PyVSC

SystemVerilog-Style Constraints and Coverage in Python

Matthew Ballance
matt.ballance@gmail.com

Agenda

- Introduction
- Constraining and randomizing data
- Collecting coverage
- PyVSC Environment Integration
- Future Work

Intro

- Randomization and functional coverage collection central to functional verification
  - Often incorporated in a verification language, such as SystemVerilog
  - Very useful capabilities outside simulation as well

- PyVSC is a Python package that provides randomization and functional coverage
  - **Py**thon **V**erification **S**timulus and **C**overage

- Key Goals:
  - SystemVerilog constraints and coverage feature set
    - Similar look and feel to provide familiarity for existing SystemVerilog users
  - Performance and capacity on-par with SystemVerilog environments
  - Ability to record coverage data for later use

Agenda

- Introduction
- Constraining and randomizing data
- Collecting coverage
- PyVSC Environment Integration
- Future Work

Capturing Randomizable Data

- Randomizable classes are decorated with the *randobj* decorator
  - No need to change the inheritance hierarchy to add randomization to an existing class
- PyVSC-specific data types for constrainable fields
  - Signed and unsigned scalar fields
  - Enum-type fields
  - Class-type fields
  - Arrays/lists
- Randomizable classes support randomization hooks
  - pre_randomize() method called before randomization
  - post_randomize() method called after

```
@vsc.randobj
class my_s(object):

    def __init__(self):
        self.a = vsc.rand_bit_t(8)
        self.b = vsc.rand_uint8_t()

    @vsc.constraint
    def ab_c(self):
        self.a < self.b
```

Capturing Class Constraints

- Class constraints are captured as statements in *constraint*-decorated methods
- Constraints methods are declarative
  - Evaluated once to build a constraint model
  - Beware use of procedural constructs

```
@vsc.constraint
def ab_c(self):
  with vsc.if_then(self.a == 1):
    self.b == 1
  with vsc.else_if(self.a == 2):
    self.b == 2
  with vsc.else_if(self.a == 3):
    self.b == 4
  with vsc.else_if(self.a == 4):
    self.b == 8
  with vsc.else_if(self.a == 5):
    self.b == 16
```

Constraints are *virtual*

- Just as in SystemVerilog, constraints can be overridden by inheritance
- Example:
  - Class *my_base_s* declares a constraint *ab_c*
  - Class *my_ext_s* inherits and declares constraint *ab_c*
  - Instances of *my_base_s* have a < b
  - Instances of *my_ext_s* have b > a

```
@vsc.randobj
class my_base_s(object):
  def __init__(self):
    self.a = vsc.rand_bit_t(8)
    self.b = vsc.rand_bit_t(8)

  @vsc.constraint
  def ab_c(self):
    self.a < self.b

@vsc.randobj
class my_ext_s(my_base_s):
  def __init__(self):
    super().__init__()
  @vsc.constraint
  def ab_c(self):
    self.a > self.b
```

Randomizing

- PyVSC provides two class methods for randomizing data
  - randomize() – just use the class constraints
  - randomize_with() – take inline constraints too
- Example:
  - Randomize in a loop
  - Add an additional constraint using the iteration variable

```
@vsc.randobj
class my_base_s(object):
  def __init__(self):
    self.a = vsc.rand_bit_t(8)
    self.b = vsc.rand_bit_t(8)
  @vsc.constraint
  def ab_c(self):
    self.a < self.b

item = my_base_s()
for i in range(10):
  with item.randomize_with() as it:
    it.a == i
```

Random Data and Constraint Features

- PyVSC provides good coverage of SystemVerilog constraint features
- Simplifies adoption by SV-knowledgeable users
- Where to go from here?
  - SystemVerilog takes a static view of descriptions
  - Python takes a very dynamic view
  - Build on SystemVerilog base, explore possibilities
    - Build constraints dynamically based on function args
    - Create new constraints during simulation

Feature	SystemVerilog	PyVSC
Algebraic constraints	Y	Y
Integer fields	Y	Y
Enum fields	Y	Y
Fixed-size arrays	Y	Y
Variable-size arrays	Y	Y
dist constraint	Y	Y
soft constraint	Y	Y
inside constraint	Y	Y
solve ordering	Y	Y
unique constraint	Y	Y
foreach constraint	Y	Y
constraint_mode	Y	Y
rand_mode	Y	Y
if/else constraint	Y	Y

Agenda

- Introduction
- Constraining and randomizing data
- Collecting coverage
- PyVSC Environment Integration
- Future Work

Declaring a Covergroup

- A PyVSC Covergroup is a class decorated with *vsc.covergroup*
- The init method specifies
  - How data will be sampled
  - What coverpoints/crosses compose the covergroup
- Calling the sample() method samples data

```
@vsc.covergroup
class my_covergroup(object):
  def __init__(self):
    self.with_sample(
      a=vsc.bit_t(4)
    )
    self.cp1 = vsc.coverpoint(
      self.a, bins={
        "a" : vsc.bin(1, 2, 4),
        "b" : vsc.bin(8, [12,15])
    })

cg_i = my_covergroup()
cg_i.sample(1)
cg_i.sample(2)
```

Passing Coverage Data via the Sample Function

- Define the parameters accepted by the sample function
  - Parameters must be a PyVSC-defined type
  - These parameters are defined as class fields
- When *sample* is called, the parameters are sampled

```
@vsc.covergroup
class my_covergroup(object):
  def __init__(self):
    self.with_sample(
      a=vsc.bit_t(4)
    )
    self.cp1 = vsc.coverpoint(
      self.a, bins={
        "a" : vsc.bin(1, 2, 4),
        "b" : vsc.bin(8, [12,15])
    })

cg_i = my_covergroup()
cg_i.sample(1)
cg_i.sample(2)
```

Binding Coverpoints to Sampling Data

- Coverpoints can be defined on Python *lambda* expressions
- This simplifies sampling data from different objects
  - No need to pass the entire object to the sample function
  - No need to build the object with PyVSC types
- Very helpful in working with arbitrary data

```
class my_obj(object):
  def __init__(self, v):
    self.a = v

@vsc.covergroup
class my_covergroup(object):
  def __init__(self):
    self.obj = None
    self.cp1 = vsc.coverpoint(
      lambda: self.obj.a, bins={
        "a" : vsc.bin(1, 2, 4),
    })
o1 = my_obj(1)
o2 = my_obj(2)
cg_i = my_covergroup()
cg_i.obj = o1
cg_i.sample()
cg_i.obj = o2
cg_i.sample()
```

Coverage Features

- PyVSC currently supports a subset of SystemVerilog constructs
- Future work will complete
- Request your favorite feature!

Feature	SystemVerilog	PyVSC
covergroup type	Y	Y
coverpoint bins	Y	Y
coverpoint ignore_bins	Y	N
coverpoint illegal_bins	Y	N
coverpoint single bin	Y	Y
coverpoint array bin	Y	Y
coverpoint auto bins	Y	Y
coverpoint transition bin	Y	N
cross auto bins	Y	Y
cross bin expressions	Y	N
cross explicit bins	Y	N
cross ignore_bins	Y	N
cross illegal_bins	Y	N

Agenda

- Introduction
- Constraining and randomizing data
- Collecting coverage
- PyVSC Environment Integration
- Future Work

PyVSC Integration Points

- PyVSC is a Python library, and can be used in any environment
  - Pros: very flexible
  - Cons: user is responsible for handling integration
- Two integration points:
  - Random seed management
  - Saving coverage data

Random Seed Management

- PyVSC uses the Python *random* package
- Setting the *random*-package seed controls PyVSC random results
- Can directly specify

```
import random
…
random.seed(0)
```

- Environments, such as cocotb, automatically set *random*-package seed
  - From environment variables
  - From simulator command-line options
  - …

Coverage – Text Report

- PyVSC provides two methods for working with simple coverage reports
- *report_coverage* prints a simple text report or saves to a file

```
import vsc
…
vsc.report_coverage(details=True)
```

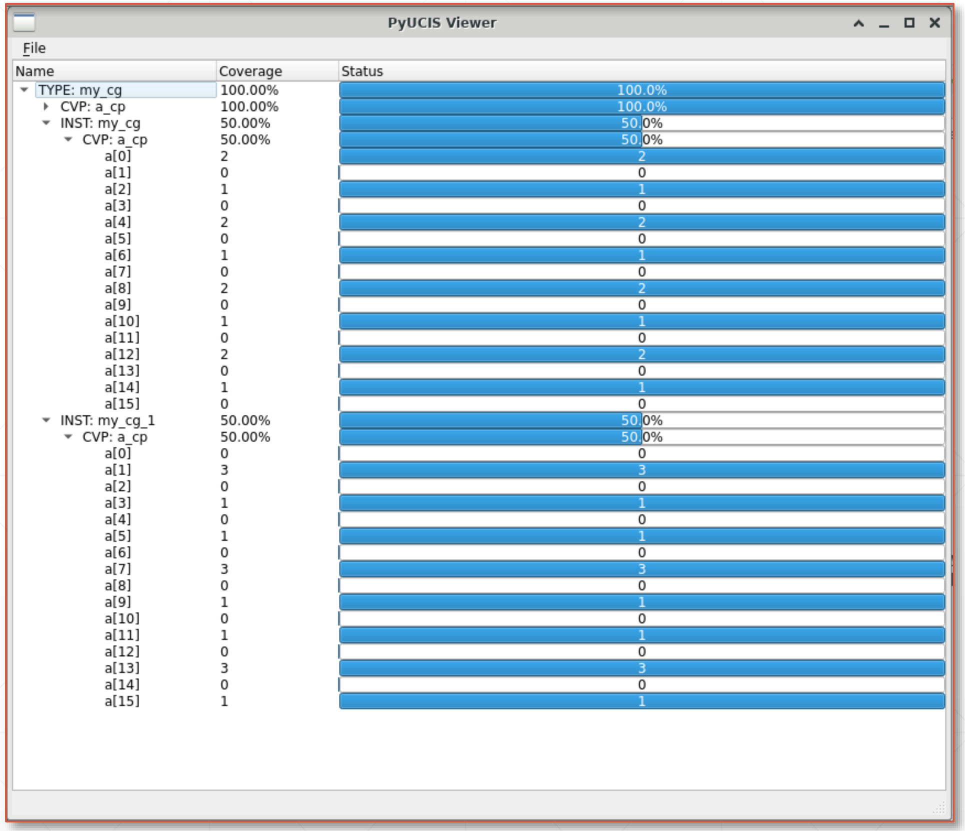
```
TYPE my_cg : 100.000000%
    CVP a_cp : 100.000000%
    INST my_cg : 100.000000%
        CVP a_cp : 100.000000%
    INST my_cg_1 : 0.000000%
        CVP a_cp : 0.000000%
```

- *get_coverage_report_model* returns an Python-object hierarchy
  - Enables creation of custom reports

Coverage – UCIS XML Report

- *write_coverage_db* saves as a UCIS XML file
  - Accellera standard for coverage interchange

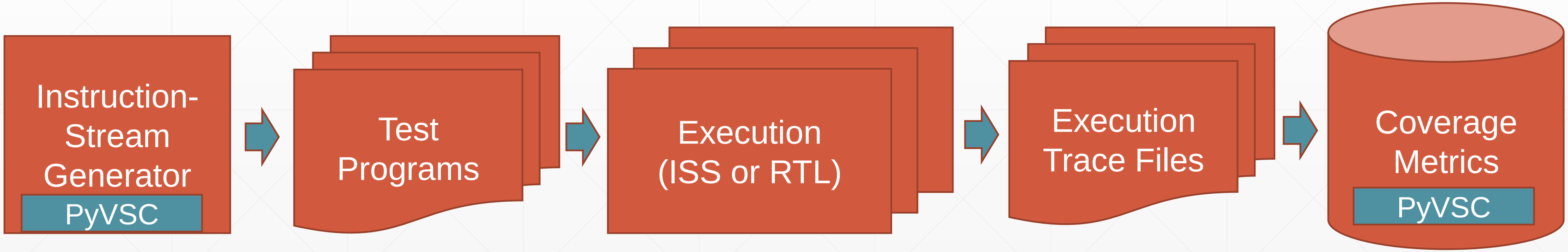
```
import vsc
…
vsc.write_coverage_db("cov.xml")
```



- UCIS XML files can be viewed graphically
  - https://github.com/fvutils/pyucis-viewer

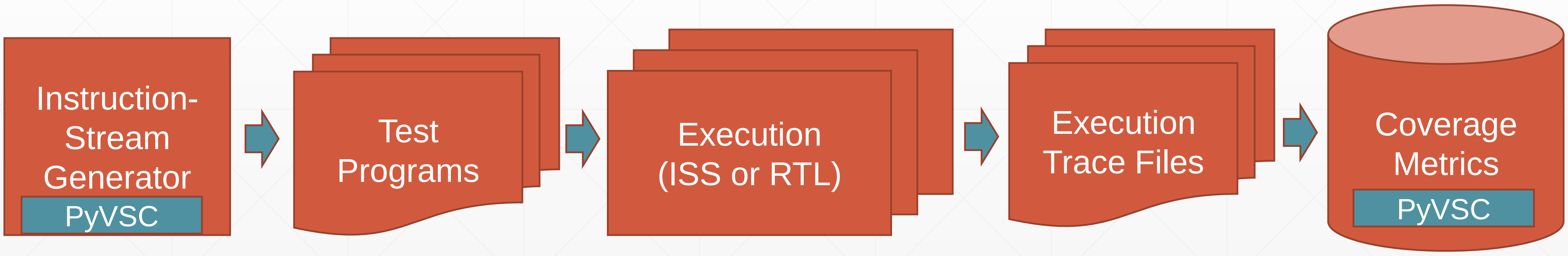
Example Use of PyVSC: Google riscv-dv

- Google riscv-dv project generates test programs for RISC-V cores
  - Uses constrained-random generation to generate programs
  - Defines coverage metrics on executed instruction scenarios
  - Provides a SystemVerilog model that uses a simulator for generation and coverage collection
  - Now provides a Python model that uses PyVSC for randomization and coverage collection
- https://github.com/google/riscv-dv



Example Use of PyVSC: Google riscv-dv

- Metrics
  - ~8k LoC for the Python-based instruction-stream generator
  - ~50 covergroups
  - ~300 coverpoints and crosses



Agenda

- Introduction
- Constraining and randomizing data
- Collecting coverage
- PyVSC Environment Integration
- Future Work

Future Work

- Complete SystemVerilog constructs
  - Missing constructs primarily in the functional coverage area
- Beyond SystemVerilog
  - Python provides opportunity to do things not possible in SystemVerilog
  - Programmatically / dynamically created constraints and coverage
  - Features for providing increased stimulus-steering
- Performance
  - Python excels as an integration language, but it not a high-speed implementation language
  - Planning to re-implement core model in C++ for increased performance/capacity
- Specific environment integrations
  - Eg cocotb integration to automate coverage-database save on exit

Getting Started with PyVSC

- Documentation: https://pyvsc.readthedocs.io
- Pre-built package on PyPi: *pip install pyvsc*
  - Linux, Mac OS X
  - Currently, no Windows support
- GitHub: https://github.com/fvutils/pyvsc
  - Source code
  - Issue trackers