Spec/Category/SpecVariable handling proposals for P1450.4

In all discussions below, the term "spec variables" refer to variables defined within a spec block, and which appear in one or more category subsections of a spec block or blocks. Further, spec variables will have one or more selector subvalues (i.e., min, typ, max, or meas), as specified by IEEE 1450-1999, chapter 19 "Spec and Selector Blocks", and selected by the Category and Selector statements of the PatternExec block (IEEE 1450-1999, chapter 16) or by parameters of P1450.4 Test or Flow blocks.

As defined in IEEE 1450-1999, spec variables (used for specifying timing or levels values) are global in scope (accessible to all levels). The combination of the category name + variable name must be named unique.

Finally, even though the categories and variables are syntactically contained within a spec block, a spec block has no semantic meaning. It is not used to provide additional scoping.

1. Status quo for dot0, dot1, dot2
   a. No changes, no additions, to what's specified in STIL dot0, dot1, dot2
   b. All spec variables are global, and combination of category+variable name MUST be unique.
   c. The set of variables within a particular category can be contained in one spec block, or spread out over multiple spec blocks.
2. Add spec block name to scoping hierarchy, and allow duplication of "category+variable name" in multiple (uniquely named) spec blocks.
   a. All spec variables and categories are scoped to a particular spec block, which MUST be named in a particular context. If variables or categories are NOT named in the currently active spec block(s), they cannot be resolved. Can reuse category and variable names in different spec blocks.
   b. Note that this is a major change from existing STIL practice, and will present some issues when translating to LTX Envision (which uses a scheme very similar to #2 above). We believe that these issues can be dealt with by creating Envision spec variable names which are a concatenation of the spec block name and spec variable name. It's a bit clumsy, but it will work. When translating Envision code to STIL (if that is ever done), no problems arise, since an Envision spec variable is global in scope, and would not be reused in different spec blocks.

Because existing languages use either opt 1 or opt 4 models, I propose that we include BOTH in dot4, allowing the user to choose which model to use via a STIL Flow 2009 statement, with optional Level1 or Level2 token. To be agnostic here, there is no default – the user MUST specify which model is being used. Comments?

```
STIL 1.0 {
    ( Flow 2009 (Level1 | Level2) ; )+
}
```

Pros and Cons of Opt. 1 and Opt. 2

- Opt 1 (strict dot0 model)
    - Pros:
        - Allows direct compatibility with dot0 (and more to the point, with some testers whose software follows that model), at the expense of greater encapsulation of data that opt 2 provides (see Cons below)
        - Easier tester translation path to some testers. Allows direct translation of spec blocks and the category names within those blocks to those types of testers - with no modification of category names, a key point.
        - Provides potentially simpler path to acceptance of dot4 – users don't have to recalibrate their mental model of how spec blocks, categories, and variables are handled in STIL.
- Cons
    - Falls short in the key area of encapsulating data within the spec block - a concept that was considered in the original discussions for dot0, but not implemented.
        - In dot0, the spec block is a syntactical container only, and carries no weight with respect to scoping of categories and variables. In effect, the spec variables in dot0 are global in scope, and the combination of category name + spec variable name MUST be unique. This may make it difficult to combine spec blocks from multiple runs of tools, in which spec blocks may use the same names for categories and variables, but the values of those variables may be different.
    - Spec variables in a given category can be defined in multiple spec blocks, rather than requiring that the variables within a single category all be defined within a single spec block. Though dot0 recommends that all spec variables belonging to a given category be defined within a single spec block, it does NOT enforce that. This could result in the variables within a single category being defined in multiple spec blocks, which could be spread out in widely separated locations of the input file or input stream – a potential program maintenance headache!
- Opt. 2 (enhanced model for dot4)
    - Pros:
        - Allows for encapsulating the category/variable definitions within the spec block. All variables within a single category MUST be defined within a single spec block.
        - Makes it potentially easier to combine spec blocks from multiple runs of tools that generate STIL data. In some

cases, two different spec blocks may define a variable using the same category name, but with a different variable for the variable. If that situation occurs, some name remapping would need to be done. By allowing a category name to be reused in multiple spec blocks, such a situation is avoided.

- Would break NO existing code that follows current STIL standards rules. Dot0 already imposes a constraint of category name uniqueness (within a single file – in multiple files, that could be an issue – see discussion in bullet point above). Likewise, if translating existing code from testers that adhere to the dot0 model back to STIL, there is also no possibility of breaking that existing code (since they, too, impose the constraint of category name uniqueness).

- Cons
  - Some issues in translating to tester software that follows the dot0 model may occur. In these cases, there are two solutions:
    - User level – insure that the multiple spec blocks that occur within a program do not reuse category names. In this case, the spec, category, and variable names of the STIL program can be translated directly (with no renaming) to the target tester software.
    - Tester translator level. If the above constraint is not met, a simple name remapping by the translation tool can be done. The key is to make the category name unique – which can be done in any number of ways that still retain the intent of the original STIL program. Two such mechanisms are:
      - Create a new category name based on the combination of spec block name + category name.
      - Append a alpha or numeric tag to the category name (such as "category_name_1") to the first occurrence of a specific category name. Subsequent occurrences would then use "category_name_2, category_name_3, etc.)
    - Note that there is already precedent in the dot4 standard for such renaming when translating to target testers – if multiple bin axes are used in the SoftBinDefs and HardBinDefs. In those cases, when translating to almost all existing systems, the hierarchy of the soft and hard bin defs would need to be flattened, with new bin names based on the combination of axis name and bin name. This situation is not completely identical, since the bin

constructs are newly introduced in dot4, whereas in the case of possible renaming of category names, that operation would introduce a change to an existing part of the STIL standard.

- o Mixed opt. 1 and opt 2 model (user selectable)
  - Pros:
    - Allows both models, with the user specifying which to use.
    - Could speed acceptance of STIL dot4 by allowing an entry path which requires no changes to user's mindset, while allowing a more advanced capability if the user chooses to take advantage of it.
    - Provides a minor short-term benefit.
  - Cons:
    - Could add confusion to the standard, with different users taking advantage of different features of STIL. There's already enough of that, and it's felt that if we can avoid introducing more "dialects" without undue burden on users, we should do so.
    - Would introduce substantial complexity in both parsers, and in a reader's ability to understand which set of rules would govern the behavior.
      - If a tester were to use dot4 as a runtime language, and if both the dot0 and dot4 rules could be allowed (with the program code specifying which was in effect), then the issue of what happens if a piece of code using the dot4 rules (which allow for repeated category names in different spec blocks, and which includes the spec blocks as part of the scoping hierarchy) were run with the rules then changed to dot0 rules (again, using an option to the STIL Flow 2009; statement) would have to be addressed – and it's felt by members of the WG (who all have experience in both writing and using such software) that such a situation, could be solved, but would introduce substantial complexity.

- Therefore, to summarize, the decision at this point is to allow the dot4 model only. This provides a cleaner long-term path, and adds minimal complexity to STIL translators which would have to take code with the dot4 spec/category/variable hierarchy and translate it (using simple name remapping schemes) to target testers which adhere to the dot0 model.

- The changes to the spec/category/variable hierarchy from dot0 to dot4 can be summarized as follows:
  - o No distribution of spec variables among multiple spec blocks. All variables belonging to a single category MUST be contained within a

single spec block. NOTE: This is a recommended, though not enforced, practice even in dot0.

o   A category name can be reused in multiple spec blocks – in contrast to dot0, which specifies that a category name MUST be unique. That constraint arises because in dot0, the spec block has no semantic weight – the spec variables (unique in category_name+spec_name), are global in scope. Dot4, on the other hand, does give semantic weight to the spec block name. All variables must be unique in the spec_block_name+category_name+variable_name – and such a variable is global in scope, as would be the truncated form (minus the spec block name) in dot0.

o   Just as spec variables in dot0 can be referred to by the form

```
category_name.spec_name.selector_value
```

(i.e., `my_category.my_var.[Min | Typ | Max | Meas ]`),

so can spec variables be specified in dot 4 by the form

```
spec_block_name.category_name.spec_name.selector_value
```

Comparison of existing languages with respect to the above proposals

LTX Envision:
Directly compatible with opt. 1. If opt. 4 is used, some name translation (most likely by appending the spec block name, and perhaps the category name, to the spec variable) must be used to avoid potential duplication of spec variable names, which is not allowed currently in Envision. (See appendix A below). Referring to the discussion in appendix A (in which we're discussing a spec variable named "Per", to be used for setting the tester period), the key phrase is "Since all parameter names are at global scope, there can only be one instance of Per in the program."

Teradyne IGXL:
Directly compatible with opt. 1. If opt. 4 is used, some name translation (most likely by appending the spec block name, and perhaps the category name, to the spec variable) must be used to avoid potential duplication of spec variable names, which is not allowed currently in IG-XL. (see Appendix B below). See the key sections regarding multiple spec sheets (the IG-XL equivalent of a spec block).

> **Multiple Specs Sheets Per Job**
> A single job can use multiple AC or DC Specs sheets. Sheets in a job can refer to any of the specs defined on any of the specs sheets in the job. Use the Job List sheet to specify the specs sheets to be used in the job.
> **Note:**
> If there are multiple AC or DC Specs sheets in a job, all spec name symbols on the sheets must be **unique**. Duplicate names among the sheets are not allowed. If multiple specs sheets in a job define the same spec symbol name, a validation error results.

These comments regarding allowing multiple spec sheets in a job (the IG-XL equivalent of a STIL TestProgram block) refer to the latest version (7.1) of IG-XL. Earlier versions allowed only one DC spec sheet and one AC spec sheet per job (and with the constraint of unique spec sheet variable names).

Advantest OTPL:
Directly compatible with opt. 4.

Schlumberger/NPTest/Credence/LTXC ASAP, Sapphire, and Diamond:
Directly compatible with opt. 4.

Teradyne IG973/IG971/IG953: Follows option 1 model. Duplicate spec names within a single test program not allowed.

Teradyne Image: procedural only, not block-structured. C-language conventions regarding variable names apply.

Appendix A

Hi Jim,

Comments in-line below.

Regards,
Gary

Hi, Gary,

Another two questions (in addition to those I sent the other day).

Is it required that any spec variable which appears in any category in a specific spec block also appear in the ParamGlobals section of that spec block? (It appears to be the case, based on this example).

In the following example, the Flownode Node[17] specifies ac spec "ACSpec.Cat16mhz" in the spec pairs section, and executes functional test GrossFunctional. Test GrossFunctional specfies pattern Functional_PatSeq; pattern Functional_PatSeq specifies waveform table LS245pats; waveform table LS245pats specifies the period using the value of the spec variable "Per". This all works, since ac spec table "ACSpec" has a parameter called "Per".

What happens if, instead, Node[17] specified ac spec block "ACSpec_2.Cat16mhz_2"? That spec table does NOT have a spec variable called "Per". Would Envision resolve this by looking in spec tables other than those specfied in the SpecPairs section of the flow node? (i.e., even though the test specified to use ACSpec_2.Cat16mhz_2, would Envision look there, not find it, then start searching other spec blocks)? If not, then in my view, the spec variables require the spec name to determine where to look to resolve the spec name to its value. Is it required to have the SpecPairs section in the flow node? If not, then I suppose that Envision *could* look in all spec blocks to help resolve the spec variable name where it is used (in the waveform table, in this case). Can you determine how this would work?

Since all parameter names are at global scope, there can only be one instance of Per in the program. The SpecPairs syntax at the flow node level really only specifies the category to use in each of the Spec tables listed, not the set of parameters available to the test object. So, in your example, the parameter Per would be looked up by name globally, and since it was associated with ACSpec, it's value would be found there. Since ACSpec did not have a category selected at the FlowNode level, the last one selected for it (in a previous FlowNode) would be used when resolving the value of Per.

Get back to me when you can (but, please, don't wait *too* long).

Thanks,

Jim

Multiple Spec objects may be defined for a test program with the requirement that all parameter names be unique across all Spec objects. Categories may be different across Spec objects, allowing for example a set of categories for timing parameters, and a separate set of categories for DC parameters.

```
Spec AC_Specs {
   Category Cat16mhz {
        Freq = "16MHz";
        Per = "1/Freq";
    }
    Category Cat25mhz {
        Freq = "25MHz";
        Per = "1/Freq";
    }
    ParamGlobals {
       Freq { Type = Hz; Comment = "Device Operating Frequency"; }
       Per { Type = S; Comment = "Device Period"; }
    }
}
Spec AC_Specs_2 {
   Category Cat16mhz_2 {
        Freq_2 = "16MHz";
        Per_2 = "1/Freq_2";
    }
    Category Cat25mhz_2 {
        Freq_2 = "25MHz";
        Per_2 = "1/Freq_2";
    }
    ParamGlobals {
       Freq_2 { Type = Hz; Comment = "Device Operating
Frequency"; }
       Per_2 { Type = S; Comment = "Device Period"; }
    }
}

Mask ACRelaxed {
   Freq = Typ;
    Per = Typ;
}

Mask ACRelaxed_2 {
   Freq_2 = Typ;
    Per_2 = Typ;
}
```

```
    WaveformTable LS245pats LS245pats {
        Period "Per";
        Cell "all_pins" 0/1 all_pins_01 {
            Data 6/7;
        }
        Cell "all_pins" r/R all_pins_rR {
            Data 6/7;
        }
    }


    PatternSequence Functional_PatSeq {
        Thread[0] = Single;
        Zipper = Zipper {
            Row { LS245pats, Disable = { Bus_Actice_to_Z,
Bus_Control_pins } }
            Row { LS245pats, Enable_ = { Bus_Z_to_Active,
Bus_Control_pins } }
            Row { LS245pats, ReadWrite = { Bus_Active } }
            Row { LS245pats, Reset = { Bus_reset, Bus_Control_pins } }
        }
        TransactionCount = 2;
        BasePeriodResolution = Expr { Type = s; }
    }


    Test Gross_Functional {
        FinderFilter = "";
        Result = Expr { String = "#"; Mode = Output; }
        LoopExpr = Expr { String = "FALSE"; }
        LoopNotify = True;
        LoopDepth = Outside;
        Mask[0] = DC_Easy;
        Mask[1] = ACRelaxed;
        Entry[0] = Functional_Levels;
        Entry[1] = Functional_PatSeq;
        PortExpr[0] = Expr { String = ".Result = tm_rslt:PASS"; }
        PortExpr[1] = Expr { String = ".Result = tm_rslt:FAIL"; }
    Title[0] = FuncTest;
        TestMethod = Ftest;
        Test_enable[0] = Expr { String = "Seq_en:DEFAULT_EXECUTION"; }
        Test_pins[0] = Expr { String = "all_pins"; }
        Test_result[0] = Expr { String = "#"; Mode = Output; }
        Pattern_index[0] = Expr { String =
"Functional_PatSeq.Thread.Single"; }
        Simulate_results[0] = Expr { String = "tm_rslt:FLOW_SIM_OFF"; }
    }


    Node[17] =     FlowNode_  {
        XCoord = (356,81);
        Port[0] {
            To = 7;
            UIFPort = 88;
        }
        Port[1] {
            To = 18;
            UIFPort = 178;
        }
```

```
        UIFInfo = 272;
        SpecPairs {
            ACSpec = Expr { String = "ACSpec.Cat16mhz"; Type =
INTEGER; }
            DC_Spec = Expr { String = "DC_Spec.T_25dC"; Type =
INTEGER; }
        }
        TestId = "42";
        PortSelect = "Gross_Functional.ExecResult";
        Exec = Gross_Functional;
        EVO_Calibration = envstuE_Fcal;
    }
```

----- Original Message -----
From: "Gary Murray" <gary_murray@ltx.com>
To: <jim_oreilly@ieee.org>
Sent: Tuesday, February 24, 2009 9:57 AM
Subject: Revised Spec syntax

```
>   Spec AC_Specs {
>   Category Cat16mhz {
>       tcyc1 = "60ns";
>       tcyc = "60nS";
>       fmax = "floor(1/(tcyc.Typ.Cat16mhz*1000000)) * 1000000";
>       tsys = "10nS";
>       tsmp = "10nS";
>       trd = "10nS";
>       tds.Min = "12nS";
>       tdh.Min = "4nS";
>       tdval.Max = "10nS";
>       tddis.Max = "0S";
>       texs.Min = "12nS";
>       texh.Min = "4nS";
>       truns.Min = "12nS";
>       trunh.Min = "0nS";
>       tfpcond.Max = "20nS";
>       tfpbusy.Max = "25nS";
>       tfpint.Max = "20nS";
>       t0 = "10nS";
>       iogb = "8nS";
>       FpSys_ph = "-tcyc/4";
>       FpSys.Min = "0nS";
>       FpSys.Typ = "5nS";
>       FpSys.Max = "15nS";
>       Half_cyc = "tcyc/2";
>       SpeedBin = "4";
>       TrigLoc = "20";
>       tdval1.Max = "10nS";
>   }
>   Category Cat25mhz {
>       tcyc1 = "40ns";
>       tcyc = "1/fmax";
>       fmax = "25MHz";
>       tsys = "3.5nS/10nS*tcyc";
```

```
>        tsmp = "3.8nS/10nS*tcyc";
>        trd = "3.3nS/10nS*tcyc";
>        tds.Min = "6.1nS/10nS*tcyc";
>        tdh.Min = "1nS/10nS*tcyc";
>        tdval.Max = "4nS/10nS*tcyc";
>        tddis.Max = "0nS/10nS*tcyc";
>        texs.Min = "5.8nS/10nS*tcyc";
>        texh.Min = "1nS/10nS*tcyc";
>        truns.Min = "7.5nS/10nS*tcyc";
>        trunh.Min = "2nS/10nS*tcyc";
>        tfpcond.Max = "5nS/10nS*tcyc";
>        tfpbusy.Max = "5nS/10nS*tcyc";
>        tfpint.Max = "5nS/10nS*tcyc";
>        t0 = "10nS";
>        iogb = "tcyc/15";
>        FpSys_ph = "-tcyc/4";
>        FpSys.Min = "0nS";
>        FpSys.Typ = "5nS";
>        FpSys.Max = "15nS";
>        Half_cyc = "tcyc/2";
>        SpeedBin = "1";
>        TrigLoc = "20";
>        tdval1.Max = "4nS/10nS*tcyc";
>    }
>    Category Cat33mhz {
>        tcyc1 = "1/fmax";
>        tcyc = "1/fmax";
>        fmax = "33MHz";
>        tsys = "3.5nS/10nS*tcyc";
>        tsmp = "3.8nS/10nS*tcyc";
>        trd = "3.3nS/10nS*tcyc";
>        tds.Min = "6.1nS/10nS*tcyc";
>        tdh.Min = "1nS/10nS*tcyc";
>        tdval.Max = "4nS/10nS*tcyc";
>        tddis.Max = "0nS/10nS*tcyc";
>        texs.Min = "5.8nS/10nS*tcyc";
>        texh.Min = "1nS/10nS*tcyc";
>        truns.Min = "7.5nS/10nS*tcyc";
>        trunh.Min = "2nS/10nS*tcyc";
>        tfpcond.Max = "5nS/10nS*tcyc";
>        tfpbusy.Max = "5nS/10nS*tcyc";
>        tfpint.Max = "5nS/10nS*tcyc";
>        t0 = "10nS";
>        iogb = "tcyc/15";
>        FpSys_ph = "-tcyc/4";
>        FpSys.Min = "0nS";
>        FpSys.Typ = "5nS";
>        FpSys.Max = "15nS";
>        Half_cyc = "tcyc/2";
>        SpeedBin = "1";
>        TrigLoc = "20";
>        tdval1.Max = "4nS/10nS*tcyc";
>    }
>    Category Cat40mhz {
>        tcyc1 = "25ns";
>        tcyc = "1/fmax";
```

```
>        fmax = "40MHz";
>        tsys = "3.5nS/10nS*tcyc";
>        tsmp = "3.8nS/10nS*tcyc";
>        trd = "3.3nS/10nS*tcyc";
>        tds.Min = "6.1nS/10nS*tcyc";
>        tdh.Min = "1nS/10nS*tcyc";
>        tdval.Max = "5nS/10nS*tcyc";
>        tddis.Max = "0nS/10nS*tcyc";
>        texs.Min = "5.8nS/10nS*tcyc";
>        texh.Min = "1nS/10nS*tcyc";
>        truns.Min = "7.5nS/10nS*tcyc";
>        trunh.Min = "2nS/10nS*tcyc";
>        tfpcond.Max = "5nS/10nS*tcyc";
>        tfpbusy.Max = "5nS/10nS*tcyc";
>        tfpint.Max = "5nS/10nS*tcyc";
>        t0 = "10nS";
>        iogb = "tcyc/15";
>        FpSys_ph = "-tcyc/4";
>        FpSys.Min = "0nS";
>        FpSys.Typ = "5nS";
>        FpSys.Max = "15nS";
>        Half_cyc = "tcyc/2";
>        SpeedBin = "1";
>        TrigLoc = "20";
>        tdval1.Max = "5nS/10nS*tcyc";
>    }
>    Category Cat50mhz {
>        tcyc1 = "20ns";
>        tcyc = "20nS";
>        fmax = "floor(1/(tcyc.Typ.Cat50mhz*1000000)) * 1000000";
>        tsys = "7nS";
>        tsmp = "7.6nS";
>        trd = "6.6nS";
>        tds.Min = "12.2nS";
>        tdh.Min = "2nS";
>        tdval.Max = "-4nS";
>        tddis.Max = "0nS";
>        texs.Min = "11.6nS";
>        texh.Min = "2nS";
>        truns.Min = "15nS";
>        trunh.Min = "4nS";
>        tfpcond.Max = "10nS";
>        tfpbusy.Max = "10nS";
>        tfpint.Max = "10nS";
>        t0 = "10nS";
>        iogb = "1.3nS";
>        FpSys_ph = "-5nS";
>        FpSys.Min = "0nS";
>        FpSys.Typ = "5nS";
>        FpSys.Max = "15nS";
>        Half_cyc = "tcyc/2";
>        SpeedBin = "4";
>        TrigLoc = "20";
>        tdval1.Max = "4ns";
>    }
>    Category Cat67mhz {
```

```
>       tcyc1 = "15ns";
>       tcyc = "15nS";
>       fmax = "floor(1/(tcyc.Typ.Cat67mhz*1000000)) * 1000000";
>       tsys = "5.25nS";
>       tsmp = "5.7nS";
>       trd = "4.95nS";
>       tds.Min = "9.15nS";
>       tdh.Min = "1.5nS";
>       tdval.Max = "3nS";
>       tddis.Max = "0nS";
>       texs.Min = "8.7nS";
>       texh.Min = "1.5nS";
>       truns.Min = "11.25nS";
>       trunh.Min = "3nS";
>       tfpcond.Max = "7.5nS";
>       tfpbusy.Max = "7.5nS";
>       tfpint.Max = "7.5nS";
>       t0 = "10nS";
>       iogb = "1nS";
>       FpSys_ph = "-3.75nS";
>       FpSys.Min = "0nS";
>       FpSys.Typ = "5nS";
>       FpSys.Max = "15nS";
>       Half_cyc = "tcyc/2";
>       SpeedBin = "3";
>       TrigLoc = "20";
>       tdval1.Max = "3ns";
>    }
>    Category Cat80mhz {
>       tcyc1 = "12.5ns";
>       tcyc = "12.5nS";
>       fmax = "floor(1/(tcyc.Typ.Cat80mhz*1000000)) * 1000000";
>       tsys = "4.375nS";
>       tsmp = "4.75nS";
>       trd = "4.125nS";
>       tds.Min = "7.625nS";
>       tdh.Min = "1.25nS";
>       tdval.Max = "2.5nS";
>       tddis.Max = "0nS";
>       texs.Min = "7.25nS";
>       texh.Min = "1.25nS";
>       truns.Min = "9.375nS";
>       trunh.Min = "2.5nS";
>       tfpcond.Max = "6.25nS";
>       tfpbusy.Max = "6.25nS";
>       tfpint.Max = "6.25nS";
>       t0 = "10nS";
>       iogb = "833pS";
>       FpSys_ph = "-3.125nS";
>       FpSys.Min = "0nS";
>       FpSys.Typ = "5nS";
>       FpSys.Max = "15nS";
>       Half_cyc = "tcyc/2";
>       SpeedBin = "2";
>       TrigLoc = "20";
>       tdval1.Max = "0ns";
```

```
>    }
>    Category Shmoo {
>        tcyc1 = "20nS";
>        tcyc.Min = "8nS";
>        tcyc.Typ = "20nS";
>        tcyc.Max = "tcyc1";
>        fmax = "1/tcyc";
>        tsys = "3.5nS/10nS*tcyc";
>        tsmp = "3.8nS/10nS*tcyc";
>        trd = "3.3nS/10nS*tcyc";
>        tds.Min = "6.1nS/10nS*tcyc";
>        tdh.Min = "1.0nS/10nS*tcyc";
>        tdval.Max = "3.5nS/10nS*tcyc";
>        tddis.Max = "0nS/10nS*tcyc";
>        texs.Min = "5.8nS/10nS*tcyc";
>        texh.Min = "1nS/10nS*tcyc";
>        truns.Min = "7.5nS/10nS*tcyc";
>        trunh.Min = "2nS/10nS*tcyc";
>        tfpcond.Max = "5nS/10nS*tcyc";
>        tfpbusy.Max = "5nS/10nS*tcyc";
>        tfpint.Max = "5nS/10nS*tcyc";
>        t0 = "tcyc/2";
>        iogb = "tcyc/15";
>        FpSys_ph = "-tcyc/4";
>        FpSys.Min = "0nS";
>        FpSys.Typ = "5nS";
>        FpSys.Max = "15nS";
>        Half_cyc = "tcyc/2";
>        SpeedBin = "1";
>        TrigLoc = "3900";
>        tdval1.Max = "3.5nS/10nS*tcyc";
>    }
>    ParamGlobals {
>        tcyc1 { Type = s; Comment = "Characterization DUT Period"; }
>        tcyc { Type = s; Comment = "DUT Period"; }
>        fmax { Type = Hz; Comment = "Device Maximum Frequency"; }
>        tsys { Type = s; Comment = "Clk2xSys Phase"; }
>        tsmp { Type = s; Comment = "Clk2xSmp Phase"; }
>        trd { Type = s; Comment = "Clk2xRd Phase"; }
>        tds { Type = s; Comment = "Databus Setup Time"; }
>        tdh { Type = s; Comment = "Databus Hold Time"; }
>        tdval { Type = s; Comment = "Databus\/Tag Valid Time"; }
>        tddis { Type = s; Comment = "Databus Disable"; }
>        texs { Type = s; Comment = "Exception Setup Time"; }
>        texh { Type = s; Comment = "Exception Hold Time"; }
>        truns { Type = s; Comment = "Run Setup Time"; }
>        trunh { Type = s; Comment = "Run Hold Time"; }
>        tfpcond { Type = s; Comment = "FpCond Valid Time"; }
>        tfpbusy { Type = s; Comment = "FpBusy Valid Time"; }
>        tfpint { Type = s; Comment = "FpInt Valid Time"; }
>        t0 { Type = s; Comment = "Tester T0 Offset"; }
>        iogb { Type = s; Comment = "I\/O Guardband"; }
>        FpSys_ph { Type = s; Comment = "FpSysOut Strobe Phase"; }
>        FpSys { Type = s; Comment = "FpSysOut Prop Delay"; }
>        Half_cyc { Type = s; Comment = "Clock Width"; }
>        SpeedBin { Type = SCALAR; }
```

```
>          TrigLoc { Type = SCALAR; }
>          tdval1 { Type = s; }
>    }
> }
>
```

Appendix B

Hi Jim,

As far as the UltraFlex goes you can have Multiple Ac & DC Specsheets but if they are used in the Same Active Job, you cannot have
Duplicate spec names.

From Version 7.10 Documentation:

Multiple Specs Sheets Per Job

A single job can use multiple AC or DC Specs sheets. Sheets in a job can refer to any of the specs defined on any of the specs sheets in the job. Use the Job List sheet to specify the specs sheets to be used in the job.

Note:

If there are multiple AC or DC Specs sheets in a job, all spec name symbols on the sheets must be **unique**. Duplicate names among the sheets are not allowed. If multiple specs sheets in a job define the same spec symbol name, a validation error results.

Category and selector names can be shared among the multiple specs sheets in a job. The uniqueness requirement applies only to the spec symbol names.

A workbook can contain any number of AC and DC Specs sheets. If the sheets are not used in the same job, they can contain duplicate spec symbol names. It is assumed that specs sheets that have duplicate spec symbol names will be used in different jobs.

I haven't used the Flex for a while but I did a quick check of the help and found this: (Sounds more limited than the UltraFlex).

----- Original Message -----
From: Jim O'Reilly
To: Phil Bergeron
Sent: Thursday, August 20, 2009 1:16 PM
Subject: IGXL AC, DC Spec sheets

Phil,

Good to talk with ya - enjoy the time off while you can.

The two issues I'd like to know about IGXL and AC/DC spec sheets are:

1. Can you have more than one AC specsheet (or more than one DC spec sheet) in a program?
2. If multiple AC or multiple DC specsheets are allowed, can variable names (symbols, in IGXL terminology - see below) be reused in more than one AC (or DC) specsheet?
3. If either of the above (or both) are true, what versions of the software first allow them?

At your convenience, let me know what you can find out.

Thanks,

Jim

AC Specs

| Symbol | Value | Selector | | Commercial | |
		Name	Val	Typ	Comment
tplh	30.E-09	max_AC	Max	10.E-09	
tphl	40.E-09	max_AC	Max	10.E-09	
tpzh	40.E-09	max_AC	Max	10.E-09	
tpzl	40.E-09	max_AC	Max	10.E-09	
tphz	40.E-09	max_AC	Max	10.E-09	
tplz	60.E-09	max_AC	Max	15.E-09	

Variable scoping rules.

1. Within a test or flow, local scope consists of local variables and parameters. Each name must be unique in that context.
2. Variables, constants, and parameters (including spec variables) declared in the local scope will hide any variables, constants, or parameters of the same name which are declared in the global scope. Global scope includes specs declared in Spec blocks, global variables declared in named or unnamed variables blocks at the global scope, TestProgram variables, and all Test or Flow objects (created by instantiating TestTypes or FlowTypes).
3. To access the variables in the upper level scope, the complete variable or spec access syntax can be used.