Hierarchical Scan Description Language Syntax Specification

Revision A
31 Aug 1992
Authors: Adam Sheppard, Giridhar V.

PRELIMINARY

ASSET InterTech, Inc.

Table of Contents

1. Identification

1.1. Product Statement

During late 1989 and 1990, representatives of Hewlett-Packard developed and promoted a language for describing boundary-scan devices, called Boundary Scan Description Language (BSDL). The intended applications of BSDL were as input to a test driver, as input to a compliance monitor, and as input to an automated boundary-scan synthesis tool. All three intended applications are automated, utilizing software to drive, validate, or even create the boundary-scan hardware.

Also during this time, Texas Instruments released the ASSET™ Scan-Based Diagnostic System, the first system to support IEEE Std 1149.1-1990. The first generation of ASSET used Configuration Files to describe 1149.1 devices, boards, and systems. ASSET Configuration Files were given to HP to aid in the development of BSDL.

HP has done an excellent job in working with industry to fan out the BSDL requirements specifications and accept inputs for improvements. BSDL has become a de-facto standard, and is supported by many ATE, CAE, and semiconductor vendors and customers.

BSDL focuses on describing only 1149.1 compliant devices. It does not address 1149.1 at the board, system, or multi-chip module levels. BSDL also needs a few device-level features added to better support interactive debug. Manipulating test registers as a stream of bits is not easy, and determining the values on device pin buses or on fields within test data registers by looking at binary or hexadecimal output is impossible. ASSET 1.X Configuration Files, as simple as they were, addressed this issue.

ASSET marketing and support personnel began being questioned about when ASSET would support BSDL. In addition, customers also began asking for enhancements to the ASSET 1.X Configuration File language. They wanted compatibility with existing BSDL files to avoid rework, and they wanted more flexibility to add convenience features for interactive use.

In response, Hierarchical Scan Description Language was created. It addresses a number of deficiencies in both BSDL and Configuration Files. HSDL supports all the features of BSDL for industry compatibility, providing support for automatic test-pattern generation, validation, and synthesis tools. HSDL also supports all the convenience features of Configuration Files, providing the ability to describe boards, name subsets of test registers, create symbol tables for test registers or fields that use symbolic, named values, prevent illegal states from being established, and so forth.

In addition, HSDL supports new features that increase its powers in the areas covered by both BSDL and Configuration Files. For automated tools, HSDL includes new features such as those for describing different status values captured by a test register and designating them as "pass" or "fail" values. For interactive use, HSDL includes new features such as those for adding descriptive text to each item in the entity.

HSDL is a strict superset of BSDL. All statements that are part of HSDL device entities but not part of BSDL are optional. Thus, BSDL is acceptable input to an HSDL translator. HSDL device entities can be made acceptable to a BSDL translator simply by feeding them to the BSDL translator and deleting all the new statements that cause syntax errors, with no loss or change in meaning.

HSDL is similar to an ongoing effort in the EDIF Test Committee, which is seeking to define BST (Boundary-Scan Test) as an area of EDIF that will describe boundary-scan devices and boards. Much of BST and its predecessor EBST has been incorporated into HSDL. BST includes netlist information (because EDIF does) and test vector information for the device and board being

described. These items were omitted from HSDL to keep it focused on the original spirit of BSDL.

TI is supporting BSDL by making BSDL files available for TI devices, and has upgraded the first generation of ASSET to take BSDL as an input. TI and Teradyne have jointly developed the Serial Vector Format (SVF) and have placed this specification in the public domain for review. The industry did not have a common serial vector format standard, and SVF was developed to address this problem. With the introduction of HSDL, TI continues its effort of working with industry to develop or promote standard data formats that make it easier for customers to adopt 1149.1.

1.2. Scope

This document describes in detail the new statements of HSDL and their meaning. It does not describe BSDL, as many sources of information on BSDL already exist and HSDL does not change any of the original BSDL statements in any way.

1.3. Important Notice

This document defines Texas Instruments' (TI) Hierarchical Scan Description Language (HSDL). Texas Instruments reserves the right to make changes to this document without notice. TI advises its customers to obtain the latest version of the relevant information to verify that the information being relied upon is current.

Although TI hereby expressly disclaims any and all warranties whatsoever regarding the HSDL Syntax Specification, TI has published this document with the intent that HSDL will be seriously considered and adopted, in whole, by the test and measurement and semiconductor marketplaces.

To obtain copies of the current HSDL Syntax Specification, please contact the Test Technology Center of Texas Instruments:

Mailing Address:	Test Technology Center P.O. Box 869305 M/S 8407 Plano, TX 75086
Phone:	(214) 575-2577
X.400 E-mail Address:	/ADMD=MCI/PRMD=TI/C=US/G=ADAM/S=SHEPPARD
MCIMail ID:	5333075

TI welcomes all feedback in order to improve the HSDL Syntax Specification as necessary. Please direct comments to Adam Sheppard:

Phone:	(214) 575-2599
X.400 E-mail Address:	/ADMD=MCI/PRMD=TI/C=US/G=ADAM/S=SHEPPARD
MCIMail ID:	5333075

1.4. Nomenclature

ambiguous..... Ambiguous values have combinations of don't-care bits such that two such values could be selected as a replacement for the same bit pattern.

ASSETâ ..... A design debug and verification tool from ASSET InterTech, Inc.

arithmetic operator..... A constraint operator that performs multiplication, division, addition, or subtraction of two values.

associative operator..... An arithmetic, logical, or relational operator whose left and right operands can be exchanged without altering the result.

attribute..... A VHDL attribute is a named item and associates some value with another named object, such as the entity.

BIST ..... Built-in self test.

BSDL ..... Boundary-Scan Description Language. Hardware description language developed by Hewlett-Packard Company.

bus ..... A collection of bits from one or more test registers or buses, chosen to represent a natural grouping of data in a design. An HSDL bus as declared in the BUS_COMPOSITION attribute.

cell..... An 1149.1 scan cell designed into a device. A BSDL constant that defines the characteristics of a scan cell.

CLAMP ..... A new instruction in 1149.1 Supplement A that shifts through the BYPASS register, but drives device outputs based on the contents of the BOUNDARY register.

constraint..... An expression that is evaluated prior to each scan operation, that determines if the values to be shifted into the hardware will violate a design constraint and cause possible hardware damage. An HSDL constraint as declared in the CONSTRAINTS attribute.

concatenated test register A test register comprised of two or more other test registers.

configuration file..... A file format used by ASSET 1.X to describe devices and modules. Two formats were used: device configuration files and module configuration files. Each consisted of five different statement types describing such things as test registers, instructions, register fields, members, paths, module buses, and constraints.

data-register scan..... A *scan operation* that includes shifting new data into test data registers from the TDI buffers and shifting captured data out into the TDO buffers while the TAP controller is in Shift-DR state. The test controller automatically shifts a number of bits equal to the combined length of the selected test data registers of all devices in the scan path.

description..... An HSDL or BSDL file containing an entity describing a unit under test. Any comments attached to an item in the HSDL entity with a ..._DESCRIPTION attribute to increase understandability.

device entity ..... An HSDL device entity describes an 1149.1 device, with test registers, instructions, and a TAP.

device under test..... A unit under test that is a device.

don't-care..... A BIT value representing a bit whose value does not matter.

DR scan..... See *data-register scan*.

dynamic path..... A set of controlled paths that may be attached to the TMS signal of the scan path or have their TMS lines forced high or low. A hardware construct described by an HSDL DYNAMIC_PATH attribute.

dynamic path case ..... A certain configuration of a dynamic path, where one of the controlled paths is attached to the TMS line.

entity ..... An HSDL entity or entity identifier as defined in the BSDL entity statement.

external path..... A connector of any type where TDO leaves the module and TDI returns without a connection between. An item described by an HSDL EXTERNAL_PATH constant.

generic ..... A parameter to the entity, used in BSDL and HSDL to contain the package type used in the design.

HIGHZ..... A new instruction in 1149.1 Supplement A that shifts through the BYPASS register, but places all three-state or bidirectional device outputs in a high-impedance state.

HSDL ..... Hierarchical Scan Description Language. A superset of BSDL developed by Texas Instruments Incorporated.

identifier..... A sequence of characters used to name an object.

implementation-defined... Behavior that is described as *implementation-defined* is subject to interpretation by each implementation of a test controller. Each test controller may behave differently when an implementation-defined construct is used.

in port..... A port that describes an input pin.

inout port ..... A port that describes a bidirectional pin.

instruction ..... An HSDL instruction opcode described by the HSDL INSTRUCTION_OPCODE attribute.

instruction-register scan.. A *scan operation* that includes shifting new data into instruction registers from the TDI buffers and shifting captured data out into the TDO buffers while the TAP controller is in Shift-IR state. The test controller automatically shifts a number of bits equal to the combined length of the instruction registers of all devices in the scan path.

IR scan ..... See *instruction-register scan*.

logical operator ..... A constraint operator that performs operations on individual bits of a value.

LSB..... Least-significant bit. The bit whose binary value is 2^0.

member..... A device or module mounted on a module. An HSDL device or module entity contained as a member within a module entity.

member connection ..... An HSDL CONNECTIONS attribute used to connect something to each of the external paths of the members of a module.

module entity..... An HSDL module entity describes an 1149.1 module, which contains other device and module entities arranged along a scan path.

MSB..... Most-significant bit. The bit whose binary value is 2^{n-1}, where *n* is the number of bits in the value.

net ..... A set of device pins that are all interconnected.

netlist ..... A list of interconnections on a device or module. See *net*.

normal mode..... A normal-mode instruction does not affect the functionality of a device in any way.

operator ..... The person performing a test. In the context of constraints, a word or special symbol that represents a logical, relational, or arithmetic operation to perform.

out port ..... A port that describes an output pin.

package..... An HSDL package identifier declared in the PIN_MAP_STRING constant.

path ..... An HSDL path declared in an STATIC_PATH, DYNAMIC_PATH, or EXTERNAL_PATH constant.

path entry..... Something that can be listed in a static path or dynamic path declaration. A static path, dynamic path, external path, member, or member's external path.

port..... Represents a named device pin. An HSDL port declared in the PORT statement.

primary scan path ..... A scan path that is a complete TDI-to-TDO connection from a TDI input of the module to a TDO output of the module, including all member devices, member modules, external paths, static paths, and dynamic paths that make up the primary scan path.

private instruction..... An instruction opcode that performs proprietary or dangerous operations when shifted into the instruction register. A private instruction cannot be shifted in by the test controller.

pure test connector..... A connector whose TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET ports are all in ports so that a test controller can be plugged into it.

pure UUT connector..... A connector whose TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET ports are all out ports so that a scannable UUT can be plugged into it.

relational operator..... A constraint operator that compares the equality or ordering of two values.

reserved word..... A language element (identifier) that is used as part of the language and that cannot be used to name an object in a declaration.

scan operation..... An operation that manipulates the IEEE Std 1149.1 test bus to perform test operations.

scan path ..... A serial TDI-to-TDO connection of devices and modules forming a complete 1149.1 test bus.

Scan Path Linker ..... A device (SN74ACT8997) from Texas Instruments that multiplexes secondary scan paths in and out of the primary scan path.

selected test connector.... The test connector that the test controller is plugged into for the duration of the test.

semantics..... Rules that specify the restrictions and meaning of a computer language.

standard practice ..... Rules followed when writing VHDL for use in BSDL or HSDL entities that simplify the computer programs that translate BSDL and HSDL.

static path ..... A serial TDI-to-TDO connection of member devices, member modules, and scan paths that all share the same TAP control signals. A scan path described by an HSDL STATIC_PATH constant.

symbol ..... An identifier that represents a list of one or more numbers or patterns. An HSDL symbol declared in the SYMBOL_TABLE constant.

symbol table ..... A collection of related symbols. An HSDL symbol table declared in the SYMBOL_TABLE constant.

syntax..... The rules that describe what characters to write to form a program. The program is meaningless without semantics.

TAP control signals..... For the purposes of this specification, the TAP control signals are Test Clock (TCK), described by the TAP_SCAN_CLOCK attribute; Test Mode Select (TMS), described by the TAP_SCAN_MODE attribute, and Test Reset (TRST), described by the TAP_SCAN_RESET attribute.

TDI symbol..... A symbol that may be used to represent a value to shift into a UUT, but not to represent a value that is shifted out.

TDO symbol..... A symbol that may be used to represent a value shifted out of a UUT, but not to represent a value to shift in.

test connector ..... A connector that a test controller can be plugged into. See *pure test connector.*

test controller ..... The computer system, hardware, and/or software used to test the unit under test by applying stimuli and observing the response.

test mode ..... A test-mode instruction affects the functionality of a device in some way, usually by overriding the device outputs or the system logic inputs.

test register ..... An HSDL test register declared in the HSDL REGISTER_ACCESS attribute.

undefined ..... Behavior that is described as *undefined* causes unpredicable behavior in a test controller. An HSDL description may not rely on undefined behavior in a construct.

unit under test ..... The HSDL entity, either a device or module, that is being tested by the test controller.

user-defined package ..... A BSDL package and package body that defines cells for use in the BOUNDARY_REGISTER attribute.

UUT ..... See *unit under test*.

UUT connector ..... A connector that a scannable UUT can be plugged into. See *pure UUT connector*.

VHDL ..... VHSIC Hardware Description Language

VHSIC ..... Very High Speed Integrated Circuit

violated constraint ..... A constraint that evaluates true. A violated constraint indicates that a hardware design constraint would be violated, resulting in possible damage to the unit under test.

1.5. Related Documents

Advanced Logic and Bus Interface Logic. Texas Instruments, 1991.

ASSET Scan-Based Diagnostics User's Guide. Texas Instruments, 1991.

HP Boundary-Scan Tutorial and BSDL Reference Guide. Hewlett-Packard Company, 1990.

IEEE Std 1149.1-1990, Test Access Port and Boundary-Scan Architecture. IEEE, 1990.

IEEE Std 1149.1-1990 Supplement A. IEEE, 1992.

IEEE Std 1149.1-1990 Unofficial Supplement B, Boundary-Scan Description Language. 1149.1 BSDL Working Group, 1992.

IEEE Std 1076-1987, VHDL Language Reference Manual. IEEE, 1987.

1.6. Constraints

HSDL supports only IEEE Std 1149.1-1990 compliant devices and modules.

A great deal of leniency is provided when describing modules, as allowed by 1149.1. Some pathologically complex modules cannot be described by HSDL in the interests of keeping the language manageable. Fully describing all modules requires a netlist for the scan paths and a simulation model for the dynamic scan paths, which is considered beyond the scope of HSDL.

The language is designed in the spirit of BSDL, and hence the pros and cons of BSDL syntax remain.

HSDL may undergo changes based on the inputs received from the industry when this document is circulated.

2.
Introduction to Hierarchical Scan Description Language

2.1. Language Elements

2.1.1. The VHDL Statement Subset

HSDL employs the same subset of VHDL statements used by BSDL. However, the VHDL statements are used in more flexible ways in HSDL than in BSDL. For example, BSDL only attaches attributes to the entity. HSDL attaches attributes to the entity, the ports, and to Symbol Table and Path constants.

2.1.2. The Entity, the Package, and the Package Body

These three different VHDL items are used in the same manner in HSDL as they are in BSDL. HSDL has two types of entities, however: device entities and module entities. A *device entity* describes an 1149.1 device, with test registers, instructions, and a TAP. A *module entity* describes an 1149.1 module, which contains other device and module entities arranged along a scan path.

A user-defined package for creating new cell types is of no use in a module entity, so a module entity cannot contain user-defined packages.

2.2. The Entity Description for a Device

A device entity in HSDL follows the same order and syntax as an entity in BSDL. The HSDL statements are optional and may appear only at predefined points in the entity.

```
entity My_IC is        -- an entity for my IC
        Generic Parameter
        Logical Port Description
        Use Statement(s) *
        [Optional Device Description] *
        [Optional Port Description(s)] *
        Package Pin Mapping
        Scan Port Identification
        TAP Description
                Device-Dependent Descriptions **
                [Optional Data Register(s)] **
                [Optional Symbol Table(s)] *
                Register Access
                [Optional Register Information] *
        Boundary Register Description
        [Optional Boundary Register Symbol(s)] *
        [Optional Bus Description(s)] *
        [Optional Constraint Description(s)] *
        [Optional Design Warning] **
end My_IC;
```

An asterisk (*) designates areas of HSDL that are new or that were enhanced from BSDL. Two asterisks (**) designate areas of HSDL that are BSDL but that were not completely discussed in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*.

The order of elements shown above is a required standard practice in order to simplify non-VHDL applications, like BSDL or HSDL translators. Each element of the entity is examined and discussed in the subsections that follow.

2.2.1. Generic Parameter

Unchanged from BSDL. See *generic (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The generic parameter is mandatory.

2.2.2. Logical Port Description

Unchanged from BSDL. See *port (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The port statement is mandatory.

2.2.3. Use Statement(s)

The **use** statement is primarily unchanged from BSDL. See *use (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The statement use STD_1149_1_1990.all; is mandatory and must appear first, followed by the optional user-defined packages.

A new package has been defined for HSDL device entities that declares all attributes and subtypes used by a device entity. In addition, it identifies the entity as an HSDL device, and not simply a BSDL device. The new package is HSDL_device, as shown in this example.

```
use HSDL_device.all;  -- identifies the entity as a device with HSDL extensions
```

2.2.4. Optional Device Description

This optional statement describes the device. It appears primarily for documentation purposes. Ideally, it should describe the functionality of the device in sufficient detail to meet the requirements for documenting devices found in Chapter 12 of IEEE Std 1149.1-1990. As a description, it may be displayed by interactive tools or by the test controller itself as a description of the device under test.

```
attribute DEVICE_DESCRIPTION of ttl74bct8374 : entity is
   "SN74BCT8374 is a Scan Test Devices with Octal D-Type Edge-Triggered " &
   "Flip-Flops. In normal mode, on the positive transition of CLK the Q " &
   "outputs take on the logic levels setup up at the D inputs.  The "     &
   "output enable OC_NEG is used to place the Q outputs in the "         &
   "high-impedance state, but does not affect the internal operations "   &
   "of the flip-flop.";
```

2.2.5. Optional Port Description(s)

Next comes optional statements for describing the ports of the device. The same syntax is used in the HSDL module entity for describing the ports of the module. The normal function of each port should be described in sufficient detail so that the operator can understand how the device normally operates.

Descriptions need not be supplied for the ports identified as scan ports by the TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET attributes (see Scan Port Identification). The description of these ports is already known and is described in full by IEEE Std 1149.1-1990. Suitable descriptions are synthesized by the HSDL translator for these ports if no description has been supplied for them.

```
attribute PORT_DESCRIPTION of OC_NEG : signal is
   "Output control of the device.  All pins on the output bus Q can be " &
   "set to high-impedance by placing a 1 on the OC_NEG pin.  Disabling " &
   "the output bus Q has no effect on the internal operation of the "    &
   "flip-flops.";
```

2.2.6. Package Pin Mapping

Unchanged from BSDL. See *attribute PIN_MAP (BSDL)* and *constant-(entity) (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*. The PIN_MAP attribute is mandatory and must appear first, followed by one or more PIN_MAP_STRING constants.

2.2.7. Scan Port Identification

Unchanged from BSDL. See *attribute TAP_SCAN_CLOCK (BSDL), attribute TAP_SCAN_IN (BSDL), attribute TAP_SCAN_MODE (BSDL), attribute TAP_SCAN_OUT (BSDL)*, and *attribute TAP_SCAN_RESET (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The statements may appear in any order. All but the TAP_SCAN_RESET attribute are required.

2.2.8. Device-Dependent Descriptions

All device-dependent descriptions are retained unchanged from BSDL. See *attribute INSTRUCTION_LENGTH (BSDL)*, *attribute INSTRUCTION_OPCODE (BSDL), attribute INSTRUCTION_CAPTURE (BSDL), attribute INSTRUCTION_DISABLE (BSDL),* and *attribute INSTRUCTION_PRIVATE (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The INSTRUCTION_LENGTH attribute is mandatory and must appear first, followed by the mandatory INSTRUCTION_OPCODE attribute and the mandatory INSTRUCTION_CAPTURE attribute. The optional INSTRUCTION_DISABLE and INSTRUCTION_PRIVATE attributes may appear next in any order. In addition, the recently introduced INSTRUCTION_GUARD attribute of BSDL is also available.

Private instructions identified by the INSTRUCTION_PRIVATE attribute cannot be shifted into the device using HSDL. Private instructions invoke proprietary or dangerous test operations in the device and cannot safely be used.

The optional INSTRUCTION_GUARD attribute of BSDL identifies an opcode that places the Bypass Register between TDI and TDO, and drives the outputs of the device using the previous contents of the Boundary-Scan Register. This is the behavior defined by the CLAMP instruction of IEEE Std 1149.1-1990 Supplement A. This statement identifies the opcode to software. An example of this type of statement appears in the BSDL for the BCT octals from Texas Instruments.

```
attribute INSTRUCTION_GUARD of ttl74bct8374 : entity is "SETBYP";
```

The optional INSTRUCTION_NORMAL and INSTRUCTION_TEST attributes of BSDL identify an opcode as either a normal-mode or test-mode instruction. Normal-mode instructions do not affect the normal operation of the device in any way. Test-mode instructions affect the normal operation, but the exact effect is unspecified. All or part of the normal operation of the device is suspended and replaced with a test operation.

The instructions BYPASS, SAMPLE, IDCODE, and USERCODE, defined by 1149.1, must be normal-mode instructions and do not need to be listed in the INSTRUCTION_NORMAL attribute. The instructions EXTEST, INTEST, RUNBIST, HIGHZ, and CLAMP, defined by

1149.1 and Supplement A, must be test-mode instructions and do not need to be listed in the INSTRUCTION_TEST attribute. If an instruction is not defined by 1149.1 and is not listed in either the INSTRUCTION_NORMAL or INSTRUCTION_TEST attributes, its effect on the hardware is unknown.

```
attribute INSTRUCTION_NORMAL of ttl74bct8374 : entity is
   "BYPASS, SAMPLE, READBN, CELLTST, SCANCN";

attribute INSTRUCTION_TEST of ttl74bct8374 : entity is
   "EXTEST, INTEST, TRIBYP, SETBYP, RUNT, READBT, TOPHIP, SCANCT";
```

An optional HSDL statement, INSTRUCTION_DESCRIPTION, follows the BSDL instruction statements and can be used to provide descriptions of each instruction listed in the INSTRUCTION_OPCODE attribute. The descriptions may be displayed by the test controller to help the operator understand the purpose and function of each instruction.

Descriptions need not be supplied for the standard instructions required by IEEE Std 1149.1-1990 and by Supplement A. These instructions include EXTEST, SAMPLE/PRELOAD, BYPASS, INTEST, RUNBIST, IDCODE, USERCODE, CLAMP, and HIGHZ. The description of these instructions is already known and is described in full by IEEE Std 1149.1-1990. Suitable descriptions are synthesized by the HSDL translator for these instructions if no description has been supplied for them. If instruction descriptions are used, it is recommended that descriptions for INTEST and RUNBIST be supplied rather than synthesized, as the operation of these instructions can vary from device to device.

```
attribute INSTRUCTION_DESCRIPTION of My_IC : entity is
      "BYPASS  ('Select BYPASS register in normal mode.')," &
      "EXTEST  ('Select Boundary-Scan register in test mode; control device" &
               "inputs and outputs using the contents of the Boundary.')," &
      "SAMPLE  ('Select Boundary-Scan register in normal mode; sample "     &
               "device inputs and outputs into the Boundary.')";
```

2.2.9. Optional Data Registers(s)

Unchanged from BSDL. See *attribute IDCODE_REGISTER (BSDL)* and *attribute USERCODE_REGISTER (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*. The optional IDCODE_REGISTER attribute must appear first, followed by the optional USERCODE_REGISTER attribute.

2.2.10. Optional Symbol Table(s)

Next the optional HSDL statements describing symbol tables may appear. Symbol tables are groups of related symbol names. Each symbol name stands for one or more symbol values.

Symbol tables are useful for describing values by name rather than by number. IEEE Std 1149.1-1990 already contains one example of a test register with an associated symbol table: the instruction register. The standard never refers to "instruction 0"; instead it refers to "the EXTEST instruction". Other test registers or subsets of test registers often have named opcodes defined in the data sheet to describe the possible values in a mnemonic way.

Symbol tables are defined separately from the elements (test registers, buses, ports) that they are associated with. If more than one test register in the entity uses the same opcodes, only one symbol table needs to be defined. That symbol table is then associated with each of the test registers.

```
constant BCR_Opcodes : SYMBOL_TABLE :=
-- symbol   (value, ..., value)
   "SAMTOG  (00)," &
   "PRPG    (01)," &
   "PSA     (10)," &
   "PSAPRPG (11)";
```

Multiple symbol tables may be created. The symbol table itself must be defined before any of the attributes for it, because all the attributes refer to both the symbol table name and the symbol names. The optional attributes for a symbol table may appear in any order following the symbol table.

Consider the uses for symbols. Symbols can be used in place of values being shifted into the device, and symbols could be displayed in place of values shifted out of the device. This suggests two possible types of symbols: symbols that can be shifted in and symbols that can be shifted out. HSDL allows symbols to be classified as used only for shifting in (SYMBOL_OF_TDI), only for shifting out (SYMBOL_OF_TDO), or both (the default).

If a symbol name is not listed as a SYMBOL_OF_TDI or as a SYMBOL_OF_TDO for the symbol table, it is considered to be both. The symbol can be used to represent a value to shift into the device, and it can be displayed instead of a value shifted out of the device.

Why make this distinction? The 1149.1 standard contains a precedent: the instruction register captures and shifts out a status value, but the value shifted into the instruction register and latched is an instruction. Two different types of patterns with two totally different meanings are thus used by the instruction register. The status values captured and shifted out are SYMBOL_OF_TDO, whereas the instructions shifted in and latched are SYMBOL_OF_TDI.

```
constant INSTRUCTIONS : SYMBOL_TABLE :=
   "EXTEST      (000)," &
   "SAMPLE      (001)," &
   "BYPASS      (1XX, 01X)," &
   "GOOD_STATUS (001)," &
   "BAD_STATUS  (XX0)";
attribute SYMBOL_OF_TDI of INSTRUCTIONS : constant is
   "EXTEST, SAMPLE, BYPASS";
attribute SYMBOL_OF_TDO of INSTRUCTIONS : constant is
   "GOOD_STATUS, BAD_STATUS";
```

Symbols designated SYMBOL_OF_TDO (either explicitly, or by defaulting to both TDI/TDO) cannot have ambiguous values. *Ambiguous values* have combinations of don't-care bits such that two symbols could be selected as a replacement for the same bit pattern. If TDO symbols were allowed to be ambiguous, the test controller would potentially need to select an infinite number of symbols for replacement, and thus the output would appear as a long list of symbol names rather than just one.

Descriptions can be associated with each symbol. The descriptions should indicate the purpose, function, and meaning of the symbol. The test controller may display this description to help the operator select a symbol for shifting in, or interpret the meaning of a symbol that has been shifted out.

```
attribute SYMBOL_DESCRIPTION of BCR_Opcodes : constant is
-- symbol  (description)
   "SAMTOG ('Samples device inputs on input bus; toggles device outputs "  &
            "from output bus.'),"                                          &
   "PRPG   ('Conducts 16-bit Pseudo-Random Pattern Generation using the "  &
            "contents of the input and output buses of the Boundary-Scan " &
            "register as an initial value.  The Q outputs of the device "  &
            "will be set to the value in the output bus.  A new pattern "  &
            "is generated on each TCK in Run-Test/Idle state.'),"          &
   "PSA    ('Conducts 16-bit Parallel Signature Analysis using the "       &
            "contents of the input and output buses of the Boundary-Scan " &
            "register as an initial value.  The Q outputs of the device "  &
            "will be set to the value in the output bus.  A new checksum"  &
            "is generated on each TCK in Run-Test/Idle state.'),"          &
   "PSAPRPG('Simultaneous 8-bit PSA on the D inputs and 8-bit PRPG on "    &
            "the Q outputs.')";
```

Finally, a TDO or TDI/TDO symbol may be designated as the "default" TDO symbol. That symbol will be displayed when no symbol's values directly match the bit pattern shifted out.

```
attribute SYMBOL_DEFAULT of BIT_RESULTS : constant is "BIT_FAILED";
```

2.2.11. Register Access

Unchanged from BSDL. See *attribute REGISTER_ACCESS (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*. The REGISTER_ACCESS attribute is mandatory.

2.2.12. Optional Register Information

HSDL provides the ability to define several additional types of information about the test registers in a device. Test register concatenation, symbol tables, descriptions, Capture-DR values, Test-Logic-Reset values, and privacy can all be described. The new attributes can occur in any order following the REGISTER_ACCESS attribute.

A major deficiency of BSDL from a data-integrity viewpoint is its inability to describe concatenated test registers. This hardware construct, where a third test register is formed from two other test registers (for example), is allowed by IEEE Std 1149.1-1990. However, in BSDL the third test register must be described as a completely separate register, sharing no hardware whatsoever. Conflicts can arise between what the test controller believes is currently in the hardware and what the *hardware* believes is currently in the hardware.

```
attribute REGISTER_COMPOSITION of My_IC : entity is
--  reg  (reg, reg[bit], reg[bit,bit], ...)
   "BCR  (REPS[1], REPS[3])," &
   "EMUL (REPS[5,12])";
```

Each test register may have a symbol table associated with it, providing a set of symbols that can be shifted into the test register and used to represent values shifted out of the test register.

```
attribute REGISTER_SYMBOLS of My_IC : entity is
--  reg  (symbol_table)
   "BCR  (BCR_SYMBOLS)," &
   "EMUL (EMUL_SYMBOLS)";
```

Each test register can have a description associated with it.

```
attribute REGISTER_DESCRIPTION of ttl74bct8374 : entity is
--  reg      (description)
   "BOUNDARY ('The Boundary-Scan register contains the cells attached "  &
             "to the pins of the device.')),"                           &
   "BYPASS   ('The BYPASS register is a one-bit register that always "   &
             "loads a 0 in the Capture-DR state.  It is used to speed "  &
             "up scanning to the UUT when a device is not used during "  &
             "a test.'),"                                                &
   "BCR      ('The Boundary Control Register is a design-specific "      &
             "test data register used to specify the test operation - "  &
             "PSA, PRPG - that will be performed by the RUNT "           &
             "instruction.')";
```

If the values captured by a test register in the Capture-DR state can be considered status values, they can be placed in the REGISTER_CAPTURE attribute. Good and bad status values can be identified, along with a description of the value. The description may be displayed for the operator to better understand the cause of success or failure.

```
attribute REGISTER_CAPTURE of ttl74bct8374 : entity is
--  reg    (capturevalue, pass/fail, description)
   "BYPASS (0, pass, 'The BYPASS register is working correctly.'),"    &
   "EMUL   (OKON, pass,  'The Emulation logic is working correctly; "  &
                           "emulation is enabled.'),"                  &
   "EMUL   (OKOFF, pass, 'The Emulation logic is working correctly; "  &
                           "emulation is disabled.'),"                 &
   "EMUL   (ERROR, fail, 'The Emulation logic is not working.')";
```

If a test register loads a constant value during Test-Logic-Reset, that value can be listed in the REGISTER_RESET attribute. This is a data integrity feature of HSDL.

```
attribute REGISTER_RESET of ttl74bct8374 : entity is
--  reg         (resetvalue)
   "INSTRUCTION (BYPASS)," &
   "BCR         (PSA)";
```

2.2.13. Boundary Register Description

Unchanged from BSDL. See *attribute BOUNDARY_CELLS (BSDL), BOUNDARY_LENGTH (BSDL)*, and *BOUNDARY_REGISTER (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*. These statements are mandatory and must appear in the order they are listed here.

2.2.14. Optional Boundary Register Symbol(s)

The optional HSDL BOUNDARY_SYMBOLS attribute may appear following all BSDL boundary register statements. The BOUNDARY_SYMBOLS attribute is used to associate symbol tables with certain types of cells attached to ports of the device. For example, a cell on a control pin could have a symbol table associated with it that defined ENABLE and DISABLE values.

```
attribute BOUNDARY_SYMBOLS of ttl74bct8374 : entity is
   "CLK    (input, clk_symbols)," &
   "OC_NEG (input, oc_symbols)";
```

2.2.15. Optional Bus Description(s)

The optional HSDL bus descriptions define buses, logical groupings of subsets of bits from one or more test registers or buses. The most common use for buses in a device is to define the fields contained within various test registers. For example, the IdCode Register consists of version number, part number, and manufacturer fields. Each of these test register subsets could be described using a bus.

```
attribute BUS_COMPOSITION of My_IC : entity is
   "version[4]       (idcode[31,28])," &
   "part_number[16]  (idcode[27,12])," &
   "manufacturer[11] (idcode[11, 1])";

attribute BUS_COMPOSITION of ttl74bct8374 : entity is
--  bus[length] (reg, reg[bit], reg[bit,bit], ...)
   "inputs[8]  (BOUNDARY[ 8,15])," &
   "outputs[8] (BOUNDARY[ 0, 7])," &
   "aux[2]     (BOUNDARY[16,17])," &
   "lfsr[16]   (BOUNDARY[ 0,15])";
```

A bus may have a symbol table associated with it. Again using the IdCode Register as an example, the manufacturer field could have a large number of symbols associated with it, describing the manufacturers.

```
attribute BUS_SYMBOLS of My_IC : entity is
   "manufacturer (List_Of_Manufacturers)";

attribute BUS_SYMBOLS of ttl74bct8374 : entity is
--  bus (symbol_table)
   "aux (AUX_SYMBOLS)";
```

Each bus in the design can optionally have a description associated with it.

```
attribute BUS_DESCRIPTION of My_IC : entity is
   "version      ('The version number of the device.')," &
   "part_number  ('The part number of the device.'),   " &
   "manufacturer ('The manufacturer''s JEDEC code.')";

attribute BUS_DESCRIPTION of ttl74bct8374 : entity is
--  bus    (description)
   "inputs  ('The eight cells of the Boundary-Scan register "  &
            "that are connected to the D inputs are combined " &
            "to form the input bus.')."                        &
   "outputs ('The eight cells of the Boundary-Scan register "  &
            "that are connected to the Q outputs are "         &
            "combined to form the output bus.')."              &
   "aux     ('The two cells of the Boundary-Scan register "    &
            "that are connected to the CLK and OC_NEG pins "   &
            "are combined to form the aux bus.')."             &
   "lfsr    ('A bus used by the bcr.')";
```

2.2.16. Optional Constraint Description(s)

The design of hardware inevitably involves some constraints, illegal conditions that cannot be established without unpredictable or damaging results. Constraints consist of logical and relational expressions that are evaluated before each scan operation. If any of the constraints are TRUE, an illegal condition has been created and no scanning can be performed until the constraint is FALSE.

```
attribute CONSTRAINTS of My_IC : entity is
--  constraint   (expression)
   "shared_cells (BCR = SAMTOG and EMUL = STEP)," &
   "bad_device   (BYPASS = 1)";
```

Constraints may also have a description associated with them. The description may be displayed by the test controller when the constraint has been violated to aid the operator in correcting the problem.

```
attribute CONSTRAINT_DESCRIPTION of My_IC : entity is
--  constraint   (description)
   "shared_cells ('Physical cells are shared by the Boundary Control " &
                 "Register and the EMUL emulation register.')."       &
   "bad_device   ('The device fails if BYPASS is a 1 (not true)!')";
```

2.2.17. Optional Design Warning

Unchanged from BSDL. The DESIGN_WARNING attribute is not discussed in the *BSDL Syntax* chapter of the *HP Boundary-Scan Tutorial and BSDL Reference Guide*, but is described in the *Introduction to BSDL* chapter under the section *Miscellaneous Declarations.* This statement is optional.

2.3. The Entity Description for a Module

A *module* is defined as a collection of devices and other modules whose TAP ports are connected to define a scan path. Since modules can contain other modules as members, a hierarchy is formed (hence the name **Hierarchical** Scan Description Language). The simplest module represents a board containing one or more devices connected into a single scan path. More complicated modules can represent multi-chip modules, backplanes, boxes, subsystems, or entire systems.

A module entity in HSDL uses much of the same syntax as a device entity. New statements have been added to list the device and module entities mounted on the module entity and to describe how these member entities are interconnected. Existing BSDL statements have been removed where they did not apply to the description of modules.

A module does not have to directly represent a physical board, box, etc. A module entity can be used to define any collection of members that is convenient. Usually, the most convenient

module entity is one that corresponds directly to a real board, but occasionally other divisions may be useful.

Modules and devices have many similiarities. Both have ports, different packaging options, Test Access Port(s), symbol tables, buses, and constraints. The main differences are that a device consists of test registers connected in a certain way and controlled by instructions, whereas a module consists of member devices and modules connected in a certain way and controlled by scan paths.

```
entity My_Board is      -- an entity for my board
        Generic Parameter
        Logical Port Description
        Use Statement(s) *
        [Optional Module Descriptions] *
        [Optional Port Description(s)] *
        Package Pin Mapping
        Scan Port Identification
        [Optional Member Description(s)] *
        [Optional Symbol Table Description(s)] *
        [Optional Bus Description(s)] *
        Path Description(s) *
                [ Optional External Path Declaration(s)] *
                Static Path Declaration(s) *
                [ Optional Dynamic Path Declaration(s)] *
        [ Optional Member Connections] *
        [Optional Constraint Description(s)] *
        [Optional Design Warning] **
end My_Board;
```

An asterisk (*) designates areas of HSDL that are new or that were enhanced from BSDL. Two asterisks (**) designate areas of HSDL that are BSDL but that were not completely discussed in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*.

In a sense, all the statements of a module entity are new, because BSDL could not describe modules. However, the syntax and meaning of many of the statements is the same. For this reason, an HSDL module entity **cannot** be converted into BSDL, because no equivalent BSDL exists.

The order of elements shown above is a required standard practice in order to simplify non-VHDL applications, like BSDL or HSDL translators. Each element of the entity is examined and discussed in the subsections that follow.

2.3.1. Generic Parameter

Unchanged from BSDL. See *generic (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The generic parameter is mandatory.

2.3.2. Logical Port Description

Unchanged from BSDL. See *port (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The port statement is mandatory.

2.3.3. Use Statement(s)

The **use** statement is primarily unchanged from BSDL. See *use (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The statement use STD_1149_1_1990.all; is mandatory and must appear first, followed by the HSDL module package.

HSDL modules cannot use user-defined packages. These packages only contain cell definitions for the boundary register, and a module does not have a boundary register.

A new package has been defined for HSDL module entities that declares all attributes and subtypes used by a module entity. In addition, it identifies the entity as an HSDL module. The new package is HSDL_module, as shown in this example.

```
use HSDL_module.all;  -- identifies the entity as an HSDL module
```

2.3.4. Optional Module Description

This optional statement describes the module. It appears primarily for documentation purposes. Ideally, it should describe the functionality of the module in sufficient detail to help the test engineer or operator understand the function, purpose, and usage of the module. As a description, it may be displayed by interactive tools or by the test controller itself as a description of the module under test.

```
attribute MODULE_DESCRIPTION of gdm: entity is
   "The General Demonstration Module (GDM) is an example board that was " &
   "designed to demonstrate some of the capabilities of an 1149.1 UUT.";
```

2.3.5. Optional Port Description(s)

Same as the HSDL device entity.

2.3.6. Package Pin Mapping

Unchanged from BSDL. See *attribute PIN_MAP (BSDL)* and *constant-(entity) (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide*. The PIN_MAP attribute is mandatory and must appear first, followed by one or more PIN_MAP_STRING constants.

2.3.7. Scan Port Identification

Syntax is unchanged from BSDL. See *attribute TAP_SCAN_CLOCK (BSDL), attribute TAP_SCAN_IN (BSDL), attribute TAP_SCAN_MODE (BSDL), attribute TAP_SCAN_OUT (BSDL)*, and *attribute TAP_SCAN_RESET (BSDL)* in the *HP Boundary-Scan Tutorial and BSDL Reference Guide.* The statements may appear in any order. All but the TAP_SCAN_RESET attribute are required.

IEEE Std 1149.1-1990 compliant devices have a single Test Access Port (TAP). Modules, however, can have many TAPs and many scan paths. Two different scan paths may actually share some of the ports of their TAPs.

The TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET attributes must be used to identify **every** port that can be connected to a member device or module. The 1149.1 standard itself shows some examples of boards with multiple TMS or multiple TDI/TDO signals.

A TAP_SCAN_IN port must be an **in** port. Likewise, a TAP_SCAN_OUT port must be an **out** port. The TAP control lines (TAP_SCAN_CLOCK, TAP_SCAN_MODE, TAP_SCAN_RESET) can be all **in** ports, all **out** ports, or all **inout** ports, depending on the

hardware design. If the connector these ports reside on is designed only to allow a scannable UUT to be plugged in to the connector, it is a *pure unit-under-test connector*, and the ports are all **out**. If the connector these ports reside on is designed only to allow a test controller to be plugged in to the connector, it is a *pure test connector*, and the ports are all **in** (like BSDL device entities). If the connector is designed to allow either a scannable UUT or a test controller to be plugged in, it may be a test connector or a unit-under-test connector, and the ports are all **inout**. In this way, there is no "magic" connector on the module that the test controller must always be plugged into.

The TAP_SCAN_CLOCK frequency specifies the maximum TCK operating frequency of the module itself, not including the devices mounted on the module. The module's physical design may have frequency limitations that place its maximum operating frequency far below that of the member devices and member modules mounted on the module. A member device or member module may likewise have a lower maximum operating frequency than the module it is mounted on. The test controller must use the lowest of all TAP_SCAN_CLOCK frequencies to accurately determine the maximum TCK frequency used to test the UUT.

Consider the following example board with five different TAPs:

```
-- Scan Port Identification
attribute TAP_SCAN_IN    of TDI  : signal is true;
attribute TAP_SCAN_MODE  of TMS  : signal is true;
attribute TAP_SCAN_OUT   of TDO  : signal is true;
attribute TAP_SCAN_RESET of TRST : signal is true;
attribute TAP_SCAN_CLOCK of LCLK : signal is (20.0e6, BOTH);

attribute TAP_SCAN_IN    of J2_TDO : signal is true;
attribute TAP_SCAN_MODE  of J2_TMS : signal is true;
attribute TAP_SCAN_OUT   of J2_TDI : signal is true;
attribute TAP_SCAN_CLOCK of J2_TCK : signal is (20.0e6, BOTH);

attribute TAP_SCAN_IN    of J3_TDO : signal is true;
attribute TAP_SCAN_MODE  of J3_TMS : signal is true;
attribute TAP_SCAN_OUT   of J3_TDI : signal is true;
attribute TAP_SCAN_CLOCK of J3_TCK : signal is (20.0e6, BOTH);

attribute TAP_SCAN_IN    of J4_TDO : signal is true;
attribute TAP_SCAN_MODE  of J4_TMS : signal is true;
attribute TAP_SCAN_OUT   of J4_TDI : signal is true;
attribute TAP_SCAN_CLOCK of J4_TCK : signal is (20.0e6, BOTH);

attribute TAP_SCAN_IN    of J5_TDO : signal is true;
attribute TAP_SCAN_MODE  of J5_TMS : signal is true;
attribute TAP_SCAN_OUT   of J5_TDI : signal is true;
attribute TAP_SCAN_CLOCK of J5_TCK : signal is (20.0e6, BOTH);
```

2.3.8. Optional Member Description(s)

Next come the declarations of all the members of the module. *Members* represent devices or other modules that are "mounted" on the module. Usually members represent components, but some boards may contain scannable daughterboards, card slots, etc. that require member modules to describe them.

A member declaration names all the parts mounted on the module, selecting an entity and a packaging option for each. The MEMBERS attribute does not indicate whether the member is a device or a module - it does not matter, and can be determined by the HSDL translator after reading the member's entity.

```
attribute MEMBERS of gdm : entity is
   "u19 (ttl74bct8244, NT_PACKAGE)," &
   "u21 (ttl74bct8244, NT_PACKAGE)," &
   "u9  (ttl74bct8244, NT_PACKAGE)," &
   "u8  (ttl74bct8244, NT_PACKAGE)," &
   "u1  (ttl74bct8244, NT_PACKAGE)," &
   "u22 (ttl74bct8373, NT_PACKAGE)," &
   "u20 (cf93279,      FK_PACKAGE)";
```

Each member can have a description associated with it. The difference between a member description and a device or module description is that a device or module description describes the particular **type** of device or module, whereas a member description describes the particular **usage** of the member within the module. For example, a device description may indicate that a device is a buffer, but a member description will indicate that the buffer is used to buffer signals being fed to a backplane bus.

```
attribute MEMBER_DESCRIPTION of PC : entity is
   "u1  ('CPU.')," &
   "u2  ('The SPL is used to isolate motherboard RAM and the 32-bit slot " &
        "from the rest of the test logic to improve failure analysis.')," &
   "u3  ('Data bus between i486 and memory, lines d0-d7.')," &
   "u4  ('Data bus between i486 and memory, lines d8-d15.')," &
   "u5  ('Data bus between i486 and memory, lines d16-d23.')," &
   "u6  ('Data bus between i486 and memory, lines d24-d32.')," &
   "u7  ('Chipset override control logic.')," &
   "u8  ('Hard disk controller override logic.')," &
   "u9  ('DSP used to oversee multimedia functions.')," &
   "u10 ('Keyboard override control logic.')" ;
```

2.3.9. Optional Symbol Table Description(s)

Same as the HSDL device entity.

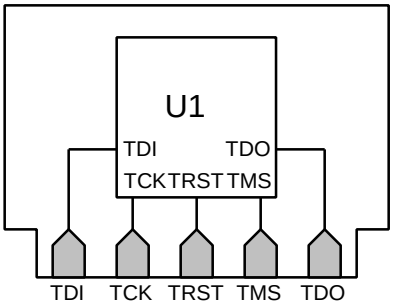
2.3.10. Optional Bus Description(s)

Same as the HSDL device entity. Buses in an HSDL module can be built of module buses, member module buses, member device buses, and member device test registers.

2.3.11. Path Descriptions

Module paths can be a confusing subject. Module paths are intended to describe the netlist of TAP signals (scan paths) on the board. Some introduction is in order.

The simplest module to discuss consists of one member device wired to a connector.



The connector has the five TAP signals (TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET) each wired appropriately to the member device. What are the important characteristics of this module?

The module contains one member, which can be defined in the MEMBERS attribute. But how is that member device controlled? A netlist of some sort needs to be constructed, connecting the signals that can be fed to the board by the test controller to the TAP ports of the device. A simple netlist just for the scan path can expand far beyond the desire of any engineer to type it in. Just the connections of TDI to TDO result in the creation of many nets.

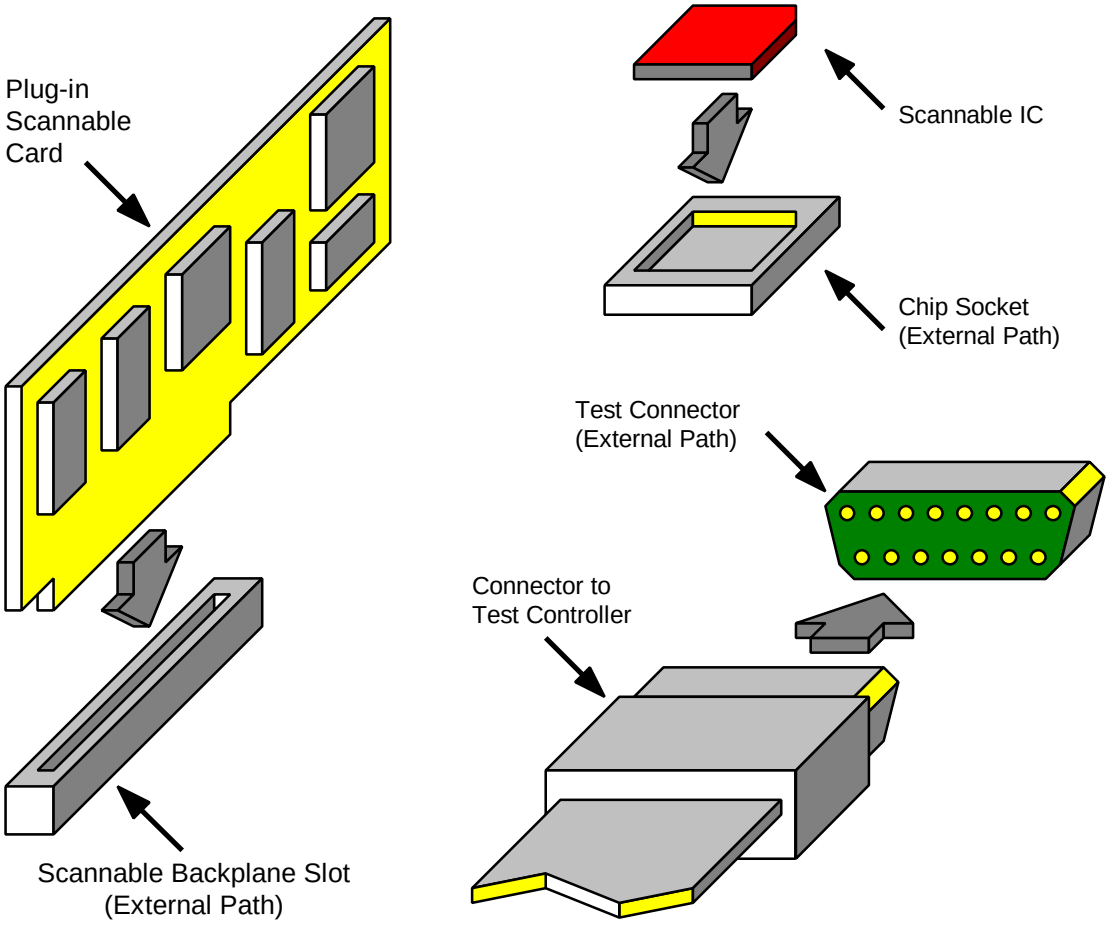
To avoid introducing a netlist concept into module entities, scan paths were created. Each path is controlled or controls one set of TAP signals (TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and optionally TAP_SCAN_RESET). Objects listed in the path are all connected. The TAP control signals of each object (TAP_SCAN_CLOCK, TAP_SCAN_MODE, and optionally TAP_SCAN_RESET) are all

wired together. The TAP_SCAN_IN of each object is wired to the TAP_SCAN_OUT of the next object.

To represent the simple board described earlier using this scheme, all that is required is to indicate that the connector and the device are in the same scan path, and that the connector has certain scan ports on it.

A connector of any type, where TDO leaves the module and TDI returns without a connection between, is an *external path*. Each external path of the module consists of a set of four or five TAP signals (TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and optionally TAP_SCAN_RESET). External paths can be used to represent any place where the scan path leaves the module. Examples of external paths: test controller interfaces, scannable card slots, scannable connectors, scannable daughterboard pins, and scannable device sockets. External paths and physical module connectors do not always have a one-to-one correspondence. The physical connector may have more than one instance of each TAP signal. In a sense, an external path represents a sort of "board-level Test Access Port", where the four or five TAP signals are received by or driven from the board. Every device has one external path: its TAP port.

Shown here are three examples of physical items found on a typical unit under test that are represented using external paths. The example on the left is a scannable card slot. The example on the upper right is a scannable chip socket. The example on the lower right is the connector used to plug the test controller into the UUT.

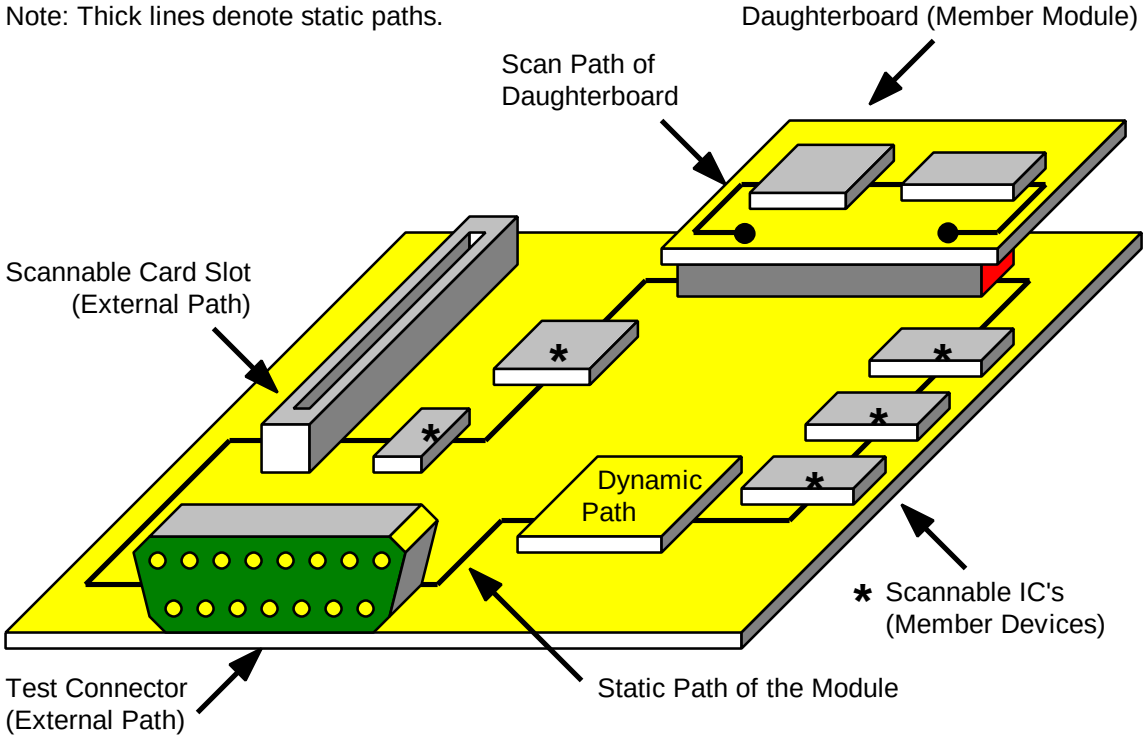


A serial TDI-to-TDO connection of member devices, member modules, and paths that all share the same TAP control signals is a *static path*. A static path represents a simple wiring-

together of members and connectors. An interesting but important fact to observe is that the static path forms a circle of TDI-to-TDO connections. Some of the items in the circle are external paths where the test controller can be plugged in. For the simple board described earlier, the static path consists of one external path and one member device.

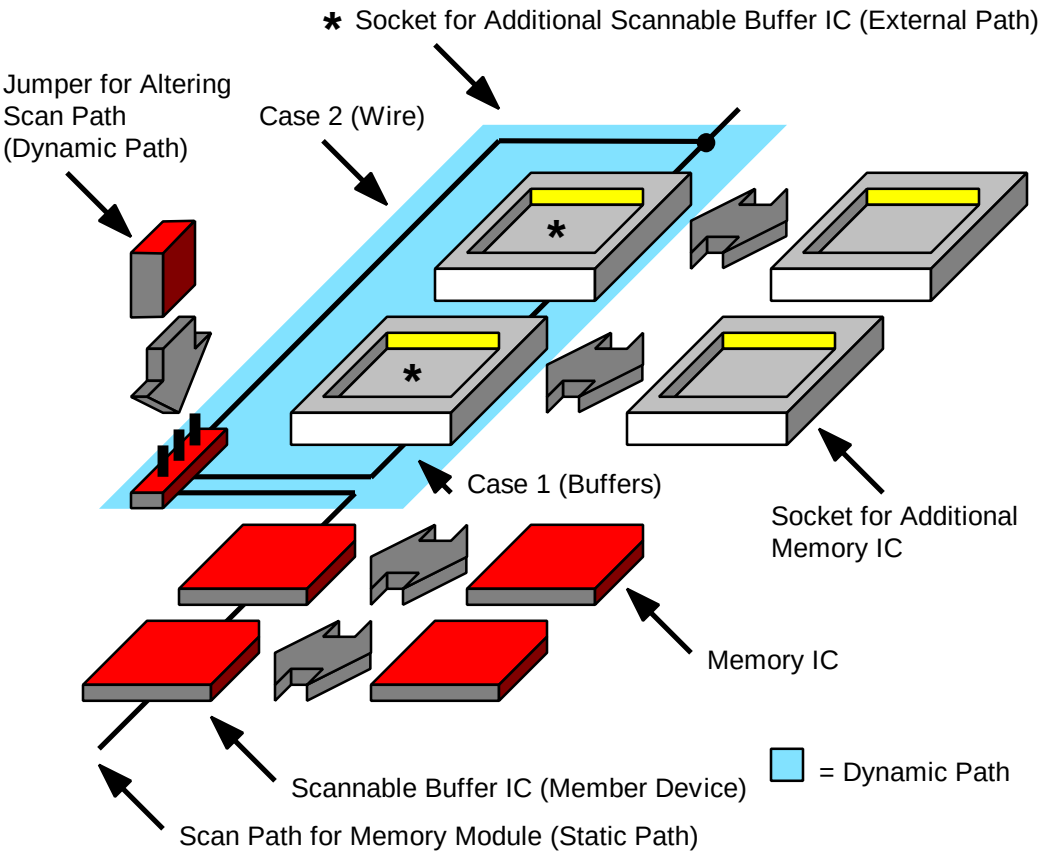
The previous example showing the simplest module is an example of a static path containing one member device and one external path. Shown here is an example of the physical items found on a typical unit under test that are represented using static paths. The example shows some devices (members), a daughterboard (member module), a test connector (external path), a scannable card slot (external path), and a block of logic labelled dynamic path. Only the five devices marked with an asterisk are member devices. The two devices shown on the daughterboard are members of the daughterboard module.

Note: Thick lines denote static paths.



A third type of path represents a limited form of ad-hoc scan path multiplexing. A *dynamic path* is a set of controlled paths that may be attached to the TMS signal of the path or have their TMS lines forced high or low. A dynamic path does not actually control the scan path multiplexing; the UUT hardware and the test software must coordinate their activities to change the configuration of the dynamic path. The state of a dynamic path simply represents the state of the hardware, and changing the dynamic path does not change the hardware.

Shown here is an example of physical items found on a typical unit under test that are represented using dynamic paths. The example shows a portion of a memory board where the amount of memory on the board can be expanded. Scannable buffers are used to add testability to the memory. The expandability is provided by two sockets for the memory and two sockets for the corresponding buffers. (The pictures of the two buffer sockets contain asterisks.) Since the buffers may or may not be present, the scan path must be routed around the buffers when they are absent to keep the scan path intact. The jumper used to perform this function controls a dynamic path. Case 1 of the dynamic path, when the buffers are installed, scans through two buffer sockets (external paths), while the case 2, when the buffers are absent, scans through a wire (empty static path). The area of the memory board represented as a dynamic path is shown with a light-colored background pattern.



Using these three types of paths, how is the netlist of TAP signals represented? Each path consists of a set of four or five TAP signals. When an external path is defined, a set of four or five TAP ports of the module must be associated with the external path. These ports represent physical pins on the module connector. When a static path is defined, at least one external path is (eventually) included in the static path. This implicitly connects the TAP signals of the scan path to the TAP ports of the external path. Static paths or dynamic paths that do not have an external path in them must be included inside another static path or dynamic path, and eventually all paths of any type are tied in to a master static path for the module that does have external paths. In this way, all TDI-to-TDO nets are identified, and all TMS signals, all TCK signals , and all optional TRST signals of the external paths are tied together to form three nets (one for TMS, one for TCK, and one for TRST).

Another way of thinking about paths is that each item in a module has its own TAP. For an external path, the TAP is defined by the ports attached to the external path. Static and dynamic paths have a TAP. Devices have TAPs, and member modules (by virtue of their external paths) also have TAPs. Listing an item in a path connects the TAP of the path to the TAP of the item, building a netlist.

In the simple case described earlier (a board with one device and one test connector), many simplifications can be made. An external path is not needed to describe this board, because only one set of TAP signals exists for the board. Thus, only one path statement is needed, a static path listing the single device on the board. The connection of the TAP signals to the static path and the creation of an external path for the board are all implied and do not need to be stated explicitly in this case.

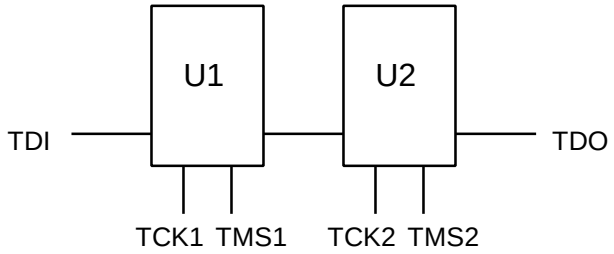
External paths, static paths, and dynamic paths form a hierarchy of paths. The lowest-level subpaths are defined first and later included in the definitions of higher-level paths, until at last the primary scan paths of the module are defined, incorporating all lower-level paths.

The example module HSDL entities shown later in this document should make the netlist and path concepts clearer.

Another statement related to external paths, the *member connection*, is used to attach members or paths to the external paths of a member module or device. For example, a backplane module has several external paths. When that backplane module is included as a member in a higher-level, system module, the system module must plug items into the external paths of the backplane member. All the external paths of the backplane **except one** must have something plugged in to them. The one remaining external path is where the backplane module is connected to the system module, usually in a static path.

The declarations for external paths, static paths, and dynamic paths may be listed in any order necessary to describe the module. Modules may be built containing more than one primary scan path. If any members contain unresolved external paths, the member connections must be listed following all path declarations.

An example of a module that cannot be described using HSDL is shown here.



In this module, the connection of devices does not correspond to the HSDL definition of a scan path; that is, serially connected members or paths sharing the **same set of TAP control signals**. These devices are serially connected, but they are connected to different TAP_SCAN_CLOCK and TAP_SCAN_MODE ports.

2.3.12. Optional External Path Declaration(s)

External paths usually represent external connectors that contain a set of TAP signals, one each of TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and optionally TAP_SCAN_RESET ports. An external path is given a name so that it can be referenced by other path declarations.

External paths are declared with a constant, and list the TAP ports associated with the external path.

```
port (LCLK           : inout bit;
      UUTCLK         : out bit;
      TMSR           : in  bit;
      GND            : linkage bit_vector (1 to 7);
      EVT0           : linkage bit;
      TDI            : in  bit;
      TDO            : out bit;
      TMS, TRST      : inout bit;

      -- ram32 slot connectors
      J2_TCK, J2_TMS : inout bit;
      J2_TDI         : out bit;
      J2_TDO         : in  bit;
      J2_DATABUS     : inout bit_vector (31 downto 0);
      J2_ADDRBUS     : out bit_vector (23 downto 0);
      J2_RW          : out bit;
        .
        .
        .
-- Scan Port Identification
attribute TAP_SCAN_IN    of TDI  : signal is true;
attribute TAP_SCAN_MODE  of TMS  : signal is true;
attribute TAP_SCAN_OUT   of TDO  : signal is true;
attribute TAP_SCAN_RESET of TRST : signal is true;
attribute TAP_SCAN_CLOCK of LCLK : signal is (20.0e6, BOTH);

attribute TAP_SCAN_IN    of J2_TDO : signal is true;
attribute TAP_SCAN_MODE  of J2_TMS : signal is true;
attribute TAP_SCAN_OUT   of J2_TDI : signal is true;
attribute TAP_SCAN_CLOCK of J2_TCK : signal is (20.0e6, BOTH);
        .
        .
        .
constant J1 : EXTERNAL_PATH := "TDI, TDO, TMS, LCLK, TRST";
constant J2 : EXTERNAL_PATH := "J2_TDI, J2_TDO, J2_TMS, J2_TCK";
```

In the example above, two external paths are shown. Both external paths use ports that are defined such that either a test controller or a scannable module or device could be plugged into them.

Note that if a board contains only one set of TAP signals, no external path need be defined. Its existence can be assumed.

A module must contain at least one external path where the TAP_SCAN_CLOCK, TAP_SCAN_MODE, and optional TAP_SCAN_RESET ports are either **in** or **inout**. Otherwise, no test connector would exist on the module for plugging in the test controller.

An optional PATH_DESCRIPTION statement can be given for each external path. The purpose and function of the external path should be described in sufficient detail so that the operator can understand what the external path represents.

```
attribute PATH_DESCRIPTION of J1 : constant is
   "J1 is the scan test connector for this board.";
```

2.3.13. Static Path Declaration(s)

Static paths define known, fixed, serially-connected member devices, member modules, or other paths. The items listed in the static path are connected serially, with the TDO of one item connected to the TDI of the item on its right. The TAP control signals (TAP_SCAN_CLOCK, TAP_SCAN_MODE, and optional TAP_SCAN_RESET) of all items listed are interconnected: all clocks on one net, all mode selects on another net, and all resets on a third net.

```
constant subpath1 :  STATIC_PATH := "u3, u4, u5, u6";
constant short :     STATIC_PATH := "";
   -- subpath2 is equivalent to "u19, u3, u4, u5, u6, u18"
constant subpath2 :  STATIC_PATH := "u19, subpath1, u18";
constant boardpath : STATIC_PATH := "J1, u1, u2, u7, J3, J4, J5, u8, dpath, u10";
```

The examples above show three static paths. Subpath1 is a simple static path consisting of four serially connected devices. Short is an empty static path, which simply shorts TDI to TDO. A short can be useful for "closing" external paths and for creating empty configurations of a dynamic path. Boardpath is a static path containing devices (beginning with "u"), external paths (beginning with "J"), and a dynamic path (beginning with "d").

Static paths are "circular" in the sense that the TDI of the first item in the static path is implicitly connected to the TDO of the last item in the static path. When a static path is listed as an item in a new path, the TDI of the first item in the first static path is connected to the TDO of the preceding item in the new path, and the TDO of the last item in the first static path is connected to the TDI of the following item in the new path. Conceptually, each static path has a TAP, and when one is listed in another, their TAPs are connected.

In the example shown here, boardpath is intended to be the primary scan path of the module. The presence of external paths in this static path is a clue. If boardpath is not included in any other path, it is (one of) the primary scan path(s) of the module.

An optional PATH_DESCRIPTION statement can be given for each static path. The purpose and function of the static path should be described in sufficient detail so that the operator can understand what portion of the unit under test is covered by the static path.

```
attribute PATH_DESCRIPTION of short : constant is
   "Short is an empty static path, representing a wire between TDI and TDO.";
```

2.3.14. Optional Dynamic Path Declaration(s)

A dynamic path describes a limited form of ad-hoc scan path multiplexing. A fixed set of items (paths or members) may be multiplexed. A dynamic path has one or more configurations, where each configuration selects **one** of the items from the set to be placed in the scan path (connected to the TAP_SCAN_MODE signal), and the other items are either in Test-Logic-Reset or Run-Test/Idle state.

```
constant dpath : DYNAMIC_PATH :=
   "0 (u9:dpath, short:1    )," &
   "1 (u9:1,     short:dpath) " ;
```

In this example, the dynamic path named dpath has two different configurations, or *dynamic path cases*. Each configuration indicates that one of the two items is connected to the TMS signal of the scan path and the other item is placed in Test-Logic-Reset. Case 0 connects the TAP_SCAN_MODE signal of member device u9 to the TAP_SCAN_MODE signal of the dynamic path, and the TAP_SCAN_MODE signal of the static path short to a fixed high (1), forcing it to Test-Logic-Reset. Case 1 connects the u9's TAP_SCAN_MODE signal to a fixed high (1) and short's TAP_SCAN_MODE signal to the dynamic path's TAP_SCAN_MODE signal.

Items in a dynamic path case that are not connected to TMS are assumed to be built so that their TDI/TDO ports are multiplexed out of the serial TDI-to-TDO chain, so that shifting operations do not involve them.

It is undefined whether the operation of multiplexing an item out of the dynamic path stops the test clock for that item or allows it to continue running. HSDL translators must assume that the test clock is stopped (this is the worst-case assumption). It is also undefined whether an item is placed in Test-Logic-Reset by manipulating its TAP controller state (pulling TMS high and clocking TCK five cycles) or by pulling TRST low. In other words, the actions of the item described by the dynamic path may not exactly describe what the hardware is doing. The only requirement is that the condition of the test controller's software

model of the item described by the dynamic path matches the condition of the hardware when it is multiplexed back into the scan path.

An optional PATH_DESCRIPTION statement can be given for each dynamic path. The purpose and function of the dynamic path should be described in sufficient detail so that the operator can understand what the dynamic path models and what it is used for.

```
attribute PATH_DESCRIPTION of dpath : constant is
   "The dynamic path dpath is used to remove the C40 from " &
   "the scan path if the device is not mounted on the board.";
```

2.3.15. Optional Member Connections

A requirement of HSDL is that all external paths of a member be connected to something, except the one external path that is used to control the member. What does this requirement mean, and why is there one exception?

Consider a backplane module intended for incorporation into a computer module. The backplane has one test connector (an external path) and multiple card slots (also external paths). The computer module declares the backplane module as a member. To plug the backplane module into one of the scan paths of the computer module, HSDL must know which one to use. How can this be determined? One of the external paths of the backplane member is listed in a static path or dynamic path, and the other external paths must have something plugged into them with the CONNECTIONS attribute.

A secondary reason for forcing all external paths to have something plugged in to them is that the test controller will only have to check the highest-level entity of the UUT for unresolved external paths when the UUT's entity is loaded by the test controller. If the user accidentally forgot to plug something into an external path five levels deep in the hierarchy, it could be weeks before the error is caught. To improve error detection in HSDL, all members with external paths must have those external paths resolved.

Sometimes it is desirable to leave external paths empty so that something may be plugged into them in a higher-level entity. To do this, a new external path is defined in the current entity and plugged in to the member's external path. This extra effort may be tedious but can avoid many errors and much complication of the test controller.

To plug items into member external paths, the CONNECTIONS attribute is used. The CONNECTIONS attribute must appear following all path declarations only if any member devices or member modules contain unresolved external paths.

```
attribute CONNECTIONS of fiction : entity is
   "u2 (ssp1:subpath1, ssp2:j2, ssp3:*, ssp4:*)";
```

In this example, the member named u2 has five external paths (remember that one of these external paths is not shown). Member u2's external path ssp1 has a static path named subpath1 attached to it. Member external path ssp2 has another external path connected to it named j2. External paths ssp3 and ssp4 are permanently not connected to anything (permanently open).

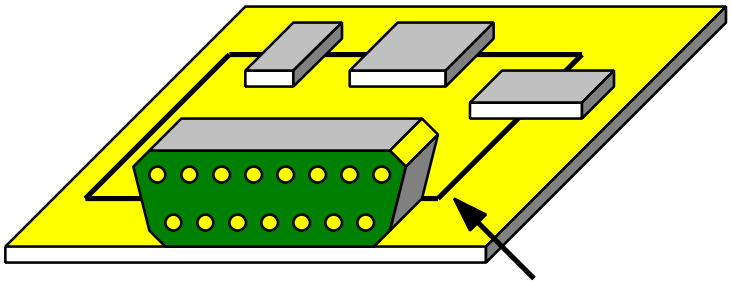
Normally this is illegal (it creates a permanent open in the scan path), but u2 happens to be a TI Scan Path Linker, SN74ACT8997. This device has four external paths that are under the control of dynamic paths, and these external paths can be multiplexed out of the scan path. Leaving the external paths permanently open implies that the test controller must forbid the operator from using certain configurations of the dynamic paths in the device.

2.3.16. Path Requirements

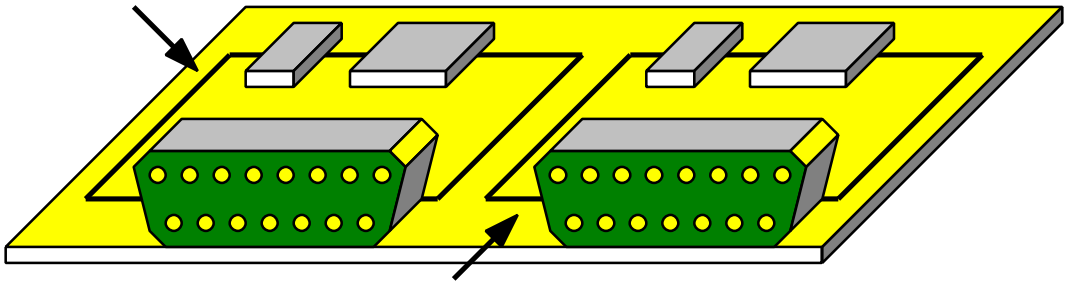
This section defines the rules that must be followed when combining EXTERNAL_PATH constants, STATIC_PATH constants, DYNAMIC_PATH constants, and the CONNECTIONS attribute to completely describe all scan paths of the unit under test.

A *primary scan path* represents a complete TDI-to-TDO connection from a TDI input of the module to a TDO output of the module. It includes all member devices, member modules, external paths, static paths, and dynamic paths that make up the primary scan path. All modules have at least one primary scan path, but may have more. The ultimate purpose of defining external, static and dynamic paths is to define the primary scan paths of the module.

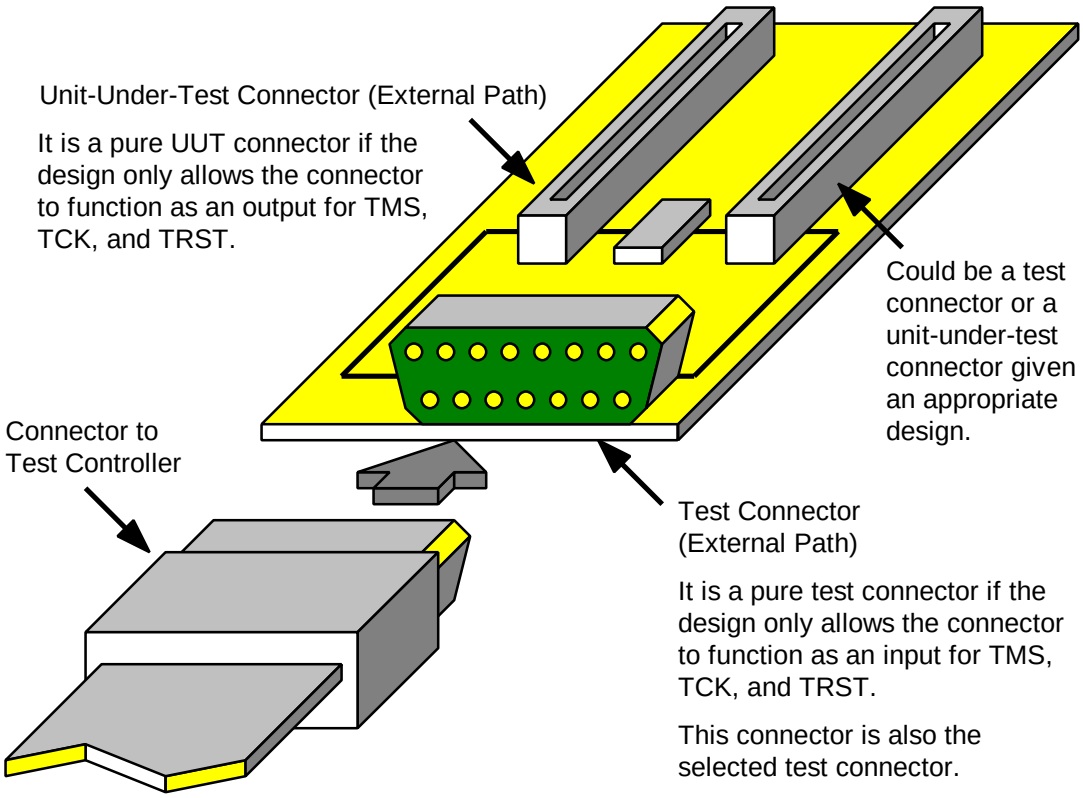
Module Containing One Primary Path



Module Containing Two Primary Paths



A *test connector* is an external path whose TAP_SCAN_CLOCK, TAP_SCAN_MODE, and, if present, TAP_SCAN_RESET ports all have mode IN or mode INOUT. Such an external path may be controlled by a test controller. A *unit-under-test connector* is an external path whose TAP_SCAN_CLOCK, TAP_SCAN_MODE, and, if present, TAP_SCAN_RESET ports all have mode INOUT or mode OUT. Such an external path may drive a unit-under-test. Every module must define at least one test connector. An external path whose TAP_SCAN_CLOCK, TAP_SCAN_MODE, and, if present, TAP_SCAN_RESET ports all have mode IN is a *pure test connector*, all have mode INOUT may be a *test connector* or a *unit-under-test connector*, or all have mode OUT is a *pure unit-under-test connector*. The *selected test connector* is the test connector that the test controller is connected to for the duration of the test.



If a module has exactly one each of TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, and TAP_SCAN_MODE ports, with an optional TAP_SCAN_RESET port, then the TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET ports must all have mode IN or mode INOUT. In this case, the module has only one test connector and no other external paths, so only one primary path is allowed, the external path declaration for the test connector may be omitted, and the test connector may be omitted from the primary scan path. The HSDL translator automatically generates the test connector by generating an external path named TAP, connecting it to the TAP_SCAN_... ports, and including it in the primary scan path. Every device also has a test connector named TAP.

A *path entry* is a static path, dynamic path, external path, member, or member external path - in other words, something that can be listed in another static path or dynamic path declaration. Every path entry in the module must be included in exactly one static path, or in exactly one member external path connection, or in exactly one dynamic path, but not in any combination thereof.

An exception is that a static path that is not included in another path defines a primary scan path. Each primary scan path must include at least one test connector (unless it can be assumed as stated previously). The test connector must be listed in the STATIC_PATH constant that defines the primary scan path.

Each primary scan path may include at most one pure test connector. Each primary scan path may also include any number of test connectors that are not pure test connectors and any number of unit-under-test connectors. If a primary scan path contains a pure test connector, it must always be used as the selected test connector.

When listing a member in a static path or dynamic path, the member external path must be used to identify the member and the external path of the member that is being connected to the static path or dynamic path. A member external path being connected to a static path or dynamic path must designate a test connector. If the member contains a pure test connector, that test connector must be connected to the static path or dynamic path. If the

member has only one test connector, the member name alone may be used, and the member test connector is implicitly connected to the static path or dynamic path.

Each external path of a member is connected to one of the primary scan paths of the member. More than one of these may be test connectors, but only one may be a pure test connector. Exactly one external path per primary scan path of a member may be connected to a static path or dynamic path. If two could be connected, a subpath would be created that could not be controlled by a test controller.

All external paths of each member must be connected to a path entry. When a member has more than one external path, the CONNECTIONS attribute must be used to connect the member external paths that were not included in a static path or dynamic path. If used, the CONNECTIONS attribute must follow all EXTERNAL_PATH, STATIC_PATH, and DYNAMIC_PATH constants.

The maximum operating frequencies for all TAP_SCAN_CLOCK ports that are part of the external paths in a primary path must be the same. The frequencies may be different for external paths in different primary paths. The port's TAP_SCAN_CLOCK frequency is interpreted as the maximum operating frequency of the primary path.

2.3.17. Optional Constraint Description(s)

Same as the HSDL device entity.

2.3.18. Optional Design Warning

Unchanged from BSDL. The DESIGN_WARNING attribute is not discussed in the *BSDL Syntax* chapter of the *HP Boundary-Scan Tutorial and BSDL Reference Guide*, but is described in the *Introduction to BSDL* chapter under the section *Miscellaneous Declarations.* This statement is optional.

3. Using HSDL

This section describes how HSDL can be used to solve selected scan description problems. Many HSDL statements are covered informally during the discussion.

3.1. Describing Architectures Above the Device Level

In HSDL, any level of scan architecture above the device level is called a module. To describe a module, an HSDL module entity is created. Module entities begin just like device entities but differ substantially after the use statements.

Begin the module with an entity statement, a generic statement, a port statement listing all the module pins, and a use STD_1149_1_1990.all statement. This part of the module description is exactly like the corresponding parts of the device description. Note that in a module, many more pins (called ports in HSDL) may need to be listed depending on the complexity of the module.

```
entity gdm is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   port (LCLK           : in  bit;
         UUTCLK         : out bit;
         TMSR           : in  bit;
         GND            : linkage bit_vector (1 to 7);
         EVT0           : linkage bit;
         TDO            : out bit;
         TMS, TDI, TRST : in  bit);

   -- Use Statement
   use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
```

Module entities contain the use HSDL_module.all statement after the statement use STD_1149_1_1990.all to indicate that a module is being described. These are the only use statements that are needed, and the only two that are allowed, in a module entity.

```
   use HSDL_module.all;      -- Get HSDL extensions for modules
```

After the two use statements, define the different packages for the module. Most modules only have one package type. Define the packages using the PIN_MAP attribute followed by PIN_MAP_STRING constants, one for each package. Pin numbering can use identifiers like J1_2 as well as numbers like 5. Identifiers are more common for boards that contain several slots, sockets, connectors, etc.

```
   -- Package Pin Mapping
   attribute PIN_MAP of gdm : entity is PHYSICAL_PIN_MAP;
   constant ONLY_PACKAGE : PIN_MAP_STRING :=
      "LCLK: 1, UUTCLK: 2, TDI: 3, TDO: 4, TMS: 5, TMSR: 6, TRST: 7," &
      "EVT0: 8, GND: (9, 10, 11, 12, 13, 14, 15)";
```

Scan port identification appears next. Include the attributes TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET (if the module has any TRST signals) for each 1149.1 signal on the module that can be accessed from a port on the module. Again, most modules only have one TAP, so only one each of the TAP_SCAN_... attributes is needed.

```
   -- Scan Port Identification
   attribute TAP_SCAN_IN of TDI : signal is true;
   attribute TAP_SCAN_MODE of TMS : signal is true;
   attribute TAP_SCAN_OUT of TDO : signal is true;
   attribute TAP_SCAN_RESET of TRST : signal is true;
   attribute TAP_SCAN_CLOCK of LCLK : signal is (20.0e6, BOTH);
```

At this point the module entity begins to look quite different from the device entity. Statements in the device entity that described the instruction register, idcode and usercode registers, instructions, user-defined test registers, and the boundary are not needed and make no sense for

a module entity. In their place, new statements describe the devices and modules contained in this module and the order of these devices and modules in the scan path.

Include the MEMBERS attribute to list each of the devices and modules that are contained in the module being described. Many modules contain only devices. Assign a different name to each member, using the reference designator from the schematic as the name to avoid confusion. The entity and package used for each member must also be defined. Look in the entity file for each device and module used to decide which package pinout is used on the hardware and list that package name in the MEMBERS attribute.

```
-- Member Description
attribute MEMBERS of gdm : entity is
   "u19 (ttl74bct8244, NT_PACKAGE)," &
   "u21 (ttl74bct8244, NT_PACKAGE)," &
   "u9  (ttl74bct8244, NT_PACKAGE)," &
   "u8  (ttl74bct8244, NT_PACKAGE)," &
   "u1  (ttl74bct8244, NT_PACKAGE)," &
   "u22 (ttl74bct8373, NT_PACKAGE)," &
   "u20 (cf93279,      FK_PACKAGE)";
```

Finally, describe the scan paths of the module. Most modules contain only a simple serial scan path that can be described with one STATIC_PATH constant. List the members in the STATIC_PATH constant in the order they appear on the module, listing the member closest to TDI first and the member closest to TDO last.

```
-- Paths of the module
constant path1 : STATIC_PATH := "u22, u1, u8, u19, u21, u20, u9";
```

The last statement of the module is the end entity statement, completing the description.

```
end gdm;
```

Only ten different types of statements are needed to describe a simple module, as shown here: entity, generic, port, use, PIN_MAP attribute, PIN_MAP_STRING constants, TAP_SCAN_... attributes, MEMBERS attribute, STATIC_PATH constant, and end.

3.2. Describing a Board, Box, Subsystem, or System

The method for describing a module is used to describe any board, box, subsystem, or system. As the module being described gets higher in the hierarchy of the unit under test, fewer members of the module are devices and more members are modules. This does not involve any new concepts - HSDL module descriptions treat a module and a device in the same way. Both are members.

Complex boards, especially backplanes used in more complex systems, may contain scannable card slots, chip sockets, card edge connectors, and even more than one test connector for plugging in the test controller. Describe these hardware constructs, which each involve an open in the scan path that is resolved later, using external paths. See *Describing a Backplane* for details.

Some boards may be designed with scannable devices that may or may not be installed. The scan path around the scannable devices may be jumpered to skip around them. Later, when the board is upgraded, the scan path is rejumpered to include the devices. This is an example of ad-hoc scan path multiplexing. Other such ad-hoc methods are possible, such as using a scannable buffer and a multiplexer to enable and disable secondary scan paths. Boards and other modules that use ad-hoc methods are described using dynamic paths. See *Controlling Ad-Hoc Scan Path Multiplexing* for details.

Module-level designs often include new buses and signals that are not directly related to the names of the signals provided by devices. For example, four different eight-bit buffers may be used to drive a data bus. A natural desire is to want to use the name of the data bus to perform testing, not the names of each of the device pins. In HSDL, bus names can be assigned to any combination of member signals. See *Assigning a Name to a Bus on a Board* for details.

Boards often have design constraints that disallow certain hazardous conditions from being established. The constraints prevent damage to the board. During testing, these design constraints may be overridden by test-mode instructions. Constraint expressions can be used to prevent damage from occurring, even when a test-mode instruction is in effect. See *Preventing Illegal Hardware Conditions* for details.

3.3. Describing a Multichip Module

In HSDL, multichip modules (MCMs) are conceptually no different from a regular board, and are described using the same set of statements. See *Describing a Board, Box, Subsystem, or System* for details.

3.4. Describing a Backplane

Complex boards, especially backplanes used in more complex systems, may contain scannable card slots, chip sockets, card edge connectors, and even more than one test connector for plugging in the test controller. Describe these hardware constructs, which each involve an open in the scan path that is resolved later, using external paths.

Define an external path using the EXTERNAL_PATH constant. Each external path has a TAP associated with it, consisting of a set of four test signals (or five, if TRST is present). Define the TAP for an external path by listing the names of each test signal in the string following the EXTERNAL_PATH.

```
constant j1 : EXTERNAL_PATH := "TDI, TDO, TMS, TCK, TRST";
```

An external path is used to describe each slot, socket, connector, and so on that contains test signals going off or coming onto the module. External paths are classified three ways, depending on whether the test signals are input only, output only, or input/output. The TDI signal is always an input and the TDO signal is always an output.

A card slot or chip socket that a scannable board or device can be plugged into is normally designed so that its TMS, TCK, and TRST signals are all outputs. In HSDL terminology, this is a unit-under-test connector. The EXTERNAL_PATH names four or five ports that are listed in TAP_SCAN_... attributes. Define the ports named by TAP_SCAN_MODE, TAP_SCAN_CLOCK, and TAP_SCAN_RESET as out ports in the port statement.

```
port (TDI            : in  bit;
      TDO            : out bit;
      TMS, TCK, TRST : out bit);  -- all outputs

   .
   .
   .

constant slot1 : EXTERNAL_PATH := "TDI, TDO, TMS, TCK, TRST";
```

A connector that the test controller can be plugged into is normally designed so that its TMS, TCK, and TRST signals are all inputs. In HSDL terminology, this is a test connector. The EXTERNAL_PATH names four or five ports that are listed in TAP_SCAN_... attributes. Define the ports named by TAP_SCAN_MODE, TAP_SCAN_CLOCK, and TAP_SCAN_RESET as in ports in the port statement.

```
port (TDI            : in  bit;
      TDO            : out bit;
      TMS, TCK, TRST : in  bit);  -- all inputs

   .
   .
   .

constant test : EXTERNAL_PATH :=  "TDI, TDO, TMS, TCK, TRST";
```

It is possible, although unlikely, that a card slot may be designed so that either a scannable card, which receives TAP signals, or a test controller, which drives TAP signals, could be plugged into it. A backplane of such slots would not require a dedicated test connector, although the design would probably be expensive in terms of real estate, speed, and so on. To describe such a slot,

define the ports named by TAP_SCAN_MODE, TAP_SCAN_CLOCK, and TAP_SCAN_RESET as inout ports in the port statement.

```
port (TDI            : in    bit;
      TDO            : out   bit;
      TMS, TCK, TRST : inout bit);  -- all inouts

   .
   .
   .

constant test : EXTERNAL_PATH := "TDI, TDO, TMS, TCK, TRST";
```

The primary scan path for a backplane is still described using a STATIC_PATH constant. The test connector must be listed in the static path, and by convention it is listed first. Then list the external path names, which define the slots, sockets, and so forth, and the member devices on the backplane in the order they appear. List the item closest to TDI right after the name of the test connector, and the item closest to TDO last.

```
constant primary : STATIC_PATH := "j1, slot1, slot2, slot3, u1";
```

3.5. Assigning a Name to a Subset of a Test Register

Test register names and contents are not usually the most convenient way to examine the data shifted into and out of a device. The instruction register and bypass register contain only one item of information, the instruction opcode and bypass bit, respectively. But many other test registers, including the identification register and the boundary register, are broken up into named subsets.

The identification register contains four subsets: a fixed '1' bit in the LSB, an 11-bit manufacturer identity field, a 16-bit part number field, and a 4-bit version field. The boundary potentially contains dozens of different fields, each corresponding to a grouping of cells connected to related device pins. For example, all output cells connected to a 32-bit data bus might need to be controlled or observed.

User-defined instructions and test registers, in order to reduce the overhead needed to implement testability features, often place many items of information in the same test register, or may even use a single test register in the hardware to load many kinds of information depending on the instruction used to shift through that register.

Define buses in the device entity, using the BUS_COMPOSITION attribute, to assign names to each of these test register subsets. Usually a bus consists of adjacent bits from one test register, making description easy. List the bus name and length, followed by the test register name and bit numbers giving the MSB and LSB of the bus. Remember from 1149.1 that bit 0 is the LSB and is closest to TDO. By reversing the MSB and LSB, the bit ordering in the bus can be the inverse of the bit order in the test register.

```
attribute BUS_COMPOSITION of My_IC : entity is
   "version[4]       (IDCODE[31,28])," &
   "part_number[16]  (IDCODE[27,12])," &
   "manufacturer[11] (IDCODE[11, 1])";

attribute BUS_COMPOSITION of ttl74bct8374 : entity is
   "inputs[8]  (BOUNDARY[ 8,15])," &
   "outputs[8] (BOUNDARY[ 0, 7])," &
   "aux[2]     (BOUNDARY[16,17])," &
   "lfsr[16]   (BOUNDARY[ 0,15])";
```

Buses can contain any number of cells in any order from any number of test registers or even buses. In cases where the hardware design has been heavily optimized the bits may be scattered throughout a test register in any order. In rare instances related bits of information may be located in different test registers. A bus can describe all these cases by using a list of bus components.

```
attribute BUS_COMPOSITION of My_FPGA : entity is
   "databus_inputs[32] (BOUNDARY[ 0], BOUNDARY[ 3], BOUNDARY[ 6], BOUNDARY[ 9]," &
                       "BOUNDARY[12], BOUNDARY[15], BOUNDARY[18], BOUNDARY[21]," &
                       "BOUNDARY[24], BOUNDARY[27], BOUNDARY[30], BOUNDARY[33]," &
                       "BOUNDARY[36], BOUNDARY[39], BOUNDARY[42], BOUNDARY[45]," &
                       "BOUNDARY[48], BOUNDARY[51], BOUNDARY[54], BOUNDARY[57]," &
                       "BOUNDARY[60], BOUNDARY[63], BOUNDARY[66], BOUNDARY[69]," &
                       "BOUNDARY[72], BOUNDARY[75], BOUNDARY[78], BOUNDARY[81]," &
                       "BOUNDARY[84], BOUNDARY[87], BOUNDARY[90], BOUNDARY[93] " ;
```

3.6. Assigning a Name to a Bus on a Board

Module-level designs often include new buses and signals that are not directly related to the names of the signals provided by devices. For example, four different eight-bit buffers may be used to drive a data bus. A natural desire is to want to use the name of the data bus to perform testing, not the names of each of the device pins. In HSDL, bus names can be assigned to any combination of member signals.

Define buses in the module entity using the BUS_COMPOSITION attribute. A module-level bus may consist of a single bit from a single device, or multiple bits from several devices. List the bus name and length, followed by the test register name(s) and bit numbers giving the MSB and LSB of each component of the bus. Remember from 1149.1 that bit 0 is the LSB and is closest to TDO. By reversing the MSB and LSB, the bit ordering of a component can be the inverse of the bit order in the underlying test register.

```
attribute BUS_COMPOSITION of My_Board : entity is
   "addrbus[32] (u1.output, u2.output, u3.output, u4.output)," &
   "databus[32] (u5.a, u6,a, u7.a, u8.a)";
```

3.7. Assigning a Symbolic Name to a Value

Some test registers and buses, like the instruction register, have commands, options, and so forth that have symbolic names. Often these names are listed on the device data sheet.

Define a symbol table of symbol names and values using the SYMBOL_TABLE attribute.

```
constant BCR_Opcodes : SYMBOL_TABLE :=
   "SAMTOG  (00)," &
   "PRPG    (01)," &
   "PSA     (10)," &
   "PSAPRPG (11)";
```

To associate the symbols with the test register or bus they belong to, use the attributes BOUNDARY_SYMBOLS, BUS_SYMBOLS, and REGISTER_SYMBOLS. See *Assigning Symbolic Names to a Test Register or Bus* for details.

3.8. Assigning Symbolic Names to a Test Register or Bus

Once a symbol table has been defined (see *Assigning a Symbolic Name to a Value* for details), it can be attached to a test register, a bus, or a device port. The symbols in the table can then be used instead of bit patterns when specifying values to shift in. The symbols are also displayed instead of bit patterns when examining values that were shifted out.

Use the REGISTER_SYMBOLS attribute to attach a symbol table to a test register.

```
attribute REGISTER_SYMBOLS of ttl74bct8374 : entity is
   "BCR (BCR_Opcodes)";
```

Use the BUS_SYMBOLS attribute to attach a symbol table to a bus.

```
attribute BUS_SYMBOLS of ttl74bct8374 : entity is
   "aux (AUX_SYMBOLS)";
```

Use the BOUNDARY_SYMBOLS attribute to attach a symbol table to a specific type of cell attached to a port. Each different cell function (input, clock, output2, output3, bidir, control, controlr, and internal) can have a different symbol table. Typically, the symbol table only makes sense for clock cells, control cells, and input and output cells on single-bit ports.

```
attribute BOUNDARY_SYMBOLS of ttl74bct8374 : entity is
   "CLK    (input, clk_symbols)," &
   "OC_NEG (input, oc_symbols)";
```

3.9. Preventing Illegal Hardware Conditions

Boards often have design constraints that disallow certain hazardous conditions from being established. The constraints prevent damage to the board. During testing, these design constraints may be overridden by test-mode instructions. Constraint expressions can be used to prevent damage from occurring, even when a test-mode instruction is in effect.

Define a constraint using the CONSTRAINTS attribute. Constraints can be defined for both modules and devices. Each constraint is given a name that can be displayed by the test controller when the constraint is violated. The constraint expression is a logical expression that, when true, indicates that a constraint has been violated. The constraints are evaluted before each scan operation shifts data into the hardware, to ensure that the data is acceptable. Alternatively, ATPG software can use the constraints to prevent the creation of vectors that would otherwise damage the hardware.

3.10. Adding Descriptions to Each Item in the Entity

Reading a BSDL or HSDL description of a device or module is not the easiest undertaking. The syntax is too cluttered and the information too terse to be immediately understandable. Textual descriptions can be added to HSDL device and module entities to overcome this.

Define a text description for both electronic documentation and help in interactive tools. Descriptions can be attached to buses (BUS_DESCRIPTION), constraints (CONSTRAINT_DESCRIPTION), device entity names (DEVICE_DESCRIPTION), instruction opcodes (INSTRUCTION_DESCRIPTION), member names (MEMBER_DESCRIPTION), module entity names (MODULE_DESCRIPTION), test register names (REGISTER_DESCRIPTION), symbol names (SYMBOL_DESCRIPTION), dynamic paths (PATH_DESCRIPTION), external paths (PATH_DESCRIPTION), and static paths (PATH_DESCRIPTION). These descriptions can be displayed by a tool when information is requested by the user.

3.11. Controlling Ad-Hoc Scan Path Multiplexing

Some boards may be designed with scannable devices that may or may not be installed. The scan path around the scannable devices may be jumpered to skip around them. Later, when the board is upgraded, the scan path is rejumpered to include the devices. This is an example of ad-hoc scan path multiplexing. Other ad-hoc methods are possible, such as using a scannable buffer and a multiplexer to enable and disable secondary scan paths. Boards and other modules that use ad-hoc methods are described using dynamic paths.

Define a dynamic path using the DYNAMIC_PATH attribute. Each case of the dynamic path lists all the possible items for all cases, and selects one for inclusion in the scan path. The others are either in Test-Logic-Reset or Run-Test/Idle state.

```
constant dpath : DYNAMIC_PATH :=
   "0 (u9:dpath, short:1    )," &
   "1 (u9:1,     short:dpath)";
```

Each item selected by the dynamic path may need to be a list of one or more members, external paths, static paths, or even dynamic paths. Since the dynamic path only allows a single name for each item, so additional static paths may need to be defined using STATIC_PATH constants. This restriction is intended to keep dynamic paths short and readable.

4. Example HSDL Device Description

This example is an HSDL description for the Texas Instruments SN74BCT8374 Octal D-Type Flip-Flop. It is based on the original BSDL description of the same device found in the *HP Boundary-Scan Tutorial and BSDL Reference Manual*.

Do not consider this to be the official HSDL description of the SN74BCT8374. It is only an example used to illustrate extensions in HSDL.

```
entity ttl74bct8374 is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   port (CLK           : in  bit;
         Q             : out bit_vector (1 to 8);
         D             : in  bit_vector (1 to 8);
         GND, VCC      : linkage bit;
         OC_NEG        : in  bit;
         TDO           : out bit;
         TMS, TDI, TCK : in  bit);

   -- Use Statement
   use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
   use HSDL_device.all;      -- Get HSDL extensions for devices

   attribute DEVICE_DESCRIPTION of ttl74bct8374 : entity is
      "SN74BCT8374 is a Scan Test Devices with Octal D-Type Edge-Triggered " &
      "Flip-Flops. In normal mode, on the positive transition of CLK the Q " &
      "outputs take on the logic levels setup up at the D inputs.  The "     &
      "output enable OC_NEG is used to place the Q outputs in the "          &
      "high-impedance state, but does not affect the internal operations "   &
      "of the flip-flop.";

   -- Port Descriptions
   attribute PORT_DESCRIPTION of D : signal is
      "Eight-bit input bus of the device.";
   attribute PORT_DESCRIPTION of Q : signal is
      "Eight-bit output bus of the device.  All outputs can be set to "       &
      "high-impedance by placing a 1 on the OC_NEG pin.  All outputs can be " &
      "updated with the values on the input bus D on a positive transition "  &
      "of CLK.";
   attribute PORT_DESCRIPTION of CLK : signal is
      "Edge-triggered flip-flop clock control of the device.  All pins on " &
      "the output bus Q can be updated with the values on the input bus D " &
      "on a positive transition of CLK.";
   attribute PORT_DESCRIPTION of OC_NEG : signal is
      "Output control of the device.  All pins on the output bus Q can be " &
      "set to high-impedance by placing a 1 on the OC_NEG pin.  Disabling " &
      "the output bus Q has no effect on the internal operation of the "    &
      "flip-flops.";

   -- Package Pin Mapping
   attribute PIN_MAP of ttl74bct8374 : entity is PHYSICAL_PIN_MAP;
   constant DW_PACKAGE : PIN_MAP_STRING :=
      "CLK:1, Q:(2,3,4,5,7,8,9,10), D:(23,22,21,20,19,17,16,15)," &
      "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";
   constant FK_PACKAGE : PIN_MAP_STRING :=
      "CLK:9, Q:(10,11,12,13,16,17,18,19), D:(6,5,4,3,2,27,26,25)," &
      "GND:14, VCC:28, OC_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";
   constant NT_PACKAGE : PIN_MAP_STRING :=
      "CLK:1, Q:(2,3,4,5,7,8,9,10), D:(23,22,21,20,19,17,16,15)," &
      "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

   -- Scan Port Identification
   attribute TAP_SCAN_IN of TDI : signal is true;
   attribute TAP_SCAN_MODE of TMS : signal is true;
```

```
   attribute TAP_SCAN_OUT of TDO : signal is true;
   attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

   -- TAP Description
1550 attribute INSTRUCTION_LENGTH of ttl74bct8374 : entity is 8;
   attribute INSTRUCTION_OPCODE of ttl74bct8374 : entity is
      "BYPASS  (11111111, 10001000, 00000101, 10000100, 00000001)," &
      "EXTEST  (00000000, 10000000)," &
      "SAMPLE  (00000010, 10000010)," &
1555  "INTEST  (00000011, 10000011)," &
      "TRIBYP  (00000110, 10000110)," &  -- Boundary Hi-Z
      "SETBYP  (00000111, 10000111)," &  -- Boundary 1/0
      "RUNT    (00001001, 10001001)," &  -- Boundary run test
      "READBN  (00001010, 10001010)," &  -- Boundary read normal
1560  "READBT  (00001011, 10001011)," &  -- Boundary read test
      "CELLTST (00001100, 10001100)," &  -- Boundary selftest normal
      "TOPHIP  (00001101, 10001101)," &  -- Boundary toggle out test
      "SCANCN  (00001110, 10001110)," &  -- BCR Scan normal
      "SCANCT  (00001111, 10001111)";    -- BCR Scan test
1565 attribute INSTRUCTION_CAPTURE of ttl74bct8374 : entity is "10000001";
   attribute INSTRUCTION_DISABLE of ttl74bct8374 : entity is "TRIBYP";
   attribute INSTRUCTION_GUARD of ttl74bct8374 : entity is "SETBYP";
   attribute INSTRUCTION_NORMAL of ttl74bct8374 : entity is
      "BYPASS, SAMPLE, READBN, CELLTST, SCANCN";
1570 attribute INSTRUCTION_TEST of ttl74bct8374 : entity is
      "EXTEST, INTEST, TRIBYP, SETBYP, RUNT, READBT, TOPHIP, SCANCT";
   attribute INSTRUCTION_DESCRIPTION of ttl74bct8374 : entity is
   -- instruction  (description)
      "BYPASS  ('Select BYPASS register in normal mode.')," &
1575  "EXTEST  ('Select Boundary-Scan register in test mode; control device" &
               "inputs and outputs using the contents of the Boundary.')," &
      "SAMPLE  ('Select Boundary-Scan register in normal mode; sample " &
               "device inputs and outputs into the Boundary.')," &
      "INTEST  ('Select Boundary-Scan register in test mode; control device" &
1580           "inputs and outputs using the contents of the Boundary.')," &
      "TRIBYP  ('Select BYPASS register in test mode; control device " &
               "outputs to high-impedance.')," &
      "SETBYP  ('Select BYPASS register in test mode; control device " &
               "outputs using the contents of the Boundary.')," &
1585  "RUNT    ('Execute the test loaded into the Boundary Control " &
               "Register in the Run-Test/Idle state.')," &
      "READBN  ('Select Boundary-Scan register in normal mode; Boundary " &
               "cell contents are not changed in Capture-DR.')," &
      "READBT  ('Select Boundary-Scan register in test mode; Boundary cell " &
1590           "contents are not changed in Capture-DR.')," &
      "CELLTST ('Select Boundary-Scan register in normal mode; Boundary " &
               "cell contents are inverted and captured in Capture-DR.')," &
      "TOPHIP  ('Select BYPASS register in test mode; toggle outputs of " &
               "device pins from Boundary while holding device inputs.')," &
1595  "SCANCN  ('Select Boundary Control Register in normal mode.')," &
      "SCANCT  ('Select Boundary Control Register in test mode.')";

   -- Symbol Table Descriptions
   constant BCR_Opcodes : SYMBOL_TABLE :=
1600 -- symbol   (value, ..., value)
      "SAMTOG  (00)," &
      "PRPG    (01)," &
      "PSA     (10)," &
      "PSAPRPG (11)";
1605 attribute SYMBOL_DESCRIPTION of BCR_Opcodes : constant is
   -- symbol  (description)
      "SAMTOG ('Samples device inputs on input bus; toggles device outputs "  &
               "from output bus.')," &
      "PRPG   ('Conducts 16-bit Pseudo-Random Pattern Generation using the "  &
1610           "contents of the input and output buses of the Boundary-Scan " &
               "register as an initial value.  The Q outputs of the device "  &
               "will be set to the value in the output bus.  A new pattern "  &
               "is generated on each TCK in Run-Test/Idle state.')," &
      "PSA    ('Conducts 16-bit Parallel Signature Analysis using the "  &
1615           "contents of the input and output buses of the Boundary-Scan " &
               "register as an initial value.  The Q outputs of the device "  &
               "will be set to the value in the output bus.  A new checksum"  &
               "is generated on each TCK in Run-Test/Idle state.')," &
      "PSAPRPG('Simultaneous 8-bit PSA on the D inputs and 8-bit PRPG on "  &
1620           "the Q outputs.')";
```

```
   constant AUX_SYMBOLS : SYMBOL_TABLE :=
      "enable  (0X)," &
      "disable (1X)";
   attribute SYMBOL_DESCRIPTION of AUX_SYMBOLS : constant is
      "enable  ('Enable device output pins in test mode.')," &
      "disable ('Set device output pins to high-impedance in test mode.')";
   constant CLK_SYMBOLS : SYMBOL_TABLE :=
      "CLK_HIGH (1), CLK_LOW (0)";
   constant OC_SYMBOLS : SYMBOL_TABLE :=
      "drive (0), threestate (1)";

   -- Registers of the device
   attribute REGISTER_ACCESS of ttl74bct8374 : entity is
      "BOUNDARY (READBN, READBT, CELLTST),"      &
      "BYPASS   (TOPHIP, SETBYP, RUNT, TRIBYP)," &
      "BCR[2]   (SCANCN, SCANCT)";       -- 2-bit Boundary Control Register
   attribute REGISTER_SYMBOLS of ttl74bct8374 : entity is
   --  reg  (symbol_table)
      "BCR  (BCR_SYMBOLS)";
   attribute REGISTER_DESCRIPTION of ttl74bct8374 : entity is
   --  reg      (description)
      "BOUNDARY ('The Boundary-Scan register contains the cells attached "  &
                "to the pins of the device.')," &
      "BYPASS   ('The BYPASS register is a one-bit register that always "   &
                "loads a 0 in the Capture-DR state.  It is used to speed "  &
                "up scanning to the UUT when a device is not used during "  &
                "a test.')," &
      "BCR      ('The Boundary Control Register is a design-specific "      &
                "test data register used to specify the test operation - "  &
                "PSA, PRPG - that will be performed by the RUNT "           &
                "instruction.')";
   attribute REGISTER_CAPTURE of ttl74bct8374 : entity is
   --  reg    (capturevalue, pass/fail, description)
      "BYPASS (0, pass, 'The BYPASS register is working correctly.')";
      -- Note that this capture value did not need to be specified.
   attribute REGISTER_RESET of ttl74bct8374 : entity is
   --  reg         (resetvalue)
      "INSTRUCTION (BYPASS)," &  -- This does not need to be specified.
      "BCR         (PSA)";

   -- Boundary Register Description
   attribute BOUNDARY_CELLS of ttl74bct8374 : entity is "BC_1";
   attribute BOUNDARY_LENGTH of ttl74bct8374 : entity is 18;
   attribute BOUNDARY_REGISTER of ttl74bct8374 : entity is
   -- num   cell   port     function  safe  [ccell  disval  rslt]
      "17  (BC_1,  CLK,     input,    X)," &
      "16  (BC_1,  OC_NEG,  input,    X)," &  -- Merged input/control
      "16  (BC_1,  *,       control,  1)," &  -- Merged input/control
      "15  (BC_1,  D(1),    input,    X)," &
      "14  (BC_1,  D(2),    input,    X)," &
      "13  (BC_1,  D(3),    input,    X)," &
      "12  (BC_1,  D(4),    input,    X)," &
      "11  (BC_1,  D(5),    input,    X)," &
      "10  (BC_1,  D(6),    input,    X)," &
      " 9  (BC_1,  D(7),    input,    X)," &
      " 8  (BC_1,  D(8),    input,    X)," &
      " 7  (BC_1,  Q(1),    output3,  X,     16,    1,      Z)," &
      " 6  (BC_1,  Q(2),    output3,  X,     16,    1,      Z)," &
      " 5  (BC_1,  Q(3),    output3,  X,     16,    1,      Z)," &
      " 4  (BC_1,  Q(4),    output3,  X,     16,    1,      Z)," &
      " 3  (BC_1,  Q(5),    output3,  X,     16,    1,      Z)," &
      " 2  (BC_1,  Q(6),    output3,  X,     16,    1,      Z)," &
      " 1  (BC_1,  Q(7),    output3,  X,     16,    1,      Z)," &
      " 0  (BC_1,  Q(8),    output3,  X,     16,    1,      Z)";
   -- outputs controlled from cell 16 set to 0 are Hi-Z.
   -- cell 16 has a merged function, both input and control.

   attribute BOUNDARY_SYMBOLS of ttl74bct8374 : entity is
   --  port   (function, symbol_table)
      "CLK    (input, clk_symbols)," &
      "OC_NEG (input, oc_symbols)";

   -- Buses of the device
   attribute BUS_COMPOSITION of ttl74bct8374 : entity is
   --  bus[length] (reg, reg[bit], reg[bit,bit], ...)
```

```
      "inputs[8]  (BOUNDARY[ 8,15])," &
      "outputs[8] (BOUNDARY[ 0, 7])," &
      "aux[2]     (BOUNDARY[16,17])," &
      "lfsr[16]   (BOUNDARY[ 0,15])";
   attribute BUS_SYMBOLS of ttl74bct8374 : entity is
   --  bus (symbol_table)
      "aux (AUX_SYMBOLS)";
   attribute BUS_DESCRIPTION of ttl74bct8374 : entity is
   --  bus    (description)
      "inputs  ('The eight cells of the Boundary-Scan register "  &
               "that are connected to the D inputs are combined " &
               "to form the input bus.')," &
      "outputs ('The eight cells of the Boundary-Scan register "  &
               "that are connected to the Q outputs are "         &
               "combined to form the output bus.')," &
      "aux     ('The two cells of the Boundary-Scan register "    &
               "that are connected to the CLK and OC_NEG pins "   &
               "are combined to form the aux bus.')," &
      "lfsr    ('A bus used by the bcr.')";

   -- Constraint Descriptions
   attribute CONSTRAINTS of ttl74bct8374 : entity is
   --  constraint   (expression)
      "bad_device   (BYPASS = 1)";            -- this is an example ONLY
   attribute CONSTRAINT_DESCRIPTION of ttl74bct8374: entity is
   --  constraint   (description)
      "bad_device   ('The device fails if BYPASS is a 1 (not true)!')";

end ttl74bct8374;
```

5.
Example HSDL Module Descriptions

5.1. HSDL for General Demonstration Module

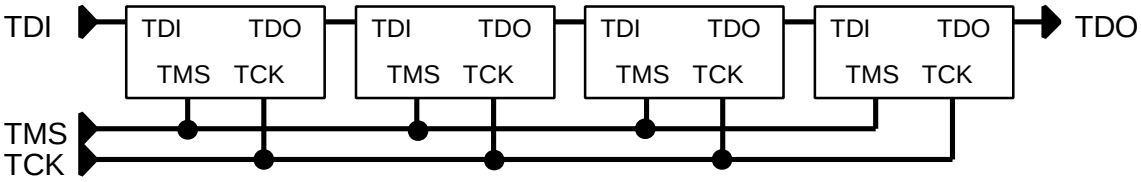
This HSDL example was written for the General Demonstration Module that can be purchased for use with Texas Instruments' ASSET Scan-Based Diagnostics System.

```
entity gdm is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   port (LCLK           : in  bit;
         UUTCLK         : out bit;
         TMSR           : in  bit;
         GND            : linkage bit_vector (1 to 7);
         EVT0           : linkage bit;
         TDO            : out bit;
         TMS, TDI, TRST : in  bit);

   -- Use Statement
   use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
   use HSDL_module.all;      -- Get HSDL extensions for modules

   attribute MODULE_DESCRIPTION of gdm: entity is
      "The General Demonstration Module (GDM) is an example board that was " &
      "designed to demonstrate some of the capabilities of an 1149.1 UUT.";

   -- Port Descriptions
   attribute PORT_DESCRIPTION of EVT0 : signal is
      "Unimplemented event-qualification line.";

   -- Package Pin Mapping
   attribute PIN_MAP of gdm : entity is PHYSICAL_PIN_MAP;
   constant ONLY_PACKAGE : PIN_MAP_STRING :=
      "LCLK: 1, UUTCLK: 2, TDI: 3, TDO: 4, TMS: 5, TMSR: 6, TRST: 7," &
      "EVT0: 8, GND: (9, 10, 11, 12, 13, 14, 15)";

   -- Scan Port Identification
   attribute TAP_SCAN_IN of TDI : signal is true;
   attribute TAP_SCAN_MODE of TMS : signal is true;
   attribute TAP_SCAN_OUT of TDO : signal is true;
   attribute TAP_SCAN_RESET of TRST : signal is true;
   attribute TAP_SCAN_CLOCK of LCLK : signal is (20.0e6, BOTH);

   -- Member Description
   attribute MEMBERS of gdm : entity is
      "u19 (ttl74bct8244, NT_PACKAGE)," &
      "u21 (ttl74bct8244, NT_PACKAGE)," &
      "u9  (ttl74bct8244, NT_PACKAGE)," &
      "u8  (ttl74bct8244, NT_PACKAGE)," &
      "u1  (ttl74bct8244, NT_PACKAGE)," &
      "u22 (ttl74bct8373, NT_PACKAGE)," &
      "u20 (cf93279,      FK_PACKAGE)";
   attribute MEMBER_DESCRIPTION of gdm : entity is
      "u19 ('Provides controllability/observability for combinational logic.')," &
      "u21 ('Provides observability of D2 hex LED inputs.')," &
      "u9  ('Provides controllability/observability for combinational logic.')," &
      "u8  ('Provides controllability/observability of AT1 LED inputs.')," &
      "u1  ('Provides test points in sequential logic.')," &
      "u22 ('Buffers the SQWV clock signal.')," &
      "u20 ('Performs floating-point conversions.')";

   -- Buses of the module
   attribute BUS_COMPOSITION of gdm : entity is
   --  bus[length] (reg, reg[bit], reg[bit,bit], ...)
      "da30[1]  (u19.inputs[7,7])," &
```

```
      "da31[1]  (u19.inputs[6,6])," &
      "da30d[1] (u19.inputs[5,5])"  ;
   attribute BUS_DESCRIPTION of gdm : entity is
   --  bus   (description)
      "da30  ('A one-bit bus.')," &
      "da31  ('Another one-bit bus.')," &
      "da30d ('A third one-bit bus.')"  ;

   -- Paths of the module
   constant path1 : STATIC_PATH := "u22, u1, u8, u19, u21, u20, u9";

   -- Constraint Descriptions
   attribute CONSTRAINTS of gdm : entity is
   --  constraint    (expression)
      "bad_direction (u20.oez = 0 and u20.dir = 1)";
   attribute CONSTRAINT_DESCRIPTION of gdm: entity is
   --  constraint    (description)
      "bad_direction ('You can''t do that with the GDM!')";

end gdm;
```

5.2. HSDL for Figure 3-1 of IEEE Std 1149.1-1990

This HSDL example was written for Figure 3-1, found on page 3-6 of IEEE Std 1149.1-1990. This is a simple serial connection of four devices.

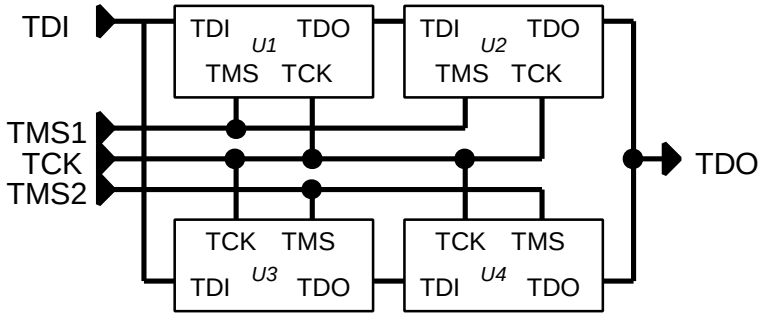



```
entity fig3_1 is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   -- external ASSET connector
   port (TCK      : in  bit;
         TDO      : out bit;
         TMS, TDI : in  bit);

   -- Use Statement
   use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
   use HSDL_module.all;      -- Get HSDL extensions for modules

   attribute MODULE_DESCRIPTION of fig3_1: entity is
      "The Fig3_1 Module (FIG3_1) is an example board that was created "  &
      "from Figure 3-1 of IEEE Std 1149.1-1990.";

   -- Package Pin Mapping
   attribute PIN_MAP of fig3_1 : entity is PHYSICAL_PIN_MAP;
   constant ONLY_PACKAGE : PIN_MAP_STRING :=
      "TDI:1, TDO:2, TCK:3, TMS:4";

   -- Scan Port Identification
   attribute TAP_SCAN_IN    of TDI  : signal is true;
   attribute TAP_SCAN_MODE  of TMS  : signal is true;
   attribute TAP_SCAN_OUT   of TDO  : signal is true;
   attribute TAP_SCAN_CLOCK of TCK  : signal is (20.0e6, BOTH);

   -- Member Description
   attribute MEMBERS of fig3_1 : entity is
      "u1  (chip, std_package)," &
      "u2  (chip, std_package)," &
      "u3  (chip, std_package)," &
      "u4  (chip, std_package) " ;
```

```
   -- Paths of the module
   constant PATH : STATIC_PATH := "u1, u2, u3, u4";

end fig3_1;
```

5.3. HSDL for Figure 3-2 of IEEE Std 1149.1-1990

This HSDL example was written for Figure 3-2, found on page 3-6 of IEEE Std 1149.1-1990. The four devices are connected in two paralleled serial chains.

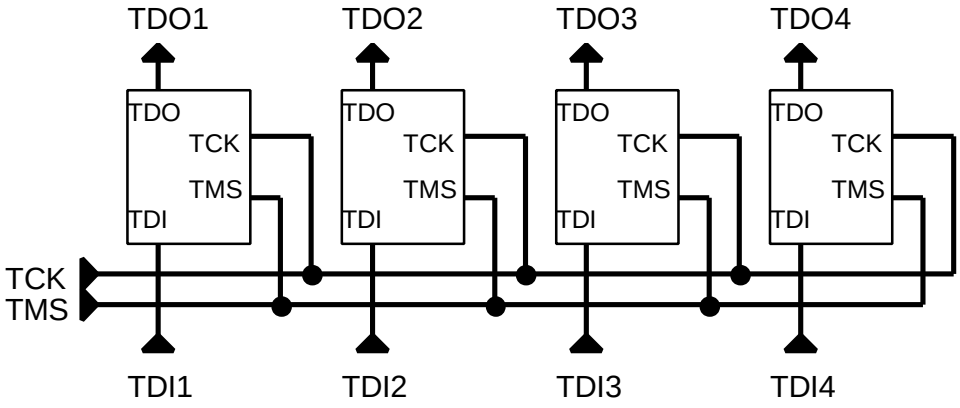



```
entity fig3_2 is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   -- external ASSET connector
   port (TCK        : in  bit;
         TDO        : out bit;
         TMS1, TMS2 : in  bit;
         TDI        : in  bit);

   -- Use Statement
   use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
   use HSDL_module.all;      -- Get HSDL extensions for modules

   attribute MODULE_DESCRIPTION of fig3_2: entity is
      "The Fig3_2 Module (FIG3_2) is an example board that was created "  &
      "from Figure 3-2 of IEEE Std 1149.1-1990.";

   -- Package Pin Mapping
   attribute PIN_MAP of fig3_2 : entity is PHYSICAL_PIN_MAP;
   constant ONLY_PACKAGE : PIN_MAP_STRING :=
      "TDI:1, TDO:2, TCK:3, TMS1:4, TMS2:5";

   -- Scan Port Identification
   attribute TAP_SCAN_IN    of TDI  : signal is true;
   attribute TAP_SCAN_MODE  of TMS1 : signal is true;
   attribute TAP_SCAN_MODE  of TMS2 : signal is true;
   attribute TAP_SCAN_OUT   of TDO  : signal is true;
   attribute TAP_SCAN_CLOCK of TCK  : signal is (20.0e6, BOTH);

   -- Member Description
   attribute MEMBERS of fig3_2 : entity is
      "u1  (chip, std_package)," &
      "u2  (chip, std_package)," &
      "u3  (chip, std_package)," &
      "u4  (chip, std_package) " ;

   -- Paths of the module
   constant J1 : EXTERNAL_PATH := "TDI, TDO, TCK, TMS1";
   constant J2 : EXTERNAL_PATH := "TDI, TDO, TCK, TMS2";

   constant PATH1 : STATIC_PATH := "J1, u1, u2";

   constant PATH2 : STATIC_PATH := "J2, u3, u4";
```

```
end fig3_2;
```

5.4. HSDL for Figure 3-3 of IEEE Std 1149.1-1990

This HSDL example was written for Figure 3-3, found on path 3-7 of IEEE Std 1149.1-1990. This module consists of four independent scan paths with common TMS and TCK signals.




```
entity fig3_3 is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   -- external ASSET connector
   port (TCK        : in  bit;
         TDO1       : out bit;
         TDO2       : out bit;
         TDO3       : out bit;
         TDO4       : out bit;
         TMS        : in  bit;
         TDI1       : in  bit;
         TDI2       : in  bit;
         TDI3       : in  bit;
         TDI4       : in  bit);

   -- Use Statement
   use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
   use HSDL_module.all;      -- Get HSDL extensions for modules

   attribute MODULE_DESCRIPTION of fig3_3: entity is
      "The Fig3_3 Module (FIG3_3) is an example board that was created "  &
      "from Figure 3-3 of IEEE Std 1149.1-1990.";

   -- Package Pin Mapping
   attribute PIN_MAP of fig3_3 : entity is PHYSICAL_PIN_MAP;
   constant ONLY_PACKAGE : PIN_MAP_STRING :=
      "TDI1:1, TDI2:2, TDI3:3, TDI4:4," &
      "TDO1:5, TDO2:6, TDO3:7, TDO4:8," &
      "TCK:9, TMS:10";

   -- Scan Port Identification
   attribute TAP_SCAN_IN    of TDI1 : signal is true;
   attribute TAP_SCAN_IN    of TDI2 : signal is true;
   attribute TAP_SCAN_IN    of TDI3 : signal is true;
   attribute TAP_SCAN_IN    of TDI4 : signal is true;
   attribute TAP_SCAN_MODE  of TMS  : signal is true;
   attribute TAP_SCAN_OUT   of TDO1 : signal is true;
   attribute TAP_SCAN_OUT   of TDO2 : signal is true;
   attribute TAP_SCAN_OUT   of TDO3 : signal is true;
   attribute TAP_SCAN_OUT   of TDO4 : signal is true;
   attribute TAP_SCAN_CLOCK of TCK  : signal is (20.0e6, BOTH);
```

```
   -- Member Description
   attribute MEMBERS of fig3_3 : entity is
      "u1  (chip, std_package)," &
      "u2  (chip, std_package)," &
      "u3  (chip, std_package)," &
      "u4  (chip, std_package) " ;

   -- Paths of the module
   constant J1 : EXTERNAL_PATH := "TDI1, TDO1, TCK, TMS";
   constant J2 : EXTERNAL_PATH := "TDI2, TDO2, TCK, TMS";
   constant J3 : EXTERNAL_PATH := "TDI3, TDO3, TCK, TMS";
   constant J4 : EXTERNAL_PATH := "TDI4, TDO4, TCK, TMS";

   constant PATH1 : STATIC_PATH := "J1, u1";
   constant PATH2 : STATIC_PATH := "J2, u2";
   constant PATH3 : STATIC_PATH := "J3, u3";
   constant PATH4 : STATIC_PATH := "J4, u4";

end fig3_3;
```

5.5. HSDL for Fictional Module

This HSDL example was written for a fictional board. The FICTION module contains nine devices, one SPL, one dedicated test connector, a dynamic path, and four external paths.

```
entity fiction is
   -- Generic Parameter
   generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

   -- Logical Port Description
   -- external ASSET connector
   port (LCLK           : inout bit;
         UUTCLK         : out bit;
         TMSR           : in  bit;
         GND            : linkage bit_vector (1 to 7);
         EVT0           : linkage bit;
         TDI            : in  bit;
         TDO            : out bit;
         TMS, TRST      : inout bit;

         -- ram32 slot connectors
         J2_TCK, J2_TMS : inout bit;
         J2_TDI         : out bit;
         J2_TDO         : in  bit;
         J2_DATABUS     : inout bit_vector (31 downto 0);
         J2_ADDRBUS     : out bit_vector (23 downto 0);
         J2_RW          : out bit;

         -- backplane slot connectors (scannable VGA feature connectors!)
         J3_TCK, J3_TMS : inout bit;
         J3_TDI         : out bit;
         J3_TDO         : in  bit;
         J3_RED         : out bit;
         J3_GRN         : out bit;
         J3_BLU         : out bit;
         J3_REDRTN      : in  bit;
         J3_GRNRTN      : in  bit;
         J3_BLURTN      : in  bit;
         J3_HSYNC       : out bit;
         J3_VSYNC       : out bit;
         J3_SYNCRTN     : in  bit;
         J3_ID          : in  bit_vector (0 to 2);

         J4_TCK, J4_TMS : inout bit;
         J4_TDI         : out bit;
         J4_TDO         : in  bit;
         J4_RED         : out bit;
         J4_GRN         : out bit;
         J4_BLU         : out bit;
         J4_REDRTN      : in  bit;
```

```
        J4_GRNRTN      : in  bit;
        J4_BLURTN      : in  bit;
        J4_HSYNC       : out bit;
        J4_VSYNC       : out bit;
        J4_SYNCRTN     : in  bit;
        J4_ID          : in  bit_vector (0 to 2);

        J5_TCK, J5_TMS : inout bit;
        J5_TDI         : out bit;
        J5_TDO         : in  bit;
        J5_RED         : out bit;
        J5_GRN         : out bit;
        J5_BLU         : out bit;
        J5_REDRTN      : in  bit;
        J5_GRNRTN      : in  bit;
        J5_BLURTN      : in  bit;
        J5_HSYN        : out bit;
        J5_VSYN        : out bit;
        J5_HSYNC       : out bit;
        J5_VSYNC       : out bit;
        J5_SYNCRTN     : in  bit;
        J5_ID          : in  bit_vector (0 to 2));

-- Use Statement
use STD_1149_1_1990.all;  -- Get Std 1149.1-1990 attributes and definitions
use HSDL_module.all;      -- Get HSDL extensions for modules

attribute MODULE_DESCRIPTION of fiction: entity is
   "The Fictional Module (FICTION) is an example board that was created "  &
   "from Adam Sheppard's imagination to help define HSDL.  It is the "     &
   "motherboard for a computer.  The motherboard contains an SPL, a "      &
   "manual scan path switch, and four slots.  This is a MultiMedia(TM) PC.";

-- Port Descriptions
attribute PORT_DESCRIPTION of EVT0 : signal is
   "Unimplemented event-qualification line.";

-- Package Pin Mapping
attribute PIN_MAP of fiction : entity is PHYSICAL_PIN_MAP;
constant ONLY_PACKAGE : PIN_MAP_STRING :=
   "LCLK: J1_1, UUTCLK: J1_2, " &
   "TDI: J1_3, TDO: J1_4, TMS: J1_5, TMSR: J1_6, TRST: J1_7," &
   "EVT0: J1_8, GND: (J1_9, J1_10, J1_11, J1_12, J1_13, J1_14, J1_15)" &

   "J2_TCK: J2_1, J2_TMS: J2_2, J2_TDI: J2_3, J2_TDO: J2_4," &
   "J2_DATABUS: (J2_5, J2_6, J2_7, J2_8, J2_9, J2_10,J2_11,J2_12," &
                "J2_13,J2_14,J2_15,J2_16,J2_17,J2_18,J2_19,J2_20," &
                "J2_21,J2_22,J2_23,J2_24,J2_25,J2_26,J2_27,J2_28," &
                "J2_29,J2_30,J2_31,J2_32,J2_33,J2_34,J2_35,J2_36)," &
   "J2_ADDRBUS: (J2_37,J2_38,J2_39,J2_40,J2_41,J2_42,J2_43,J2_44," &
                "J2_45,J2_46,J2_47,J2_48,J2_49,J2_50,J2_51,J2_52," &
                "J2_53,J2_54,J2_55,J2_56,J2_57,J2_58,J2_59,J2_60)," &
   "J2_RW: J2_61," &

   "J3_TCK: J3_16, J3_TMS: J3_17, J3_TDI: J3_18, J3_TDO: J3_19, " &
   "J3_RED: J3_1, J3_GRN: J3_2, J3_BLU: J3_3, " &
   "J3_REDRTN: J3_6, J3_GRNRTN: J3_7, J3_BLURTN: J3_8," &
   "J3_HSYNC: J3_13, J3_VSYNC: J3_14, J3_SYNCRTN: J3_10, " &
   "J3_ID: (J3_4, J3_12, J3_11)," &

   "J4_TCK: J4_16, J4_TMS: J4_17, J4_TDI: J4_18, J4_TDO: J4_19, " &
   "J4_RED: J4_1, J4_GRN: J4_2, J4_BLU: J4_3, " &
   "J4_REDRTN: J4_6, J4_GRNRTN: J4_7, J4_BLURTN: J4_8," &
   "J4_HSYNC: J4_13, J4_VSYNC: J4_14, J4_SYNCRTN: J4_10, " &
   "J4_ID: (J4_4, J4_12, J4_11)," &

   "J5_TCK: J5_16, J5_TMS: J5_17, J5_TDI: J5_18, J5_TDO: J5_19, " &
   "J5_RED: J5_1, J5_GRN: J5_2, J5_BLU: J5_3, " &
   "J5_REDRTN: J5_6, J5_GRNRTN: J5_7, J5_BLURTN: J5_8," &
   "J5_HSYNC: J5_13, J5_VSYNC: J5_14, J5_SYNCRTN: J5_10, " &
   "J5_ID: (J5_4, J5_12, J5_11)";
```

```
   -- Scan Port Identification
   attribute TAP_SCAN_IN    of TDI  : signal is true;
   attribute TAP_SCAN_MODE  of TMS  : signal is true;
   attribute TAP_SCAN_OUT   of TDO  : signal is true;
   attribute TAP_SCAN_RESET of TRST : signal is true;
   attribute TAP_SCAN_CLOCK of LCLK : signal is (20.0e6, BOTH);

   attribute TAP_SCAN_IN    of J2_TDO : signal is true;
   attribute TAP_SCAN_MODE  of J2_TMS : signal is true;
   attribute TAP_SCAN_OUT   of J2_TDI : signal is true;
   attribute TAP_SCAN_CLOCK of J2_TCK : signal is (20.0e6, BOTH);

   attribute TAP_SCAN_IN    of J3_TDO : signal is true;
   attribute TAP_SCAN_MODE  of J3_TMS : signal is true;
   attribute TAP_SCAN_OUT   of J3_TDI : signal is true;
   attribute TAP_SCAN_CLOCK of J3_TCK : signal is (20.0e6, BOTH);

   attribute TAP_SCAN_IN    of J4_TDO : signal is true;
   attribute TAP_SCAN_MODE  of J4_TMS : signal is true;
   attribute TAP_SCAN_OUT   of J4_TDI : signal is true;
   attribute TAP_SCAN_CLOCK of J4_TCK : signal is (20.0e6, BOTH);

   attribute TAP_SCAN_IN    of J5_TDO : signal is true;
   attribute TAP_SCAN_MODE  of J5_TMS : signal is true;
   attribute TAP_SCAN_OUT   of J5_TDI : signal is true;
   attribute TAP_SCAN_CLOCK of J5_TCK : signal is (20.0e6, BOTH);

   -- Member Description
   attribute MEMBERS of fiction : entity is
      "u1  (i486,        FK_PACKAGE)," &
      "u2  (sn74act8997, FK_PACKAGE)," &
      "u3  (sn74bct8245, FK_PACKAGE)," &
      "u4  (sn74bct8245, FK_PACKAGE)," &
      "u5  (sn74bct8245, FK_PACKAGE)," &
      "u6  (sn74bct8245, FK_PACKAGE)," &
      "u7  (sn74bct8244, FK_PACKAGE)," &
      "u8  (sn74bct8244, FK_PACKAGE)," &
      "u9  (tms320c40,   FK_PACKAGE)," &
      "u10 (sn74bct8373, FK_PACKAGE)"  ;
   attribute MEMBER_DESCRIPTION of fiction : entity is
      "u1  ('CPU.')," &
      "u2  ('The SPL is used to isolate motherboard RAM and the 32-bit slot " &
           "from the rest of the test logic to improve failure analysis.')," &
      "u3  ('Data bus between i486 and memory, lines d0-d7.')," &
      "u4  ('Data bus between i486 and memory, lines d8-d15.')," &
      "u5  ('Data bus between i486 and memory, lines d16-d23.')," &
      "u6  ('Data bus between i486 and memory, lines d24-d32.')," &
      "u7  ('Chipset override control logic.')," &
      "u8  ('Hard disk controller override logic.')," &
      "u9  ('DSP used to oversee multimedia functions.')," &
      "u10 ('Keyboard override control logic.')" ;

   -- Symbol Table Descriptions

   -- Buses of the module

   -- Paths of the module
   constant J1 : EXTERNAL_PATH := "TDI, TDO, TMS, LCLK, TRST";
   attribute PATH_DESCRIPTION of J1 : constant is
      "J1 is the scan test connector for this board.";

   constant J2 : EXTERNAL_PATH := "J2_TDI, J2_TDO, J2_TMS, J2_TCK";
   constant J3 : EXTERNAL_PATH := "J3_TDI, J3_TDO, J3_TMS, J3_TCK";
   constant J4 : EXTERNAL_PATH := "J4_TDI, J4_TDO, J4_TMS, J4_TCK";
   constant J5 : EXTERNAL_PATH := "J5_TDI, J5_TDO, J5_TMS, J5_TCK";

   constant subpath1 : STATIC_PATH := "u3, u4, u5, u6";
   constant short : STATIC_PATH := "";
   attribute PATH_DESCRIPTION of short : constant is
      "Short is an empty static path, representing a wire between TDI and TDO.";
```

```
   constant dpath : DYNAMIC_PATH :=
      "0 (u9:dpath, short:1    )," &
      "1 (u9:1,     short:dpath)";
   attribute PATH_DESCRIPTION of dpath : constant is
      "The dynamic path dpath is used to remove the C40 from " &
      "the scan path if the device is not mounted on the board.";

   constant boardpath : STATIC_PATH :=
      "J1, u1, u2, u7, J3, J4, J5, u8, dpath, u10";

   attribute CONNECTIONS of fiction : entity is
      "u2 (ssp1:subpath1, ssp2:j2, ssp3:*, ssp4:*)";

   -- Constraint Descriptions

end fiction;
```

A.
Hierarchical Scan Description Language Syntax

A.1. List of HSDL Statements

attribute *BOUNDARY_SYMBOLS* (HSDL devices)

Description

This optional "attribute" statement associates a symbol table with a boundary-scan register cell attached to a device port. The TDI or TDI/TDO symbols named in the symbol table can then be shifted into the specified cells, and the TDI/TDO or TDO symbols named in the symbol table can be used as replacements for bit patterns that are captured and shifted out of the specified cells. The cells "behind" the device ports can be controlled symbolically by using names instead of numbers. Usability is increased because the test engineer no longer needs to remember the bit patterns, just the names.

For simple device ports with only one boundary-scan cell attached to them, the symbol table can be thought of as being associated with the device port itself. For device ports with two or more cells attached to them, the situation is more complex. A symbol table can be associated with each type of cell attached to the port. For example, a simple bidirectional cell can consist of three cells: an input cell, an output cell, and a control cell.

The connection between boundary-scan register cells and device ports is made in the BOUNDARY_REGISTER attribute of BSDL. The input and output cell functions are directly attached to the device port; the control cell is indirectly attached to the device port through the optional <Disable Cell> portion of the cell's description. Thus, cells with the functions input, output2, output3, control, controlr, clock, and bidir can have a symbol table associated with them.

Internal cells cannot be associated with a symbol table using this attribute because they are not attached to a device port. A symbol table may be associated with an internal cell by defining a one-bit bus on that internal cell (using the BUS_COMPOSITION attribute) and associating a symbol table with the bus.

Syntax

attribute BOUNDARY_SYMBOLS of <*device id*> : entity is
"<*Boundary Symbol List*>";

<*Boundary Symbol List*> is:
<*Boundary Symbol*>
<*Boundary Symbol*>, <*Boundary Symbol*>
<*Boundary Symbol*>,...<*Boundary Symbol*>

<*Boundary Symbol*> is:
<*Port Name*> (<*Function*>, <*Symbol Table Name*>)

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Port Name>	A <VHDL identifier> giving the name of a previously defined port to associate a symbol table with.
<Function>	A <VHDL identifier> giving the function of the port to associate a symbol table with.
<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table to attach to the test register. A symbol table may be attached to any number of test registers.

Examples

```
attribute BOUNDARY_SYMBOLS of ttl74bct8374 : entity is
   "CLK    (input, clk_symbols)," &
   "OC_NEG (input, oc_symbols)";
```

See Also attribute BUS_SYMBOLS (HSDL), attribute SYMBOL_OF_TDI (HSDL), attribute SYMBOL_OF_TDO (HSDL), constant SYMBOL_TABLE (HSDL).

attribute *BUS_COMPOSITION* (HSDL)

Description

This optional "attribute" statement defines logical buses that may be present in the entity. A bus is a collection of bits from one or more test registers, chosen to represent a natural grouping of data signals in a design. Obvious examples of buses include the address and data buses common to microprocessor designs. Other examples include the manufacturer, part number, and version fields of the IDCODE register, and the various status and control bits that may be present in internal, user-defined test data registers.

A bus can be created any time it would be more convenient or more natural to refer to scan cells in an order that differs from their physical ordering in a test register. This can be a common requirement, given that the design and layout of test registers is often controlled by concerns like reducing test logic overhead and conserving valuable "real estate". The bits in the resulting test registers may be in an arbitrary order not corresponding to their logical meaning.

When a bus is modified by the test controller, the corresponding bits in the test registers that make up the bus are also modified to reflect the change. When the value of a bus is retrieved by the test controller, the corresponding bits in the test registers that make up the bus are retrieved. In this way, a bus is a logical view into the physical test register.

Both device and module entities may define buses. A device bus is composed of subsets of one or more test registers or buses defined in the device entity. A module bus is similar, but the test registers and buses on which it is based can also be contained in module members.

Defining a bus is very similar to the method for defining concatenated test registers, using the REGISTER_COMPOSITION attribute. The two statements share similar syntax.

Syntax

attribute BUS_COMPOSITION of <*entity id*> : entity is
  "<*Bus Definition List*>";

<*Bus Definition List*> is:
    <*Bus Definition*>
    <*Bus Definition*>, <*Bus Definition*>
    <*Bus Definition*>,...<*Bus Definition*>

<*Bus Definition*> is:
    <*Bus Name*>[<*Bus Width*>] (<*Bus Composition*>)

<*Bus Width*> is:
    <*VHDL Integer*>

<*Bus Composition*> is:
    <*Bus Component*>
    <*Bus Component*>, <*Bus Component*>
    <*Bus Component*>,...<*Bus Component*>

<*Bus Component*> is:
    <*Bus Qualifier*>.<*Bus Bits*>
    <*Bus Bits*>

<*Bus Qualifier*> is:
    <*Member Name*>
    <*Member Name*>.<*Bus Qualifier*>

<*Bus Bits*> is:
<*Register*>
<*Register Cell*>
<*Register Cell Range*>
<*Bus Name*>
<*Bus Single Bit*>
<*Bus Bit Range*>

<*Bus Single Bit*> is:
<*Bus Name*> [<*Bus Bit*>]

<*Bus Bit Range*> is:
<*Bus Name*> [<*Bus MSB*>, <*Bus LSB*>]

<*Bus MSB*> is:
<*Bus Bit*>

<*Bus LSB*> is:
<*Bus Bit*>

<*Bus Bit*> is:
<*VHDL Integer*>

Parameters

<entity id>	A <VHDL identifier> giving the name of the entity.
<Bus Definition>	Each bus definition describes how a bus is composed of the cells of other test registers or buses.
<Bus Name>	A <VHDL identifier> that defines the name of the bus being created.
<Bus Width>	A <VHDL Integer> that defines the width in bits of the bus being created. The width of the new bus must be equal to the sum of the widths of each of the bus components.
<Bus Component>	The bus component names the bits of each test register or bus that is a component of the bus being defined. In a device entity, the bus component may not have a bus qualifier, because all test registers and buses referenced must come from the device entity (a device has no members).
<Bus Qualifier>	The bus qualifier names the member that the bus bits come from. If necessary, more than one bus qualifier may be used to search deeper into a hierarchy of members to locate the bus bits. The first qualifier is the name of a member of the current module; the next qualifier is a member of that member's entity, and so forth until the desired module or device level in the hierarchy is reached.
<Bus Bits>	Each set of bits in the bus can be a complete test register or bus, a one-bit subset of a test register or bus, or a subset of a test register or bus including a range of bits.

<Register>	A <VHDL identifier> giving the name of a previously defined test register.
<Register Cell>	A single cell of a test register is a component of the bus.
<Register Cell Range>	Multiple cells of a test register form a component of the bus.
<Bus Single Bit>	A single bit of a bus is a component of the new bus.
<Bus Bit Range>	Multiple cells of a bus form a component of the new bus.
<Bus MSB>	The bit offset of the bus that will be the most significant bit of this component of the new bus. This need not be the most significant bit of the bus bit range. If the bus MSB is less than the bus LSB, the significance of the bits are reversed in the new bus.
<Bus LSB>	The bit offset of the bus that will be the least significant bit of this component of the new bus. This need not be the least significant bit of the bus bit range. If the bus MSB is less than the bus LSB, the significance of the bits are reversed in the new bus.
<Bus Bit>	A <VHDL Integer> giving the bit offset in the bus. 0 indicates the LSB of the bus.

Examples

```
attribute BUS_COMPOSITION of ttl74bct8374 : entity is
   "inputs[8]  (BOUNDARY[ 8,15])," &
   "outputs[8] (BOUNDARY[ 0, 7])," &
   "aux[2]     (BOUNDARY[16,17])," &
   "lfsr[16]   (BOUNDARY[ 0,15])";
```

```
attribute BUS_COMPOSITION of My_IC : entity is
   "version[4]       (idcode[31,28])," &
   "part_number[16]  (idcode[27,12])," &
   "manufacturer[11] (idcode[11, 1])";
```

```
attribute BUS_COMPOSITION of My_Board : entity is
   "addrbus[32] (u1.output, u2.output, u3.output, u4.output)," &
   "databus[32] (u5.a, u6,a, u7.a, u8.a)";
```

See Also Attribute IDCODE_REGISTER (BSDL), attribute MEMBERS (HSDL modules), attribute REGISTER_ACCESS (BSDL), attribute REGISTER_COMPOSITION (HSDL devices), attribute USERCODE_REGISTER (BSDL).

attribute *BUS_DESCRIPTION* (HSDL)

Description This optional "attribute" statement provides descriptions of the buses in an entity. Any or all of the buses defined in the entity may have a description. The description should provide enough information to the user so that no additional documentation is necessary in order to understand the use and meaning of the bus.

Syntax

attribute BUS_DESCRIPTION of <*device id*> : constant is
"<*Bus Descriptions*>";

<*Bus Descriptions*> is:
<*Bus Description*>
<*Bus Description*>, <*Bus Description*>
<*Bus Description*>,...<*Bus Description*>

<*Bus Description*> is:
<*Bus Name*> ('<*Description*>')

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Bus Name>	A <VHDL identifier> giving the name of a previously defined bus. A bus may not be listed more than once.
<Description>	A string providing a description of the use and meaning of the bus.

Examples

```
attribute BUS_DESCRIPTION of ttl74bct8374 : entity is
   "inputs  ('The eight cells of the Boundary-Scan register "   &
             "that are connected to the D inputs are combined " &
             "to form the input bus.')," &
   "outputs ('The eight cells of the Boundary-Scan register "   &
             "that are connected to the Q outputs are "         &
             "combined to form the output bus.')," &
   "aux     ('The two cells of the Boundary-Scan register "     &
             "that are connected to the CLK and OC_NEG pins "   &
             "are combined to form the aux bus.')," &
   "lfsr    ('The sixteen cells of the Boundary-Scan register " &
             "that are combined during PSA/PRPG operations to " &
             "form the Linear Feedback Shift Register.')";
```

See Also constant BUS_COMPOSITION (HSDL).

attribute *BUS_SYMBOLS* (HSDL)

Description

This optional "attribute" statement associates a symbol table with a bus. The TDI or TDI/TDO symbols named in the symbol table can then be shifted into the specified bus, and the TDI/TDO or TDO symbols named in the symbol table can be used as replacements for bit patterns that are captured and shifted out of the specified bus. The bus can be controlled symbolically by using names instead of numbers. Usability is increased because the test engineer no longer needs to remember the bit patterns, just the names.

Syntax

attribute BUS_SYMBOLS of <*device id*> : entity is
"<*Bus Symbol List*>";

<*Bus Symbol List*> is:
<*Bus Symbols*>
<*Bus Symbols*>, <*Bus Symbols*>
<*Bus Symbols*>,...<*Bus Symbols*>

<*Bus Symbols*> is:
<*Bus Name*> (<*Symbol Table Name*>)

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Bus Symbol List>	Each bus of the entity may have one symbol table associated with it. Neither all the buses nor all the symbol tables must be listed.
<Bus Name>	A <VHDL identifier> giving the name of a previously defined bus to attach a symbol table to.
<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table to attach to the bus. A symbol table may be attached to any number of buses.

Examples

```
attribute BUS_SYMBOLS of ttl74bct8374 : entity is
   "aux (AUX_SYMBOLS)";
```

See Also

attribute BUS_COMPOSITION (HSDL), attribute SYMBOL_OF_TDI (HSDL), attribute SYMBOL_OF_TDO (HSDL), constant SYMBOL_TABLE (HSDL).

attribute *CONNECTIONS* (HSDL modules)

Description

This "attribute" statement describes how the external paths of members are connected to the paths and devices of the current module. All external paths of all members must be connected to something when they are included in a module. This improves the diagnostic abilities of the HSDL translator, preventing unconnected external paths from "slipping through" until the UUT description is loaded into the test controller.

If an external path is to remain unconnected (for later connection in a higher-level module), it can be connected to a new external path defined in the current module. The decision to connect the external path can thus be postponed until the next higher-level module.

Syntax

attribute CONNECTIONS of <*module id*> : entity is
"<*Member Path Table*>";

<*Member Path Table*> is:
<*Member Path Entry*>
<*Member Path Entry*>, <*Member Path Entry*>
<*Member Path Entry*>,...<*Member Path Entry*>

<*Member Path Entry*> is:
<*Member Name*> (<*External Path List*>)

<*External Path List*> is:
<*External Path Entry*>
<*External Path Entry*>, <*External Path Entry*>
<*External Path Entry*>,...<*External Path Entry*>

<*External Path Entry*> is:
<*External Path Name*> : <*External Path Connection*>

<*External Path Connection*> is:
<*Static Path Entry*>
*

Parameters

<module id>	A <VHDL identifier> giving the name of the entity.
<Member Path Table>	A string connecting the external paths of each member to a path or device. Every member that has external paths that have not already been connected to a path or device must be listed here.
<Member Path Entry>	A member path entry lists the name of a member and a list of its external paths, attaching each to a path or device.
<External Path List>	An external path list lists each external path and connects it to a path or device. Every external path of the member that is not already connected must be listed.
<External Path Entry>	An external path entry associates an external path of the member with a static path, dynamic path, external path, member, or external path of

a member. The TAP signals of the static path entry are connected to the TAP signals of the external path. The external path is considered to be driving the

<External Path Name> A <VHDL Identifier> giving the name of an existing public external path of the member.

<External Path Connection> An external path connection identifies what the external path of the member is connected to. An external path may be connected to a static path, a dynamic path, an external path, a member, or an external path within a member. These connections model different hardware configurations that might occur.

Connecting an external path to a device indicates that the external path was probably a chip socket.

Connecting an external path to a module indicates that the external path was probably a daughterboard connector or a backplane connector.

Connecting an external path to a static path indicates that the member controlling the external path is probably a device or module with a secondary scan path. A secondary scan path is a slave path that can be configured in and out of the scan path by scanning commands to the master. The Texas Instruments SN74ACT8997 Scan Path Linker and SN74ACT8999 Scan Path Selector are examples of such devices. An example of a module like this might be a multi-chip module that controls a secondary scan path.

Connecting an external path to a dynamic path or a member's external path is simply a more complicated example of connecting an external path to a static path.

Connecting an external path to an external path defined in the current module is the method for postponing the connection to a higher-level module.

An asterisk (*) denotes an external path that is "open". Such an external path is not connected to any scannable hardware, and thus cannot be scanned through. The test controller can then prevent the use of any scan paths including this external path. This facility is useful for members whose external paths are controlled by a dynamic path. Examples include the TI SN74ACT8997 and SN74ACT8999 scan path linkers and selector that control secondary scan paths. If an external path of these devices is not connected, the test controller should

disallow the selection of that case in the dynamic path of the device.

Every external path of every member must be connected to something. This connection may be a short (an empty static path), something scannable (a path, member, or member external path), or nothing at all (an asterisk). The connection can be accomplished by placing the member in a STATIC_PATH or DYNAMIC_PATH list, or by listing the member in the CONNECTIONS attribute.

A member's external path must be connected to an external path defined in the current module in order to leave the connection undefined, so that it may be connected in a higher-level module.

Examples

```
constant subpath1 : STATIC_PATH := "u1, u2";
constant subpath2 : STATIC_PATH := "u3, u4, u5";
-- Attach subpath 1/2 to secondary scan path 1/3 of SN74ACT8997
-- Leave secondary scan path 2/4 permanently unconnected
attribute CONNECTIONS of My_Board : entity is
   "u9 (ssp1:subpath1, ssp2:*, ssp3:subpath2, ssp4:*)";
```

See Also constant DYNAMIC_PATH (HSDL modules), constant EXTERNAL_PATH (HSDL modules), constant STATIC_PATH (HSDL modules).

attribute *CONSTRAINTS* (HSDL)

Description

This optional "attribute" statement describes illegal conditions (constraints) of the hardware, conditions that the test controller is not allowed to create by scanning bit patterns into the hardware. These constraints usually mirror those that are prevented by design in the normal, functional mode of the hardware, but which can be easily circumvented using scan.

An example of a typical constraint is preventing more than one driver at a time from driving onto a bus. Usually, the hardware is specifically designed to prevent this occurrence. However, since boundary scan can take over the operation of the hardware, the hardware design that prevented two drivers from driving onto the bus is now overridden by scan. Constraints can be coded into the HSDL entity to repeat the same constraints designed into the hardware, thus preventing damage.

Constraints are written in the form of logical expressions that must evaluate to FALSE before a scan operation is allowed to take place. Each constraint is evaluated against the data to be shifted in, and if any of the constraints is TRUE, the test controller stops scanning and informs the operator that a constraint has been violated. The operation in progress is aborted, and scanning is forbidden until once again all constraints evaluate to FALSE.

Logical expressions are used in preference to illegal bit patterns, because the bit patterns could be quite lengthy and redundant. If a specific bit pattern is considered illegal, it can be checked for with an expression. Thus, expressions are more "expressive".

Constraints can appear in both device and module entities. Module constraints are often related to bus drivers, and tend to prevent conflicts between devices. Device constraints may take the form of values that should not be scanned into certain test registers or specific combinations of values in different test registers. Private instructions represent a form of constraint, in that a private instruction is a bit pattern that can never be scanned into the instruction register.

A constraint can use the operators shown in the following table. In many ways, these operators are like their counterparts in VHDL. The precedence, associativity, and meaning is the same. The operand values and their results differ, because the values in HSDL can include don't-care bits.

Operators

Operator	Rank	Description
a AND *b*	6	Logical AND
a OR *b*	6	Logical Inclusive OR
a XOR *b*	6	Logical Exclusive OR
a NAND *b*	6	Logical Negated AND
a NOR *b*	6	Logical Negated OR
a = *b*	5	Relational Equality
a <> *b*	5	Relational Inequality
a < *b*	5	Relational Less Than
a > *b*	5	Relational Greater Than
a <= *b*	5	Relational Less Than or Equal
a >= *b*	5	Relational Greater Than or Equal
a + *b*	4	Arithmetic Addition
a - *b*	4	Arithmetic Subtraction
- *a*	3	Arithmetic Negation
a * *b*	2	Arithmetic Multiplication
a / *b*	2	Arithmetic Division
NOT *a*	1	Logical Negation

Operators fall into the following groups: logical, relational, or arithmetic, and are shown in order of increasing precedence. *Precedence* defines the order of evaluation of operators. Operators with higher precedence are evaluated before operators with lower precedence. In the table, the logical negation operator (NOT) has the highest precedence with rank 1, and the other logical operators have the lowest precedence with rank 6. Most operators at the same precedence level evaluate their operands left-to-right. The exceptions are the logical negation (NOT) and arithmetic negation (-) operators, which are evaluated right-to-left. Parentheses can be used to override the usual order of evaluation.

An *associative* operator is one where the left and right operands can be exchanged without changing the result. The associative operators are identified in the discussion of each operator.

The logical operators are AND, OR, XOR, NAND, NOR, and NOT. All but NOT are associative operators. The logical operators are applied to each bit of the operands to determine the result. Their results are shown in the following tables.

Logical Operators

a	b	a AND b	a OR b	a XOR b	a NAND b	a NOR b
1	1	1	1	0	0	0
1	0	0	1	1	1	0
0	0	0	0	0	1	1

a	NOT a
1	0
0	1

The relational operators are =, <>, <, >, <=, >=. Only = and <> are associative operators. The relational operators are applied to all bits of the operand to

determine the result. The results of applying these operators to single bits are shown in the following tables.

Relational Operators

a	b	a = b	a <> b
1	1	1	0
1	0	0	1
0	0	1	0

a	b	a < b	a > b	a <= b	a >= b
1	1	0	0	1	1
1	0	0	1	0	1
0	1	1	0	1	0
0	0	0	0	1	1

The arithmetic operators are *, /, +, and -. Only multiplication, addition, and subtraction are associative operators. The arithmetic operators are applied to all bits of the operand to determine the result.

Operands are implicitly extended with leading zeroes during constraint evaluation, if necessary. The maximum number of bits that can be supported in the operands of a constraint is implementation-defined, and may vary depending on the constraint operator being applied.

If the final result of constraint evaluation contains any 1 bits, the result is true and the constraint has been violated.

If the value to be shifted in to a test register, bus, or port contains don't-care or unknown bits, and a constraint refers to that test register, bus, or port, the result of constraint evaluation is true and the constraint has been violated. The reasoning is that a constraint on a test register means that the contents of that test register are cared about; therefore don't-care and unknown bits in the value to shift into the test register are invalid.

Syntax

attribute CONSTRAINTS of <*entity id*> : entity is
"<*Constraint List*>";

<*Constraint List*> is:
<*Constraint*>
<*Constraint*>, <*Constraint*>
<*Constraint*>,...<*Constraint*>

<*Constraint*> is:
<*Constraint Name*> (<*Constraint Expression*>)

<*Constraint Name*> is:
<*VHDL identifier*>

<*Constraint Expression*> is:
<*Relational Expression*>
<*Relational Expression*> AND <*Constraint Expression*>
<*Relational Expression*> OR <*Constraint Expression*>
<*Relational Expression*> XOR <*Constraint Expression*>
<*Relational Expression*> NAND <*Constraint Expression*>
<*RelationalExpression*> NOR <*Constraint Expression*>

<*Relational Expression*> is:
<*Negative Expression*>
<*Negative Expression*> = <*Negative Expression*>
<*Negative Expression*> <> <*Negative Expression*>
<*Negative Expression*> < <*Negative Expression*>
<*Negative Expression*> > <*Negative Expression*>
<*Negative Expression*> <= <*Negative Expression*>
<*Negative Expression*> >= <*Negative Expression*>

<*Simple Expression*> is:
<*Negative Expression*>
<*Negative Expression*> + <*Simple Expression*>
<*Negative Expression*> - <*Simple Expression*>

<*Negative Expression*> is:
<*Term*>
- <*Term*>

<*Term*> is:
<*Factor*> * <*Term*>
<*Factor*> / <*Term*>

<*Factor*> is:
<*Primary Expression*>
NOT <*Primary Expression*>

<*Primary Expression*> is:
<*Constraint Item*>
(<*Constraint Expression*>)

<*Constraint Item*> is:
<*Bus Component*>
<*Port Component*>
<*Constraint Value*>

<*Port Component*> is:
<*Port Qualifier*> . <*Port Bits*>
<*Port Bits*>

<*Port Qualifier*> is:
<*Member Name*>
<*Member Name*> . <*Port Qualifier*>

<*Port Bits*> is:
<*Port Name*>
<*Port Single Bit*>
<*Port Bit Range*>

<*Port Single Bit*> is:
<*Port Name*> [<*Port Bit*>]

<*Port Bit Range*> is:
<*Port Name*> [<*Port MSB*>, <*Port LSB*>]

<*Port MSB*> is:
<*Port Bit*>

<*Port LSB*> is:
<*Port Bit*>

<*Port Bit*> is:
<*VHDL Integer*>

<*Constraint Value*> is:
<*Pattern*>
<*Symbol Name*>

Parameters

<entity id>	A <VHDL identifier> giving the name of the entity.
<Constraint Name>	A <VHDL identifier> defining the name of a new constraint. This name is used by the test controller when it informs the operator that a constraint has been violated.
<Constraint Expression>	A logical expression that may include logical operators to check combinations of values. When the expression evaluates to TRUE, the test controller halts and informs the operator that the constraint with the specified name has been violated.
<Relational Expression>	An expression that checks for equality or ordering of a constraint item and a constraint value.
<Simple Expression>	An expression that adds or subtracts a constraint item and a constraint value.
<Negative Expression>	An expression that inverts the sign of a constraint item or a constraint value.
<Term>	An expression that multiplies or divides a constraint item and a constraint value.
<Factor>	An expression that inverts the bits of a constraint item or a constraint value.
	<Primary Expression> A constraint item specifying scan cells to check. The primary expression may also be a parenthesized constraint expression, so that the usual evaluation order may be overridden.
<Constraint Item>	The constraint item specifies which bits are to be checked before shifting in, and can specify all or part of a test register, bus, or port. The constraint item may be contained in a member.
<Bus Component>	The bus component names the bits of a test register or bus that is checked by the constraint expression. In a device entity, the bus component may not have a bus qualifier, because all test registers and buses referenced must come from the device entity (a device has no members). See *attribute BUS_COMPOSITION (HSDL)* for more details.

<Port Component>	The port component names the bits of a port that is checked by the constraint expression. In a device entity, the port component may not have a port qualifier, because all ports referenced must come from the device entity (a device has no members). A port defined in a module may not be checked by a constraint expression, because module entities do not define the connections of ports to scan cells.
<Port Qualifier>	The port qualifier names the device member that the port bits come from. If necessary, more than one port qualifier may be used to search deeper into a hierarchy of members to locate the port bits. The first qualifier is the name of a member of the current module; the next qualifier is a member of that member's entity, and so forth until the device level in the hierarchy is reached.
<Port Bits>	Each set of bits in the port can be a complete port, a one-bit subset of a port, or a subset of a port including a range of bits.
<Port Single Bit>	A single bit of a port is checked.
<Port Bit Range>	Multiple cells of a port are checked.
<Port MSB>	The bit offset of the port that is the most significant bit checked. This need not be the most significant bit of the port bit range. If the port MSB is less than the port LSB, the significance of the bits is reversed.
<Port LSB>	The bit offset of the port that is the least significant bit checked. This need not be the least significant bit of the port bit range. If the port MSB is less than the port LSB, the significance of the bits is reversed.
<Port Bit>	A <VHDL Integer> giving the bit offset in the port. 0 indicates the LSB of the port.
<Constraint Value>	A pattern or symbol name that defines the value to check against the value to be shifted into the test register, bus, or port. The pattern may include don't-care bits (X). The symbol name must be contained in the symbol table attached to the <Constraint Item>, and must be a TDI symbol or a TDI/TDO symbol.

Examples

```
attribute CONSTRAINTS of My_Board : entity is
   "bus_conflict (u1.enable = ACTIVE AND u2.enable = ACTIVE)," &
   "backplane_conflict (u7.OC_NEG + u8.OC_NEG + u9.OC_NEG < 2)";
```

See Also attribute INSTRUCTION_PRIVATE (BSDL).

attribute *CONSTRAINT_DESCRIPTION* (HSDL)

Description This optional "attribute" statement provides descriptions of the constraints in an entity. Any or all of the constraints defined in the entity may have a description. The description should provide enough information so that the operator can understand the cause and reason of the constraint violation and how to correct it.

Syntax

attribute CONSTRAINT_DESCRIPTION of <*entity id*> : entity is
"<*Constraint Descriptions*>";

<*Constraint Descriptions*> is:
<*Constraint Description*>
<*Constraint Description*>, <*Constraint Description*>
<*Constraint Description*>,...<*Constraint Description*>

<*Constraint Description*> is:
<*Constraint Name*> ('<*Description*>')

Parameters

<entity id>	A <VHDL identifier> giving the name of the entity.
<Constraint Name>	A <VHDL identifier> giving the name of a previously defined constraint. A constraint may not be listed more than once.
<Description>	A string providing a description of the reason for the constraint, the nature of the constraint violation, and how to correct the violation. The test controller displays the constraint description, if available, along with the constraint name when a constraint is violated.

Examples

```
attribute CONSTRAINT_DESCRIPTION of My_Board : entity is
   "bus_conflict ('Drivers u1 and u2 are both enabled to "   &
               "drive the data bus.  Disable one of the " &
               "drivers and resume testing.')";
```

See Also attribute CONSTRAINTS (HSDL).

attribute *DEVICE_DESCRIPTION* (HSDL devices)

Description This optional "attribute" statement may appear following the "use" statements, and can be used to describe the type of the device itself. This description can be used to partially meet Documentation Requirement 12.3.1.a of IEEE Std 1149.1-1990.

Syntax

attribute DEVICE_DESCRIPTION of <*device id*> : entity is
"<*Device Description*>";

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Device Description>	A text string describing the device. The string may be arbitrarily long. No specific meaning is attached to its contents. The string should contain information describing the overall operation of the device, and may be used by the engineer to gain an understanding of the module. Tools may display this string when queried about the device.

Examples

```
attribute DEVICE_DESCRIPTION of My_IC : entity is
   "My_IC is a two-bit adder that computes the sum " &
   "of A and B on the rising edge of CLK.";
```

See Also attribute MODULE_DESCRIPTION (HSDL modules)

attribute *DYNAMIC_PATH_RESET* (HSDL modules)

Description

This "attribute" statement describes what happens to a dynamic path when the UUT passes through the Test-Logic-Reset state of the TAP state diagram. The state of the hardware controlling the dynamic path may be altered when this happens. If the hardware controlling the dynamic path happens to be a scannable device, the device will be placed in a normal mode and the configuration of the dynamic path will change.

If the dynamic path does not change state when the UUT passes through Test-Logic-Reset, a DYNAMIC_PATH_RESET attribute is not needed.

Syntax

attribute DYNAMIC_PATH_RESET of <*Dynamic Path Name*> : constant is "<*Dynamic Case Value*>";

Parameters

<Dynamic Path Name>	A <VHDL identifier> giving the name of a previously defined dynamic path.
<Dynamic Case Value>	A <VHDL Integer> specifying which case value of the dynamic path becomes active when the UUT is reset. The integer must match one of the dynamic case values of the dynamic path named by this statement. This case value also specifies the initial state of the dynamic path, because the UUT is in Test-Logic-Reset state on power-up.

Examples

```
attribute DYNAMIC_PATH_RESET of c40switch : constant is "1";
```

See Also

attribute REGISTER_RESET (HSDL devices), constant DYNAMIC_PATH (HSDL modules).

attribute *INSTRUCTION_DESCRIPTION* (HSDL devices)

Description

This optional "attribute" statement describes the function and purpose of the instruction opcodes of a device. The description can be used to meet the IEEE Std. 1149.1-1990 Documentation Requirement 12.3.1.b.ii. It should describe the operation of the instruction, any initialization needed, test registers altered, algorithms employed, and so forth. The description should be complete enough to use the instruction without additional documentation.

An instruction description is not required for the predefined instructions in IEEE Std 1149.1-1990 and Supplement A. These include SAMPLE/PRELOAD, EXTEST, INTEST, BYPASS, RUNBIST, CLAMP, and HIGHZ. An appropriate description is created for each of these instructions if no description is defined in the entity.

If the instructions INTEST and RUNBIST are available in the device, a description should be supplied by the user in lieu of the one created by the HSDL translator. The INTEST and RUNBIST instructions have many design-specific parameters that cannot be adequately described in HSDL.

Syntax

attribute INSTRUCTION_DESCRIPTION of <*device id*> : entity is
"<*Instruction Description Table*>";

<*Instruction Description Table*> is:
<*Instruction Description Entry*>
<*Instruction Description Entry*>, <*Instruction Description Entry*>
<*Instruction Description Entry*>,...<*Instruction Description Entry*>

<*Instruction Description Entry*> is:
<*Opcode Name*> ('<*Instruction Description*>')

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Instruction Description Table>	This parameter is a string containing instruction opcode names and associated descriptions.
<Instruction Description Entry>	Each instruction description entry associates an instruction opcode with a description.
<Opcode Name>	The opcode name gives the name of an instruction opcode previously defined in the INSTRUCTION_OPCODE attribute. An opcode name need not appear in the instruction description, but each opcode name may only appear once.
<Instruction Description>	A string describing the operation of the associated opcode. The string may be arbitrarily long. No specific meaning is attached to its contents. The string should contain information describing the overall operation of the module, and may be used by the engineer to gain an understanding of the module. Tools may display this string when queried about the module.

Examples

```
attribute INSTRUCTION_DESCRIPTION of My_IC : entity is
   "EXTEST  ('Select Boundary-Scan register in test mode; " &
             "control device inputs and outputs using the " &
             "contents of the Boundary.')," &
   "TRIBYP  ('Select BYPASS register in test mode; " &
             "control device outputs to high-impedance.')" ;
```

See Also attribute INSTRUCTION_DISABLE (BSDL), attribute INSTRUCTION_GUARD (BSDL), attribute INSTRUCTION_NORMAL (HSDL), attribute INSTRUCTION_OPCODE (BSDL), attribute INSTRUCTION_TEST (HSDL).

attribute *INSTRUCTION_NORMAL* (HSDL devices)

Description

This optional "attribute" statement identifies instruction opcodes of the device that do not affect the normal operation of the device in any way. The device continues to operate as if the test circuitry was inactive.

Instruction opcodes can be classified in three ways: normal-mode, test-mode, and unknown. A normal-mode instruction is one that does not affect the normal operation of the device, and is identified by listing it in the INSTRUCTION_NORMAL attribute. A test-mode instruction is one that affects the normal operation in some way by placing the device into a test mode, and is identified by listing it in the INSTRUCTION_TEST attribute. An instruction whose effects are not known cannot be listed in either attribute, so it is "unknown" whether the device operates in test or normal mode when it is loaded.

The instructions BYPASS, SAMPLE, IDCODE, and USERCODE, defined by 1149.1, must be normal-mode instructions and do not need to be listed in the INSTRUCTION_NORMAL attribute.

Syntax

attribute INSTRUCTION_NORMAL of <*device id*> : entity is
"<*Instruction List*>";

<*Instruction List*> is:
<*Instruction Name*>
<*Instruction Name*>,...<*Instruction Name*>

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Instruction Name>	A <VHDL identifier> giving the name of a previously defined instruction opcode which is classified as a normal-mode instruction. An instruction opcode cannot appear more than once. An instruction opcode cannot appear in both the INSTRUCTION_NORMAL attribute and the INSTRUCTION_TEST attribute.

Examples

```
attribute INSTRUCTION_NORMAL of ttl74bct8374 : entity is
   "BYPASS, SAMPLE, READBN, CELLTST, SCANCN";
```

See Also

attribute INSTRUCTION_OPCODE (BSDL), attribute INSTRUCTION_TEST (HSDL devices).

attribute *INSTRUCTION_TEST* (HSDL devices)

Description

This optional "attribute" statement identifies instruction opcodes of the device that affect the normal operation of the device in some way. All or part of the normal operation of the device is suspended and replaced with a test operation.

Instruction opcodes can be classified in three ways: normal-mode, test-mode, and unknown. A normal-mode instruction is one that does not affect the normal operation of the device, and is identified by listing it in the INSTRUCTION_NORMAL attribute. A test-mode instruction is one that affects the normal operation in some way by placing the device into a test mode, and is identified by listing it in the INSTRUCTION_TEST attribute. An instruction whose effects are not known cannot be listed in either attribute, so it is "unknown" whether the device operates in test or normal mode when it is loaded.

Whether a test-mode instruction affects the device output pins or affects the inputs to the device system logic is unspecified; in other words, the exact nature of the test mode is not described. A test-mode instruction is simply one that alters the device in some way.

The instructions EXTEST, INTEST, RUNBIST, HIGHZ, and CLAMP, defined by 1149.1, must be test-mode instructions and do not need to be listed in the INSTRUCTION_TEST attribute. Note that although 1149.1 does not specify the exact nature of system logic inputs and device outputs for these instructions, it does specify that they alter the normal operation of the device.

The instructions listed in the INSTRUCTION_DISABLE and INSTRUCTION_GUARD attributes must be test-mode instructions, but do not need to be listed in the INSTRUCTION_TEST attribute.

Syntax

attribute INSTRUCTION_TEST of <*device id*> : entity is
"<*Instruction List*>";

<*Instruction List*> is:
<*Instruction Name*>
<*Instruction Name*>,...<*Instruction Name*>

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Instruction Name>	A <VHDL identifier> giving the name of a previously defined instruction opcode which is classified as a test-mode instruction. An instruction opcode cannot appear more than once. An instruction opcode may not appear in both the INSTRUCTION_NORMAL attribute and the INSTRUCTION_TEST attribute.

Examples

```
attribute INSTRUCTION_TEST of ttl74bct8374 : entity is
   "EXTEST, INTEST, TRIBYP, SETBYP," &
   "RUNT,   READBT, TOPHIP, SCANCT"  ;
```

See Also

attribute INSTRUCTION_DISABLE (BSDL), attribute INSTRUCTION_GUARD (BSDL), attribute INSTRUCTION_NORMAL (HSDL devices), attribute INSTRUCTION_OPCODE (BSDL)

attribute *MEMBERS* (HSDL modules)

Description

This optional "attribute" statement identifies the members of the module. Members are the devices and submodules that are wired together to create the module. Each member has a unique name and is a certain type of device or module using a specific packaging option.

A module does not have to contain member devices or modules. One example of such a module is a dummy board that simply contains a short from TDI to TDO. Another, more realistic example is a system backplane that contains no scannable devices, only scannable card slots.

Syntax

attribute MEMBERS of <*module id*> : entity is
"<*Member Table*>";

<*Member Table*> is:
<*Member Entry*>
<*Member Entry*>, <*Member Entry*>
<*Member Entry*>, ... <*Member Entry*>

<*Member Entry*> is:
<*Member Name*> (<*Entity Name*> <*Package Name*>)

Parameters

<module id>	A <VHDL identifier> giving the name of the entity.
<Member Table>	This parameter is a string containing member definitions.
<Member Entry>	Each Member Entry parameter defines one of the members of the module, its name, its type (entity), and its package.
<Member Name>	A <VHDL identifier> defining a unique, user-selected name for the member.
<Entity Name>	The name of an existing entity, described in another HSDL or BSDL file.
<Package Name>	The name of a package defined as a PIN_MAP_STRING constant in the entity given by Entity Name.

Examples

```
attribute MEMBERS of My_Board : entity is
   "u1 (sn74bct8245, FK_PACKAGE)," &
   "u2 (sn74bct8240, DW_PACKAGE) " ;
```

See Also entity (BSDL), constant-(entity) (BSDL).

attribute *MEMBER_DESCRIPTION* (HSDL modules)

Description This optional "attribute" statement can be used to describe the purpose of each of the members of a module.

Syntax

attribute MEMBER_DESCRIPTION of <*module id*> : entity is
"<*Member Description Table*>";

<*Member Description Table*> is:
<*Member Description Entry*>
<*Member Description Entry*>, <*Member Description Entry*>
<*Member Description Entry*>,...<*Member Description Entry*>

<*Member Description Entry*> is:
<*Member Name*> ('<*Member Description*>')

Parameters

<module id>	A <VHDL identifier> giving the name of the entity.
<Member Description Table>	This parameter is a string that associates a description with one or more of the module members.
<Member Description Entry>	Each Member Description Entry gives the name of a member and associates a comment with that member.
<Member Name>	A <VHDL identifier> that gives the name of one of the members defined in the MEMBERS attribute. Each Member Name may appear no more than once in the <Member Description Table>. Not every member needs to have a description associated with it.
<Member Description>	A text string describing the member. The string may be arbitrarily long. No specific meaning is attached to its contents. The string should contain information describing the purpose of the member in the module, and may be used by the engineer to gain an understanding of the module. Tools may display this string when queried about the member.

Examples

```
attribute MEMBER_DESCRIPTION of My_Board : entity is
   "u1 ('The 245 in U1 is used as a backplane driver.')," &
   "u2 ('The 240 in U2 inverts the panel switches.')"     ;
```

See Also attribute DEVICE_DESCRIPTION (HSDL devices), attribute MEMBERS (HSDL modules), attribute MODULE_DESCRIPTION (HSDL modules).

attribute *MODULE_DESCRIPTION* (HSDL modules)

Description This optional "attribute" statement may appear following the "use" statements, and can be used to describe the type of the module itself. This description can be used to partially meet Documentation Requirement 12.3.1.a of IEEE Std 1149.1-1990.

Syntax

attribute MODULE_DESCRIPTION of <*module id*> : entity is
"<*Module Description*>";

Parameters

<module id>	A <VHDL identifier> giving the name of the entity.
<Module Description>	A text string describing the module. The string may be arbitrarily long. No specific meaning is attached to its contents. The string should contain information describing the overall operation of the module, and may be used by the engineer to gain an understanding of the module. Tools may display this string when queried about the module.

Examples

```
attribute MODULE_DESCRIPTION of My_Board : entity is
   "My_Board contains four megabytes of RAM, with a " &
   "32-bit address bus and 32-bit data bus.";
```

See Also attribute DEVICE_DESCRIPTION (HSDL devices).

attribute *PATH_DESCRIPTION* (HSDL modules)

Description

This optional "attribute" statement can be used to describe the purpose and function of each of the paths of the module. A separate path description is used to describe each of the paths of the module.

Syntax

attribute PATH_DESCRIPTION of <*Path Name*> : constant is
"<*Path Description*>";

<*Path Name*> is:
<*Static Path Name*>
<*Dynamic Path Name*>
<*External Path Name*>

Parameters

<Path Name>	A <VHDL identifier> giving the name of one of the static, dynamic, or external paths defined in the module. Each path can have at most one description attached to it.
<Path Description>	A string describing the purpose and function of the path, and may be used by the engineer to gain an understanding of the module.. The string may be arbitrarily long. No specific meaning is attached to its contents. Tools may display this string when queried about the path.

Examples

```
attribute PATH_DESCRIPTION of J1 : constant is
   "J1 is the scan test connector for this board.";

attribute PATH_DESCRIPTION of short : constant is
   "Short is an empty static path, representing a wire " &
   "between TDI and TDO.";

attribute PATH_DESCRIPTION of dpath : constant is
   "The dynamic path dpath is used to remove the C40 from " &
   "the scan path if the device is not mounted on the board.";
```

See Also

constant DYNAMIC_PATH (HSDL modules), constant EXTERNAL_PATH (HSDL modules), constant STATIC_PATH (HSDL modules).

attribute *PORT_DESCRIPTION* (HSDL)

Description

This optional "attribute" statement can be used to describe the purpose and function of each of the ports of the entity (device or module). A separate port description is used to describe each of the ports of the entity.

A port description is not required for the scan ports of the entity. Scan ports are those identified in the TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET attributes. An appropriate description is created for each of the scan ports if none is defined in the entity.

Syntax

attribute PORT_DESCRIPTION of <*Port Name*> : signal is
"<*Port Description*>";

Parameters

<Port Name>	A <VHDL identifier> giving the name of one of the ports.
<Port Description>	A string describing the purpose and function of the port. The string should describe the functional operation of the port, not the test operation. The string may be arbitrarily long. No specific meaning is attached to its contents. The string should contain information describing the overall operation of the module, and may be used by the engineer to gain an understanding of the module. Tools may display this string when queried about the module.

Examples

```
attribute PORT_DESCRIPTION of Q : signal is
   "Eight-bit output bus of the device.  All outputs can be "  &
   "set to high-impedance by placing a 1 on the OC_NEG pin.  " &
   "All outputs can be updated with the values on the input "  &
   "bus D on a positive transition of CLK.";
```

See Also

attribute TAP_SCAN_CLOCK (BSDL), attribute TAP_SCAN_IN (BSDL), attribute TAP_SCAN_MODE (BSDL), attribute TAP_SCAN_OUT (BSDL), attribute TAP_SCAN_RESET (BSDL), port (BSDL).

attribute *REGISTER_CAPTURE* (HSDL devices)

Description

This optional "attribute" statement describes the possible values that a test register of a device can capture, along with a description and an indication of whether each value represents a pass or failure value. This attaches rudimentary diagnostic ability to a test register and can be useful for test registers that capture a status value. The BSDL attribute INSTRUCTION_CAPTURE provides this ability for the instruction register.

Values captured by a test register that are not listed in the REGISTER_CAPTURE attribute are considered don't-care values; that is, they are neither pass nor failure values.

Syntax

attribute REGISTER_CAPTURE of <*device id*> : entity is
"<*Register Capture List*>";

<*Register Capture List*> is:
<*Register Capture*>
<*Register Capture*>, <*Register Capture*>
<*Register Capture*>,...<*Register Capture*>

<*Register Capture*> is:
<*Register*> (<*Capture Value*> , <*Pass Fail*>)
<*Register*> (<*Capture Value*> , <*Pass Fail*> , '<*Capture Description*>')

<*Capture Value*> is:
<*Pattern*>
<*Symbol Name*>

<*Pass Fail*> is:
PASS
FAIL

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Register Capture List>	A string describing the capture values of each test register. A test register may appear more than once in the list, to describe all pass and fail values.
<Register>	A <VHDL identifier> giving the name of a previously defined test register.
<Capture Value>	A value captured by the test register. Ambiguous values are not allowed. Ambiguous values are values containing don't-care bits such that two different values could represent the same binary pattern. In this case, ambiguity could arise from two conflicting patterns, or from a pattern and a symbol representing the same value. Undefined capture values for a test register are assumed to be Fail values.
<Pattern>	A bit pattern, optionally containing Xs to represent don't-care bits, that is captured by the

	test register. The same bit pattern may not appear twice in the capture list.
<Symbol Name>	A symbol name representing a value that is captured by the test register. The symbol must be a TDI/TDO symbol or a TDO symbol. The same symbol name may not appear twice in the capture list.
<Pass Fail>	A <VHDL identifier> describing whether the capture value is considered acceptable (PASS) or unacceptable (FAIL).
<Capture Description>	A string describing the meaning of the capture value. For example, if a specific bit of the capture value represents the success or failure of power-up BIST, identify the bit. Two different capture values would be created for this case: one that is a Pass value and another that is a Fail value. The test controller can use this string to describe the capture result.

Examples

```
attribute REGISTER_CAPTURE of My_IC : entity is
   "BYPASS (0, pass, 'The BYPASS register is working.')," &
   "BYPASS (1, fail, 'The BYPASS register is broken.')";
```

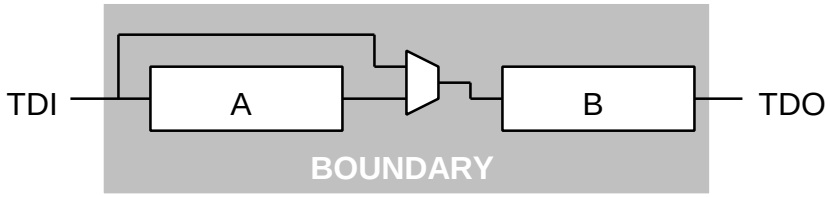
See Also attribute INSTRUCTION_CAPTURE (BSDL), attribute REGISTER_ACCESS (BSDL), attribute SYMBOL_OF_TDI (HSDL), attribute SYMBOL_OF_TDO (HSDL), constant SYMBOL_TABLE (HSDL).

attribute *REGISTER_COMPOSITION* (HSDL devices)

Description

This optional "attribute" statement describes test registers that are composed of portions of other test registers. Even though IEEE Std 1149.1-1990 permits the construction of a test register that is the concatenation of other test registers, BSDL cannot describe the new test register except as a completely new and separate register, sharing no cells with its component test registers. This can cause data-integrity problems for the test controller.

Consider the example shown here.



In this design, the boundary-scan register is a concatenation of two other registers, A and B. The instruction UseB selects only the B register for a DR-Scan operation. No instruction selects only the A register because the hardware design does not allow it. If the test controller loads a UseB instruction, scans through B, and then loads EXTEST, it may not correctly understand what is currently loaded into the boundary-scan register if the boundary-scan register is **not** described as a concatenation of A and B. This would lead to incorrect results in the best case, and seriously damaged hardware in the worst.

The boundary-scan register above could be described two different ways:

BOUNDARY is a concatenation of A and B.
B is a subset of the cells of BOUNDARY.

To avoid introducing more names for test registers that cannot be scanned through separately (like register A shown above), the REGISTER_COMPOSITION attribute works from the second perspective. Instead of describing the BOUNDARY as a concatenation of A and B, test register B is described as a subset of the cells of BOUNDARY. In certain hardware designs, both A and B could be made scannable, so the BOUNDARY could be described as a concatenation of A and B.

The decision as to which way to describe concatenated registers is based on the hardware design and how it is described in BSDL. The only way to name a test register is to define it in the REGISTER_ACCESS attribute along with the instructions that select it. If the test register has no instructions (like register A above), it cannot be named. Thus register B must be described as a subset of BOUNDARY.

Syntax

attribute REGISTER_COMPOSITION of <*device id*> : entity is
"<*Composition Table*>";

<*Composition Table*> is:
<*Composition Entry*>
<*Composition Entry*>, <*Composition Entry*>
<*Composition Entry*>,...<*Composition Entry*>

<*Composition Entry*> is:
<*Concatenated Register*> (<*Composition List*>)

<*Concatenated Register*> is:
<*Register*>

<*Composition List*> is:
<*Composition*>
<*Composition*>, <*Composition*>
<*Composition*>,...<*Composition*>

<*Composition*> is:
<*Register*>
<*Register Cell*>
<*Register Cell Range*>

<*Register Cell*> is:
<*Register*> [<*Register Bit*>]

<*Register Cell Range*> is:
<*Register*> [<*Register MSB*>, <*Register LSB*>]

<*Register MSB*> is:
<*Register Bit*>

<*Register LSB*> is:
<*Register Bit*>

<*Register Bit*> is:
<*VHDL Integer*>

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Composition Entry>	Each composition entry describes how a concatenated test register is composed of the cells of other test registers. The instruction register may not be composed of other test registers, nor may any test register be composed of parts of it.
<Concatenated Register>	A <VHDL identifier> giving the name of a test register previously declared in the REGISTER_ACCESS attribute. The predefined test registers BOUNDARY, BYPASS, and IDCODE can also be used. A test register cannot both be composed of and a part of another test register; i.e. A can be composed of B **or** B can be composed of A, but not both.
<Composition List>	The composition list defines which cells of which test registers are concatenated in what order to form the concatenated test register. The compositions are listed with the most significant composition as the first and the least significant composition as the last (MSB-to-LSB ordering). A specific cell (bit of a test register) may not occur more than once in a composition (otherwise a circular test register is created).
<Composition>	Each component of a concatenated register can be a complete test register, a one-cell subset of

		a test register, or a subset of a test register including a range of cells.
	<Register Cell>	A single cell of the test register is a component of the concatenated register.
	<Register Cell Range>	Multiple cells of the test register form a component of the concatenated register.
	<Register MSB>	The register bit offset of the cell that will be the most significant bit of this component of the concatenated register. This need not be the most significant bit of the register cell range. If the register MSB is less than the register LSB, the significance of the cells is reversed in the concatenated register.
	<Register LSB>	The register bit offset of the cell that will be the least significant bit of this component of the concatenated register. This need not be the least significant bit of the register cell range. If the register MSB is less than the register LSB, the significance of the cells is reversed in the concatenated register.
	<Register Bit>	A <VHDL Integer> giving the cell offset in the test register. 0 indicates the cell of the test register closest to its TDO output.

Examples

```
attribute REGISTER_ACCESS of My_IC : entity is
   "B[5] (UseB)";
attribute REGISTER_COMPOSITION of My_IC : entity is
   "B    (BOUNDARY[4,0])";
```

See Also attribute BUS_COMPOSITION (HSDL), attribute REGISTER_ACCESS (BSDL).

attribute *REGISTER_DESCRIPTION* (HSDL devices)

Description

This optional "attribute" statement provides descriptions of the test registers in a device. Any or all of the test registers defined in the device may have a description. The description should provide enough information to the user so that no additional documentation is necessary in order to understand the use and meaning of the test register.

Syntax

attribute REGISTER_DESCRIPTION of <*device id*> : constant is
"<*Register Descriptions*>";

<*Register Descriptions*> is:
<*Register Description*>
<*Register Description*>, <*Register Description*>
<*Register Description*>,...<*Register Description*>

<*Register Description*> is:
<*Register*> ('<*Description*>')

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Register>	A <VHDL identifier> giving the name of a previously defined test register. A test register may not be listed more than once.
<Description>	A string providing a description of the use and meaning of the test register. The string should be descriptive enough to meet IEEE Std 1149.1-1990 Documentation Requirement 12.3.1.b.iv.

Examples

```
attribute REGISTER_DESCRIPTION of ttl74bct8374 : entity is
   "BCR ('The Boundary Control Register is a design-specific " &
         "test data register used to specify the test "       &
         "function performed by the RUNT instruction.')";
```

See Also constant REGISTER_ACCESS (BSDL).

attribute *REGISTER_RESET* (HSDL devices)

Description

This optional "attribute" statement indicates the patterns loaded into test registers that are reset during Test-Logic-Reset state and at power-up. It ensures data integrity between the test controller and the UUT for user-defined test registers. An example of this is already defined in IEEE Std 1149.1-1990 for the instruction register. The standard states that the instruction register always loads the IDCODE instruction, or if not defined, the BYPASS instruction, on power-up or Test-Logic-Reset.

Syntax

attribute REGISTER_RESET of <*device id*> : entity is
"<*Register Reset List*>";

<*Register Reset List*> is:
<*Register Reset*>
<*Register Reset*>, <*Register Reset*>
<*Register Reset*>,...<*Register Reset*>

<*Register Reset*> is:
<*Register*> (<*Reset Value*>)

<*Reset Value*> is:
<*Pattern*>
<*Symbol Name*>

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Register>	A <VHDL identifier> giving the name of a previously defined test register. The device modifies the named test register on power-up and Test-Logic-Reset to the indicated value. Only the test register that are modified need to be listed. A test register may not be listed more than once.
<Reset Value>	A <VHDL integer> giving the value that is loaded into the test register on power-up and Test-Logic-Reset. The pattern or symbol name should not contain any don't-care bits.

Examples

```
attribute REGISTER_RESET of My_IC : entity is
   "INSTRUCTION (BYPASS)," &
   "BCR         (PSA)";
```

See Also attribute REGISTER_ACCESS (BSDL).

attribute *REGISTER_SYMBOLS* (HSDL devices)

Description This optional "attribute" statement associates a symbol table with a test register. The TDI or TDI/TDO symbols named in the symbol table can then be shifted into the specified test register, and the TDI/TDO or TDO symbols named in the symbol table can be used as replacements for bit patterns that are captured and shifted out of the specified test register. The test register can be controlled symbolically by using names instead of numbers. Usability is increased because the test engineer no longer needs to remember the bit patterns, just the names.

Syntax

attribute REGISTER_SYMBOLS of <*device id*> : entity is
"<*Register Symbol List*>";

<*Register Symbol List*> is:
<*Register Symbols*>
<*Register Symbols*>, <*Register Symbols*>
<*Register Symbols*>,...<*Register Symbols*>

<*Register Symbols*> is:
<*Register*> (<*Symbol Table Name*>)

Parameters

<device id>	A <VHDL identifier> giving the name of the entity.
<Register Symbol List>	Each test register of the device may have one symbol table associated with it. Neither all the test registers nor all the symbol tables must be listed.
<Register>	A <VHDL identifier> giving the name of a previously defined test register to attach a symbol table to.
<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table to attach to the test register. A symbol table may be attached to any number of test registers.

Examples

```
attribute REGISTER_SYMBOLS of <device id> : entity is
   "BCR (BCR_Opcodes)";
```

See Also attribute BUS_SYMBOLS (HSDL), attribute SYMBOL_OF_TDI (HSDL), attribute SYMBOL_OF_TDO (HSDL), constant SYMBOL_TABLE (HSDL).

attribute *SYMBOL_DEFAULT* (HSDL)

Description This optional "attribute" statement classifies one of the symbols of a symbol table as the default symbol. The default symbol matches any value not specifically assigned to any symbol in the symbol table. A default symbol is analogous to the BYPASS instruction. The BYPASS instruction is selected for any instruction pattern not specifically assigned to an instruction in the INSTRUCTION_OPCODE attribute.

Syntax

attribute SYMBOL_DEFAULT of <*Symbol Table Name*> : constant is
"<*Symbol Name*>";

Parameters

<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table.
<Symbol Name>	A <VHDL identifier> giving the name of the symbol in the specified symbol table to be classified as the default. This symbol is used by the test controller to represent any value not specified assigned to a symbol in the symbol table.

Examples

```
attribute SYMBOL_DEFAULT of BCR_Opcodes : constant is "PSA";
```

See Also attribute INSTRUCTION_OPCODE (BSDL), constant SYMBOL_TABLE (HSDL).

attribute *SYMBOL_DESCRIPTION* (HSDL)

Description

This optional "attribute" statement provides descriptions of the symbols in a symbol table. Any or all of the symbols defined in the symbol table may have a description. The description should provide enough information to the user so that no additional documentation is necessary in order to understand the use and meaning of the symbol name.

Syntax

attribute SYMBOL_DESCRIPTION of <*Symbol Table Name*> : constant is
"<*Symbol Description Table*>";

<*Symbol Description Table*> is:
<*Symbol Description Entry*>
<*Symbol Description Entry*>, <*Symbol Description Entry*>
<*Symbol Description Entry*>,...<*Symbol Description Entry*>

<*Symbol Description Entry*> is:
<*Symbol Name*> ('<*Symbol Description*>')

Parameters

<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table.
<Symbol Description Table>	A string associating each symbol name in the symbol table with a description. Each symbol name may be listed at most one time.
<Symbol Description Entry>	Each entry provides a description for a symbol.
<Symbol Name>	A <VHDL identifier> giving the name of a symbol previously defined in the named symbol table.
<Symbol Description>	A string providing a description of the use and meaning of the symbol.

Examples

```
attribute SYMBOL_DESCRIPTION of BCR_Opcodes : constant is
   "SAMTOG  ('Samples device inputs on input bus; toggles "    &
             "device outputs from output bus.')),"             &
   "PRPG    ('Conducts 16-bit Pseudo-Random Pattern "         &
             "Generation using the contents of the input and " &
             "output buses of the Boundary-Scan register as "  &
             "an initial value.  The Q outputs of the device " &
             "are set to the value in the output bus.  A new " &
             "pattern is generated on each TCK in "            &
             "Run-Test/Idle state.'),"                         &
   "PSA     ('Conducts 16-bit Parallel Signature Analysis "   &
             "using the contents of the input and output "     &
             "buses of the Boundary-Scan register as an "      &
             "initial value.  The Q outputs of the device "    &
             "are set to the value in the output bus.  A "     &
             "new checksum is generated on each TCK in "       &
             "Run-Test/Idle state.'),"                         &
   "PSAPRPG ('Simultaneous 8-bit PSA on the D inputs and "    &
             "8-bit PRPG on the Q outputs.')";
```

See Also constant SYMBOL_TABLE (HSDL).

attribute *SYMBOL_OF_TDI* (HSDL)

Description

This optional "attribute" statement categorizes certain symbol names in a symbol table as being valid only for shifting into TDI and applying to the system logic. Symbols listed in this attribute can be shifted in and applied to the UUT, but are never used as replacements for values that are captured and shifted out of the UUT.

TDI symbols have fewer restrictions than TDO symbols or TDI/TDO symbols. The values associated with two TDI symbols can be ambiguous and can even be identical. For example, the SAMPLE and PRELOAD instructions have identical instruction bit patterns. The ambiguity is acceptable because the test controller knows what bit pattern to shift in when given a TDI symbol name.

Syntax

attribute SYMBOL_OF_TDI of <*Symbol Table Name*> : constant is
"<*TDI Symbol List*>";

<*TDI Symbol List*> is:
<*Symbol Name*>
<*Symbol Name*>, <*Symbol Name*>
<*Symbol Name*>,...<*Symbol Name*>

Parameters

<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table.
<TDI Symbol List>	A list of the symbol names that are only valid for shifting into TDI. Symbols not appearing in the SYMBOL_OF_TDI or SYMBOL_OF_TDO attributes are valid for both TDI and TDO. A TDI symbol name may not also appear in the SYMBOL_OF_TDO attribute. A symbol name may not appear more than once in the list.
<Symbol Name>	A <VHDL identifier> giving the name of a symbol in the specified symbol table to be designated as a TDI symbol.

Examples

```
attribute SYMBOL_OF_TDI of BIST_Symbols : constant is
   "TEST_BOUNDARY, TEST_CORE, FRY_DEVICE, BACKDRIVE_OUTPUTS";
```

See Also attribute SYMBOL_OF_TDO (HSDL), constant SYMBOL_TABLE (HSDL).

attribute *SYMBOL_OF_TDO* (HSDL)

Description

This optional "attribute" statement categorizes certain symbol names in a symbol table as being valid only for represented bit patterns that are captured from the system logic and shifted out of TDO. Symbols listed in this attribute are used as replacements for values that are captured and shifted out of the UUT, but can never be shifted in and applied to the system logic.

TDO symbols cannot be ambiguous or identical. Ambiguous values are values containing don't-care bits such that two different values could represent the same binary pattern. Ambiguity is unacceptable for TDO symbols because when a bit pattern is captured and shifted out, the test controller cannot determine which symbol name to use as a replacement for the bit pattern. Two different symbols both represent that value, so a unique replacement is not possible. For this reason, ambiguous TDO symbols are not allowed.

Syntax

attribute SYMBOL_OF_TDO of <*Symbol Table Name*> : constant is
"<*TDO Symbol List*>";

<*TDO Symbol List*> is:
<*Symbol Name*>
<*Symbol Name*>, <*Symbol Name*>
<*Symbol Name*>,...<*Symbol Name*>

Parameters

<Symbol Table Name>	A <VHDL identifier> giving the name of a previously defined symbol table.
<TDO Symbol List>	A list of the symbol names that are only valid for replacing bit patterns shifted out of TDO. Symbols not appearing in the SYMBOL_OF_TDI or SYMBOL_OF_TDO attributes are valid for both TDI and TDO. A TDO symbol name may not also appear in the SYMBOL_OF_TDI attribute. A symbol name may not appear more than once in the list.
<Symbol Name>	A <VHDL identifier> giving the name of a symbol in the specified symbol table to be designated as a TDO symbol.

Examples

```
attribute SYMBOL_OF_TDO of BIST_Symbols : constant is
   "BIST_WORKED, ALU_FAILURE, CU_FAILURE, TOTAL_FAILURE";
```

See Also attribute SYMBOL_OF_TDI (HSDL), constant SYMBOL_TABLE (HSDL).

constant *DYNAMIC_PATH* (HSDL modules)

Description

This "constant" declaration is one of three that defines the scan paths of a module. This statement defines a dynamic path, a scan path whose contents are known at the time the HSDL file is written, but the placement of those contents in the scan path varies during the application of the test. Items in the dynamic path may be switched in and out of the scan path by controlling the connection of their TMS signals.

The dynamic path is useful for describing limited ad-hoc reconfiguration of the scan path in which only the TMS signals are modified. When an item in a dynamic path is active, the TMS signal of the item is connected to the TMS signal of the dynamic path as a whole. When inactive, the TMS signal of the item may be pulled high or low, placing the item in Test-Logic-Reset or Run-Test/Idle state, respectively.

The modification of TMS signals is outside the control of HSDL. The dynamic path merely describes how the scan path reconfiguration of the UUT operates, not how to control it. This control must be performed elsewhere.

This statement cannot describe scan paths whose clocks are stopped "in place" and resynchronized with the TAP state diagram later. TAP signals cannot be switched between scan paths with this statement. A dynamic path simply allows items to be placed into and removed from the scan path, and to describe what happens to these items when they are removed from the scan path.

A dynamic path bears a strong resemblance to the case statement in many programming languages. Each case statement value selects a different configuration of the dynamic path.

Syntax

constant <*Dynamic Path Name*> : DYNAMIC_PATH :=
  "<*Dynamic Path Table*>";

<*Dynamic Path Table*> is:
    <*Dynamic Path Case*>,...<*Dynamic Path Case*>

<*Dynamic Path Case*> is:
    <*Dynamic Case Value*> (<*Dynamic Path List*>)

<*Dynamic Case Value*> is:
    <*VHDL Integer*>

<*Dynamic Path List*> is:
    <*Dynamic Path Configuration*>
    <*Dynamic Path Configuration*>, <*Dynamic Path Configuration*>
    <*Dynamic Path Configuration*>,...<*Dynamic Path Configuration*>

<*Dynamic Path Configuration*> is:
    <*Static Path Entry*> : <*Dynamic Path State*>

<*Dynamic Path State*> is:

    <*Dynamic Path Name*>
    1
    0

Parameters

<Dynamic Path Name>	A <VHDL identifier> defining the name of the dynamic path. The name must be unique.
<Dynamic Path Table>	This string describes the different configurations allowed for the dynamic path. At least two cases must be present. If only one case is needed, use a static path.
<Dynamic Path Case>	Each dynamic path case defines a case value and describes the configuration of the items in the dynamic path when the dynamic path case is active.
<Dynamic Case Value>	A <VHDL Integer> that identifies the scan path configuration. Each dynamic case value must be unique within the DYNAMIC_PATH constant. At execution time, the test controller must be kept informed of the current case value for the dynamic path, so that it may conduct scans appropriately. Other software, hardware, etc. is needed to ensure that the scan path in the hardware is actually changed to reflect the current configuration.
<Dynamic Path List>	The dynamic path list describes each item contained in the dynamic path and its test-mode-select state. The dynamic path list for each dynamic case value must repeat the same dynamic path items. The state of every dynamic path item must be described in every case.
<Dynamic Path Configuration>	Each configuration lists a dynamic path item and indicates the state of the item by describing the state of its TAP_SCAN_MODE port.
<Static Path Entry>	Each dynamic path item can be any single item that can be contained in a static path: a member device or module, another path described in the module, or an external path in a member module.
<Dynamic Path State>	The dynamic path state indicates the state of the TAP_SCAN_MODE port of the dynamic path item. Using the name of the dynamic path indicates that the dynamic path item is connected to the TAP_SCAN_MODE port of the scan path. A 1 or 0 indicates that the TAP_SCAN_MODE port of the dynamic path item is either high or low, respectively, placing the device in Test-Logic-Reset or Run-Test/Idle. The dynamic path state keeps the test controller database consistent with the hardware state by informing the test controller of the current state of the hardware. Thus the test controller can apply any REGISTER_RESET actions which

may be present in the devices of the dynamic path.

Examples

```
constant short : STATIC_PATH := "";
constant c40switch : DYNAMIC_PATH :=  -- switch 'C40 out of path
   "0 (u29: 0, short: c40switch)," &  -- 'C40 is in Run-Test/Idle
   "1 (u29: c40switch, short: 0)"  ;  -- 'C40 is in scan path
```

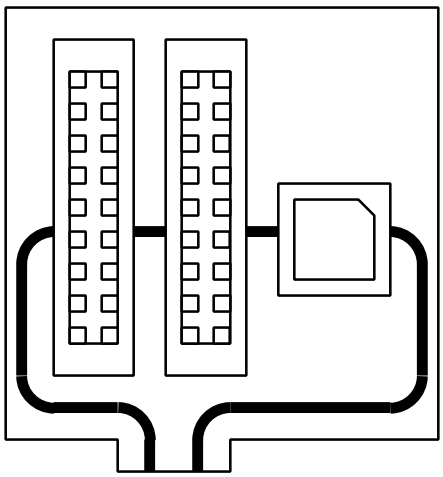
See Also attribute DYNAMIC_PATH_RESET (HSDL modules), attribute REGISTER_RESET (HSDL devices), attribute TAP_SCAN_MODE (BSDL), constant STATIC_PATH (HSDL modules).

constant *EXTERNAL_PATH* (HSDL modules)

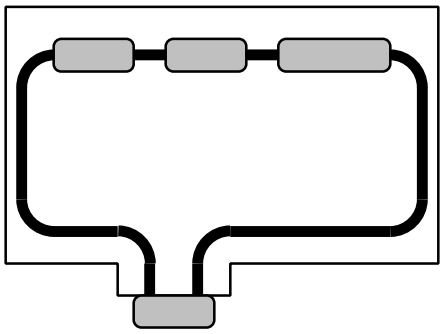
Description

This optional "constant" declaration is one of three that defines the scan paths of the module. This statement defines an external path, a scan path whose contents may not be known at the time the HSDL file is written. An external path often represents a scannable backplane slot, which a scannable board can be inserted into at a later time. When describing the backplane, the slots are described as external paths and thus any scannable board can be inserted at a later time. This scannable board is also described by an HSDL module file, and either a higher-level HSDL module file or the test controller can be used to indicate which HSDL module is plugged into the slot.

External paths can also represent chip sockets, daughterboard connectors, test controller scan connectors, and so forth. Consider the board shown below.



This example board contains two backplane slots, one chip socket, and one card-edge connector, all of which are scannable. No scannable devices or modules are wired onto the board, only connectors for plugging in other things later. From the HSDL point of view, this module appears as shown here.



So the module is composed of four external paths and no members. Every board has at least one external path, the one containing the ports used by the test controller to control the board. External paths "complete" the board scan path, making it a circular path that includes the test controller.

The definition of an external path is such that, under the proper circumstances, either the test controller or a scannable module could be plugged into the external path. Conceptually, and given the proper hardware design, the test

controller could control the UUT from any external path on the board. An external path contains one set of TAP signals, as described by the attributes TAP_SCAN_IN, TAP_SCAN_OUT, TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET. If a module has more than one set of TAP signals, all external paths must be defined to establish which path is associated with which set of signals. If the module has only one set of TAP signals, no external path definition is necessary, because the signals associated with the board scan path can be deduced. In this case, the board has only one external path, and its definition is optional to simplify writing HSDL for a typical board.

Syntax

constant <*External Path Name*> : EXTERNAL_PATH :=
"<*Port Name List*>";

<*Port Name List*> is:
<*Port Name*>, <*Port Name*>, <*Port Name*>, <*Port Name*>
<*Port Name*>, <*Port Name*>, <*Port Name*>, <*Port Name*>, <*Port Name*>

Parameters

<External Path Name>	A <VHDL identifier> defining the name of the external path. This name must be unique. By default, the external path is public.
<Port Name List>	This string identifies the ports of the module that control or are controlled by the external path. Four or five port names may be listed: one TAP_SCAN_IN port, one TAP_SCAN_OUT port, one TAP_SCAN_CLOCK port, one TAP_SCAN_MODE port, and optionally one TAP_SCAN_RESET port. Each port name listed must appear in one of the TAP_SCAN_IN / TAP_SCAN_OUT / TAP_SCAN_CLOCK / TAP_SCAN_MODE / TAP_SCAN_RESET attribute statements. The port names may be listed in any order.
<Port Name>	The port name listed must adhere to the following rules: • A port listed in the TAP_SCAN_IN attribute must have mode "in". • A port listed in the TAP_SCAN_OUT attribute must have mode "out". • Ports listed in the TAP_SCAN_CLOCK, TAP_SCAN_MODE, and TAP_SCAN_RESET attributes must **all** have mode *in*, *out*, or *inout*. If all are defined as *out*, the external path is a pure unit-under-test connector. A module may be connected to the external path, but a test controller may not. The external path is designed to control a scan path. If all are defined as *in*, the external path is a pure test connector. A test controller may be connected to the external path, but a module may not. The external path is

designed to allow a test controller to control the module. If all are defined as *inout*, the external path may be either a test connector or a unit-under-test connector, and either a module or a test controller may be connected.

Examples

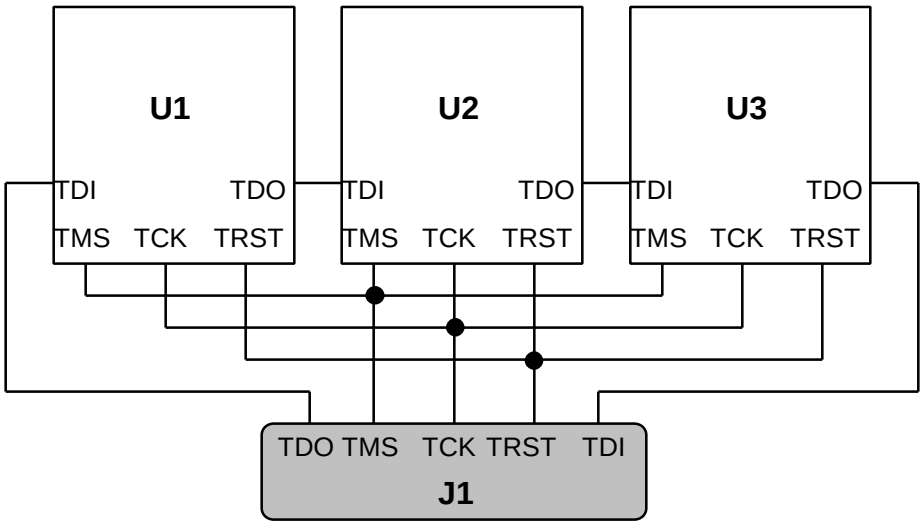
```
port (TDI, TCK, TMS, TRST: in bit; TDO: out bit);
   .
   .
   .
attribute TAP_SCAN_IN    of TDI  : signal is true;
attribute TAP_SCAN_OUT   of TDO  : signal is true;
attribute TAP_SCAN_CLOCK of TCK  : signal is (20.0e6, both);
attribute TAP_SCAN_MODE  of TMS  : signal is true;
attribute TAP_SCAN_RESET of TRST : signal is true;
   .
   .
   .
-- scannable connector J1
constant j1 : EXTERNAL_PATH := "TDI, TDO, TCK, TMS, TRST";
```

See Also constant STATIC_PATH (HSDL modules), attribute TAP_SCAN_CLOCK (BSDL), attribute TAP_SCAN_IN (BSDL), attribute TAP_SCAN_MODE (BSDL), attribute TAP_SCAN_OUT (BSDL), attribute TAP_SCAN_RESET (BSDL), port (BSDL).

constant *STATIC_PATH* (HSDL modules)

Description

This "constant" declaration is one of three that define the scan paths of a module. This statement defines a static path, a scan paths whose contents are known and fixed at the time the HSDL file is written. A static path describes the usual interconnection of scannable devices and boards: the TDI/TDO pins connected serially, with the TMS, TCK, and TRST of all devices and boards wired together. The figure here shows a typical static path.



The static path depicted here has four items in it: three devices and one external path. Since an external path describes the ports used to control it, the static path containing the external path completely describes the netlist of the test bus. All connectivity information for TDI, TDO, TMS, TCK, and TRST signals can be derived.

Note that a static path appears circular when examining its TDI-to-TDO connections. If all four entries in the example static path shown here were external paths with suitable wiring, the choice of which external path represents the TDI input to and TDO output from the module would be arbitrary. Any of the external paths could be chosen to plug the test controller into, in which case TDI begins at that point and TDO ends at that point.

On simple boards consisting of one external path and a straightforward serial connection between scannable devices and modules, only one path needs to be defined: a static path simply listing all the scannable devices and modules in order. In this case, an external path containing the TAP signals for the module is implicit, and it is assumed to be the first entry in the static path.

Syntax

constant <*Static Path Name*> : STATIC_PATH :=
  "<*Static Path List*>";

or

constant <*Static Path Name*> : STATIC_PATH := "";

<*Static Path List*> is:
    <*Static Path Entry*>
    <*Static Path Entry*>, <*Static Path Entry*>
    <*Static Path Entry*>,...<*Static Path Entry*>

<*Static Path Entry*> is:
<*Member Name*>
<*Static Path Name*>
<*Dynamic Path Name*>
<*Qualified External Path Name*>

<*Qualified External Path Name*> is:
<*External Path Name*>
<*Member Name*>.<*External Path Name*>

Parameters

<Static Path Name>	A <VHDL identifier> defining the name of the static path. This name must be unique.
<Static Path List>	The string that defines the composition and ordering of the static path. The leftmost static path entry is closest to TDI; the rightmost static path entry is closest to TDO. An empty static path list identifies a scan path containing nothing; i.e. a "short" in the scan path. This construct can be useful when resolving external paths and when constructing dynamic paths.
<Static Path Entry>	Each entry identifies a device, a static path, a dynamic path, or an external path that is included in this static path. The TAP signals of the entry are implicitly connected to the TAP signals of the path. The left-to-right ordering of entries indicates the TDI-to-TDO ordering of the scan path.
<Qualified External Path Name>	The name of an external path. The external path can be defined in this module or within a member of this module. External paths appearing in a static path describe the connections of the TAP ports of the module to the TAP signals of the scan path.

Examples

```
constant boardpath : STATIC_PATH := "J1, U1, U2, U3";
```

See Also attribute MEMBERS (HSDL modules), constant DYNAMIC_PATH (HSDL modules), constant EXTERNAL_PATH (HSDL modules).

constant *SYMBOL_TABLE* (HSDL)

Description

This optional "constant" declaration creates a symbol table. A symbol table defines symbol names and equates them to a list of values. The symbol name may be used in place of any of the values after it is defined. Symbols increase usability and readability. Instruction opcodes are similar to symbols, in that each instruction name represents any of the possible instruction bit patterns. A symbol table can later be associated with any test register, any bus, or any port of the entity. Thus, symbolic names can be defined for a constant to be shifted into or shifted out of any element of the design.

Instruction names are examples of symbols intended for shifting into a device and applying to the device logic. In HSDL this type of symbol is called a TDI symbol, because it is intended to be shifted into TDI. Symbols may also be defined that are intended to represent data captured by the device and shifted out through TDO; this type of symbol is called a TDO symbol.

Why make a distinction between TDI and TDO symbols? IEEE Std 1149.1-1990 contains the precedent. The data being shifted into the instruction register and applied to the "test logic" has a different meaning from the data captured by the instruction register and shifted out of the device. The former is an instruction, the latter is status information. For the instruction register, TDI symbols are instruction names and TDO symbols are status values. Other, design-specific test registers may exhibit similar properties.

When a table is defined, the symbols are assumed to be both TDI and TDO symbols. They can be further defined as TDI or TDO symbols by listing them in the SYMBOL_OF_TDI and SYMBOL_OF_TDO attributes.

Syntax

constant <*Symbol Table Name*> : SYMBOL_TABLE :=
  "<*Symbol Definitions*>";

<*Symbol Definitions*> is:
    <*Symbol Definition*>
    <*Symbol Definition*>, <*Symbol Definition*>
    <*Symbol Definition*>,...<*Symbol Definition*>

<*Symbol Definition*> is:
    <*Symbol Name*> (<*Symbol Values*>)

<*Symbol Values*> is:
    <*Symbol Value*>
    <*Symbol Value*>, <*Symbol Value*>
    <*Symbol Value*>,...<*Symbol Value*>

<*Symbol Value*> is:
    <*Pattern*>

Parameters

<Symbol Table Name>	A <VHDL identifier> defining the name of the symbol table. This name must be unique.
<Symbol Definitions>	A string listing the names of the symbols being defined and their associated values.
<Symbol Definition>	Each symbol definition defines a symbol name and associates values with it. A symbol name can only be defined once per symbol table, but

may appear with different values in another symbol table.

<Symbol Name> — A <VHDL identifier> defining the name of the symbol. The symbol table creates a new scope of names, so the symbol name must be unique within the symbol table but can be identical to the other symbols names or even the names of other elements of the entity. A well-known example is the BYPASS instruction, which is a symbol with the same name as a test register.

<Symbol Values> — A list of values associated with the symbol. None of the values in the symbol table as a whole may be duplicated or ambiguous. Ambiguous values are values containing don't-care bits such that two different values could represent the same binary pattern.

<Symbol Value> — A value associated with the symbol.

<Pattern> — The pattern is a binary string that defines the symbol value. X's are allowed in the pattern to represent don't-care bits.

Examples

```
constant BCR_Opcodes : SYMBOL_TABLE :=
   "SAMTOG  (00)," &
   "PRPG    (01)," &
   "PSA     (10)," &
   "PSAPRPG (11)";
```

See Also attribute BOUNDARY_SYMBOLS (HSDL devices), attribute BUS_SYMBOLS (HSDL), attribute INSTRUCTION_OPCODE (BSDL), attribute REGISTER_SYMBOLS (HSDL devices), attribute SYMBOL_DEFAULT (HSDL), attribute SYMBOL_DESCRIPTION (HSDL), attribute SYMBOL_OF_TDI (HSDL), attribute SYMBOL_OF_TDO (HSDL).

use *HSDL_device.all* (HSDL devices)

Description This "use" statement identifies the VHDL package needed to define the attributes of an HSDL device. This package defines several extensions to BSDL, making possible the use of the new HSDL device features. The package contents are primarily ignored by an HSDL translator, but are necessary in a VHDL environment.

Syntax

use HSDL_device.all;

Parameters

None.

Examples

```
use HSDL_device.all;
```

See Also use (BSDL), use HSDL_module.all (HSDL modules).

use *HSDL_module.all* (HSDL modules)

Description This "use" statement identifies the VHDL package needed to define the attributes of an HSDL module. This package defines several extensions to BSDL, making possible the definition of boards, boxes, multi-chip modules, subsystems, and systems. The package contents are primarily ignored by an HSDL translator, but are necessary in a VHDL environment.

Syntax

use HSDL_module.all;

Parameters

None.

Examples

```
use HSDL_module.all;
```

See Also use (BSDL), use HSDL_device.all (HSDL devices).

B.
HSDL Standard VHDL Package Definitions

These packages are used by HSDL to define the extra attributes and subtypes used in HSDL device and module entities. The package contents are not used by an HSDL translator, but are necessary so that an HSDL description can be used successfully in a VHDL environment.

Note that only packages are needed for HSDL; no package bodies are required.

Consideration was given to having a third package containing the attributes and subtypes common to both device and module HSDL files, but the number of HSDL packages needed per file would then have been two, reducing ease of use.

B.1. HSDL_device Standard VHDL Package

```
package HSDL_device is

-- Device Description Declarations
   attribute DEVICE_DESCRIPTION : string;

-- Port Description Declarations
   attribute PORT_DESCRIPTION : string;

-- Instruction Register Declarations
   attribute INSTRUCTION_NORMAL : string;
   attribute INSTRUCTION_TEST : string;
   attribute INSTRUCTION_DESCRIPTION : string;

-- Symbol Table Declarations
   subtype SYMBOL_TABLE is string;
   attribute SYMBOL_OF_TDI : string;
   attribute SYMBOL_OF_TDO : string;
   attribute SYMBOL_DEFAULT : string;
   attribute SYMBOL_DESCRIPTION : string;

-- Register Information Declarations
   attribute REGISTER_COMPOSITION : string;
   attribute REGISTER_CAPTURE : string;
   attribute REGISTER_RESET : string;
   attribute REGISTER_SYMBOLS : string;
   attribute REGISTER_DESCRIPTION : string;

-- Boundary Register Declarations
   attribute BOUNDARY_SYMBOLS : string;

-- Bus Declarations
   attribute BUS_COMPOSITION : string;
   attribute BUS_SYMBOLS : string;
   attribute BUS_DESCRIPTION : string;

-- Constraint Declarations
   attribute CONSTRAINTS : string;
   attribute CONSTRAINT_DESCRIPTION : string;

end HSDL_device ;
```

B.2. HSDL_module Standard VHDL Package

```
package HSDL_module is

-- Module Description Declarations
   attribute MODULE_DESCRIPTION : string;

-- Port Description Declarations
   attribute PORT_DESCRIPTION : string;

-- Member Declarations
   attribute MEMBERS : string;
   attribute MEMBER_DESCRIPTION : string;
```

```
   -- Symbol Table Declarations
      subtype SYMBOL_TABLE is string;
      attribute SYMBOL_OF_TDI : string;
      attribute SYMBOL_OF_TDO : string;
      attribute SYMBOL_DEFAULT : string;
      attribute SYMBOL_DESCRIPTION : string;

   -- Bus Declarations
      attribute BUS_COMPOSITION : string;
      attribute BUS_SYMBOLS : string;
      attribute BUS_DESCRIPTION : string;

   -- Path Declarations
      subtype EXTERNAL_PATH is string;
      subtype DYNAMIC_PATH is string;
      attribute DYNAMIC_PATH_RESET : string;
      subtype STATIC_PATH is string;

   -- Connection Declarations
      attribute CONNECTIONS : string;

   -- Constraint Declarations
      attribute CONSTRAINTS : string;
      attribute CONSTRAINT_DESCRIPTION : string;

   end HSDL_module;
```