# UVM-SystemC – Functional coverage & constrained randomization

Stephan Gerth

Stephan.Gerth@bosch-sensortec.com

BOSCH

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

# Agenda

- UVM-SystemC Updates
- Functional Coverage with FC4SC
- Wrap-Up

# Why UVM-SystemC

- Elevate verification beyond block-level towards system-level
  - System verification and software-driven verification executed by teams not familiar with SystemVerilog and its simulation environment
  - Tests coded in C or C++ as system and SW engineers use an (open source) tool-suite for embedded system design and SW development
- Structured ESL verification environment
  - verification environments for Virtual Platforms and Virtual Prototypes not conforming to following verification environments
  - Key: Benefits if the system-level verification environment is UVM compliant and can be reused later by the IC verification team
- Extendable, fully open source, and future proof
  - Based on Accellera's Open Source SystemC simulator
  - As SystemC is C++, libraries can be integrated easily (e.g. CRAVE, FC4SC)

# UVM-SystemC Updates

- UVM-SystemC 1.0 beta3 release
  - Register API
  - Bugfixes & SystemC 2.3.3 support
  - Ubus example
  - Automatic objection mechanism
- UVM verification ecosystem add-ons
  - Integration of CRAVE via UVM-SystemC layer available
  - Integration of AMIQ‘s functional coverage implementation (FC4SC) as supplemental material

# Functional Coverage for SystemC

Based on slides by Dragos Dospinescu (AMIQ)

# Functional Coverage for SystemC

- What is FC4SC
- Coverage definition API
- Coverage options and sampling API
- Output & visualisation
- Documentation
- What can be improved
- Basic mechanisms demonstrated on SFIFO example

# What is FC4SC (1)

- C++11 header only library:
  - built from scratch, with no 3rd party library dependencies
  - Based on IEEE 1800 - 2012 SystemVerilog Standard
  - https://github.com/amiq-consulting/fc4sc
- Features:
  - Coverage model construction
  - Coverage sampling control & options
  - Runtime coverage queries
  - Coverage database saving

# What is FC4SC (2)

Coverage DB management tools

1) Coverage DB visualisation tool (JavaScript):

   *fc4sc/tools/gui/index.html*

1) Coverage DB merge tool (Python):

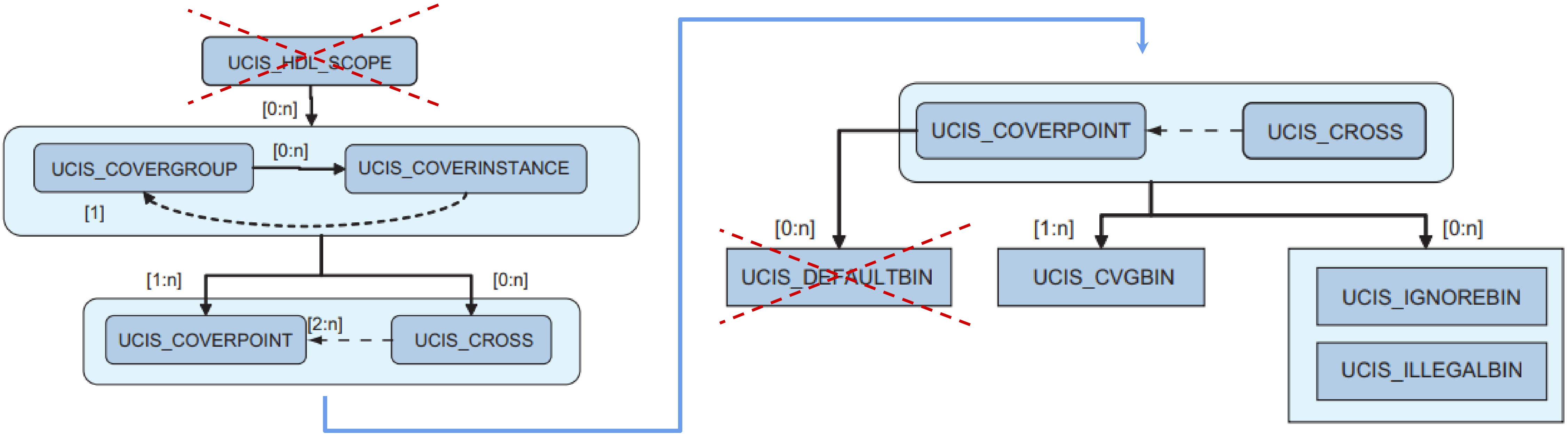
   *fc4sc/tools/coverage_merge/merge.py*

Easy to use; just

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

# Coverage definition API: overview

- Follows UCIS DB coverage data model:
- Elements: bin, coverpoint, cross, covergroup



Crossed out elements are not currently part of the implementation

# Coverage definition API: covergroup

```
class cvg_ex: public covergroup
{
public:
   CG_CONS(cvg_ex) {
    /*user code*/
   }
};
```

```
cvg_ex cg1("cg1");

cvg_ex cg2("cg2");
```

```
#define CG_CONS(type, args...) \
  using covergroup::sample; \
  type(std::string inst_name = "", ##args) : fc4sc::covergroup(#type, __FILE__, __LINE__, inst_name)
```

# Coverage definition API: coverpoint (1)

- Register the coverpoint into the covergroup
- Bind sample expression & condition
- Add bins

*This code is part of the user's covergroup definition*

Name & data type

Sample expression & condition

```
COVERPOINT(int, datacp, data*2, flag!=0)
{
    // bin definitions
};
```

# Coverage definition API: bins (basic)

```
bin<int>("less_than_8",
    1,
    interval(2, 3),
    interval(7, 5)
);

illegal_bin<int>("10", 10);
ignore_bin<int>("100", 100);
```

Multiple bin types → different sampling behavior

! name (std::string) → first argument is **mandatory**

! values / intervals → leading arguments **at least one**

# Coverage definition API: bins (complex #1)

```
// 2 bins inside [0:255]
bin_array<int>("split",
    2,
    interval(0, 255)
);
```

*Expands to multiple separate bins inside the coverpoint*

```
bin<int>("split[0]", interval(0, 128)),
bin<int>("split[1]", interval(129, 255))
```

# Coverage definition API: bins (complex #2)

```
auto fibonacci = [](size_t N) -> std::vector<int>
{
  int f0 = 1, f1 = 2; // initialize start number
  std::vector<int> result(N, f0);
  // calculate following fibonacci numbers
  for (size_t i = 1; i < N; i++) {
      std::swap(f0, f1);
      result[i] = f0;
      f1 += f0;
  }
  return result;
};
COVERPOINT(int, bin_array_cvp, value) {
      bin_array<int>("fib", fibonacci(5))
};
```

```
bin<int>("fib[0]", 1),
bin<int>("fib[1]", 2),
bin<int>("fib[2]", 3),
bin<int>("fib[3]", 5),
bin<int>("fib[4]", 8)
```

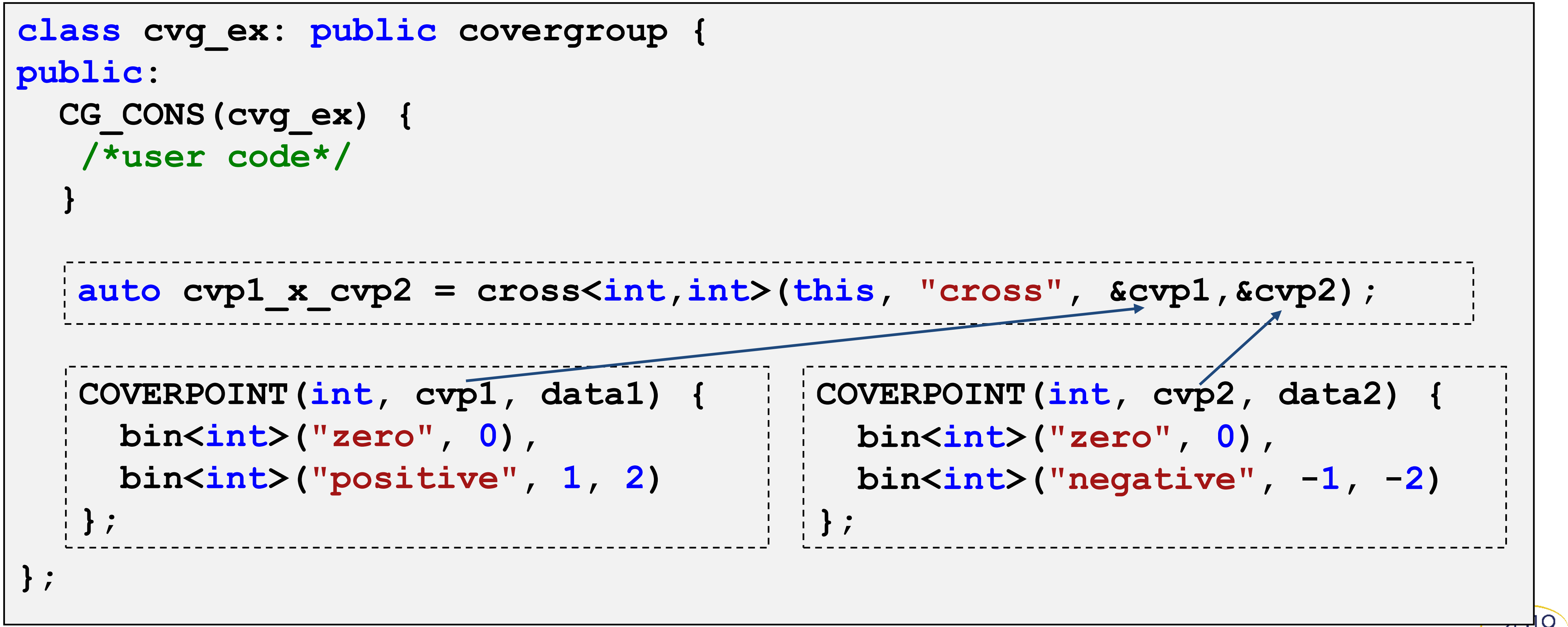
# Coverage definition API: bins + coverpoint

→ *bins are added at the coverpoint definition*

```
COVERPOINT(int, datacp, data * 2, flag != 0)
{
   illegal_bin<int>("illegal_3", 3),
   ignore_bin<int>("ignore_2", 2),
   bin<int>("four", 4),
   bin<int>("other", 11, interval(5,10), interval(20,30))
};
```

*The order and number of bins are arbitrary!*

# Coverage definition API: cross



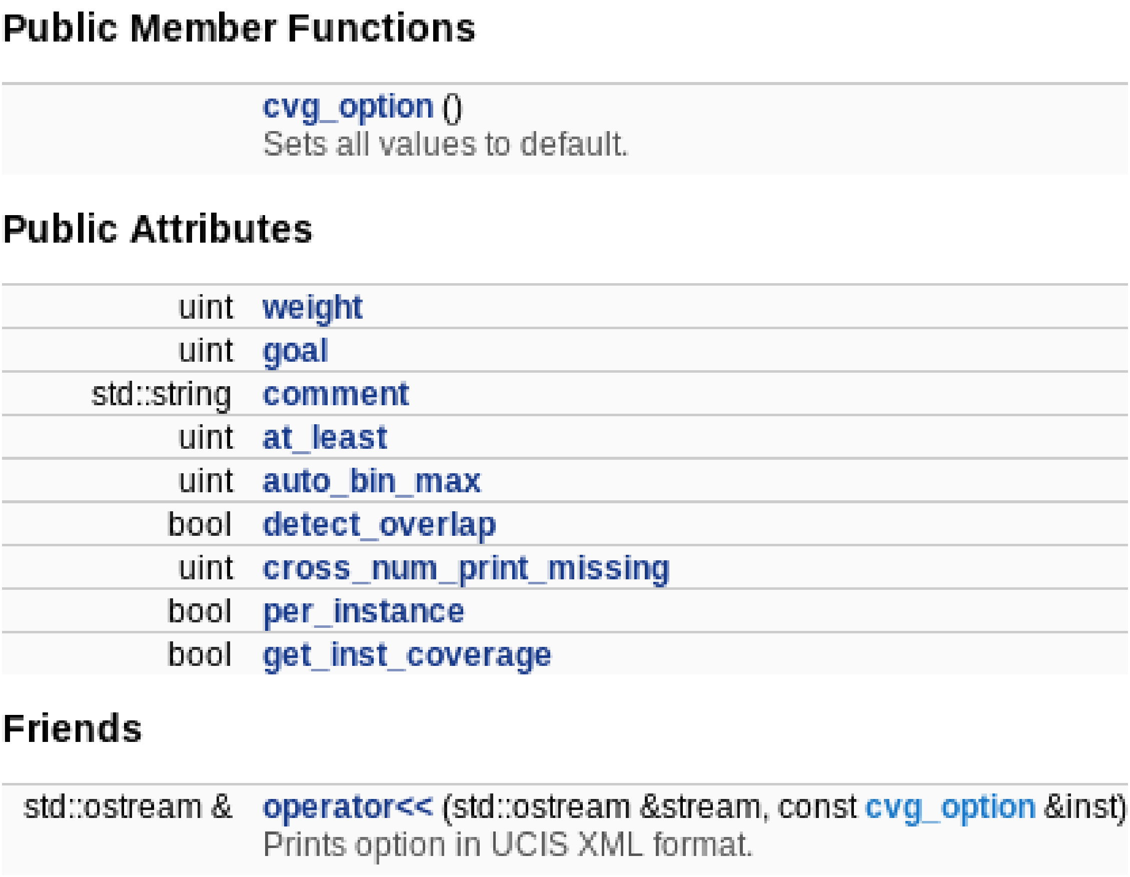
```
class cvg_ex: public covergroup {
public:
  CG_CONS(cvg_ex) {
   /*user code*/
  }

  auto cvp1_x_cvp2 = cross<int,int>(this, "cross", &cvp1,&cvp2);

  COVERPOINT(int, cvp1, data1) {
    bin<int>("zero", 0),
    bin<int>("positive", 1, 2)
  };

  COVERPOINT(int, cvp2, data2) {
    bin<int>("zero", 0),
    bin<int>("negative", -1, -2)
  };
};
```

# Coverage options & sampling API (1)



## Public Member Functions

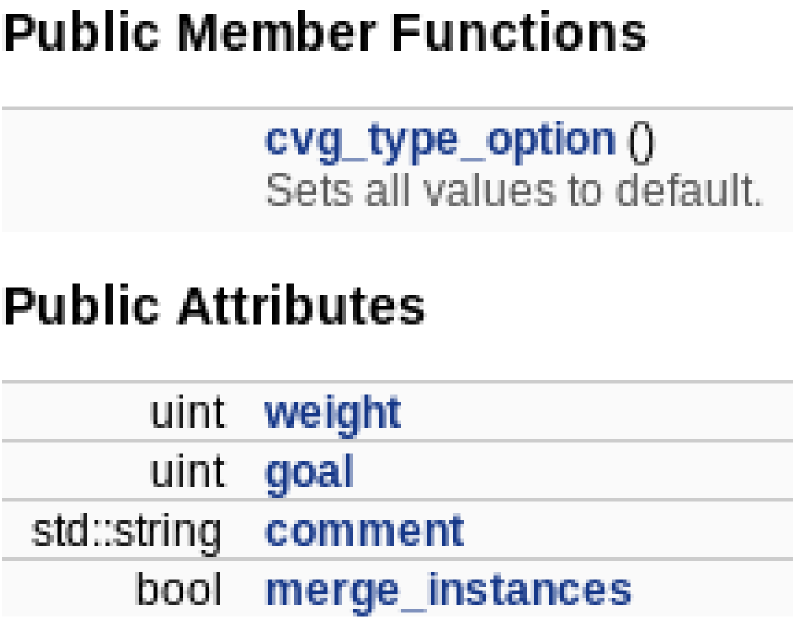
| | |
|---|---|
| | cvg_option ()<br>Sets all values to default. |

## Public Attributes

| | |
|---|---|
| uint | weight |
| uint | goal |
| std::string | comment |
| uint | at_least |
| uint | auto_bin_max |
| bool | detect_overlap |
| uint | cross_num_print_missing |
| bool | per_instance |
| bool | get_inst_coverage |

## Friends

| | |
|---|---|
| std::ostream & | operator<< (std::ostream &stream, const cvg_option &inst)<br>Prints option in UCIS XML format. |



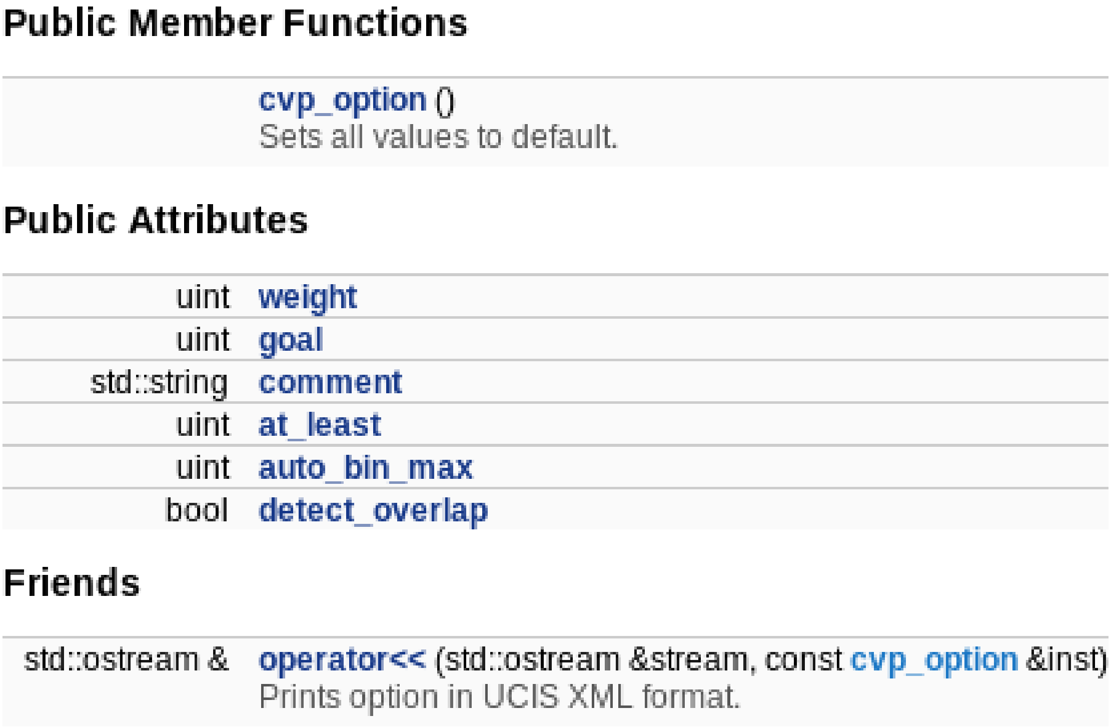
## Public Member Functions

| | |
|---|---|
| | cvg_type_option ()<br>Sets all values to default. |

## Public Attributes

| | |
|---|---|
| uint | weight |
| uint | goal |
| std::string | comment |
| bool | merge_instances |

# Coverage options & sampling API (2)



### Public Member Functions

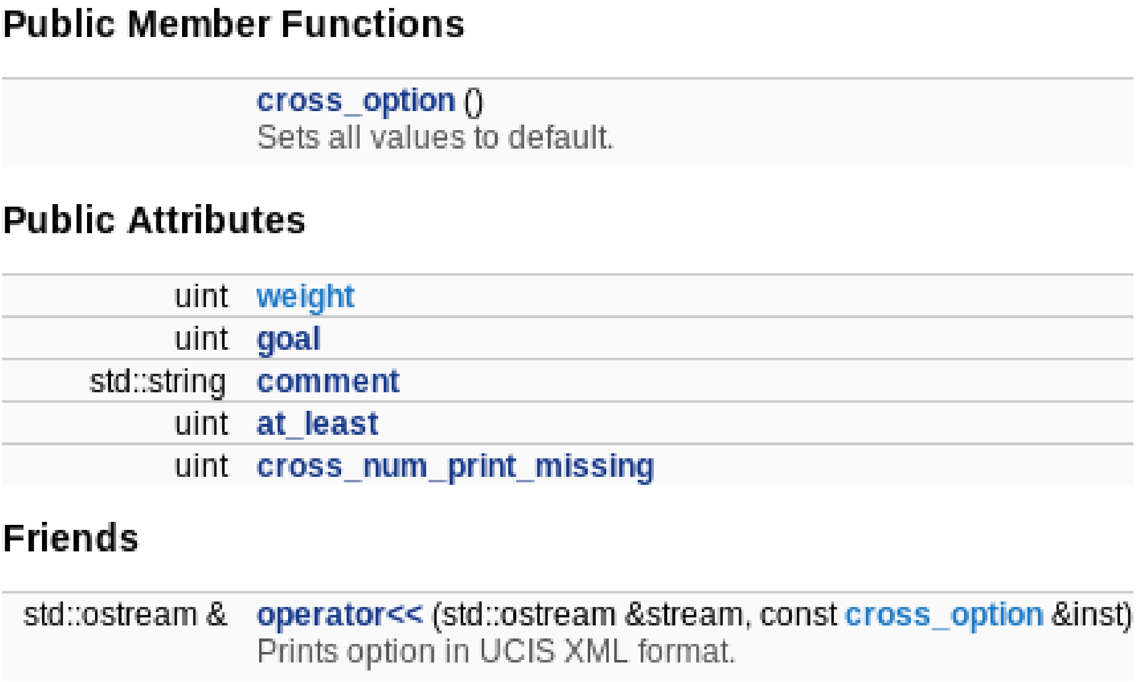
| | |
|---|---|
| | cvp_option ()<br>Sets all values to default. |

### Public Attributes

| | |
|---|---|
| uint | weight |
| uint | goal |
| std::string | comment |
| uint | at_least |
| uint | auto_bin_max |
| bool | detect_overlap |

### Friends

| | |
|---|---|
| std::ostream & | operator<< (std::ostream &stream, const cvp_option &inst)<br>Prints option in UCIS XML format. |



### Public Member Functions

| | |
|---|---|
| | cross_option ()<br>Sets all values to default. |

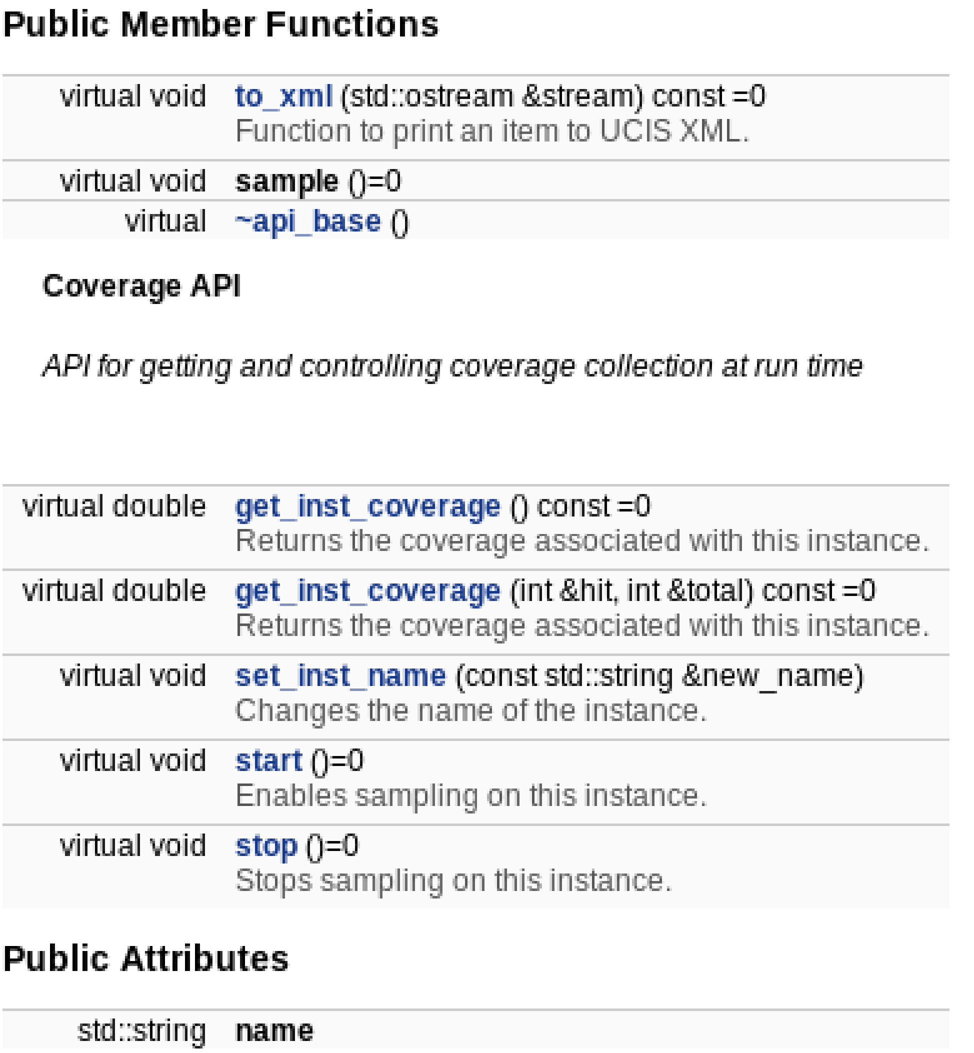
### Public Attributes

| | |
|---|---|
| uint | weight |
| uint | goal |
| std::string | comment |
| uint | at_least |
| uint | cross_num_print_missing |

### Friends

| | |
|---|---|
| std::ostream & | operator<< (std::ostream &stream, const cross_option &inst)<br>Prints option in UCIS XML format. |

# Coverage options & sampling API (3)



### Public Member Functions

| | |
|---|---|
| virtual void | **to_xml** (std::ostream &stream) const =0<br>Function to print an item to UCIS XML. |
| virtual void | **sample** ()=0 |
| virtual | **~api_base** () |

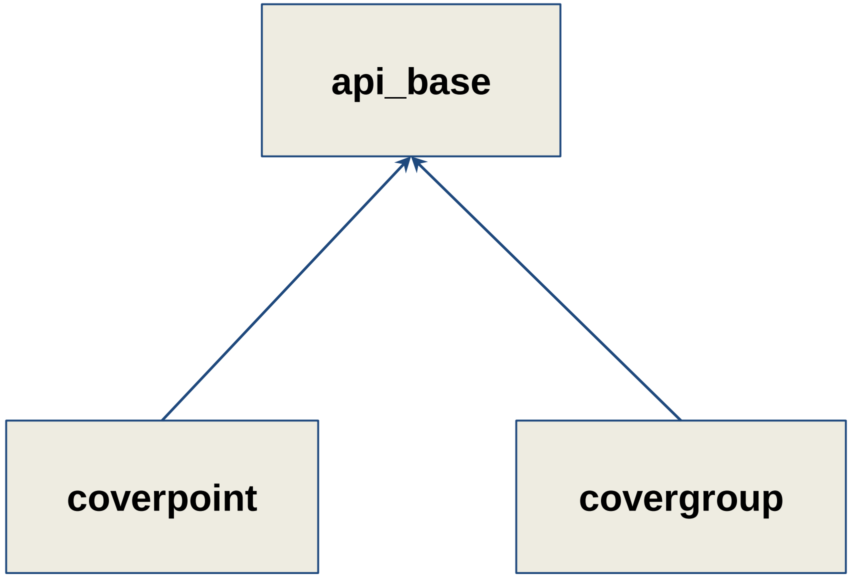
**Coverage API**

*API for getting and controlling coverage collection at run time*

| | |
|---|---|
| virtual double | **get_inst_coverage** () const =0<br>Returns the coverage associated with this instance. |
| virtual double | **get_inst_coverage** (int &hit, int &total) const =0<br>Returns the coverage associated with this instance. |
| virtual void | **set_inst_name** (const std::string &new_name)<br>Changes the name of the instance. |
| virtual void | **start** ()=0<br>Enables sampling on this instance. |
| virtual void | **stop** ()=0<br>Stops sampling on this instance. |

### Public Attributes

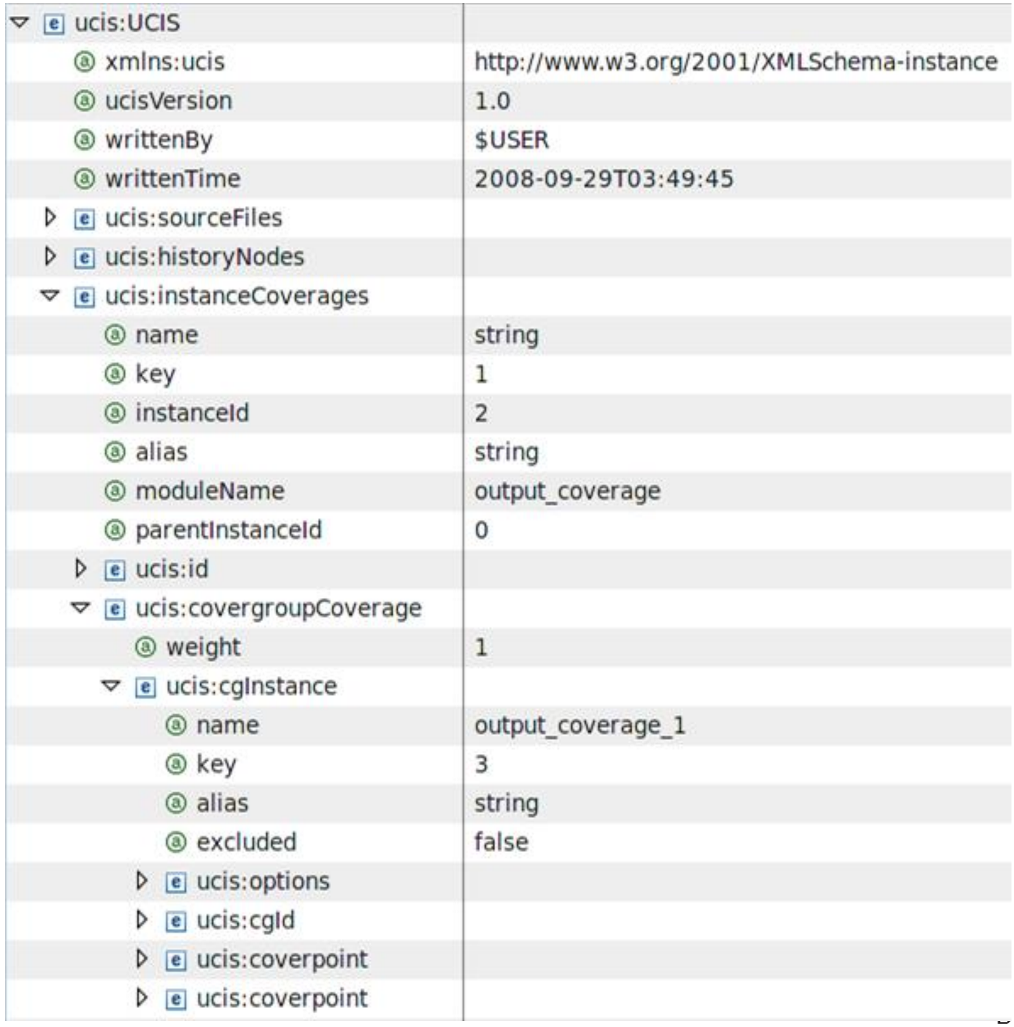
| | |
|---|---|
| std::string | **name** |



# Output & visualization

Generate output (from code):

*fc4sc::global::coverage_save("out.xml");*



```
ucis:UCIS
    xmlns:ucis                        http://www.w3.org/2001/XMLSchema-instance
    ucisVersion                       1.0
    writtenBy                         $USER
    writtenTime                       2008-09-29T03:49:45
    ucis:sourceFiles
    ucis:historyNodes
    ucis:instanceCoverages
        name                          string
        key                           1
        instanceId                    2
        alias                         string
        moduleName                    output_coverage
        parentInstanceId              0
        ucis:id
        ucis:covergroupCoverage
            weight                    1
            ucis:cgInstance
                name                  output_coverage_1
                key                   3
                alias                 string
                excluded              false
                ucis:options
                ucis:cgId
                ucis:coverpoint
                ucis:coverpoint
```

# Documentation

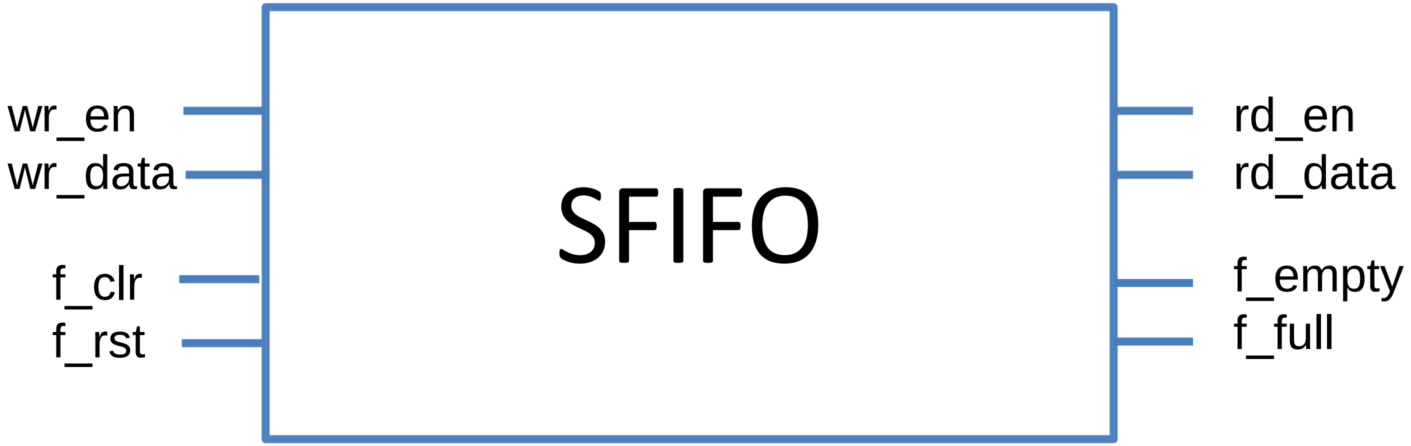
1) Doxygen
2) PDF User guide
3) github.com/amiq-consulting/fc4sc repository releases notes

# What can be improved

- Coverpoint definition API
- Custom types parametrization for *bin*, *coverpoint*, *cross*?
- Add default bins
- Add cross bins filtering
- Add cross sampling condition
- Add coverage model visitor
- Better UCIS DB support
- More support of coverage options

# SFIFO example

- Synchronous FIFO
- Coverage of data & status signals



# UVM-SystemC Wrap-Up

# UVM-SystemC Wrap-Up

- CRAVE integration layer to be part of UVM-SystemC PoC
- Functional Coverage w/ FC4SC
  - Integration of AMIQ‘s functional coverage implementation (FC4SC) as supplemental material
  - API standardization for functional coverage major topic for next year
- Sound verification environment using state of the art techniques

- Input and support from interested parties welcome!

# UVM-SystemC Wrap-Up

- References
  - SystemC Verification Working Group
    - https://www.accellera.org/activities/working-groups/systemc-verification
  - UVM-SystemC
    - https://accellera.org/images/downloads/drafts-review/uvm-systemc-1.0-beta3.tar.gz
  - FC4SC
    - https://github.com/amiq-consulting/fc4sc
  - CRAVE
    - http://www.systemc-verification.org/crave

# Questions