3 FUNCTION SIGNATURES

3.1 AMI_Init

3.1.1 Declaration

```
long AMI_Init (double *impulse_matrix,
               long row_size,
               long aggressors,
               double sample_interval,
               double bit_time,
               char *AMI_parameters_in,
               char **AMI_parameters_out,
               void **AMI_memory_handle,
               char **msg)
```

3.1.2 Arguments

3.1.2.1 impulse_matrix

'impulse_matrix' is the channel impulse response matrix. The impulse values are in volts and are uniformly spaced in time. The sample spacing is given by the parameter ‘sample_interval’.

The impulse_matrix is stored in a single dimensional array of floating point numbers which is formed by concatenating the columns of the impulse response matrix, starting with the first column and ending with the last column. The matrix elements can be retrieved/identified using

```
  impulse_matrix[idx] = element (row, col)
  idx = col * number_of_rows + row
  row – row index , ranges from 0 to row_size-1
  col – column index, ranges from 0 to aggressors
```

The first column of the impulse_matrix is the impulse response for the primary channel. The rest are the impulse responses from aggressor drivers to the victim receiver.

The AMI_Init function may return a modified impulse response by modifying the first column of impulse_matrix. If the impulse response is modified, the new impulse response is expected to represent the filtered response. The number of items in the matrix should remain unchanged.

The aggressor columns of the matrix should not be modified.

3.1.2.2 row_size

The number of rows in the impulse_matrix.

3.1.2.3 aggressors

The number of aggressors in the impulse_matrix.

3.1.2.4 sample_interval

This is the sampling interval of the impulse_matrix. Sample_interval is usually a fraction of the highest data rate (lowest bit_time) of the device. Example:

```
Sample_interval = (lowest_bit_time/64)
```

3.1.2.5 bit_time

The bit time or unit interval (UI) of the current data, e.g., 100 ps, 200 ps etc. The shared library may use this information along with the impulse_matrix to initialize the filter coefficients.

3.1.2.6 AMI_parameters (_in and _out)

Memory for AMI_parameters_in is allocated and de-allocated by the EDA platform. The memory pointed to by AMI_parameters_out is allocated and de-allocated by the model. This is a pointer to a string. All the input from the IBIS AMI parameter file are passed using a string that been formatted as a parameter tree.

Examples of the tree parameter passing is:

```
(dll
  (tx
    (taps 4)
    (spacing sync)
  )
)
```

and

```
(root
  (branch1
    (leaf1 value1)
    (leaf2 value2)
    (branch2
      (leaf3 value3)
      (leaf4 value4)
    )
    (leaf5 value5 value6 value7)
  )
)
```

The syntax for this string is:

1. Neither names nor individual values can contain white space characters.
2. Parameter name/value pairs are always enclosed in parentheses, with the value separated from the name by white space.
3. A parameter value in a name/value pair can be either a single value or a list of values separated by whitespace.
4. Parameter name/value pairs can be grouped together into parameter groups by starting with an open parenthesis followed by the group name followed by the concatenation of one or more name/value pairs followed by a close parenthesis.

| 5. Parameter name/values pairs and parameter groups can be freely
| intermixed inside a parameter group.
| 6. The top level parameter string must be a parameter group.
| 7. White space is ignored, except as a delimiter between the parameter name
| and value.
| 8. Parameter values can be expressed either as a string literal, decimal
| number or in the standard ANCI 'C' notation for floating point numbers
| (e.g., 2.0e-9). String literal values are delimited using a double
| quote (") and no double quotes are allowed inside the string literals.
| 9. A parameter can be assigned an array of values by enclosing the
| parameter name and the array of values inside a single set of
| parentheses, with the parameter name and the individual values all
| separated by white space.
|
| The modified BNF specification for the syntax is:
|
| <tree>:
| <branch>
|
| <branch>:
| (<branch name> <leaf list>)
|
| <leaf list>:
| <branch>
| <leaf>
| <leaf list> <branch>
| <leaf list> <leaf>
|
| <leaf>:
| (<parameter name> whitespace <value list>)
|
| <value list>:
| <value>
| <value list> whitespace <value>
| <value>:
| <string literals>
| <decimal number>
| <decimal number>e<exponent>
| <decimal number>E<exponent>
|
| 3.1.2.7 AMI_memory_handle
|
| Used to point to local storage for the algorithmic block being modeled and
| shall be passed back during the AMI_GetWave calls. e.g. a code snippet may
| look like the following:
|
| my_space = allocate_space(sizeof_space);
| status = store_all_kinds_of_things(my_space);
| *serdes_memory_handle = my_space;
|
| The memory pointed to by AMI_handle is allocated and de-allocated by the
| model.
|
| 3.1.2.8 msg (optional)
|
| Provides descriptive, textual message from the algorithmic model to the EDA
| platform. It must provide a character string message that can be used by

| EDA platform to update log file or display in user interface.
|
3.1.3 Return Value
|
| 1 for success
| 0 for failure
|
|
3.2 AMI_GetWave
|
3.2.1 Declaration
|
```
| long AMI_GetWave (double *wave,
|                   long wave_size,
|                   double *clock_times,
|                   char **AMI_parameters_out,
|                   void *AMI_memory);
```
|
3.2.2 Arguments
|
3.2.2.1 wave
|
| A vector of a time domain waveform, sampled uniformly at an interval
| specified by the 'sample_interval' specified during the init call. The
| wave is both input and output. The EDA platform provides the wave. The
| algorithmic model is expected to modify the waveform in place by applying
| a filtering behavior, for example, an equalization function, being
| modeled in the AMI_Getwave call.
|
| Depending on the EDA platform and the analysis/simulation method chosen,
| the input waveform could include many components. For example, the input
| waveform could include:
|
| - The waveform for the primary channel only.
| - The waveform for the primary channel plus crosstalk and amplitude noise.
| - The output of a time domain circuit simulator such as SPICE.
|
| It is assumed that the electrical interface to either the driver or the
| receiver is differential. Therefore, the sample values are assumed to be
| differential voltages centered nominally around zero volts. The
| algorithmic model's logic threshold may be non-zero, for example to model
| the differential offset of a receiver; however that offset will usually be
| small compared to the input or output differential voltage.
|
| The output waveform is expected to be the waveform at the decision point of
| the receiver (that is, the point in the receiver where the choice is made
| as to whether the data bit is a "1" or a "0"). It is understood that for
| some receiver architectures, there is no one circuit node which is the
| decision point for the receiver. In such a case, the output waveform is
| expected to be the equivalent waveform that would exist at such a node
| were it to exist.
|
3.2.2.2 wave_size
|
| Number of samples in the waveform vector.