0pt0.6pt

Patterns Module User's Manual

Roselyne Chotin-Avot

1 Description

The patterns module of *Stratus* is a set of *Python* classes and methods that allows a procedural description of input pattern file for the logic simulator. The *Stratus* `Pattern` method produces a pattern description file as output. The file generated by `Pattern` method is in pat format, so IT IS STRONGLY RECOMMENDED TO SEE pat(5) manual BEFORE TO USE IT.

2 Syntax

From a user point of view, `Pattern` method is a pattern description language using all standard *Python* facilities. Here follows the description of the `Pattern` method.
A pat format file can be divided in two parts : declaration and description part.
The declaration part is the list of inputs, outputs, internal signals and registers. Inputs are to be forced to a certain value and all the others are to be observed during simulation.
The description part is a set of patterns, where each pattern defines the value of inputs and outputs. The pattern number represents actually the absolute time for the simulator.
Similarly, a `Pattern` method can be divided in two parts : declaration and description part. Methods related to the declaration must be called before any function related to the description part.

2.1 Declaration part

The first thing you should do in this part is to instantiate the class `Patwrite` to have access to all patterns declaration and description methods. The constructor of this class take as parameters the name of pattern output file and the *Stratus* cell that is described (see `PatWrite` 3.1).
Then, this part allows you to declare the inputs, the outputs, and internal observing points (see `declar`3.4 and `declar_interface` 3.3).

2.2 Description part

After all signals are declared, you can begin the description part (see `pattern_begin` 3.9). In this part you have to define input values which are to be applied to the inputs of the circuit or output values which are to be compare with the values produced during the simulation. (see `affect` **??**, `affect_any` 3.7, `affect_int` 3.5 and `affect_fix` 3.6). `Pattern` method describes the stimulus by event : only signal transitions are described. After each event there is a new input in the pattern file (see `addpat` 3.8). Last thing you should do in this part is to generate the output file (see `pattern_end` 3.10).

3 Methods

3.1 PatWrite

This class is used to create patterns for *Stratus* models. Currently it only supports Alliance ".pat" pattern format. Patterns time stamps are in the "absolute date" format, "relative date" isn't allowed. Legal time unit are ps (default), ns, us and ms. The constructor takes as parameters the pattern output filename and an optional reference to Stratus cell.

3.2 declar

Adds a connector from a Stratus model to the pattern interface. Writes the corresponding connector declaration in the pattern file with name, arity and direction automatically extracted from the connector properties.
Supported Stratus connectors are:

- SignalIn,
- SignalOut (only supported if used as an output),
- VddIn,
- VssIn,
- CkIn,
- SignalInOut,
- TriState (always an output),
- Signals.

3.2.1 Parameters

- connector : can either be a reference to a stratus net or a string containing the name of the stratus net.
- format : optional format for the connectors values into the pattern file, accepted values are :
  - 'B': binary (default),
  - 'X': hexadecimal,
  - 'O': octal.

3.3 declar_interface

Adds all the connectors from a Stratus model to the pattern interface. Write the corresponding connector declaration in the pattern file with name, arity and direction directly taken from the connector proprieties.

3.3.1 Parameters

- cell : the tested Stratus model reference. Optional if a reference to the tested Stratus model was given during instanciation3.1.
- format : optional format for the connectors values into the pattern file, accepted values are :
  - 'B': binary (default),
  - 'X': hexadecimal,
  - 'O': octal.

3.4 declar

Affect a string value to a connector.

3.4.1 Parameters

- connector : *Stratus* connector
- value : string to affect to connector

3.5 affect_int

Affect an integer (CA2) value to a connector. Convert the 2's complement value to the corresponding binary value. The binary size is taken from the connector arity. If the connector is an output, the binary value is preceded by "?".

3.5.1 Parameters

- connector : *Stratus* connector.
- value : 2's complement value to affect to the connector.

3.6 affect_fix

Affect a fixed point value to a connector. Convert the floating point input value to the corresponding fixed point value with word_length=connector.arity() and integer_word_length=iwl. If the connector is an output, the binary value is preceded by "?".

3.6.1 Parameters

- connector : *Stratus* connector.
- value : floating point value to convert and asign to connector.
- iwl : integer word length

3.7 affect_any

Disable comparison between this connector value and the one calculated during simulation.

3.7.1 Parameters

- connector : *Stratus* connector.

3.8 addpat

Adds a pattern in the pattern file.

3.9 pattern_begin

Mark the end of the interface declaration and the beginning of the test vectors.

3.10 pattern_end

Mark the end of the test vectors and of the patterns file.

4 Example

`Pattern` method for an addaccu

```
def Pattern(self):
    # initialisation
    pat = PatWrite(self._name+'.pat',self)

    # declaration of ports
    pat.declar(self.ck, 'B')
    pat.declar(self.load, 'B')
    pat.declar(self.input, 'X')
    pat.declar(self.output, 'X')
    pat.declar(self.vdd, 'B')
    pat.declar(self.vss, 'B')

    # use of pat.declar_interface(self) has the same effect

    # description beginning
    pat.pattern_begin()

    # affect vdd and vss values
    pat.affect_int(self.vdd,1)
    pat.affect_int(self.vss,0)

    # first pattern : load an initial value
    pat.affect_int(self.input,5)
    pat.affect_int(self.load,1)
    pat.affect_int(self.ck,0)
    # add the pattern in the pattern file
    pat.addpat()
    # compute next event
    pat.affect_int(self.ck,1)
    pat.addpat()

    # compute 22 cycle of accumulation
    pat.affect_int(self.load,0)
```

```
for i in range(1,22):
   pat.affect_int(self.ck,0)
   pat.addpat()
   pat.affect_int(self.ck,1)
   pat.affect_int(self.output,i+5)
   pat.addpat()

# end of the description
pat.pattern_end()
```