

Functional Coverage For SystemC (FC4SC)

Dragoş Dospinescu,
Teodor Vasilache

Contact: contributors@amiq.com

Presentation Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **AMIQ Consulting** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

Agenda

1. What is FC4SC
2. FC4SC features overview
3. Coverage constructs
4. Coverage options, control & DB interrogation
5. Coverage database management
6. Roadmap
7. Q&A Session

What is FC4SC (1)

- C++11 header only library providing functional coverage capabilities
- No dependency on any 3rd party library
- Based on *IEEE 1800 - 2012 SystemVerilog Standard*

What is FC4SC (2)

1. Download library: <https://github.com/amiq-consulting/fc4sc>

2. Include it in your project:

```
#include "fc4sc.hpp"
```

Done!

FC4SC features overview

1. Coverage constructs: bin, coverpoint, cross, covergroup
 2. Coverage options: weight, goal, at_least
 3. Coverage control: start(), stop()
 4. Runtime database interrogation
 5. Database saving
 6. Database management tools
- UCIS DB format

Constructs: bin

Bin: collection of values and intervals

FC4SC

```
bin<int>(
    "less_than_8", // bin name
    1,             // 1
    interval(2,3), // [2:3]
    interval(7,4) // [4:7]
);
bin_array<int>("split",
    3,           // 3 bins
    interval(0,255) // [0:255]
);
illegal_bin<int>("illegal", 10);
ignore_bin<int>("ignore", 100);
```

SystemVerilog

```
bins less_than_8 = {
    1,
    [2:3],
    [4:7]
};
bins split[3] = {
    [0:255]
};
illegal_bins illegal = {10};
ignore_bins ignore = {100};
```

Constructs: coverpoint (1)

- Contains bins with data of interest
- Handles sampling
- illegal bin -> ignore bin -> regular bin

FC4SC

```
COVERPOINT(int, datacp, data)
{
  bin<int>("pos", interval( 0, 10)),
  bin<int>("neg", interval(-10, 0)),
  illegal_bin<int>("zero", 0)
};
```

SystemVerilog

```
datacp : coverpoint data
{
  bins pos = {[0:10]};
  bins neg = {[-10:0]};
  illegal_bins zero = {0};
}
```


Constructs: coverpoint (2)

FC4SC

SystemVerilog

Sample expression

```
COVERPOINT (int, cp, data*2, flag!=0)
{
    // ...
};
```

```
coverpoint (data*2) iff (flag!=0)
{
    // ...
}
```

Sample condition

Both are evaluated at sample point!

Constructs: cross

- Cartesian product of coverpoints' bins
- Behaves the same as a coverpoint in all regards

FC4SC

```
COVERPOINT(int, cp1, d1) {  
    bin<int>("zero", 0),  
    bin<int>("positive", 1, 2, 3)  
};  
COVERPOINT(int, cp2, d2) {  
    bin<int>("zero", 0),  
    bin<int>("negative", -1, -2, -3)  
};  
auto cp1_x_cp2 = cross<int, int>(  
    "cp1_x_cp2", &cp1, &cp2);
```

SystemVerilog

```
cp1 : coverpoint d1 {  
    bins zero = {0};  
    bins positive = {1,2,3};  
}  
cp2 : coverpoint d2 {  
    bins zero = {0};  
    bins negative = {-1,-2,-3};  
}  
cvp1_x_cvp2 : cross cvp1, cvp2;
```

Constructs: covergroup

- Ties together all coverage constructs
- Dispatches sampling data to coverpoints and crosses

FC4SC

```
class cvg_ex: public covergroup {  
public:  
    int d1;  
    COVERPOINT(int, cp1, d1) {  
        bin<int>("zero", 0),  
        bin<int>("positive", 1, 2, 3)  
    };  
    CG_CONS(cvg) { /*constructor*/ }  
};
```

SystemVerilog

```
covergroup cvg_ex {  
    cp1 : coverpoint d1 {  
        bins zero = {0};  
        bins positive = {1,2,3};  
    }  
}
```

Options, Control & DB interrogation (1)

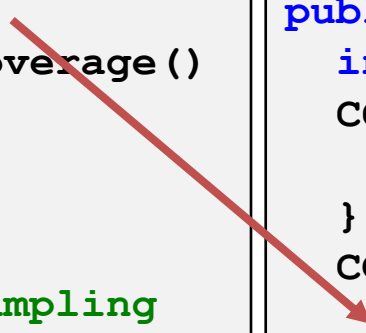
- Options are used for:
 - adjusting coverage distributions
 - setting coverage goals
 - adding comments
- Control: *starting* and *stopping* sampling
- DB interrogation (at runtime):
 - getting coverage percentage (type/instance)
 - getting number of hits
- Usable on: covergroup, coverpoint, cross

Options, Control & DB interrogation (2)

```
cvg_ex cvg; // create
cvg.d1 = 0; // update d1
cvg.sample(); // sample
EXPECT_EQ(cvg.get_inst_coverage()
, 50); // 50% covered

cvg.stop(); // stop sampling
cvg.d1 = 2;
cvg.sample();
EXPECT_EQ(cvg.get_inst_coverage()
, 50); // still 50%
```

```
class cvg_ex: public covergroup
{
public:
    int d1;
    CG_CONS(cvg, int w = 100) {
        this->option.weight = w;
    }
    COVERPOINT(int, cp1, d1) {
        bin<int>("zero", 0),
        bin<int>("positive", 1, 2, 3)
    };
};
```

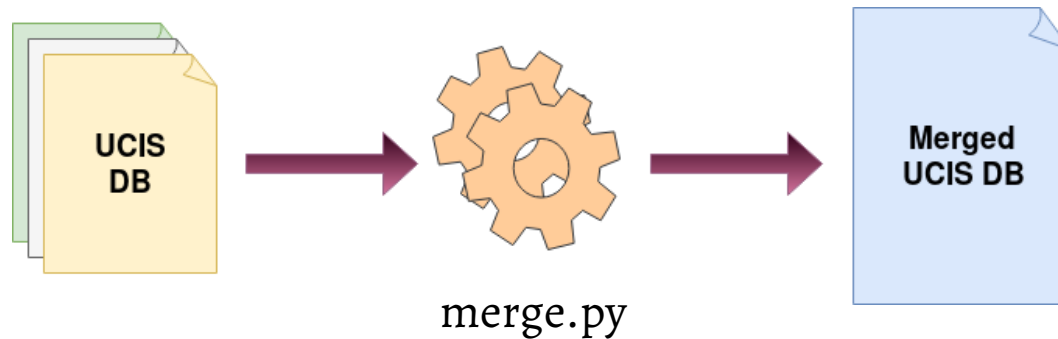


Database management: creation

- Generate XML, respecting UCIS DB Schema:
`fc4sc::global::coverage_save("cov_db.xml");`
- Reports can be generated at any point during runtime!

Database management: merge

Merge = aggregate the coverage data collected from different executions



```
$> python merge.py top/directory merged_db.xml
```

Database management: reporting

JavaScript app: *fc4sc/gui/index.html*

/home/drados/Desktop/git/fc

Contents of the file:

Show full



Show partial



Show empty



Covergroup types

[Back to top](#)

[output_coverage](#)

[fsm_coverage](#)

[stimulus_coverage](#)

output_coverage

75.00%	output_coverage_1				
	100.00%	data_ready_cvp "value"			
		zero	0	1	✓
		positive	[1:2147483646]	18	✓
		negative	[-2147483647:-1]	5	✓
		illegal_zero	0	1	✗
	50.00%	output_valid_cvp "valid"			
		valid	1	24	✓
		invalid	0	0	✗

Roadmap

1. Better UCIS DB support
2. Default bins
3. Transition coverage
4. Cross filtering (with)
5. Cross bins definition (binsof , intersect)
6. Different output formats (Json, XML etc.)

Q & A