1450.4 meeting minutes – 12/17/09

Attendees: Jim O'Reilly, Markus Seuring, Bruce Parnas, Ernie Wahl, Ajay Khoche

Not present:
Agenda:

- **NOTE: Summary includes discussion at the meeting, as well as a number of followup emails which were sent through the P1450.4 email reflector).**
- IEEE Meeting Preamble (No discussion of proprietary information).
- Discussion items
  - Discussion of keywords and units to be used for real_var_type (lines 24-37 of D29 syntax document).
    - Use keywords based on Units (Watts, instead of Power, for instance), using multiple keywords (DegC/DegF, Meters/Inches) for similar types that can have different units.
  - Conversion between Boolean and Double/Integer, Double/Integer and Boolean.
    - Double excluded. No direct conversions between Boolean and Double (can still convert Double->Integer->Boolean or Boolean->Integer->Double in multiple assignment steps, if desired – with all the caveats regarding Double->Integer conversion)
    - Boolean to Integer: False = 0, True =1
    - Integer to Boolean: 0 = False, non-zero = True
    - No mathematical operations allowed on booleans; logical operations are allowed:
      - =, !=, <, >, <=, >=, !, &&, ||
  - What operations are allowed on integers?
    - +, -, /, *, %, ==, >, <, >=, <=, != (modulus can be applied only to integer, all others can be applied to Double as well).
  - Allow enums? Yes or no?
    - Yes, per C/C++ rules – with one exception. Cannot assign starting value of enum – they ALWAYS will start with 0!!!
      - Disallow direct conversions between enums and Boolean, enums and Double. As above for Boolean, can convert from enum->Integer->Double or Double->Integer->enum in multiple assignment steps (with all the caveats regarding Double->Integer conversion)
      - Conversions from enums to Integer allowed, conversions from Integer to enum allowed within range enum values (if converting from a integer to enum outside the range specified by enum, error results).
      - Enum value will be either string name (enum literal name) or numerical value, depending on context. If assigned to an integer, get a numerical value. If assigned to a string value (or used in as string context, such as a print statement), get a string value.
  - How will uninitialized selector fields of Spec variables be handled?
    - In Dot0, accessing an uninitialized field other than Meas is a compile time error; in Dot4 access is permitted. Any operation involving NaN results in NaN and may result in a runtime error.
    - Any Min/Typ/Max selector field not explicitly declared in the Spec block - using either the assignment form (for Typ only), or the block form (for any combination of Min/Typ/Max) will have the value of **None**. The .Meas field,will also have a value of **None**, UNLESS it's explicitly assigned a numerical value in the spec block definition, or until its assigned a numerical value elsewhere (i.e., within a Test or Flow). Only the .Meas field is read-write (i.e., can be assigned a value outside the declaration). All other fields are read-only, and can be given a value ONLY at declaration time. If any field has the value **None** and is used in a time_expr or dc_expr, it is an error. *Note that* **None** *is already going to be used in another context (as an initial value for a Test*

(TestType) or Flow (FlowType) parameter, as defined in the type definition. If a parameter is initialized to **None** *and is not overridden to a value other than* ***None*** *during instantiation, it is ignored during execution.*

- Anything uninitialized (spec/category stuff) will have None.

- Explicit vs. implicit initialization of variables at declaration time.
  - Variables can be initialized at declaration. They can be initialized to a valid value, or to None. If not initialized at declaration time, variables are implicitly initialized to None. The meaning of None depends on variable type (see below).
- Do we want to support a NotInitialized member function for variable types
  - Using NaN to indicate NotInitialized does not work for all variable types
  - Instead, we'll use None to indicate "not initialized" for all variable types. The actual value used for None in implementations is not specified, but the following are recommended:
    Integer – **None** (Could be implemented as None = MAXINT)
    Boolean – **None** (Could be implemented as ternary (True/False/None))
    stil_data_type – **None**
    real_var_type – **None** (Could be implemented as None = NaN)
    String – **None** (Could be implemented with sentinel string)
  - Enum – **None** (Could be implemented with extra enum member)
- Is it allowed to define a spec variable without units?
  - At least two languages support this (EnVision and SmarTest). Therefore, we should support this.
- Is it allowed to define Meas-only variables?
  - No language currently supports Meas-only variables; at least one other selector field (Min, Typ, or Max) must also be defined. Since a global or local variable can serve the same purpose as a Meas-only spec variable, Meas-only spec variables will not be allowed. Units are established for a spec variable by including them with the numerical value in the spec variable value assignment, or by using the Units keyword within the block form of the spec variable declaration (see syntax below).
    - The units for all selector fields of a spec variable must be the same. These units can be established by including them with the numerical value in the spec variable value assignment, OR by specifying the units using the Units keyword within the block form of the spec variable declaration. If there's any mismatch in the units for various components of a spec variable (units only, NOT scaling factor – i.e., mV and V are identical units), that condition is an error.
    - At least one of the Min, Typ, or Max selector fields of a variable MUST be defined, in addition to the Meas selector field; the units which apply to the Meas selector field must also be the same as those for any of the other selector fields.
- Syntax for Spec block definition:

spec_expr = time_expr | dc_expr | real_expr // time_expr and dc_expr will include units; real_expr will not
units_expr = <string>, where <string> can contain any of the following: A, DegC, DegF, F, db, H, Hz, i, m, Ohms, s, V, W, None, or any combination of the above

Spec (SPEC_BLOCK_NAME) { // this block statement defines variable values for a given category
(**Category** CAT_NAME {
(VAR_NAME = *spec_expr*;)* // define only the Typ value
(VAR_NAME { ((**Min** *spec_expr*;) (**Typ** *spec_expr*;) (**Max** *spec_expr*;))+ (**Meas** *spec_expr*;) (**Units** *units_expr*;) })*
})+
| (**Variable** VAR_NAME {
(CAT_NAME = *spec_expr*;)* // define only the Typ value
(CAT_NAME { ((**Min** *spec_expr*;) (**Typ** *spec_expr*;) (**Max** *spec_expr*;))+ (**Meas** *spec_expr*;) (**Units** *units_expr*;) })*
})+
}

- Rules:
  - The combination of spec block name+category name+spec variable name MUST be unique. This is an extension of the dot0 rule stating

that the combination of category name+spec variable name must be unique.
        - Incremental definition of the various categories in a spec block, or the various variables in a spec block category, is permitted (as it is in dot0), but NOT recommended. Even with incremental definition, the constraint regarding uniqueness of spec block name+category_name+spec variable name MUST be adhered to.
        - At least one of the selector fields Min, Typ, or Max must be defined, in addition to Meas. Any selector field not mentioned in the spec block definition will have the value None.
        - Units can be assigned by using the *time_expr* or *dc_expr* (both of which include units), or by using *real_expr* (which does not include units) in conjunction with the **Units** keyword.
        - Units for all selector field values must be the same. If some selector fields have units specified (using *time_expr* or *dc_expr*), but others do not (using *real_expr*), the units specified by the selector fields with units will apply to the selector fields without units. If no units are specified for any selector fields, and the keyword Units is not used, the spec variable is unitless (i.e. <var_name>.Units will return None). If the keyword Units IS used and specifies None, the spec variable is also unitless.
    - Do we want to support array arithmetic ? e.g., SpecVariable +-scalar or SpecVariable +-SpecVariable. Might be used for guardbands.
        - dot0 spec var automatically refers to Typ if there's no selector and Typ is all that is defined. Can't do that in dot4 if array arithmetic is permitted.
        - No. Spec variable selector fields are not arrays. There's no need to support array arithmetic for this type of operation - though, since we DO support arrays, there might be some value to supporting array arithmetic (particularly as we look ahead to mixed-signal test methods, etc.). However, for flow-related issues, I don't think it's needed, and we should probably not include any array arithmetic.

- Open issues - are there other open issues that should be considered? A review of the open issues list can guide us here.
    - http://spreadsheets.google.com/ccc?key=pEI1-gPUmt2ZTw_kcCTgnKw&inv=jim_oreilly@ieee.org&t=93304845348855187 1&guest.
    - If logged into your google account, can edit. If not, can only view.

- Next Meeting 01/07/10.

For reference STIL .4 information can be found at the IEEE STIL website:
http://grouper.ieee.org/groups/1450/ (select the P1450.4 link from the table) or use the direct link
http://grouper.ieee.org/groups/1450/dot4/index.html