STIL P1450.4 Binning

Summarized below are the key points of binning from several different commercially available testers.

Agilent 93000 Binning:

Bin architecture:

A bin is defined as an unnamed list of 7 elements

- A 2 character major bin string
- An minor bin string // over_on | not_over_on
- An OOC rule
- Pass/Fail status // good | bad
- Reprobe flag // reprobe | noreprobe
- Color // 0-7 or black | white | red | yellow | green | cyan | blue | magenta
- Physical (hard) bin number // 0-999
- Override flag

The OOC element is optional, so for purposes of this discussion, we'll ignore that element.

There are two types of test execution flow-nodes

- run
- run-and-branch

and two types of bin flow-nodes:

- Stop bin (can be good or bad)
- Otherwise bin (if not defined by user, a system default is used).

The display icons for test execution and bin flow-nodes are shown below in Fig. 1.

Bin map(s):

List of bin structures described above. Only one bin map available in the program. As far as I know, there are no limits to the number of bins a binmap can contain - though, with the exception of softbin_string, the range of values each structure member can take on would seem to limit the number of permutations. Nonetheless, as each unique bin is defined (and used), it is simply placed in the binlist.

Containment hierarchy (relationship to flow):

- A bin is essentially a terminal flow-node.
- A flow does not need to contain bin assignment flow-nodes; if not, a default "otherwise" bin is predefined.
- No other flow-nodes can follow a bin flow-node (i.e, a bin node has an input but no output).
- The input to a bin flow-node is from the output of one (and only one) flow-node (this is really more a function of the testflow, rather than the binning – the bin just happens to be a flow-node).
- Binning is tied to a bin flow-node. Bin assignment (via a bin flow-node) selects one of N bins in the binlist (i.e., hard binning and soft binning happen together, not separately).
- Two types of test execution flow-nodes, each of which can be followed by another flow-node, or a bin node.
    - run
    - run-and-branch

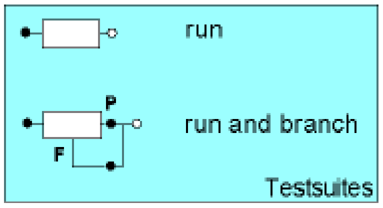
Bin/stop relationship:

Two modes:

- Stop on fail:
  Disables/powers down, skips over remaining tests and testblocks, sets bin based on the bin flow-node connected to the the fail port of the failing test. If no bin flow-node is present, the flow continues to the next execute flow-node. At the end of the test flow, if no bin has been assigned, the otherwise bin is assigned.
- Override on.
  Flow continues past the fail bin to the flow-node immediately following the failing test (as indicated by a dotted line in the testflow – see Fig. 2 below). If there are no more tests after the stop bin, the otherwise bin is assigned.

Bin strategies:

- Bin flow nodes are used in testflow:
  Bin based on the bin assigned from a bin-flow-node (assign fail-bin, pass, bin, or otherwise bin, depending on pass/fail results and overon setting)
- Bin flow nodes are not used in testflow:
  Only the otherwise bin is used, and always assigned at the end of the testflow.



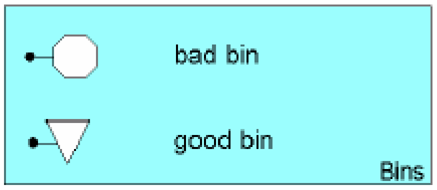


Fig. 1 Agilent 93000 - Display icons for test execution and bin flow-nodes

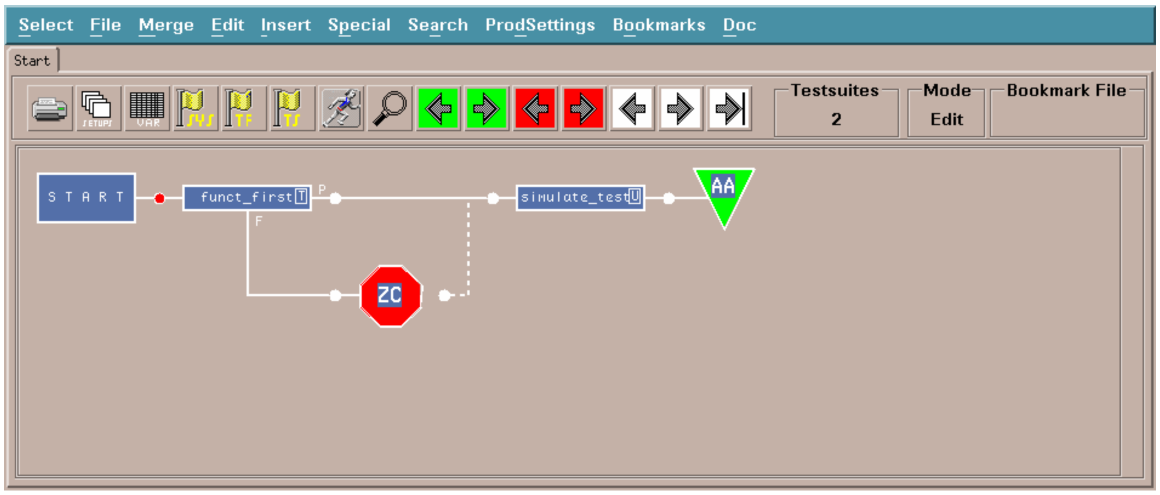


Fig. 2 Agilent 93000 - Graphical representation of test flow including pass/fail bins and run/run-and-branch flow nodes

```
test_flow

  run_and_branch(funct_first)    then
  {
  }
  else
  {
      stop_bin "ZC", "fp_cont_fail", , bad,noreprobe,red, 7, not_over_on;  // Fail bin
  }
  run(simulate_test_time__2_);
  stop_bin "AA", "p_good_part", , good,noreprobe,green, 1, not_over_on;  // pass bin
end
----------------------------------------------------------------
binning

otherwise bin = "DB", "otherwise_bin", , bad,noreprobe,cyan, 99, not_over_on;  // otherwise bin
end
```

Fig. 3 Agilent 93000 - Text representation of graphical test flow shown in Fig. 2

Retest handling:
If a part is binned into a reprobe bin, it is retested once. The specific action taken prior to reprobe is determined by a REPROBE_ACTION statement which immediately follows the device test loop in the application model , as shown in Fig. 4 below:

```
{ lot:
   { WAFER [(wafermap),({wafer_id})] wafer:
      ACTIONS
         * wafer_id = AUTOINCREMENT;
         { DEVICE die:
            {
               DEVICE_TEST
            }  // End device test block
            REPROBE_ACTION = PROB_HND_CALL(chuck_down_then_up);
         }  // End die test
   }  // End wafer loop
}  // End lot test
```

Fig. 4 Agilent 93000 – Example of an application model which specifies a REPROBE_ACTION

Depending on the intelligence of the handling equipment, and the degree of control available to reprobe a die (typically, by lowering and then re-raising the wafer prober chuck) or to replunge a part in an autohandler, it may be necessary to have test program agree on which bins are reprobe bins. If it is necessary, and the tester and handling equipment don’t agree on the reprobe bins, the parts will not be binned correctly.

Teradyne Binning (Not sure which system – J953 or earlier?)

Bin Architecture:

A bin is defined as a structure containing softbin number, softbin string, hardbin number, hardbin string, and a pass/fail/error classification, e.g.,

1 "GRADE_1" hbin =1 "GRADE_1" pass,

Bin maps:

Array of 256 instances of bin struct described above in "Bin Architecture". Only one bin map allowed per program.

Containment hierarchy (relationship to flow):

soft binning is tied to a test, however, a test may be an otherwise empty container. Soft binning is separate from hard binning.

Bin/stop relationship:

3 modes

a) stop on fail: disables and powers down affected site, skips over remaining tests and testblocks, and proceeds to hardware binning
b) continue on fail: registers the first bin failed, no intra-chiptest count (maintains chip to chip summary, e.g. failed bin 15 twice)
c) continue on error: not sure what happens

Bin strategies: there are 3

a) pass binning: set soft bin on test pass
b) fail binning: set soft bin on test fail
c) disqualify binning: unset soft bin(s) on test fail

Credence Binning (Not sure which system)

Bin Architecture:

A bin is defined as a structure containing softbin number, and hardbin number.

Bin maps:

Array of 128 instances of bin struct described above in "Bin Architecture". Only one bin map allowed per program.

Containment hierarchy (relationship to flow):

soft binning is tied to a test, however, a test may be an otherwise empty container. Soft binning is separate from hard binning.

Bin/stop relationship:

there is 1 mode, stop on fail. There is a per test ignore fail option but then no binning takes place.

Bin strategies: there are 2

a) fail binning: set soft bin on test fail
b) unconditional binning: set soft bin unconditionally

Schlumberger ITS9000

Bin Architecture:

A bin is defined as a structure containing three elements

- virtual (soft) bin number
  The virtual bin number will be mapped to a hardware bin number in the binmap. The virtual bin number is used to set the working value of the virtual bin number as the flow passes through the port of a flow node. The soft bin number or a device is usually set to the value of the working virtual bin number at the end of test.
- Bit bin number.
  Each bin object has a bit bin number (which normally defaults to 0). The bit bin number is used when it is necessary to calculate the soft bin based on the path of the flow through the test. At key places in the test flow, the bit bin number is assigned to a bit mask. As the flow passes through a flow node with a bit bin number assigned, this mask is logically ored into a working bit bin mask. At the end of test, the working value of the bit bin number is added to the working virtual bin number to calculate the soft bin number for the device. The final soft-bin number is then mapped to a hardware bin in the bin map.
- Flow id bit number
  The Flow id is used when it is necessary to replace the normal binning mechanism with a user defined mechanism. To track the flow, a unique Flow Id Bit Number [0 .. 4095] set at the input or output ports of each flow node. As the flow passes through a port of a flow now with a Flow id Bit number assigned, a bit is ored into into the Flow id bit array (4096 bits) at the bit number specified. This array can be access by a user routine at end of test to override the normal binning mechanism.

Bin maps:

The bin map has two sections – a software bin section and a hardware bin section.

- Hardware bin section
  - Defines the available hardware bins, including
  - Hardware bin name
  - PASS or FAIL
  - Reprobe this bin or not (if so, the maximum allowable reprobe count)
  - bin number (to be sent to the prober or handler).
- Software bin section
  - Assigns a name to each soft bin number (specified in each bin object).
  - Maps each soft bin to a hardware bin (any number of soft bins can be mapped to a single hard bin).

Multiple binmaps can be used in the test program. Only one at a time, of course, can be active.

Containment hierarchy (relationship to flow):

- A bin can be attached to either the input or output of a testflow node.
- The virtual bin number and bit bin number of each bin must be selected so that soft bin calculated at the end of test matches a soft bin in the selected bit map.
- When a bin is assigned at a flow-node port, only the soft-bin is assigned. At the end of the flow, the most-recently-assigned soft bin is mapped to a hard bin via the currently-selected bin map.
- A flow does not need to contain a bin assignment or a bin map to run. If no binmap exists or no bin assignment is done in the flow, then the test program's "current bin" (i.e., the most-recently assigned bin) is a null bin object.
- A test program can have more than one bin map. However the only one bin map can be selected at a time.

Bin/stop relationship:

- Bin assignment and test flow stop/continue execution are separate. The soft bin assignments are made as the flow passes through flow node ports with a bin attached. The device is binned according to the soft bin assignment at the end of testing (when a terminal node is reached).

- Stop on fail is not supported. The flow always continues until it reaches a terminal flow node. In the typical case, however, a test will have one fail node and at least one pass node. The flow is constructed so that a test-flow node's failing port leads directly to a terminal node (a "Stop" segment, in ITS9000 terminology). The ITS9000 software does allow setting breakpoints or pauses at various points in the test flow (including breakpoints if a specific test fails).

- Overide on Fail is supported.

Bin strategies:

- Bin Assignment Strategy
  The virtual bin number is used to assign the working soft bin number as the flow passes through the port of a flow node. At the end of test, the last soft bin number assigned is used to look up the soft bin and hard bin information in the selected bin map.

- Path Dependent Strategy
  This strategy is used when it is necessary to calculate the soft bin based on the path of the flow through the test. At key places in the test flow, the bit bin number is assigned to a bit mask.

  As the flow passes through a flow node with a bit bin number assigned, this mask is logically ored into a working bit bin mask. At the end of test, the working value of the bit bin number is added to the working virtual bin number to calculate the soft bin number for the device. This soft bin number is used to look up the soft bin and hard bin information in the selected bin map.

- User Defined Strategy
  This strategy is used when it is necessary to replace the native binning strategy with a user defined bining strategy. On popular way to do this is by tracking the complete flow path by assigning unique Flow Id Bit Number [0 .. 4095] set at the input port or at all output ports of each flow node. As the flow passes through a port of a flow now with a Flow id Bit number assigned, a bit is ored into into the Flow id bit array (4096 bits) at the bit number specified. This array can be access by a user routine at end of test to calculate the soft bin number based on the complete path of execution.

Retest handling:
If the RETEST attribute (see Fig. 9 below) for a particular hard bin is 0, the part is not retested. If the RETEST count is N (where N > 0), the part is retested N times. The test system keeps track of how many times a part should be retested, and maintains the pass/fail/parts tested tallys correctly. In general, the tester and the handling equipment MUST be in agreement as to which hard bins get reprobed, and how many times a particular bin can be reprobed before giving up on the current die or part and moving on to the next one.

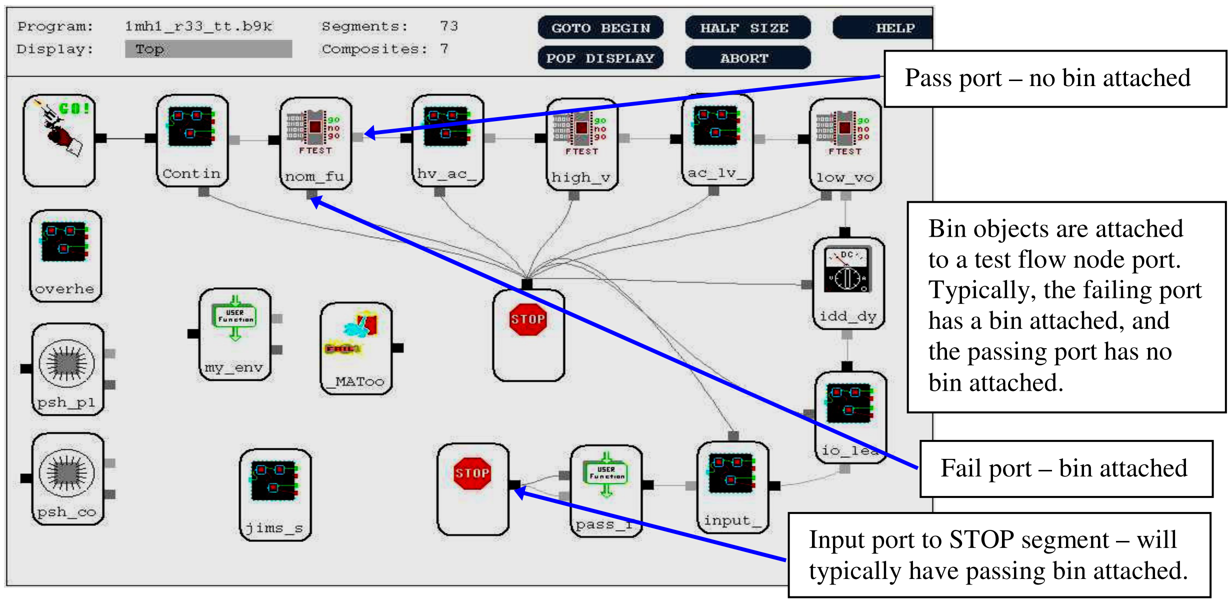
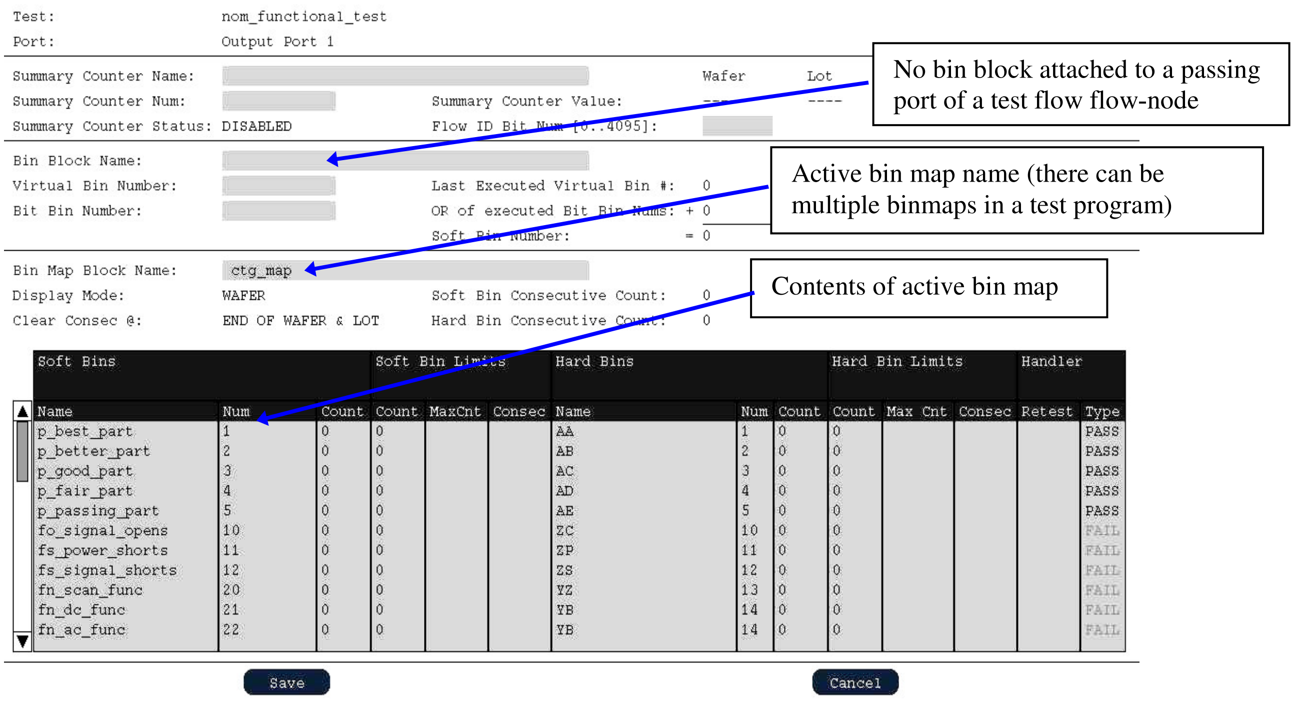

Program: 1mh1_r33_tt.b9k
Display: Top
Segments: 73
Composites: 7
GOTO BEGIN
HALF SIZE
HELP
POP DISPLAY
ABORT
GO!
Contin
nom_fu
hv_ac_
high_v
ac_lv_
low_vo
overhe
my_env
MAToo
STOP
idd_dy
psh_pl
io_lea
psh_co
jims_s
STOP
pass_1
input_
Pass port – no bin attached
Bin objects are attached to a test flow node port. Typically, the failing port has a bin attached, and the passing port has no bin attached.
Fail port – bin attached
Input port to STOP segment – will typically have passing bin attached.

Fig. 5 Schlumberger ITS9000 – Graphical representation of test flow



No bin block attached to a passing port of a test flow flow-node

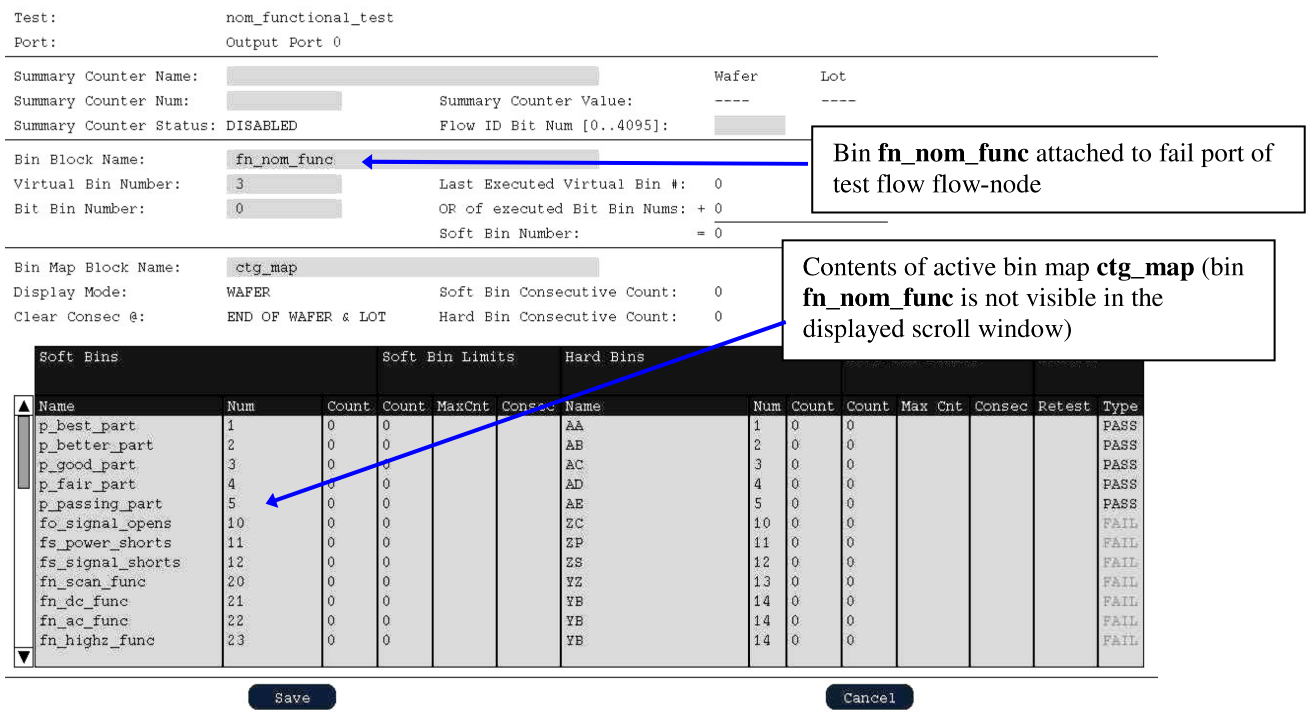
Active bin map name (there can be multiple binmaps in a test program)

Contents of active bin map

```
Test:                      nom_functional_test
Port:                      Output Port 1

Summary Counter Name:                                             Wafer      Lot
Summary Counter Num:                    Summary Counter Value:    ----       ----
Summary Counter Status: DISABLED        Flow ID Bit Num [0..4095]:

Bin Block Name:
Virtual Bin Number:                     Last Executed Virtual Bin #:    0
Bit Bin Number:                         OR of executed Bit Bin Nums: +  0
                                        Soft Bin Number:             =  0

Bin Map Block Name:     ctg_map
Display Mode:           WAFER           Soft Bin Consecutive Count:     0
Clear Consec @:         END OF WAFER & LOT   Hard Bin Consecutive Count:  0
```

Soft Bins			Soft Bin Limits			Hard Bins			Hard Bin Limits			Handler	
Name	Num	Count	Count	MaxCnt	Consec	Name	Num	Count	Count	Max Cnt	Consec	Retest	Type
p_best_part	1	0	0			AA	1	0	0				PASS
p_better_part	2	0	0			AB	2	0	0				PASS
p_good_part	3	0	0			AC	3	0	0				PASS
p_fair_part	4	0	0			AD	4	0	0				PASS
p_passing_part	5	0	0			AE	5	0	0				PASS
fo_signal_opens	10	0	0			ZC	10	0	0				FAIL
fs_power_shorts	11	0	0			ZP	11	0	0				FAIL
fs_signal_shorts	12	0	0			ZS	12	0	0				FAIL
fn_scan_func	20	0	0			YZ	13	0	0				FAIL
fn_dc_func	21	0	0			YB	14	0	0				FAIL
fn_ac_func	22	0	0			YB	14	0	0				FAIL

Fig. 6 Schlumberger ITS9000 – Bin/Binmap window opened at passing port – no bin attached



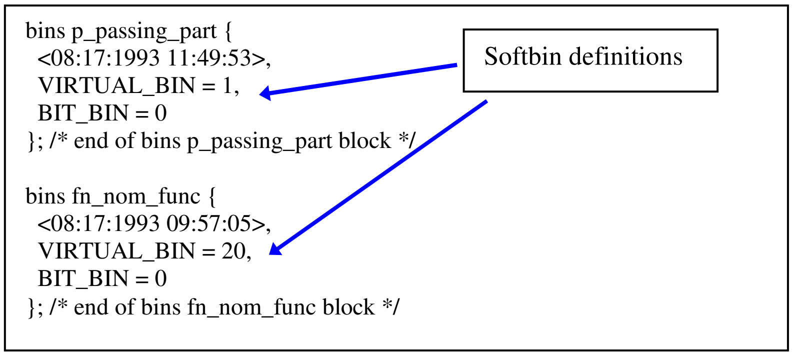
Bin **fn_nom_func** attached to fail port of test flow flow-node

Contents of active bin map **ctg_map** (bin **fn_nom_func** is not visible in the displayed scroll window)

```
Test:                      nom_functional_test
Port:                      Output Port 0

Summary Counter Name:                                             Wafer      Lot
Summary Counter Num:                    Summary Counter Value:    ----       ----
Summary Counter Status: DISABLED        Flow ID Bit Num [0..4095]:

Bin Block Name:         fn_nom_func
Virtual Bin Number:     3               Last Executed Virtual Bin #:    0
Bit Bin Number:         0               OR of executed Bit Bin Nums: +  0
                                        Soft Bin Number:             =  0

Bin Map Block Name:     ctg_map
Display Mode:           WAFER           Soft Bin Consecutive Count:     0
Clear Consec @:         END OF WAFER & LOT   Hard Bin Consecutive Count:  0
```

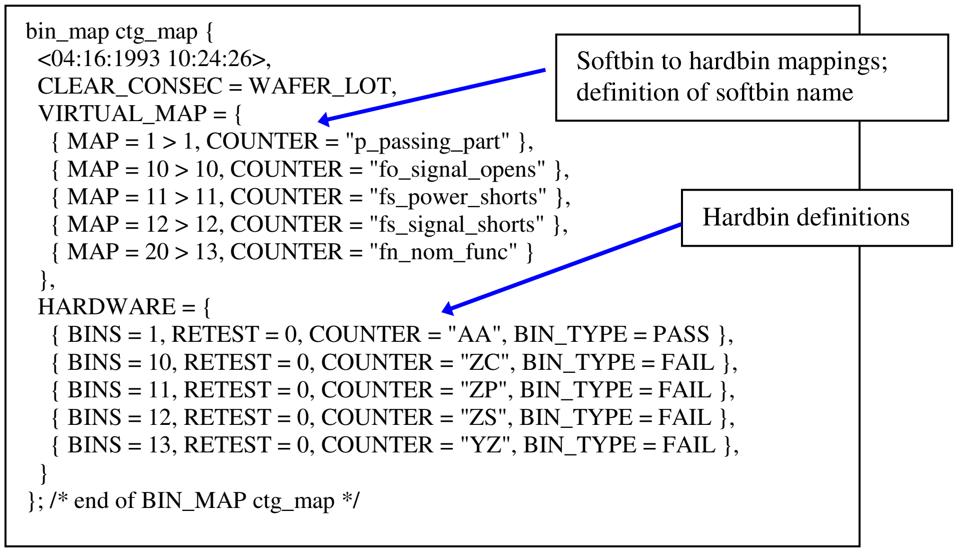
Soft Bins			Soft Bin Limits			Hard Bins							
Name	Num	Count	Count	MaxCnt	Consec	Name	Num	Count	Count	Max Cnt	Consec	Retest	Type
p_best_part	1	0	0			AA	1	0	0				PASS
p_better_part	2	0	0			AB	2	0	0				PASS
p_good_part	3	0	0			AC	3	0	0				PASS
p_fair_part	4	0	0			AD	4	0	0				PASS
p_passing_part	5	0	0			AE	5	0	0				PASS
fo_signal_opens	10	0	0			ZC	10	0	0				FAIL
fs_power_shorts	11	0	0			ZP	11	0	0				FAIL
fs_signal_shorts	12	0	0			ZS	12	0	0				FAIL
fn_scan_func	20	0	0			YZ	13	0	0				FAIL
fn_dc_func	21	0	0			YB	14	0	0				FAIL
fn_ac_func	22	0	0			YB	14	0	0				FAIL
fn_highz_func	23	0	0			YB	14	0	0				FAIL

Fig. 7 Schlumberger ITS9000 – Bin/Binmap window opened at failing port – bin fn_nom_func attached


Softbin definitions

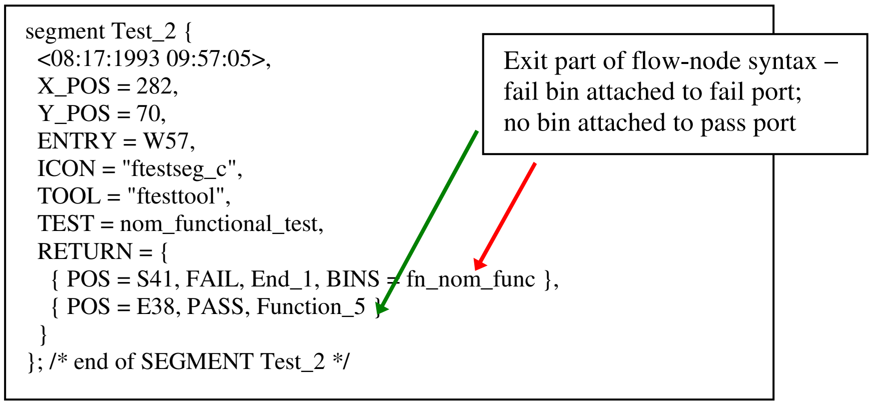
```
bins p_passing_part {
  <08:17:1993 11:49:53>,
  VIRTUAL_BIN = 1,
  BIT_BIN = 0
}; /* end of bins p_passing_part block */

bins fn_nom_func {
  <08:17:1993 09:57:05>,
  VIRTUAL_BIN = 20,
  BIT_BIN = 0
}; /* end of bins fn_nom_func block */
```

Fig. 8 Schlumberger ITS9000 – Test program syntax example for bin blocks


Softbin to hardbin mappings; definition of softbin name
Hardbin definitions

```
bin_map ctg_map {
  <04:16:1993 10:24:26>,
  CLEAR_CONSEC = WAFER_LOT,
  VIRTUAL_MAP = {
    { MAP = 1 > 1, COUNTER = "p_passing_part" },
    { MAP = 10 > 10, COUNTER = "fo_signal_opens" },
    { MAP = 11 > 11, COUNTER = "fs_power_shorts" },
    { MAP = 12 > 12, COUNTER = "fs_signal_shorts" },
    { MAP = 20 > 13, COUNTER = "fn_nom_func" }
  },
  HARDWARE = {
    { BINS = 1, RETEST = 0, COUNTER = "AA", BIN_TYPE = PASS },
    { BINS = 10, RETEST = 0, COUNTER = "ZC", BIN_TYPE = FAIL },
    { BINS = 11, RETEST = 0, COUNTER = "ZP", BIN_TYPE = FAIL },
    { BINS = 12, RETEST = 0, COUNTER = "ZS", BIN_TYPE = FAIL },
    { BINS = 13, RETEST = 0, COUNTER = "YZ", BIN_TYPE = FAIL },
  }
}; /* end of BIN_MAP ctg_map */
```

Fig. 9 Schlumberger ITS9000 – Test program syntax example for bin map


Exit part of flow-node syntax – fail bin attached to fail port; no bin attached to pass port

```
segment Test_2 {
  <08:17:1993 09:57:05>,
  X_POS = 282,
  Y_POS = 70,
  ENTRY = W57,
  ICON = "ftestseg_c",
  TOOL = "ftesttool",
  TEST = nom_functional_test,
  RETURN = {
    { POS = S41, FAIL, End_1, BINS = fn_nom_func },
    { POS = E38, PASS, Function_5 }
  }
}; /* end of SEGMENT Test_2 */
```

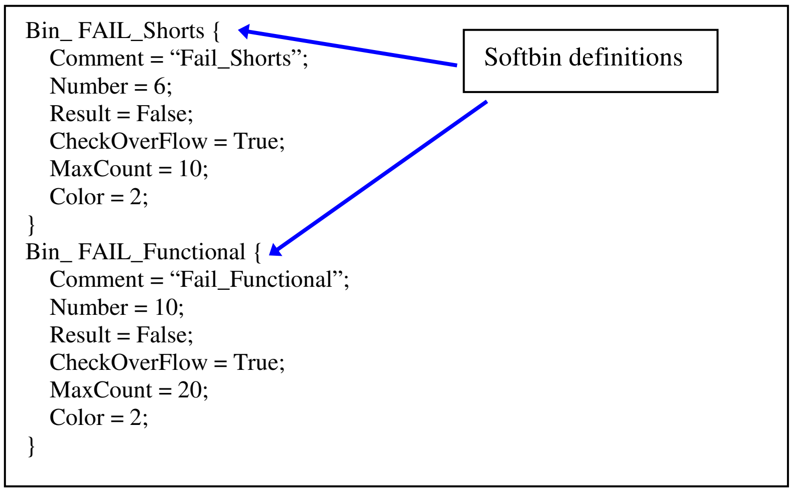
Fig. 10 Schlumberger ITS9000 – Test program syntax example for flow-node

LTX Envision Binning:

Bin architecture:

A bin is defined as a named object containing six elements, as shown below:

- Comment
- Number
- Result
- MaxCount
- CheckOverFlow
- Color



```
Bin_ FAIL_Shorts {
   Comment = “Fail_Shorts”;
   Number = 6;
   Result = False;
   CheckOverFlow = True;
   MaxCount = 10;
   Color = 2;
}
Bin_ FAIL_Functional {
   Comment = “Fail_Functional”;
   Number = 10;
   Result = False;
   CheckOverFlow = True;
   MaxCount = 20;
   Color = 2;
}
```

Fig. 11 LTX Envision test program syntax example for (soft) bins

```
Bin_ <name> {
            Comment = “<comment>”;
            Number = <bin>;
            Result = <pass_fail>;
            MaxCount = <max_count>;
            CheckOverFlow = <bypass>;
            Color = <color>;
}
```

Fig. 12 LTX Envision test program syntax definition for (Soft) Bins

Bin map(s):

The bin map is simply a list of (soft) bin name to hard bin number assignments.

Multiple binmaps can be defined in the test program. Only one at a time, of course, can be active, as shown below in Fig. 15.

```
BinMap Training_LS245_BM {
    Bin PASS = 1;
    Bin FAIL_Shorts = 4;
    Bin FAIL_Functional = 6;
    Bin evResetBin = 0;
}
```

Fig. 13 LTX Envision test program syntax example for binmaps

```
BinMap <name> {
   Bin <Bname> = <Hbin>;
   Bin <Bname> = <Hbin>;
   Bin <Bname> = <Hbin>;
}
```

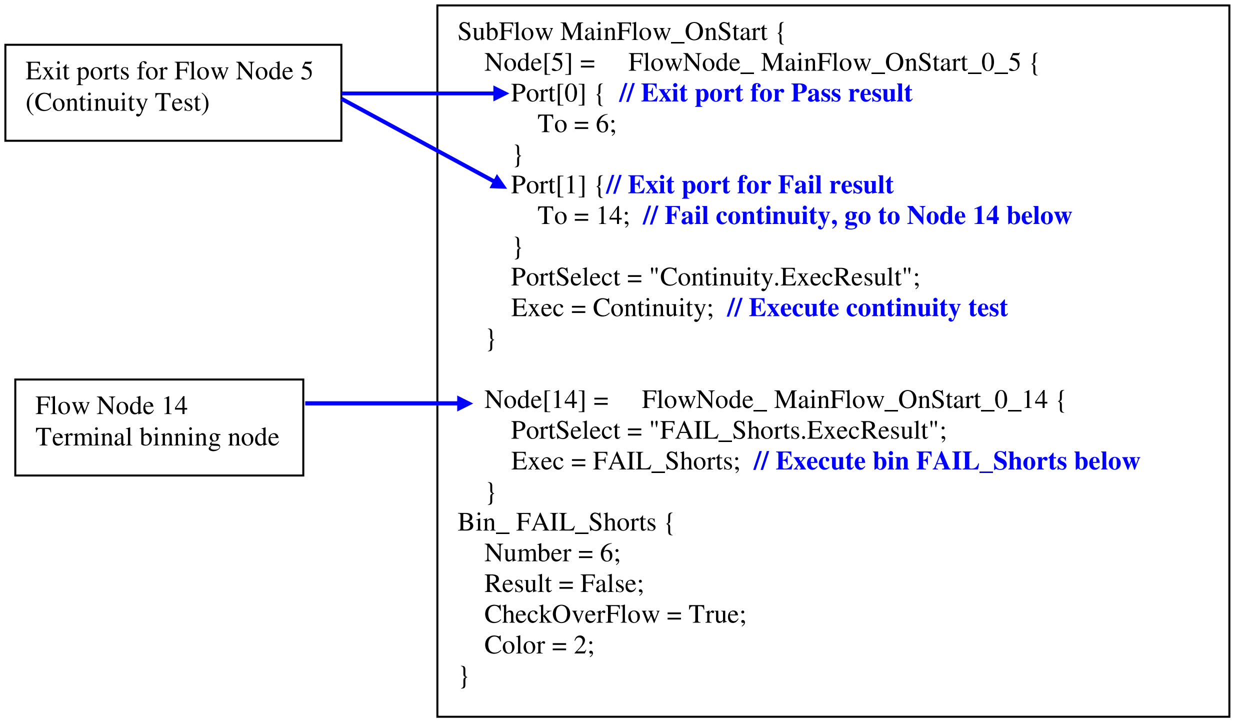
Fig. 14 LTX Envision test program syntax definition for binmaps

```
TestProg Training_LS245 {
        AdapterBoard = DIP_LS245;
        Flow_ = MainFlow;
        BinMap = Training_LS245_BM;
        <snip, snip>
}
```

Fig. 15 LTX Envision test program example showing binmap selection

Containment hierarchy (relationship to flow):
A bin is an executable object contained within a terminal flow-node. The terminal flow node is made terminal by virtue of having no exit ports. Creating a new bin generates this type of terminal flow-node. See Fig. 16 below for an example of the syntax of the bins and the terminal flownode associated with bins. Fig. 17 below shows an example of a flowtool canvas which include normal flownodes and binning flownodes.

- A bin is essentially a terminal flow-node
- The bins available for use as a terminal flownode are selected from a bin map. A test program can have multiple bin maps (but obviously, only one can be selected at any given time).
- All bins used in a testflow must be contained in the currently selected bin map.
- It seem reasonable to assume that every flow (subflow, in Envision terminology) which will be executed from an entry point must ventually terminate all its paths at a bin node.


Exit ports for Flow Node 5 (Continuity Test)
Flow Node 14 Terminal binning node

```
SubFlow MainFlow_OnStart {
    Node[5] =     FlowNode_ MainFlow_OnStart_0_5 {
        Port[0] {  // Exit port for Pass result
            To = 6;
        }
        Port[1] {// Exit port for Fail result
            To = 14;  // Fail continuity, go to Node 14 below
        }
        PortSelect = "Continuity.ExecResult";
        Exec = Continuity;  // Execute continuity test
    }

    Node[14] =     FlowNode_ MainFlow_OnStart_0_14 {
        PortSelect = "FAIL_Shorts.ExecResult";
        Exec = FAIL_Shorts;  // Execute bin FAIL_Shorts below
    }
Bin_ FAIL_Shorts {
    Number = 6;
    Result = False;
    CheckOverFlow = True;
    Color = 2;
}
```

Fig. 16 LTX Envision test program example showing bins as a terminal flownode

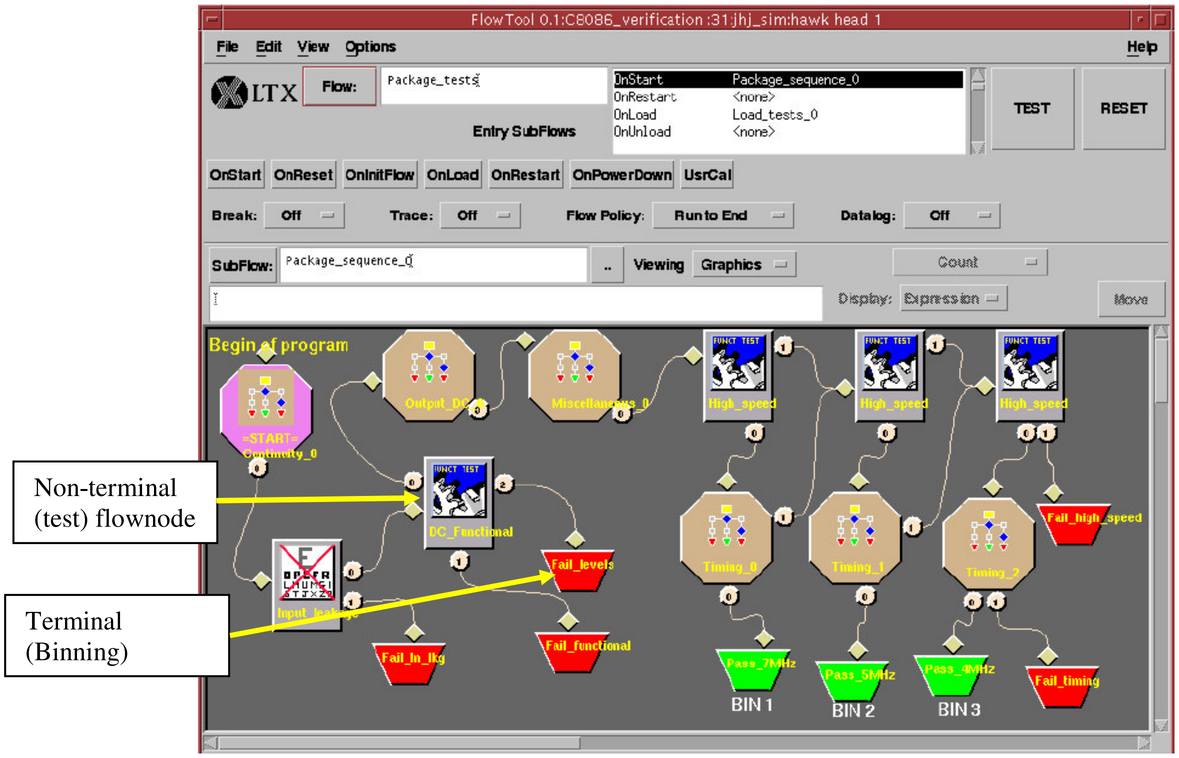


Fig. 17 LTX Envision Flowtool example display showing non-terminal (test) and terminal (binning) flownodes

Bin/stop relationship:

Bin strategies:

Retest handling:
Envision handles retests by including an EntryPoint called OnRestart. This EntryPoint is similar to the OnStart EntryPoint, except that it is triggered - either by the operator, upon request of a retest, or (presumably) by the handling equipment – when a retest is requested. It seems likely that, in this scenario, it's the handling equipment's responsibility (or the operator's, if manual testing) to know under what conditions (a device being binned into a particular set of hardbins, for instance) a retest should be requested.

I don't know for sure, but I would expect that the system must keep track of when a retest is requested, and does not update the typical system summary tallies (# devices run, # devices passed, # devices failed) – or decrements them – when a retest is requested. The LTX Envision manual for developing prober/handler interfaces includes a description of a restart method belonging to the class library used for external equipment control:

> ***EVXAStatus restart(WAIT_MODE waitMode = EVXA::WAIT)***
> Similar to ***start()*** except the serial number is not incremented. This is like the operator pressing the Restart button.

This description seems to bear out the assumptions above – it says nothing, however, about the system device tallies (total devices tested, number of passing devices, number of failing devices) mentioned above.

It would also be my guess that the flow executed by OnRestart would probably be identical to the flow executed by OnStart We may want to consider including an OnRetest (or OnRestart) EntryPoint.

Advantest OTPL Binning:

Bin architecture:

Bin map(s):

Containment hierarchy (relationship to flow):

Bin/stop relationship:

Bin strategies:

Retest handling:

Bin Definitions

The *Bin Definitions Sub-language* defines *bins*, a collection of counters that summarize the results of testing many DUTs. During the course of testing a DUT, it can be set to any bin. As testing proceeds, the DUT may be set to another bin. The bin that the DUT is finally set to is the last such setting at the end of the test. The counter for this final bin is incremented at the end of the test of this DUT.
A separate file with bin definitions should have the extension `.bdefs`.
Bin definitions are *hierarchical*. Pictorially a collection of bins may represent the following hierarchy

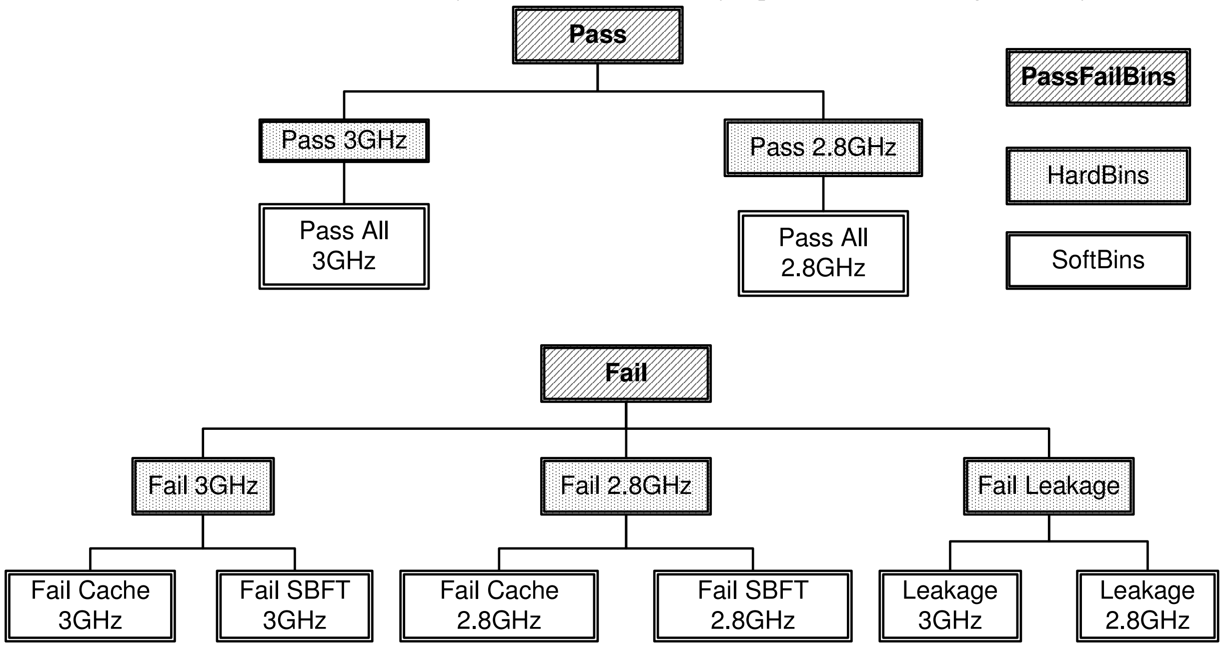


Figure Error! No text of specified style in document.**-1 Bin Defnitions Hierarchy**

Bins in a Bin Definitions hierarchy (a **`BinDefs`** declaration) are organized into *BinGroups*, which define the set of bins at a level in the hierarchy. In Figure **Error! No text of specified style in document.**-1 at the outermost level there is a *BinGroup*, say PassFailBins with two bins named Pass and Fail. Then at the next level there is a BinGroup, say HardBins, some of which are under the Pass bin, and others which are under the Fail bin. This next level of bins are said to be a *refinement* of the PassFailBins BinGroup. Finally, there are a large number of bins in the SoftBins BinGroup, a refinement of HardBins BinGroup.

1.1 Example of Bin Definitions

Below is an OTPL example describing the hierarchy of bins in Figure **Error! No text of specified style in document.**-1:

```
# ----------------------------------------------------------
# File cpuxbins.bdefs
# ----------------------------------------------------------

Version 1.0;

BinDefs
{
    # The PassFailBins are an outermost level of
    # bins.  They are not a refinement of any other
    # bins.
    BinGroup PassFailBins
    {
        Bin Pass 0: “Count of passing DUTS.”;
        Bin Fail 1: “Count of failing DUTS.”;
    }

    # The HardBins are a next level of refinement.
    # HardBins are a refinement of the PassFailBins
    # declared just before.
    BinGroup HardBins
    {
        Bin Pass3GHz    10:“DUTs passing 3GHz”,   Pass;
        Bin Pass2_8GHz  11:“DUTs passing 2.8GHz”, Pass;
        Bin Fail3GHz    12:“DUTs failing 3GHz”,   Fail;
        Bin Fail2_8GHz  13:“DUTs failing 2.8GHz”, Fail;
        Bin FailLeakage 14:“DUTs failing leakage”,Fail;
    }

    # The SoftBins are a next level of refinement.
    # SoftBins are a refinement of HardBins declared
    # just before.
    BinGroup SoftBins
    {
        LeafBin PassAll3GHz      20:
          “Good DUTs at 3GHz”,   Pass3GHz;
        LeafBin FailCache3GHz    21:
          “Cache Fails at 3GHz”, Fail3GHz;
        LeafBin FailSBFT3GHz     22:
          “SBFT Fails at 3GHz”,  Fail3GHz;
        LeafBin FailLeakage3GHz  23:
          “Leakages at 3GHz”,    FailLeakage;
        LeafBin PassAll2_8GHz    24:
          “Good DUTs at 2.8GHz”, Pass2_8GHz;
        LeafBin FailCache2_8GHz   25:
          “Cache Fails at 2.8GHz”, Fail2_8GHz;
        LeafBin FailSBFT2_8GHz    26:
          “SBFT Fails at 2.8GHz”,Fail2_8GHz;
        LeafBin FailLeakage2_8GHz 27:
          “Leakages at 2.8GHz”,  FailLeakage;
    }
   # The keyword SortBinGroup identifies one of the preceding
   # BinGroup’s as the one that is used for the sorter.  The
bin
   # numbers of bins of that group will be used by the
hardware
   # for sorting.
   SortBinGroup = HardBins;
}
```

1.2 Static Semantics of Bin Definitions

The OTPL description of BinDefs defines a simple linear ordering of BinGroups. Each BinGroup except for the the first in the BinDefs is a *refinement* of the *predecessor* (preceding) BinGroup. BinGroups represent *levels* in the hierarchy of bin definitions. Each BinGroup declares a number of bins that are constrained by the following rules.

- ❑ There are two kinds of bins declared in BinGroups, identified by the keywords **`Bin`** and **`LeafBin`**.

- ❑ The keyword **`Bin`** declares an internal bin that can be further refined.

- ❑ The keyword **`LeafBin`** declares a bin that cannot be further refined.

- ❑ If a BinGroup has a predecessor (i.e. it is not the first BinGroup in the BinDefs), leaf and non-leaf (or *base*) bins in it have to be refinements of base bins of its predecessor BinGroup. Each bin declaration in the BinGroup must specify the base (non-leaf) bin of the predecessor BinGroup that it is a refinement of.

- ❑ Bins of the first BinGroup (with no predecessor) cannot be refinements of any other bins.

- ❑ Each (leaf or non-leaf) bin declaration has the following:
  - A *name* which an identifier. Bin names must be unique within a BinGroup. A bin can have the same name as another bin in another BinGroup. However, this practice leads to confusion and is therefore discouraged.
  - A *number* which is the bin number. Bin numbers are unique within a BinGroup, but may be the equal to numbers in another BinGroup.
  - A *description* which describes what this bin summarizes.
  - If the enclosing BinGroup has a predecessor, then the bin of the predecessor that it is a refinement of.

- ❑ A BinDefs declaration completes by specifying the `SortBinGroup` as one of the preceding declared BinGroups. The SortBinGroup is the group of bins that is recognized for sorting purposes. The SortBinGroup is used with the SetBin statement as described below.

The above rules constrain BinGroups to be the set of bins at a given level. Note that a BinGroup can have both leaf and base bins.

BinGroup names are global in scope. There can only be a single BinDefs block in the test plan and all its imports. Thus, each instance of a TestPlan is associated with a single set of Bins and BinGroups which count the DUTS tested in that instance of the TestPlan.

In the example presented above, the two bins in `PassFailBins` BinGroup are named `Pass` and `Fail`. The five bins in `HardBins` are named `Pass3GHz`, `Pass2_8GHz`, `Fail3GHzFail`, `Fail2_8GHz`, `FailLeakage`. Bin names are an identifier.

Of the five `HardBins`, the bins `Pass3GHz` and `Pass2_8GHz` are both refinements of the `Pass` bin of the `PassFailBins` BinGroup. The rest of the `HardBins` are refinements of the `Fail` bin of the `PassFailBins` BinGroup.

Finally, there are eight `SoftBins`. The two failures at 3GHz for SBFT and Cache are refinements of the `HardBin.Fail3GHz`. Likewise the two failures at 2.8GHz for SBFT and Cache are refinements of the `HardBin.Fail2_8GHz`. Both the failures due to Leakage are refinements of the same bin, `HardBin.FailLeakage`, regardless of the speed at which they occurred.

Bins are assigned to DUTs in a Test Plan FlowItem, described below. A TestPlan FlowItem has a *Result Clause* in which the test plan describes the actions and transition to take place as the result of getting a particular result back from executing a test. It is at this point that a *SetBin* statement can occur:

```
# A FlowItem Result clause.  It is described later.
Result 0
{
    # Action to be taken on getting a 0 back from
    # executing a test.

    # Set the bin to SoftBin.PassAll3GHz expressing that the
    # DUT was excellent.
    SetBin SoftBins.PassAll3GHz;
}
```

1.3 Dynamic Semantics of Bin Definitions

Many **`SetBin`** statements could execute during the course of running a test plan on a DUT. The OASIS runtime will ensure that the following rules are satisfied:

- When the test plan is finally completed, the OASIS runtime will increment the counter for the *final* bin that is set for that DUT, and for all its refinements. Only the final **`SetBin`** statement of all that are executed has any effect.

- The set of bins which will be incremented represents the nodes of a path in the bin hierarchy tree from a root bin to the leaf bin on which the **`SetBin`** statement occurred.

- At most one bin from each BinGroup will be in the set of bins incremented.

- There will be at most one bin from the SortBinGroup which will be incremented. The bin number of this bin is passed to the tester hardware in order to enable it to sort the DUT by putting it into the appropriate physical bin.

- The setting of this final bin in the execution of the test plan happens through a Commit operation of the BinGroupMgr object in the TestPlan. This Commit operation is called by the Test Plan Server at the end of TestPlan execution.

- ❑ The setting of this final bin can be *rolled back* by the user from a GUI application, using the Rollback operation. This is needed when the user decides that the test needs to be rerun. It undoes the bin setting of the prior Commit.

- ❑ A call to Rollback is ignored while a test it running, and has no effect.

Consider again the example presented earlier. Suppose a DUT had the following **`SetBin`** statements executed during the course of its test:

```
SetBin SoftBins.FailSBFT3GHz;
SetBin SoftBins.PassAll2_8GHz;
```

This DUT passed the 3GHz Cache and Leakage tests, but failed the SBFT test, and so was assigned to the `FailSBFT3GHz` bin. It was then tested at 2.8GHz, and all the tests passed. So the final bin assignment is to the `PassAll2_8GHz` bin, which is in the BinGroup SoftBins.

When the test plan completes, the final assignment to the leaf bin will cause its counter to be incremented, and also the counter of each bin that it is a refinement of. Thus, this final assignment causes a single bin to be incremented in each BinGroup (i.e. at each level). In the above example, this final assignment will increment the counters of the following bins:

- Bin SoftBins.PassAll2_8GHz
- which is a refinement of HardBins.Pass2_8Ghz
- which is a refinement of PassFailBins.Pass

Thus, when the test plan completes, OASIS will increment the counter of the final bin assignment of the DUT, and for all other bins it is a refinement of.

A **`SetBin`** statement is allowed only on a bin declared as a **`LeafBin`**. It is illegal to set a base (non-leaf) bin. The counter incrementing semantics above assures that:

- If the bin is a **`Leaf Bin`**, it is the number of times a **`SetBin`** statement was executed for this bin at the end of testing a DUT.
- If the bin is a not a **`LeafBin`**, it is the sum of the counters of the bins that it is a refinement of.

Thus, in the above example, `SoftBins.FailSBFT3GHz` is allowed in a **`SetBin`** statement, but `HardBins.Fail3GHz` is not allowed in the statement. The counter for `HardBins.FailLeakage` is the sum of the counters for `SoftBins.FailLeakage3GHz` and `SoftBins.FailLeakage2_8GHz`.

The set of bins whose counters are incremented as a result of this final assignment includes a single bin from each BinGroup. In particular, a single bin of the designated SortBinGroup will have its counter incremented. The bin number of this particular bin of the SortBinGroup will be passed on to the tester hardware in order to sort the DUT into a physical bin.