9] =1.0;  f_xy[9] =1.5;
        y[10]=1.0;  x[10]=2.0;  f_xy[10]=2.0;
        y[11]=1.0;  x[11]=4.0;  f_xy[11]=3.0;
    end
    I(a, b) <+ $table_model(0, V(a,b), y, x, f_xy);
end
endmodule

```

2. VHDL-AMS function

The function that is in accordance with the Verilog-AMS approach could be declared as follows

```

function INTERPOLATION_2D (
    XV0    : REAL;
    XV1    : REAL;
    X0     : REAL_VECTOR;
    X1     : REAL_VECTOR;
    Y      : REAL_VECTOR;
    METHOD  : INTERPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_S1);
    LEFT   : EXTRAPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_L);
    RIGHT  : EXTRAPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_L))
return REAL;

```

The function and the used declarations can be found in the appendix.

The function INTERPOLATION_2D delivers an interpolation or extrapolation value for the arguments $XV0$ and $XV1$

$$result = f(XV0, XV1) \quad \text{with } f : R \times R \rightarrow R$$

The data points are given by the arrays $X0$ and $X1$. The associated values are provided in the array Y . Thus

$$Y(I) = f(X0(I), X1(I)) \quad \text{for } I \text{ in the range of } X0, X1 \text{ and } Y \text{ resp.}$$

The arrays $X0$ and $X1$ must be ordered and fulfill the following conditions

1. $X0(I) \leq X0(I+1)$ for I in $X0'$ LEFT to $X0'$ RIGHT - 1
2. $X0(I) = X0(I+1) \Rightarrow X1(I) < X1(I+1)$
3. $X1$ should be “finer/faster” separated than $X0$

The first and second conditions must be fulfilled (necessary conditions for the application of the function). This conditions assure that in a simple way “*the interpolation and extrapolation schemes used may be specified on a per dimension basis*”.

The third condition corresponds to the the Verilog-AMS requirement “*x[equals X1] is the fastest changing or innermost dimension associated with the sampled function f(x,y) and y[equals X0] is the slowest changing or outermost dimension.*”

3. Example

The following test-bench demonstrates the use of the function. It uses the same data base as described in section 1.

```

library IEEE;
use WORK.TABLE_INTERPOLATION_PKG.all;

entity BENCH_TABLE is end BENCH_TABLE;

architecture A2 of BENCH_TABLE is
  -- Example from Verilog-AMS LRM section 9.20.6
  constant Y      : REAL_VECTOR (0 to 11) := (0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                                                0.5, 0.5, 0.5,
                                                1.0, 1.0, 1.0);

  constant X      : REAL_VECTOR (0 to 11) := (1.0, 2.0, 3.0, 4.0, 5.0, 6.0,
                                                1.0, 3.0, 5.0,
                                                1.0, 2.0, 4.0);

  constant F_XY   : REAL_VECTOR (0 to 11) := (0.5, 1.0, 1.5, 2.0, 2.5, 3.0,
                                                1.0, 2.0, 3.0,
                                                1.5, 2.0, 3.0);

  quantity Q0    : REAL;
  quantity Q05   : REAL;
  quantity Q075  : REAL;
  quantity Q1    : REAL;
  quantity Q2    : REAL;

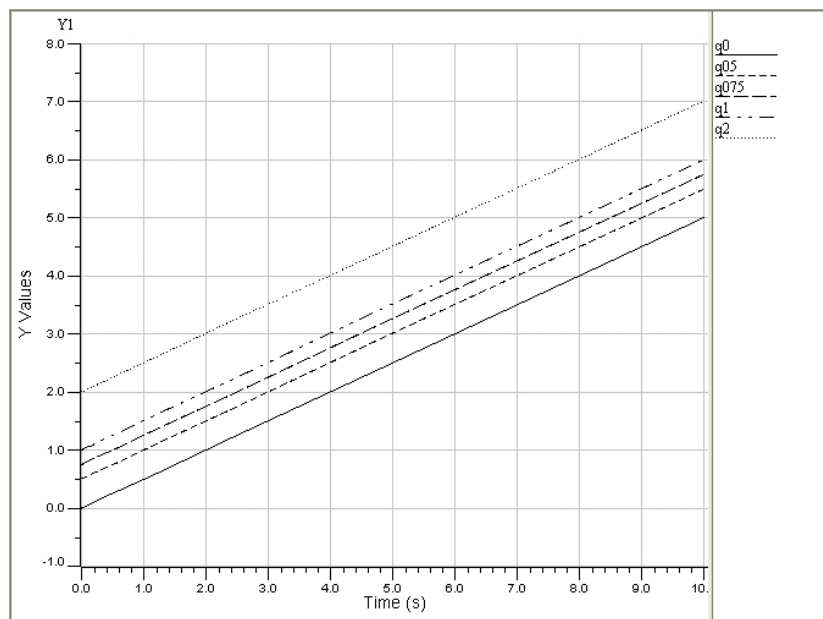
begin

  Q0  == INTERPOLATION_2D (0.0,  NOW, Y, X, F_XY);
  Q05 == INTERPOLATION_2D (0.5,  NOW, Y, X, F_XY);
  Q075 == INTERPOLATION_2D (0.75, NOW, Y, X, F_XY);
  Q1  == INTERPOLATION_2D (1.0,  NOW, Y, X, F_XY);
  Q2  == INTERPOLATION_2D (2.0,  NOW, Y, X, F_XY);

end architecture A2;

```

Results



4. VHDL-AMS aspects

In a similar way as for the two-dimensional case the functions can be declared and implemented for the 3-, 4-dimensional case and so on. However, a better solution would be declare a general function. This requires (to my opinion, Ha.) a declaration of the function interface that does not fix the dimension.

Multidimensional array declaration

In the n-dimensional case the independent values of the data points could be provided by an array that contains n-dimensional real vectors. However declaration of a vector type that would be needed is only possible for constraint arrays:

```
constant DIM : INTEGER := 3;  
type TABLE_DATA is array (NATURAL range <>) of REAL_VECTOR (0 to DIM-1);
```

Pointer array

To overcome this restriction the data could be provided using an array of pointers. The corresponding type declarations could look like

```
type REAL_VECTOR_PTR is access REAL_VECTOR;  
type REAL_VECTOR_PTR_VECTOR is array (NATURAL range <>) of REAL_VECTOR_PTR;
```

However, data objects of a pointer type cannot be declared as constants in the declarational part of an architecture and cannot be handed over as constants to a function.

Conclusion

It seems that there is no other solution than provide different interpolation functions for different dimensions.

References

- [1] Verilog-AMS Language Reference Manual Version 2.3
Accellera, August 4, 2008.

Ha./January 12, 2011

Appendix

Version 0.2 of the TABLE_INTERPOLATION_PKG

```

library IEEE;
use IEEE.MATH_REAL.all;
use IEEE.MATH_COMPLEX.all;

package TABLE_INTERPOLATION_PKG is

  /**
  -- Declaration of a complex array.
  */
  type COMPLEX_VECTOR is array (NATURAL range <>) of COMPLEX;

  /**
  -- Declaration of enumeration type INTERPOLATION_METHOD.
  -- This is only a preliminary declaration for test purposes.
  -- The final declaration will be changed.
  */
  type INTERPOLATION_METHOD is (TDM_D, TDM_S0, TDM_S1, TDM_S2, TDM_S3, TDM_RATIONAL);

  /**
  -- Declaration of enumeration type INTERPOLATION_METHOD.
  -- This is only a preliminary declaration for test purposes.
  -- The final declaration will be changed.
  */
  type EXTRAPOLATION_METHOD is (TDM_C, TDM_L, TDM_E, TDM_EXTRAP, TDM_P);

  /**
  -- Declaration of array type INTERPOLATION_METHOD_VECTOR.
  */
  type INTERPOLATION_METHOD_VECTOR is array (NATURAL range <>) of INTERPOLATION_METHOD;

  /**
  -- Declaration of array type EXTRAPOLATION_METHOD_VECTOR.
  */
  type EXTRAPOLATION_METHOD_VECTOR is array (NATURAL range <>) of EXTRAPOLATION_METHOD;

  /**
  -- Real-valued function INTERPOLATION_1D.
  -- <p>The function carries out real-valued one-dimensional interpolation.
  -- The function requires that the condition
  --  $X(X'LEFT) < X(X'LEFT+1) < \dots < X(X'RIGHT)$ 
  -- is fulfilled,
  -- </p>
  --
  -- <p>Interpolation method TDM_RATIONAL is not supported.</p>
  --
  -- @arg XV      Argument for which the value shall be determined.
  -- @arg X       Vector to characterize the X (independent) values of knots
  -- @arg Y       Vector to characterize the associated Y (independent) values of knots
  -- @arg METHOD   Interpolation method for  $X(X'LEFT) \leq XV \leq X(X'RIGHT)$ 
  -- @arg LEFT    Method for extrapolation on the left if  $XV < X(X'LEFT)$ 
  -- @arg RIGHT   Method for extrapolation on the right if  $XV > X(X'RIGHT)$ 
  -- @return      Function value for XV
  */
  function INTERPOLATION_1D (XV      : REAL;
                             X       : REAL_VECTOR;
                             Y       : REAL_VECTOR;
                             METHOD    : INTERPOLATION_METHOD := TDM_S1;
                             LEFT     : EXTRAPOLATION_METHOD := TDM_L;
                             RIGHT    : EXTRAPOLATION_METHOD := TDM_L)
    return REAL;

  /**
  -- Complex-valued function INTERPOLATION_1D.
  -- <p>The function carries out complex-valued one-dimensional interpolation.
  -- The function requires that the condition
  --  $X(X'LEFT) < X(X'LEFT+1) < \dots < X(X'RIGHT)$ 
  -- is fulfilled.
  -- </p>
  --
  */

```

```

-- <p>If the interpolation method is TDM_RATIONAL the extrapolation
-- methods TDM_EXTRAP and TDM_E are supported.</p>
--
--
-- @arg XV      Argument for which the value shall be determined.
-- @arg X       Vector to characterize the X (independent) values of knots
-- @arg Y       Vector to characterize the associated Y (independent) values of knots
-- @arg METHOD   Interpolation method for X(X'LEFT) <= XV <= X(X'RIGHT)
-- @arg LEFT    Method for extrapolation on the left if XV < X(X'LEFT)
-- @arg RIGHT   Method for extrapolation on the right if XV > X(X'RIGHT)
-- @return     Function value for XV
--*/
function INTERPOLATION_1D (XV      : REAL;
                          X       : REAL_VECTOR;
                          Y       : COMPLEX_VECTOR;
                          METHOD    : INTERPOLATION_METHOD := TDM_S1;
                          LEFT     : EXTRAPOLATION_METHOD := TDM_E;
                          RIGHT    : EXTRAPOLATION_METHOD := TDM_E)

return COMPLEX;

--/**
-- Real-valued function INTERPOLATION_2D.
-- <p>The function carries out real-valued 2-dimensional interpolation.
-- The function requires a special order for the arrays X0 and X1.
-- </p>
--
-- <p>Interpolation method TDM_RATIONAL is not supported.</p>
--
-- @arg XV0     First Argument for which the value shall be determined.
-- @arg XV1     Second Argument for which the value shall be determined.
-- @arg X0      Vector to characterize the first component (independent) values of knots
-- @arg X1      Vector to characterize the second component (independent) values of knots
-- @arg Y       Vector to characterize the associated Y (independent) values of knots
-- @arg METHOD   Interpolation methods for the first and second component resp.
-- @arg LEFT    Methods for extrapolation on the left for the first and second component resp.
-- @arg RIGHT   Methods for extrapolation on the right for the first and second component
-- resp.
-- @return     Function value for XV = (XV0, XV1)
--*/
function INTERPOLATION_2D (XV0     : REAL;
                          XV1     : REAL;
                          X0      : REAL_VECTOR;
                          X1      : REAL_VECTOR;
                          Y       : REAL_VECTOR;
                          METHOD    : INTERPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_S1);
                          LEFT     : EXTRAPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_L);
                          RIGHT    : EXTRAPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_L)
)

return REAL;

end package TABLE_INTERPOLATION_PKG;

```

```

package body TABLE_INTERPOLATION_PKG is

--/**
-- Real-valued function INTERPOLATION_2D.
-- <p>The function carries out real-valued 2-dimensional interpolation.
-- The function requires a special order for the arrays X0 and X1
-- </p>
--
-- <p>Interpolation method TDM_RATIONAL is not supported.</p>
--
-- @arg XV0      First Argument for which the value shall be determined.
-- @arg XV1      Second Argument for which the value shall be determined.
-- @arg X0       Vector to characterize the first component (independent) values of knots
-- @arg X1       Vector to characterize the second component (independent) values of knots
-- @arg Y        Vector to characterize the associated Y (independent) values of knots
-- @arg METHOD    Interpolation methods for the first and second component resp.
-- @arg LEFT     Methods for extrapolation on the left for the first and second component resp.
-- @arg RIGHT    Methods for extrapolation on the right for the first and second component
--              resp.
-- @return      Function value for XV = (XV0, XV1)
--*/
function INTERPOLATION_2D (XV0      : REAL;
                          XV1      : REAL;
                          X0       : REAL_VECTOR;
                          X1       : REAL_VECTOR;
                          Y        : REAL_VECTOR;
                          METHOD    : INTERPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_S1);
                          LEFT     : EXTRAPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_L);
                          RIGHT    : EXTRAPOLATION_METHOD_VECTOR (0 to 1) := (others => TDM_L)
)
return REAL is

    function NUMBER_OF_CONSECUTIVE_POINTS (X : REAL_VECTOR)
    return INTEGER is
        constant M      : POSITIVE := X'LENGTH - 1;
        variable SUM    : INTEGER;
        variable RESULT : INTEGER;
    begin
        SUM := 0;
        RESULT := 0;
        for I in 0 to M-1 loop
            if X(I) < X(I+1) then
                SUM := SUM + 1;
            end if;
            if X(I) > X(I+1) then
                if SUM + 1 > RESULT then
                    RESULT := SUM + 1;
                end if;
                SUM := 0;
            end if;
        end loop;

        if SUM + 1 > RESULT then
            RESULT := SUM + 1;
        end if;

    return RESULT;
end function NUMBER_OF_CONSECUTIVE_POINTS ;

    function CHECK_DATA_X0_X1 (X0, X1 : REAL_VECTOR)
    return BOOLEAN is
        constant M      : POSITIVE := X0'LENGTH - 1;
        variable RESULT : BOOLEAN := TRUE;
    begin
        for I in 0 to M - 1 loop
            if X0(I) > X0(I+1) then
                RESULT := FALSE;
                exit;
            elsif I = 0 then
                null;
            elsif X0(I) = X0(I-1) then
                if X1(I) <= X1(I-1) then
                    RESULT := FALSE;
                    exit;
                end if;
            end if;
        end loop;
    end if;
end function CHECK_DATA_X0_X1 ;

```

```

        end if;
    end loop;
return RESULT;
end function CHECK_DATA_X0_X1;

constant M          : POSITIVE := X0'LENGTH - 1;

-- Comment:
-- Due to implementation limitations of the used simulator the following
-- declarations are used instead of the alias construct:

variable X0_ALIAS   : REAL_VECTOR (0 to M)           := X0;
variable X1_ALIAS   : REAL_VECTOR (0 to X1'LENGTH - 1) := X1;
variable F_ALIAS    : REAL_VECTOR (0 to Y'LENGTH - 1) := Y;

variable X_ALIAS    : REAL_VECTOR (0 to M);
variable Y_ALIAS    : REAL_VECTOR (0 to M);

variable IDX        : INTEGER;
variable DO_INTERPOLATION : BOOLEAN := FALSE;
variable RESULT     : REAL;
begin

-- Handling of irregular situations

assert Y'LENGTH = X0'LENGTH
    report "X0 and Y must be of the same length."
    severity ERROR;

assert Y'LENGTH = X1'LENGTH
    report "X1 and Y must be of the same length."
    severity ERROR;

-- Check of data order

assert CHECK_DATA_X0_X1 (X0_ALIAS, X1_ALIAS)
    report "Data order not correct."
    severity ERROR;

assert NUMBER_OF_CONSECUTIVE_POINTS (X0_ALIAS) <= NUMBER_OF_CONSECUTIVE_POINTS (X1_ALIAS)
    report "X0 finer separated than X1"
    severity WARNING;

-- Determine result

-- X1 line

IDX          := 0;
X_ALIAS (0) := X1_ALIAS (0);
Y_ALIAS (0) := F_ALIAS (0);
for I in 1 to M loop
    if X0_ALIAS (I) = X0_ALIAS (I-1) or IDX = -1 then
        IDX          := IDX + 1;
        X_ALIAS (IDX) := X1_ALIAS (I);
        Y_ALIAS (IDX) := F_ALIAS (I);
    end if;

    if I <= M - 1 then
        if X0_ALIAS (I) < X0_ALIAS (I+1) then
            DO_INTERPOLATION := TRUE;
        end if;
    else
        DO_INTERPOLATION := TRUE;
    end if;

    if DO_INTERPOLATION then
        F_ALIAS (I - IDX)
            := INTERPOLATION_1D (
                XV      => XV1,
                X       => X_ALIAS (0 to IDX),
                Y       => Y_ALIAS (0 to IDX),
                METHOD   => METHOD (1),
                LEFT    => LEFT (1),
                RIGHT   => RIGHT (1)
            );
        IDX := -1;
        DO_INTERPOLATION := FALSE;
    end if;
end loop;
end function;

```

```
        end if;
    end loop;

    -- X0 line

    IDX          := 0;
    X_ALIAS (0) := X0_ALIAS(0);
    Y_ALIAS (0) := F_ALIAS(0);

    for I in 0 to M loop
        if X_ALIAS (IDX) < X0_ALIAS (I) then
            IDX := IDX + 1;
            X_ALIAS (IDX) := X0_ALIAS (I);
            Y_ALIAS (IDX) := F_ALIAS (I);
        end if;
    end loop;

    RESULT := INTERPOLATION_1D (
        XV => XV0,
        X  => X_ALIAS (0 to IDX),
        Y  => Y_ALIAS (0 to IDX),
        METHOD => METHOD (0),
        LEFT => LEFT (0),
        RIGHT => RIGHT (0)
    );

    return RESULT;
end function INTERPOLATION_2D;

end package body TABLE_INTERPOLATION_PKG;
```