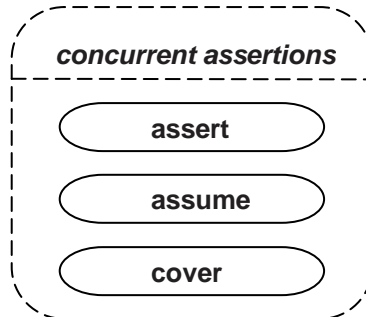


Purpose: The purpose is to fix issues with assertion VPI diagrams.

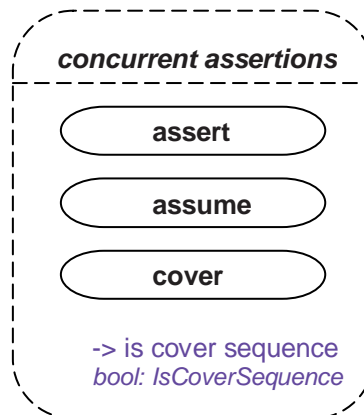
1. Clarify that `vpiIdentifier` iterator in the property and sequence declaration shall return the list of arguments in the declaration.
2. Add `vpiArgument` as an iterator to properties similar to sequences.
3. `vpiArgument` should be a `property_expr` for property instances and a `sequence_expr` for sequence instances (missed in Mantis 1730)
4. In several diagrams `vpiDefLineNo` is a “str”. This should be an “int”. It affect 36.44, 36.45, 36.46, 36.48.
5. There is overlap in the definition of `vpiDefLineNo` and `vpiDef`. Add a note that they are the same for property and sequence declarations.
6. 'block identifier' makes no sense for sequence and property declarations. These are not 'labeled statements'. In diagram 36.43 and 36.46, 'block identifier' should be replaced with 'name'.
7. `vpiArgument` should only come out of the sequence and property instance where they are defined (definitions are bold), so they should not be shown in the declaration diagrams.
8. It was clarified that you can only control verification statements, which then enables the instances within them. Clarifications were also stated w.r.t. interpretation of start times, pass, and fail of a sequence or property.
9. It was clarified what callbacks apply to sequence and property instances. The callbacks on the property and sequence instances are `cbAssertionStart`, `cbAssertionSuccess` and `cbAssertionFailure` only.
10. bool: `vpiIsCoverSequence` was added under `cover` on diagram 36.43 for distinguishing cover property and cover sequence (missed in Mantis 1768)
11. Added a note to the editor to make the Immediate assertion a section of its own instead of being included with the `sequence_expr` diagrams in 36.47.
12. Added `vpiClockedProp` to the header file and placed it under `vpiClockedSeq` . Also deleted `vpiActualArgExpr` that is defined in the header file, but is not an object in any of the diagrams so should be removed
13. Added `vpiClockedSeq` diagram to 36.48

REPLACE in 36.43 Concurrent Assertions (note to editor: only the part that changes is shown)



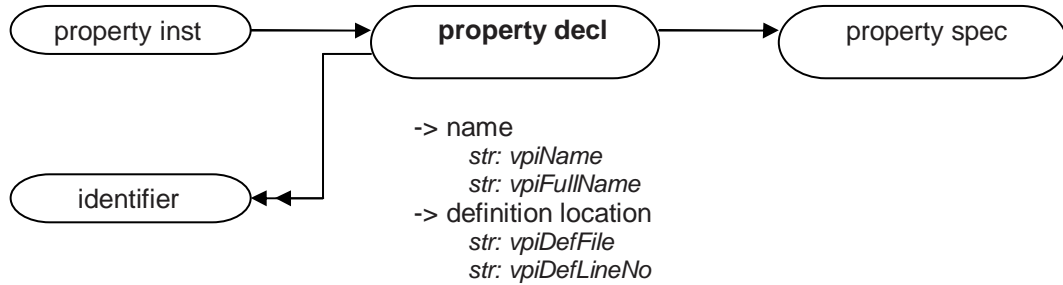
-> definition location
str: vpiDefFile
int: vpiDefLineNo
-> block identifier
str: vpiName
str: vpiFullName
-> is clock inferred
bool: vpilsClockInferred

WITH

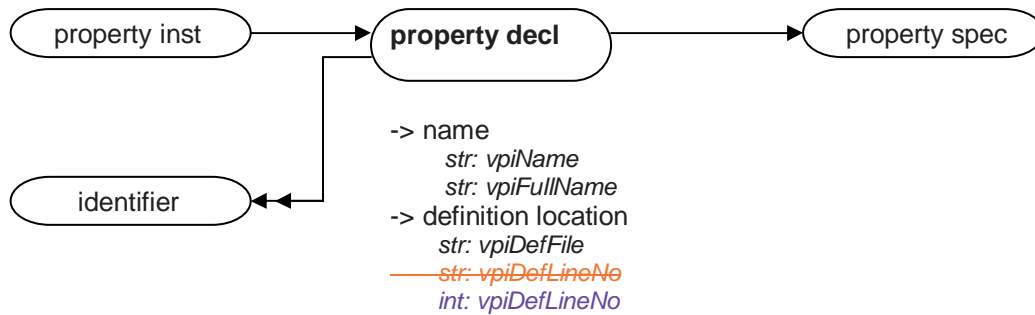


-> definition location
str: vpiDefFile
int: vpiDefLineNo
-> ~~block identifier~~-name
str: vpiName
str: vpiFullName
-> is clock inferred
bool: vpilsClockInferred

IN 36.44 Property Declaration REPLACE



WITH



Details:

- 1) The **vpiIdentifier** iterator shall return the property declaration arguments in the order that the formals for the property are declared.
- 2) **vpiLineNo** and **vpiFile** for a property declaration shall be the same as **vpiDefFile** and **vpiDefLineNo**.

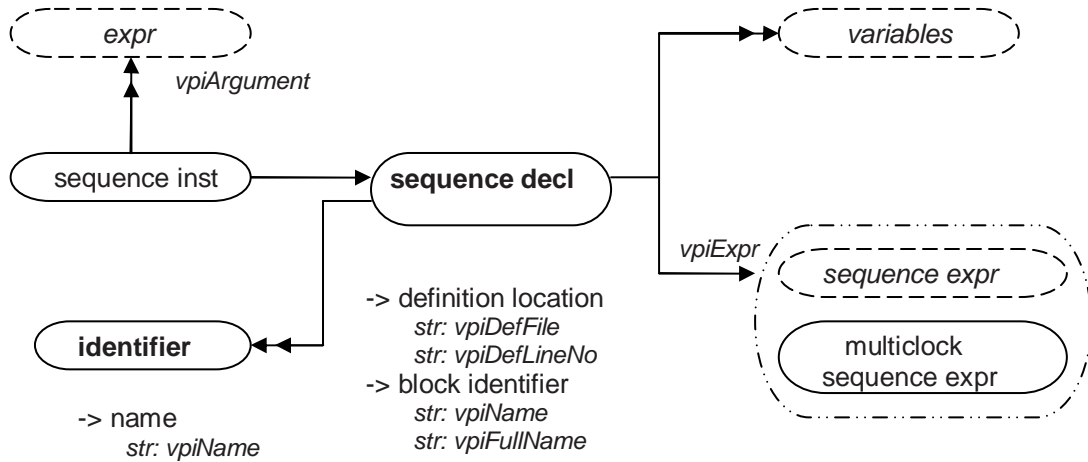
REPLACE in 36.45 Property specification

str: *vpiDefLineNo*

WITH

~~str: *vpiDefLineNo*~~
int: *vpiDefLineNo*

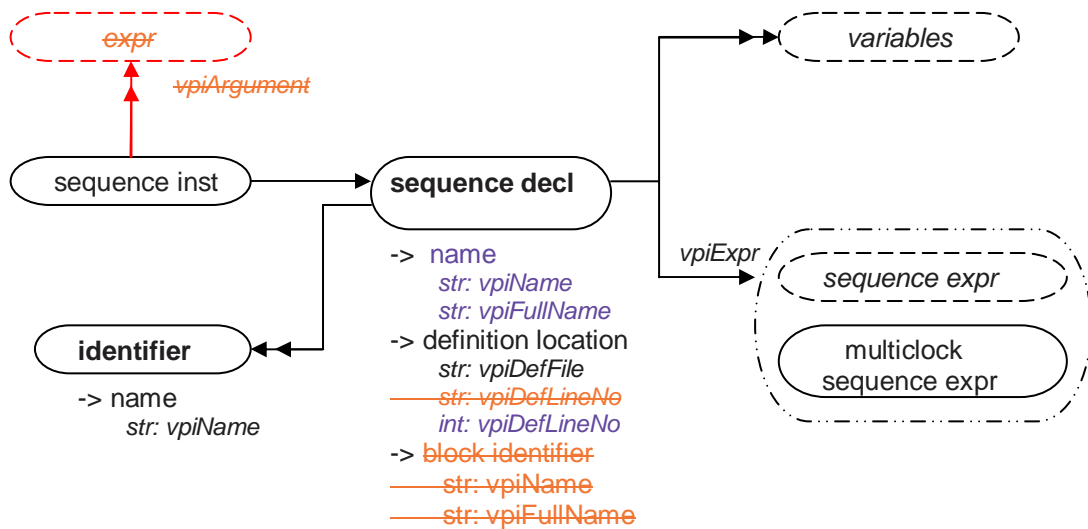
IN 36.46 Sequence declaration REPLACE



Details:

1) The **vpiArgument** iterator shall return the sequence instance arguments in the order that the formals for the sequence are declared, so that the correspondence between each argument and its respective formal can be made. If a formal has a default value, that value shall appear as the argument should the instantiation not provide a value for that argument.

WITH

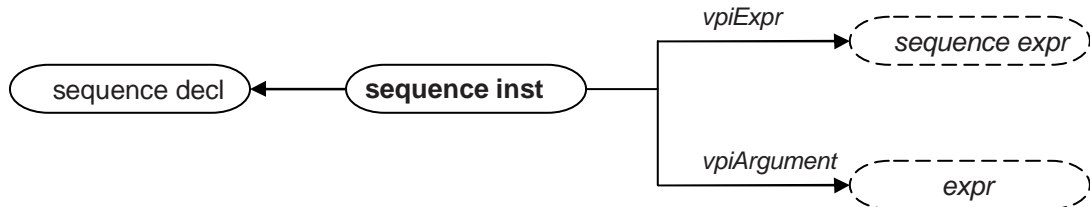


1) The **vpiArgument** iterator shall return the sequence instance arguments in the order that the formals for the sequence are declared, so that the correspondence between each argument and its respective formal can be made. If a formal has a default value, that value shall appear as the argument should the instantiation not provide a value for that argument.

1) The **vpiIdentifier** iterator shall return the sequence declaration arguments in the order that the formals for the sequence are declared.

2) **vpiLineNo** and **vpiFile** for a sequence declaration shall be the same as **vpiDefFile** and **vpiDefLineNo**.

REPLACE in 36.47 Sequence Expression (only affected parts of the diagram are shown)



WITH

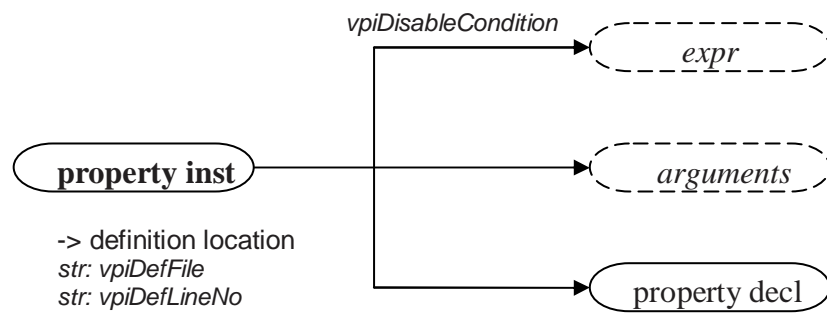


Details:

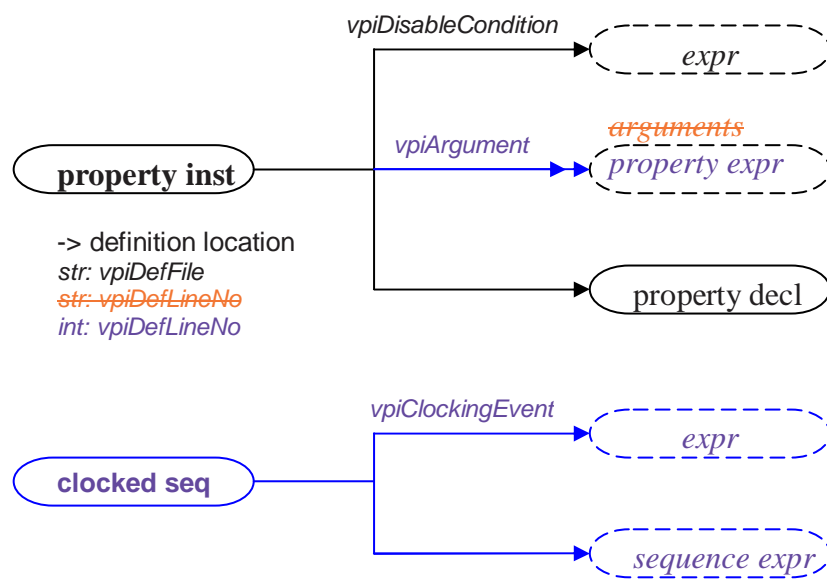
1) The **vpiArgument** iterator shall return the sequence instance arguments in the order that the formals for the sequence are declared, so that the correspondence between each argument and its respective formal can be made. If a formal has a default value, that value shall appear as the argument should the instantiation not provide a value for that argument.

Move from “36.47 Sequence expression” the immediate assert definition to a new section titled “Immediate Assertions” that follows 36.48. The notes in 36.47 still remain only in 36.47 – they do not get copied. Note that immediate assume and immediate cover from Mantis 1729 will also reside in this new section. Re-number subsequent sections accordingly.

REPLACE from 36.48 Multiclock sequence expression (only the parts that change are shown)



WITH



Details:

1) The **vpiArgument** iterator shall return the property instance arguments in the order that the formals for the property are declared, so that the correspondence between each argument and its respective formal can be made. If a formal has a default value, that value shall appear as the argument should the instantiation not provide a value for that argument.

REPLACE

38.3.2 Obtaining static assertion information

The following information about an assertion is considered to be static:

- Assertion name
- Instance in which the assertion occurs
- Module definition containing the assertion
- Assertion type
 - Sequence
 - Assert
 - Assume
 - Cover
 - Property
 - ImmediateAssert
- Assertion source information: the file, line, and column where the assertion is defined
- Assertion `clocking` block/expression

WITH

38.3.2 Obtaining static assertion information

The following information about an assertion is considered to be static:

- Assertion name
- Instance in which the assertion occurs
- Module definition containing the assertion
- Assertion type
 - Sequence [Instance](#)
 - Assert
 - Assume
 - Cover
 - Property [Instance](#)
 - ImmediateAssert
- Assertion source information: the file, line, and column where the assertion is defined
- Assertion `clocking` block/expression

In 38.4.2 REPLACE

where *reason* is any of the following.

- **cbAssertionStart**. An assertion attempt has started. For most assertions, one attempt starts each and every clock tick.
- **cbAssertionSuccess**. An assertion attempt reaches a success state.
- **cbAssertionFailure**. An assertion attempt fails to reach a success state.
- **cbAssertionStepSuccess**. Progress one step an attempt. By default, step callbacks are not enabled on any assertions; they are enabled on a per-assertion/per-attempt basis (see [38.5.2](#)), rather than on a per-assertion basis.
- **cbAssertionStepFailure**. Fail to progress by one step along an attempt. By default, step callbacks are not enabled on any assertions; they are enabled on a per-assertion/per-attempt basis (see [38.5.2](#)), rather than on a per-assertion basis.
- **cbAssertionDisable**. The assertion is disabled (e.g., as a result of a control action).
- **cbAssertionEnable**. The assertion is enabled.
- **cbAssertionReset**. The assertion is reset.
- **cbAssertionKill**. An attempt is killed (e.g., as a result of a control action).
- **cbAssertionDisablePassAction**. The pass action is disabled for vacuous and nonvacuous success for the assertion (e.g., as a result of control action).
- **cbAssertionEnablePassAction**. The pass action is enabled for vacuous and nonvacuous success for

the assertion (e.g., as a result of control action).

— **cbAssertionDisableFailAction**. The fail action is disabled for the assertion (e.g., as a result of control action).

— **cbAssertionDisableVacuousAction**. The pass action is disabled for vacuous success of the assertion (e.g., as a result of control action).

— **cbAssertionEnableNonvacuousAction**. The pass action is enabled for nonvacuous success of the assertion (e.g., as a result of control action).

These callbacks are specific to a given assertion; placing such a callback on one assertion does not cause the

callback to trigger on an event occurring on a different assertion. If the callback is successfully placed, a handle to the callback is returned. This handle can be used to remove the callback via `vpi_remove_cb()`. If there were errors on placing the callback, a `NULL` handle is returned. As with all other calls, invoking this function with invalid arguments has unpredictable effects.

WITH

— **cbAssertionStart**. An assertion attempt has started. For most assertions, one attempt starts each and every clock tick. For property and sequence instances the start is the start of evaluation of the property or sequence instance.

— **cbAssertionSuccess**. An assertion attempt reaches a success state. For property or sequence instances, success is a match.

— **cbAssertionFailure**. An assertion attempt fails to reach a success state. For property or sequence instances, failure is no match.

— **cbAssertionStepSuccess**. Progress one step along an attempt. A step is defined as progress along the flattened assertion (e.g. rewriting algorithm defined in [Note to Editor --- insert reference to the new section in F.3.1 titled “Rewriting sequence and property instances”]). By default, step callbacks are not enabled on any assertions; they are enabled on a per-assertion/per-attempt basis (see 38.5.2), rather than on a per-assertion basis. This does not apply to property or sequence instances.

— **cbAssertionStepFailure**. Fail to progress by one step along an attempt. A step is defined as progress along the flattened assertion (e.g. rewriting algorithm defined in [Note to Editor --- insert reference to the new section in F.3.1 titled “Rewriting sequence and property instances”]). By default, step callbacks are not enabled on any assertions; they are enabled on a per-assertion/per-attempt basis (see 38.5.2), rather than on a per-assertion basis. This does not apply to property or sequence instances.

— **cbAssertionDisable**. The assertion is disabled (e.g., as a result of a control action). This does not apply to property or sequence instances.

— **cbAssertionEnable**. The assertion is enabled. This does not apply to property or sequence instances.

— **cbAssertionReset**. The assertion is reset. This does not apply to property or sequence instances.

— **cbAssertionKill**. An attempt is killed (e.g., as a result of a control action). This does not apply to property or sequence instances.

— **cbAssertionDisablePassAction**. The pass action is disabled for vacuous and nonvacuous success for the assertion (e.g., as a result of control action). This does not apply to property or sequence instances.

— **cbAssertionEnablePassAction**. The pass action is enabled for vacuous and nonvacuous success for the assertion (e.g., as a result of control action). This does not apply to property or sequence instances.

— **cbAssertionDisableFailAction**. The fail action is disabled for the assertion (e.g., as a result of control action). This does not apply to property or sequence instances.

— **cbAssertionDisableVacuousAction**. The pass action is disabled for vacuous success of the assertion (e.g., as a result of control action). This does not apply to property or sequence instances.

— **cbAssertionEnableNonvacuousAction**. The pass action is enabled for nonvacuous success of the assertion (e.g., as a result of control action). This does not apply to property or sequence instances.

These callbacks are specific to a given assertion; placing such a callback on one assertion does not cause the callback to trigger on an event occurring on a different assertion. If the callback is successfully placed, a handle to the callback is returned. This handle can be used to remove the callback via `vpi_remove_cb()`. If there were errors on placing the callback, a `NULL` handle is returned. ~~As with all other calls, invoking this function with invalid arguments has unpredictable effects.~~

REPLACE

38.5.2 Assertion control

To obtain assertion control information, use **vpi_control()** with one of the operators in this subclause.

WITH:

38.5.2 Assertion control

To obtain assertion control information for verification statements (e.g. assume, assert, cover) , use **vpi_control()** with one of the operators in this subclause. Only verification statement handles are valid here, not sequence or property instances.

REPLACE On page 1126 of N.2 Source code (for sv_vpi_user.h)

```
...
#define vpiMethod 645
#define vpiIsClockInferred 649
...
```

WITH

```
...
#define vpiMethod 645
#define vpiIsClockInferred 649
#define vpiIsCoverSequence [Editor to fill in]
...
```

Also REPLACE On page 1124 of Annex N.2 Source code for object types(for sv_vpi_user.h)

```
...
#define vpiMulticlockSequenceExpr 658
#define vpiClockedSeq 659
#define vpiPropertyInst 660
#define vpiSequenceDecl 661
#define vpiActualArgExpr 663
#define vpiSequenceInst 664
...
```

WITH

```
...
#define vpiMulticlockSequenceExpr 658
#define vpiClockedSeq 659
#define vpiClockedProp [Editor to fill in]
#define vpiPropertyInst 660
#define vpiSequenceDecl 661
#define vpiActualArgExpr 663
#define vpiSequenceInst 664
```