18. Environment block

The Environment block contains constructs to cross-reference STIL information with other environments, for example, a simulation or layout netlist environment. The Environment block may be used by tools that operate in the specified Environment and access STIL data. The Environment block may also contain additional information specific for the particular environment.

Because the Environment block references information not contained in the STIL environment, no STIL blocks reference the Environment block. The Environment block is self-contained, but internally it may reference other STIL blocks as necessary to associate STIL information with the external context.

The Environment block may contain additional blocks that have not been defined as part of the STIL.1 standard and that do not need to be declared (with a UserKeywords statement) before they appear in the Environment block. These undeclared blocks are allowed with the same restrictions as a UserKeyword block. It is expected that the definition of these undeclared blocks is found in association with an extension definition. See Clause 7 for extension reference constructs. These blocks, when present, shall follow all STIL syntax requirements for UserKeyword blocks. These blocks contain information specific to an Environment context (in STIL syntax form).

18.1 Environment syntax

```
Environment (ENV_NAME) {                                                        (1)
    ( InheritEnvironment ENV_NAME ; )*                                          (2)
    ( NameMaps (MAP_NAME) {                                                     (3)
        ( InheritNameMap (ENV_NAME::) (MAP_NAME) ; )*                           (4)
        ( InheritNameMap (ENV_NAME::) (MAP_NAME) {
            ( Prefix “PREFIX_STRING”; )                                         (5)
            ( Separator “SEPARATOR_STRING”; )                                   (6)
            ( ScanCells  { (SCAN_CELL_NAME ;)*  } )*                            (7)
        } )*   // end InheritNameMap
        ( Prefix “PREFIX_STRING”; )                                             (8)
        ( Separator “SEPARATOR_STRING”; )                                       (9)
        ( ScanCells  { ((SCAN_CELL_NAME) “MAP_STRING”; )* } )*                  (10)
        ( Signals { (SIG_NAME “MAP_STRING”; )* } )*                             (11)
        ( SignalGroups (DOMAIN_NAME) { (GROUP_NAME “MAP_STRING”; )* } )*        (12)
        ( Variables { (VAR_NAME "MAP_STRING”; )* } )*                           (13)
        ( AllNames { (ANY_NAME “MAP_STRING"; )* } )*                            (14)
    } )*   // end NameMaps
    ( FileReference “FILE_PATH_NAME”; )*                                        (15)
    ( FileReference “FILE_PATH_NAME” {
        Type FILE_TYPE ;                                                        (16)
        Format FILE_FORMAT ;                                                    (17)
        Version “VERSION_NUMBER” ;                                              (18)
    } )*   // end FileReference
    (ENVIRONMENT_DEFINED_STATEMENTS)*                                           (19)
}            // end Environment
```

(1) **Environment:** Defines an Environment block.

ENV_NAME: Optional name of the Environment block. A single global Environment block may be defined without a name.

(2) **InheritEnvironment**: Statement to reference a previously defined Environment block, to incorporate the definitions of that block as part of the current block. Only statements with defined keywords are inherited, that is, SignalGroups, NameMaps, and FileReference statements or blocks. See the InheritNameMap statement for information on how NameMaps are inherited. If a FileReference block has the same name locally as an inherited FileReference, the local definition will be used. ENVIRONMENT_DEFINED_STATEMENTS are not inherited unless the inheritance rules are specified by some other STIL extension and that extension is identified in the file header. An un-named

Environment block is not automatically inherited and cannot be explicitly inherited (because it has no reference name).

(3) **NameMaps**: Defines one or more blocks containing references of STIL names to names defined in an external Environment. The MAP_NAME is optional when a single NameMaps block is present; the MAP_NAME is required if subsequent NameMaps blocks are present. The MAP_NAME shall be unique across all NameMaps blocks in a single environment when present.

(4) **InheritNameMap**: Statement to reference a previously defined NameMaps block inside an Environment block, to incorporate the definitions of the NameMaps section of that block as part of the current block.

If a NameMaps block in the current environment is being referenced, only the name of that block is required. If a NameMaps block in another environment block is being referenced, then the ENV_NAME:: is required, followed by the name of the desired block. If the NameMaps block to be inherited is the global block, then the MAP_NAME is not required. A local definition of a SCAN_CELL_NAME, SIGNAL_NAME, GROUP_NAME, VAR_NAME, or ANY NAME will override any inherited definitions for those names.

An alternative form of InheritNameMap uses the { } format and allows for additional capabilities relative to the scan-cells information.

(5) **Prefix**: This statement, when used within the context of InheritNameMap, causes the PREFIX_STRING to be prepended to each MAP_STRING definition that is inherited.

(6) **Separator**: This statement, when used within the context of InheritNameMap, causes the SEPARATOR_STRING to be used on each MAP_STRING definition that is inherited.

(7) **ScanCells**: This statement, when used within the context of InheritNameMap, is used to define a list of SCAN_CELL_NAMES. This list of names is to be correlated to a list of MAP_STRINGS in the inherited block and along with the prefix and separator from the mapped name definitions for each cell. It is an error condition if the number of MAP_STRINGS is less than the number of SCAN_CELL_NAMES. If the number of MAP_STRINGS is greater than the number of SCAN_CELL_NAMES, the additional MAP_STRINGS are ignored. Note that the correlation between cell and map strings is by position in the list.

(8) **Prefix**: This statement, when used within the context of NameMaps, causes the PREFIX_STRING to be prepended to each MAP_STRING definition that occurs within the current NameMaps block.

(9) **Separator**: This statement, when used within the context of NameMaps, causes the SEPARATOR_STRING to be used on each MAP_STRING definition that occurs within the current NameMaps block.

(10) **ScanCells**: This statement, when used within the context of NameMaps, can take two forms: one with the SCAN_CELL_NAME defined and one without.

In the case in which the SCAN_CELL_NAME is defined, a direct mapping of the name of the cell is made to the defined MAP_STRING. The MAP_STRING parameter is required.

In the case in which only one parameter is defined, then it is interpreted as the MAP_STRING. It is the form to be used when the list of MAP_STRINGs is to be inherited.

(11) **Signals**: Required if SIG_NAME map information is provided.

SIG_NAME "MAP_STRING": Statement to map the STIL SIG_NAME defined in a Signals block, into the name used in the external Environment.

(12) **SignalGroups**: Required if GROUP_NAME map information is provided.

GROUP_NAME "MAP_STRING": Statement to map the STIL GROUP_NAME defined in a SignalGroups block, into the name used in the external Environment.

(13) **Variables**: Required if VAR_NAME map information is provided.

VAR_NAME "MAP_STRING": Statement to map the STIL VAR_NAME defined in a Category or Spec block, into the name used in the external Environment.

(14) **AllNames**: This optional block supports an Environment that does not partition name spaces with the same structure as STIL and supports mapping of names from additional STIL constructs (such as objects in user-defined blocks).

ANY_NAME "MAP_STRING": Statement to map any STIL NAME, into the name used in the external Environment.

(15) **FileReference** "FILE_PATH_NAME"**:** The FileReference statement is used within an Environment block to specify various other files associated information. The content and application of the referenced files is not specified in STIL, but it is the responsibility of the tool or application using the Environment block. For instance, if the patterns are in STIL, then they would be Included, but if the patterns were in WGL or some tester-specific format, this mechanism would be used to reference that data.

(16) **Type** FILE_TYPE: Specifies the type of this file. The allowed file types are not defined as part of this standard.

(17) **Format** FILE_FORMAT: Specifies the format of this file type. The allowed file types are not defined as part of this standard.

(18) **Version** "VERSION_NUMBER": A quoted string identifying the version of this file. The format and information of the VERSION_NUMBER is dependent on the file type and format, and it is not defined here.

(19) ENVIRONMENT_DEFINED_STATEMENTS: These are semicolon-terminated or braced statements that follow the STIL requirements but do not require a UserKeyword statement to identify these blocks. The environment-defined statement and semantics associate with the specification for the extension as identified in the STIL {} block.

18.2 MAP_STRING syntax

The target MAP_STRING maps STIL names into an external environment. If the target name contains either a double quote or a backslash, then the following syntax shall be used in the target string. It is an error for any character other than double quote or backslash to follow a backslash character.

double-quote: "blah_blah"\"blah_blah"

backslash: "blah_blah\\blah_blah"

error: "blah_blah\blah_blah"

A typical usage of this backslash notation is in mapping strings into Verilog names. The following example is of this transformation:

MAP_STRING in STIL: "\\\"abc[15]"

Resultant string : \"abc[15]

NOTE—Per STIL.0 definition, double quotes are only delimiters and are not allowed as part of the string.

18.3 NameMaps example

The NameMaps block relates STIL names to external environments. An example of an application of this construct, for a Verilog-style environment, is shown in Figure 4.

```
module top_test;// Testbench Module
   integer pattern;
   wire [0:2] PO; // Primary Outputs Vector
   reg [0:3] PI;// Primary Inputs Vector
   wire A, B1, C1, D11, Q11, Q21;// Signals
   assign A = PI[0], B1 = PI[1], C1 = PI[2], D11 = PI[3];
   assign PO[0] = Q11, PO[1] = Q21;
        // Design Instance
   top dut (.A(A),.B1(B1),.C1(C1),.D11(D11),.Q11(Q11),.Q21(Q21));
endmodule
```

Figure 4—Verilog test bench example for NameMaps

```
STIL 1.0 { Design 2005; }
Header {
  Source "STD 1450.1-2005";
  Ann {* sub-clause 18.3 *}
}

Signals { "A" In; "B1" In; "C1" In; "D11" In; }
SignalGroups { _PI = ' "A" + "B1" + "C1" + "D11" '; }

Environment MY_VERILOG_TESTBENCH {
  NameMaps VECTOR_ASSOCIATIONS {
    Signals {
      "A" "top_test.PI[0]";
      "B1" "top_test.PI[1]";
      "C1" "top_test.PI[2]";
      "D11" "top_test.PI[3]";
    }
    SignalGroups {
      _PI "top_test.PI";
      _PO "top_test.PO";
    }
    Variable { _PATCOUNT "PATTERN"; }
  } // end NameMaps

  NameMaps WIRE_ASSOCIATIONS {
    Signals {
      "A" "top_test.A";
      "B1" "top_test.B1";
      "C1" "top_test.C1";
      "D11" "top_test.D11";
    }
    SignalGroups {
      _PI "top_test.PI";
      _PO "top_test.PO";
    }
    Variable { _PATCOUNT "PATTERN"; }
  } // end NameMaps
} // end Environment
```

18.4 Compact scan-cell mapping using InheritNameMap

```
STIL 1.0 { Design 2005; }
Header {
  Source "STD 1450.1-2005";
  Ann {* sub-clause 18.4 *}
}
Environment HIERARCHICAL {
  // A hierarchical NameMap for scan cells
  // See the FLAT example below to see the expanded net names
  NameMaps C {
    ScanCells {"CELL1"; "CELL2";} // Only the MAP_STRING portion of the map
  }
  NameMaps D {
    InheritNameMap C {
      Prefix "C3"; // String to prepend to MAP_STRINGS in inherited NameMap
      Separator "/";
    }
    ScanCells {"CELL3";}
  }
  NameMaps TOP {
    Separator "/";
    InheritNameMap C {
      Prefix "C1";
      ScanCells {C[0..1];} // Only the SCAN_CELL_NAME portion of the map
    }
    InheritNameMap C {
      Prefix "C2";
      ScanCells {C[2..3];}
    }
    InheritNameMap D {
      Prefix "D1";
      ScanCells {C[5..7];}
    }
    ScanCells {C[4] "FOO";}
  }
}

Environment FLAT {
  // This is a flat representation of NameMap for scan cells
  // This mapping corresponds to the above HIERARCHICAL mapping
  NameMaps {
    ScanCells {
      C[0] "C1/CELL1";
      C[1] "C1/CELL2";
      C[2] "C2/CELL1";
      C[3] "C2/CELL2";
      C[4] "FOO";
      C[5] "D1/C3/CELL1";
      C[6] "D1/C3/CELL2";
      C[7] "D1/CELL3";
    }
  }
}
```

Annex C

(informative)

Tester channel map

This clause makes use of the NameMap block as defined in Clause 17 of STIL.1.

In STIL.1, there is the definition of how to specify a mapping of Signals and other objects from the STIL pattern interchange environment to some other environment. This facility can be used to specify the mapping of a STIL test program to a set of tester channels on a tester. This clause is informational only as no new syntax is defined herein.

C.1 Tester channel map—syntax

```
Environment (ENV_NAME) {
   ( NameMaps (MAP_NAME) {
      ( Signals { (SIG_NAME “MAP_STRING”; )* } )*
   } // end NameMaps
} // end Environment
```

C.2 Tester channel map—example

```
STIL 1.0 { Design 2005; TRC 2007; }
Header {
 Source "IEEE Std 1450.3-2007";
 Ann {* clause C.2 *}
}

Environment ATE_SC212 {
 NameMaps WAFER {
   Signals {
     // sig-name “channel, type”;
     SIG1 “13, INPUT”;
     SIG2 “45, BIDI”;
     SIG3 “46, OUT”;
   } // end Signals
 } // end NameMaps WAFER
 NameMaps PACKAGE {
   Signals {
     SIG1 “42, INPUT”;
     SIG2 “19, BIDI”;
     SIG3 “16, OUT”;
   } // end Signals
 } // end NameMaps PACKAGE
 NameMaps MULTISITE {
   Signals {
     SIG1 "INPUT 13 48 96";
     SIG2 "BIDI 45 83 99";
     SIG3 " OUT 46 85 128";
```

```
    } // end Signals
} // end NameMaps MULTISITE
NameMaps MULTISITE_PINGPONG {
  Signals {
    SIG1 "INPUT (13 48) (96 106)";
    SIG2 "BIDI  (45 83) (99 107)";
    SIG3 "OUT   (46 85) (128 126)";
  } // end Signals
} // end NameMaps MULTISITE_PINGPONG
} // end Environment
```

Using NameMaps for specifying external pins as pin information for test program generation.

Explanation:

The following are described using NameMaps as they are required for external pins as pin information for test program generation. Describe the following using an Assign block:

- I/O cell name to be connected
- Internal cell name to be connected and the node name of the internal cell
- In-circuit instance name when the same cell names are used and one needs to be specified
- All individual connection information when an external pin is used by multiple internal cells

Also describe the following to an I/O cell and internal cell using a Buffer block:

- I/O attributes
- Input level group （TTL,CMOS, and so on ）
- Output level group
- Output current capability group
    - switch count and current capability for cells with the changeable output current capability
- Control information of controllable Pull-up/down
- Differential buffer or the information to identify a pair of external pins used in a set

Syntax:

```
NameMaps {
    PinAssign {
        //"      1        " "   2       3      4       5        6       7      8        " //
        //"ext_pinname" "io_cell, pin1, pin2, instance, pin3, core, instance"//
        "(1) external pin name" "(2) I/O cell name, (3) pin name, (4) pin name 2,
            (5) instance name of I/O cell, (6) pin name 3, (7) internal core name, (8) instance name";
    : :
    }  //end PinAssign
    Buffer
    {
        //"cell_name" "pin_name, power, ground, direction, drivetype, structure, etc" //
        "(1) cell name" "(2) multi-pad-pin name, (3)power supply type, (4) ground type,
            (5) I/O attributes, (6) drivetype, (7) structure info, (8) other";
    : :
    } //end Buffer
} //end NameMaps
```

Example:

```
NameMaps { // NameMaps is used to describe IEEE-unstandardized description
    PinAssign  { // Block to describe the device pin information
        "COMPACTOR_SCOMP" " TTL , , , .AUIA_DAT,  ,  ,  ,";
        "COMPACTOR_SPREAD" " TTL, , , .AUIA_DAT, , , ,";
        "RESET_CONTROL" " TTL, , , .AUIB_DAT, , , ,";
    }//End PinAssign
    Buffer { // Block to describe I/O cell buffer information
        "TTL" " ,VDDC/VDDS/VDDS2,VSS/VSS2,input ,ttl33 , , ";
        "SMTTIF" " ,VDDC/VDDS/VDDS2,VSS/VSS2,input ,smtt33 , , ";
        "TTLPD" " ,VDDC/VDDS/VDDS2,VSS/VSS2,input ,ttl33 ,pull_down, ";
    } //End Buffer
} //End NameMaps
```

Using NameMaps to describe device external pins as wiring information for device interface boards

The following are described using NameMaps as they are required for device external pins as wiring information for test board and device external pins:

- Tester channel number of final test board
- Channel number of probe card
- Package pin number and matrix number
- To a power pin, a power supply line name to be assigned
- To a GND pin, a GND line name
- To a signal pin, a power supply line name and a connecting GND line name
- Parallel connection information of multiple power supply
- Utility power supply (for relay control etc.) information
- Relay information

Syntax

```
Environment ATE_Name { // ATE_name target tester name
    NameMaps {
        Signals {
            // "Sig-name"
            "pad_name,pin_number,package_channel,prob_channel,Vs_type,Gnd_type,Sub1,Sub2,";
            "(1) signal name"" (2) pad name, (3) pin number, (4) package pin number, (5) probe
            pin number, (6) power supply type,(7) GND type,(8) extension 1, ,extension n";
            : :
        } // end Signals
        Dps {
            //VS name "Unit name";
            (9) power supply name "(10) unit name":
             :
        } //end Dps
        Relay {
            (11) relay name;
        } //end Relay
    } // end NameMaps
} // end Environment
```

Example:

```
Environment ATE_A{ //Describes a target ATE name
    NameMaps { //NameMaps is used to describe IEEE-unstandardized description
        Signals{ //This is a block to define wiring information of signal pin
            "COMPACTOR_SCOMP" "Pad1,1,X1Y1,20,VDDS,VSS2,,, ";
            "COMPACTOR_SPREAD" "Pad2,2,X1Y2,21,VDDS,VSS2,,, ";
            "RESET_CONTROL" "Pad3,3,X1Y3,22,VDDS,VSS2,,, ";
        }//End Signals
        Dps{ //This is a block to define wiring information of tester's power supply.
            VS1 "DPS1" ; //Described in order of device power supply "tester power supply";.
            VS2 "DPS2+DPS4" ; //Parallel connection of tester power supply
            VS3 "DPS3" ;
            AVDD "DPSH1" ; //Utility power supply
        }//End Dps
        Relay{ // This is a block to define wiring information of tester relay.
            PCON1 ; // Tester relay name;
            PCON2 ; // Bypass condenser relay
            LOAD[0-255]; // Pin relay
            CW[0-15]; // Control word
        } //End Relay
    } //End NameMaps
} //End Environment
```