Python Streams User Manual

Jonathan P Dawson

December 19, 2010

Contents

1 The Concepts **3**

2 The Language **4**

2.1 Systems . . . 4
2.2 Streams . . . 4
2.2.1 Counter(Start, Stop, Step) . . . 4
2.2.2 InPort(name, bits) . . . 4
2.2.3 Output() . . . 5
2.2.4 Repeater(value) . . . 5
2.2.5 Scanner() . . . 5
2.2.6 Sequence(seqeunce) . . . 5
2.2.7 SerialIn . . . 6
2.2.8 Stimulus . . . 6
2.3 Stream Combinators . . . 6
2.3.1 Array . . . 6
2.3.2 Decoupler . . . 6
2.3.3 HexPrinter . . . 6
2.3.4 Lookup . . . 6
2.3.5 Printer . . . 6
2.3.6 Process . . . 6
2.3.7 Resizer . . . 6
2.4 Stream Expressions . . . 6
2.4.1 operator precedence . . . 6
2.5 Sinks . . . 8
2.5.1 Asserter . . . 8
2.5.2 Console . . . 8
2.5.3 OutPort . . . 8
2.5.4 Response . . . 8
2.5.5 SerialOut . . . 8
2.5.6 SVGA . . . 8
2.6 Processes . . . 8
2.6.1 Output . . . 8
2.6.2 Variable . . . 8
2.6.3 VariableArray . . . 8
2.7 Process Statements . . . 8
2.7.1 Block . . . 8
2.7.2 Break . . . 8
2.7.3 Constant . . . 8

2.7.4 Continue 8
2.7.5 DoUntil 8
2.7.6 DoWhile 8
2.7.7 Evaluate 8
2.7.8 If 8
2.7.9 Loop 8
2.7.10 Print 8
2.7.11 Scan 8
2.7.12 Until 8
2.7.13 Value 8
2.7.14 Variable 8
2.7.15 WaitUs 8
2.7.16 While 8
2.7.17 Expressions 8
2.8 Process Expressions 8
2.9 Patterns of Use 8

Chapter 1

The Concepts

Chapter 2

The Language

2.1 Systems

2.2 Streams

2.2.1 Counter(Start, Stop, Step)

This versatile Stream emits numbers in the sequence $[start, start + step, start + 2 * step, ..., stop, start, ...]$.

```
Example
Counter(1, 3, 1)
=> [1, 2, 3, 1, 2, 3, 1, 2, 3, ...,]
Counter(3, 1, -1)
=> [3, 2, 1, 3, 2, 1 ...,]
```

2.2.2 InPort(name, bits)

The `InPort` Stream emits numbers aquired from a physical io port in a target device.

`name` is a string parameter specifying a name to identify the port in the target implementation.

Since it is not possible to automatically determine the number of bits needed, the width of the port must be explicitly set using the `bits` parameter.

No handshaking is implemented, the value emited by an `InPort` Stream is the value present on the io port at the time the stream transaction completes. The `InPort` Stream will automatically generate logic to synchronise the io port to the local clock domain.

Example:

```
Example
1  dip_switch = InPort("sw0", 8)
2  minimal_system = System(OutPort("led0", dip_switch))
```

2.2.3 Output()

While inputs to a process are created implicitly by reading from a stream, process outputs must be explicitly created by creating an `Output`.

```
Example
1  output1 = Output()
2  output2 = Output()
3  data = Variable(0)
4  Process(bits,
5    Loop(
6      stream.read(data),
7      stream.write(output1),
8      stream.write(output2),
9    )
10 )
```

2.2.4 Repeater(value)

The `Repeater` Stream emits `value` repeatedly.

```
Example
a = Repeater(1) + Repeater(1)
=> [2, 2, 2, 2, 2, 2, 2 ...]
```

2.2.5 Scanner()

2.2.6 Sequence(seqeunce)

The `Sequence` Stream emits each item in `sequence` in turn. When the end of the sequence is reached, the whole process repeates starting again from the first item.

```
Example
Sequence(0, 1, 1, 3, 4)
=> [0, 1, 1, 3, 4, 0, 1, 1 ...]
```

2.2.7 SerialIn

2.2.8 Stimulus

2.3 Stream Combinators

2.3.1 Array

2.3.2 Decoupler

2.3.3 HexPrinter

2.3.4 Lookup

2.3.5 Printer

2.3.6 Process

2.3.7 Resizer

2.4 Stream Expressions

A Stream Expression can be formed by combining Streams or Stream Expressions with the following ooperators:

`+, -, *, \/, %, &, |, ^, <<, >>, ==, !=, <, <=, >, >=`

Each data item in the resulting Stream Expression will be evaluated by removing a data item from each of the operand streams, and applying the operator function to these data items.

Generaly speaking a Stream Expression will have enough bits to contain any possible result without any arithmetic overflow. The one exception to this is the left shift operator where the result is always truncated to the size of the left hand operand. Stream expressions may be explicitly truncated or sign extended using the `Resizer` (section 2.3.6).

If one of the operands of a binary operator is not a Stream, Python Streams will attempt to convert this operand into an integer. If the conversion is successfull, a `Repeater` (section 2.2.3) stream will be created using the integer value. The repeater stream will be used in place of the non-stream operand. This allows constructs such as `a = 47+InPort(12, 8)` to be used as a shorthand for `a = Repeater(47)+InPort(12, 8)` or `count = Counter(1, 10, 1)+3*2` to be used as a shorthand for `count = Counter(1, 10, 1)+Repeater(5)`. Of course `a=1+1` still yields the integer 2 rather than a stream.

The operators provided in the Python Streams library are summarised in the table below. The bit width field specifies how many bits are used for the result based on the number of bits in the left and right hand operands.

2.4.1 operator precedence

The operator precedence is inherited from the python language. The following table summarizes the operator precedences, from lowest precedence (least binding) to highest precedence (most binding). Operators in the same row have the same precedence.

Operator	Function	Bit Width
`+`	Signed Add	$max(left, right) + 1$
`-`	Signed Subtract	$max(left, right) + 1$
`*`	Signed Multiply	$left + right$
`//`	Signed Floor Division	$max(left, right) + 1$
`%`	Signed Modulo	$max(left, right)$
`&`	Bitwise AND	$max(left, right)$
`|`	Bitwise OR	$max(left, right)$
`^`	Bitwise XOR	$max(left, right)$
`<<`	Arithmetic Left Shift	$left$
`>>`	Arithmetic Right Shift	$left$
`==`	Equality Comparison	1
`!=`	Inequality Comparison	1
`<`	Signed Less Than Comparison	1
`<=`	Signed Less Than or Equal Comparison	1
`>`	Signed Greater Than Comparison	1
`>=`	Signed Greater Than or Equal Comparison	1

Table 2.1: Operator summary

Operator	Description
`==, !=, <, <=, >, >=`	Comparisons
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<, >>`	Shifts
`+, -`	Addition and subtraction
`*, //, %`	multiplication, division and modulo

Table 2.2: Operator precedence

2.5 Sinks

2.5.1 Asserter

2.5.2 Console

2.5.3 OutPort

2.5.4 Response

2.5.5 SerialOut

2.5.6 SVGA

2.6 Processes

2.6.1 Output

2.6.2 Variable

2.6.3 VariableArray

2.7 Process Statements

2.7.1 Block

2.7.2 Break

2.7.3 Constant

2.7.4 Continue

2.7.5 DoUntil

2.7.6 DoWhile

2.7.7 Evaluate

2.7.8 If

2.7.9 Loop

2.7.10 Print

2.7.11 Scan

2.7.12 Until

2.7.13 Value

2.7.14 Variable

2.7.15 WaitUs

2.7.16 While

2.7.17 Expressions

2.8 Process Expressions

2.9 Patterns of Use