UVM-SystemC – Functional coverage & constrained randomization

Stephan Gerth (Bosch), Dragos Dospinescu (AMIQ), Muhammad Hassan (DFKI / Uni Bremen), Thilo Vörtler (Coseda), Manuel Soto (Fraunhofer IIS/EAS)
Agenda

• UVM-SystemC Updates
• Functional Coverage with FC4SC
• Constrained Randomization with CRAVE
• Application Examples
• 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

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

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

coverage definition API: covergroup

cvg_ex cg1(“cg1”);
cvg_ex cg2(“cg2”);
Coverage definition API: coverpoint (1)

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

```
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)

```cpp
// 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))
```
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))
};
Coverage definition API: bins + coverpoint

→ *bins are added at the coverpoint definition*

```c
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!*
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

```cpp
cvg_option()
Sets all values to default.
```

Public Attributes

```cpp
tuint 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
```

Public Member Functions

```cpp
cvg_type_option()
Sets all values to default.
```

Public Attributes

```cpp
tuint weight
uint goal
std::string comment
```

Friends

```cpp
std::ostream & operator<< (std::ostream &stream, const cvg_option &inst)
Prints option in UCIS XML format.
```
### Public Member Functions

**cvp_option()**
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

```cpp
std::ostream & operator<<(std::ostream & stream, const cvp_option & inst)
Prints option in UCIS XML format.
```

---

### Public Member Functions

**cross_option()**
Sets all values to default.

### Public Attributes

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

### Friends

```cpp
std::ostream & operator<<(std::ostream & stream, const cross_option & inst)
Prints option in UCIS XML format.
```
Coverage options & sampling API (3)

Public Member Functions

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

Public Attributes

- std::string name
Output & visualization

Generate output (from code):

```cpp
fc4sc::global::coverage_save("out.xml");
```
1) Doxygen
2) PDF User guide
3) [github.com/amiq-consulting/fc4sc](https://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
Constrained Randomization

Muhammad Hassan

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project CONVERS under contract no. 16ES0656.
Constrained Randomization

• Randomization as the basis of verification
  – Constraints on stimulus
  – Faster and accurate coverage closure

• Bugs detection due to
  – unexpected combinations of inputs
  – extreme input values

• Detects
  – Hidden bugs
  – Corner cases
Basic of CRAVE

• Constrained Random Verification Environment

• Syntax and semantics closely followed SystemVerilog IEEE 1800 std

• Random objects

• Random variables

• Hard/soft constraints

• Efficient constraint solvers
CRAVE Architecture

- C++ Constraint Specification
- Intermediate Representation (Constrained Objects)
- Pre-generation Analyses
- Generation
- Post-generation Analyses

- Constraint Partitioner
- Soft Constraint Analyzer
- Distribution Solver
- Solver Parallelizer
- Constraint Debugger
- Coverage Analyzer

Multi-solver backend

- SMT Solvers
- BDD Solver
Random Variable and Constraints

class item;
    rand int v;
    constraint c { v < 10; }
endclass

class item : public crv_sequence_item {
    crv_variable<int> v; //Random Variable
    crv_constraint c { v() < 10 }; //Constraint
    item(crv_object_name) {}
};
class item;
  rand int v;
  constraint c {v dist {[0,9]:=1};}
endclass

class item : public crv_sequence_item {
  item(crv_object_name) {} 
  crv_variable<int> v; //Variable
  crv_constraint c {dist(v(),
    make_distribution(range<int>(0,9))}); //Distribution
};
Weighted Range

class item;
    rand int v;
    constraint c {v dist {
        [0,4]:/40, // |[0:4]|=5 => 40/5=8 per value
        [5,9]:/60 // |[5:9]|=5 => 60/5=12 per value
    }}
endclass

class item : public crv_sequence_item {
    item(crv_object_name) {}
    crv_variable<int> v; //Variable
    crv_constraint c {dist(v(),
        make_distribution(weighted_range<int>(0, 4, 40),
        weighted_range<int>(5, 9, 60)))); //Weighted Range
};

SystemVerilog

<table>
<thead>
<tr>
<th>v</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>0.08</td>
<td>0.08</td>
<td>0.08</td>
<td>0.08</td>
<td>0.08</td>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
</tr>
</tbody>
</table>

SystemC
Ranges and Distribution

class item;
    rand int src_addr, dest_addr;
    constraint c_src_addr_range{src_addr dist{
        [0:9]:=1,
        [90,99]:=1
    }}
    constraint c_dest_addr_range{dest_addr dist{
        [0:2]/60,
        [10,12]/30,
        [100,103]/10
    }}
endclass

: = operator indicates the weight is the same for all values
: / operator indicates the weight is distributed across all values

<table>
<thead>
<tr>
<th>src</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src</th>
<th>90</th>
<th>91</th>
<th>92</th>
<th>93</th>
<th>94</th>
<th>95</th>
<th>96</th>
<th>97</th>
<th>98</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

endclass
Ranges and Distribution

class item;
  rand int src_addr, dest_addr;

<table>
<thead>
<tr>
<th>dest</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>100</th>
<th>101</th>
<th>102</th>
<th>103</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>2/10</td>
<td>2/10</td>
<td>2/10</td>
<td>1/10</td>
<td>1/10</td>
<td>1/10</td>
<td>0.025</td>
<td>0.025</td>
<td>0.025</td>
<td>0.025</td>
</tr>
</tbody>
</table>

constraint c_dest_addr_range{dest_addr dist { [0:2] :: 60, // |[0:2]|=3 => 60/3=2 per value [10,12] :: 30, // |[10:12]|=3 => 30/3=1 per value [100,103] :: 10 // |[100:103]|=4 => 10/4=2.5 per value }

endclass

:= operator indicates the weight is the same for all values

:/ operator indicates the weight is distributed across all values
class item : public crv_sequence_item {
public:
    item(crv_object_name) {}  
crv_constraint c_src_addr_range{dist(src_addr()),
    make_distribution(range<int>(0, 9), range<int>(90, 99))};

crv_constraint c_dest_addr_range{dist(dest_addr()),
    make_distribution(weighted_range<int>(0, 2, 60),
    weighted_range<int>(10, 12, 30),
    weighted_range<int>(100, 103, 10))};

crv_variable<int> src_addr, dest_addr;
};
Ranges and Distribution

class item : public crv_sequence_item {
public:
    item(crv_object_name) {}
    crv_constraint c_src_addr_range{dist(src_addr()),
        make_distribution(range<int>(0, 9), range<int>(90, 99))};

    crv_constraint c_dest_addr_range{dist(dest_addr()),
        make_distribution(weighted_range<int>(0, 2, 60),
            weighted_range<int>(10, 12, 30),
            weighted_range<int>(100, 103, 10))};
    crv_variable<int> src_addr, dest_addr;
};

Output:
src_addr: 94 dest_addr: 12
src_addr: 93 dest_addr: 12
src_addr: 0 dest_addr: 102
...
enum {AUDI,BMW} car_type_enum;
enum {RED,GREEN} color_enum;

class item
    rand car_type_enum car;
    rand color_enum color;
    constraint car_color {(car==AUDI)->(color != RED);
                            (car==BMW)->(color != BLUE);}
endclass

CRAVE_BETTER_ENUM(car_type_enum, AUDI = 1, BMW = 2);
CRAVE_BETTER_ENUM(color_enum, RED, GREEN);

class item : public crv_sequence_item {
    crv_variable<car_type_enum> car;
    crv_variable<color_enum> color;
    crv_constraint car_color{
        if_then(car() == car_type_enum::AUDI, color() != RED),
        if_then(car() == car_type_enum::BMW, color() != BLUE)};
};
class Packet;
    bit [31:0] src, dst, data[8];
    rand bit [7:0] kind;
    constraint c{src > 10; src < 15; kind dist {[0,255]:=1}}
endclass

Packet p;
initial begin
    p=new();
    if (!p.randomize()) // Returns a “0” if problem
        $finish;
    transmit(p);
end

class Packet : public crv_sequence_item {
    public:
        Packet(crv_object_name) {}
        crv_constraint c { src() > 10, src() < 15, dist(kind()),
                            make_distribution(range<
                                uint8_t>(0,
                            255)));
        crv_variable<
                    sc_bv<32>> src, dst, data[8];
        crv_variable<
                    sc_bv<8>> kind;
};
Packet Example

Output:
00000000000000000000000000001100 01010111
00000000000000000000000000001100 10010111
00000000000000000000000000001100 00100110
00000000000000000000000000001100 11100111
...

class Packet : public crv_sequence_item {
public:
    Packet(crv_object_name) {}    
crv_constraint c { src() > 10, src() < 15, dist(kind(),
                        make_distribution(range<uint8_t>(0, 255))));
crv_variable<sc_bv<32>> src, dst, data[8];
crv_variable<sc_bv<8>> kind;
};
CRAVE Constraint Operators

```cpp

crv_variable<type> x0, x1, x2;
crv_constraint plus{x0() == x1() + x2()};
crv_constraint minus{x2() == x0() - x1()};
crv_constraint mul{x2() == x1() * 2};  
crv_constraint div{x1() == x2() / 2};
crv_constraint nequal{x1() != x2() || x1() == x2()};
crv_constraint rshift{x2() == x1() << 1};
crv_constraint lshift{x1() == x2() >> 1};
crv_constraint range0{x1() >= 0 && x1() <= 5};
crv_constraint range1{x1() >= 0 && x1() <= x2()};
crv_constraint not{!x0() == x1()};
crv_constraint neg{~x() == x1()};
crv_constraint and{x2() == x0() & x1()};
crv_constraint or{x2() == x0() | x1()};
crv_constraint xor{x2() == x0() ^ x1()};
```
Soft Constraints

```systemverilog
class item;
    rand int src_addr, dest_addr;
    constraint hard_c{src_addr <= 20; dest_addr <= 100};
    constraint soft_c{soft src_addr % 4 == 0};
endclass
```

```systemc
struct item : public crv_random_obj {
    crv_variable<int> src_addr;
    crv_variable<int> dest_addr;

    crv_constraint hard_c{ src_addr() <= 20, dest_addr() <= 100 };
    crv_soft_constraint soft_c{ src_addr() % 4 == 0 };

    item(crv_object_name) {} 
};
```

© Accellera Systems Initiative 39
UBus randomization with CRAVE

Muhammad Hassan

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project CONVERS under contract no. 16ES0656.
UBus

• Motivation
  – Well-known SystemVerilog example

• DUT & Testbench setup
  – DUT functionality
  – Testbench layout (Agents, etc.)
UBus

- UBus
  - simple non-multiplexed
  - Synchronous
  - no pipelining
  - Address bus: 16 bit wide
  - Data bus: 8 bit wide
- N number of Masters & Slaves supported

- Three sequential phase data transfer
  - Arbitration
  - Address
  - data

- UVM example provided in the UVM Users Guide (http://accellera.org/downloads/standards/uvm)
class ubus_transfer extends uvm_sequence_item;
rand bit [15:0] addr;
rand ubus_rw_enum read_write;
rand int unsigned size;
rand bit [7:0] data[];
rand bit [3:0] wait_state[];
rand int unsigned error_pos;
rand int unsigned transmit_delay = 0;
...

constraint c_read_write {
    read_write inside { READ, WRITE }
}

constraint c_size {
    size inside {1,2,4,8};
}

constraint c_data_wait_size {
    data.size() == size;
    wait_state.size() == size;
}

constraint c_transmit_delay {
    transmit_delay <= 10;
}

class ubus_transfer : public uvm_randomized_sequence_item {
public:
    crv_variable<ubus_rw_enum> read_write;
    crv_variable<sc_bv<16>> addr;
    crv_variable<unsigned> size;
    crv_vector<sc_bv<8>> data;
    crv_vector<sc_bv<4>> wait_state;
    crv_variable<unsigned> error_pos;
    crv_variable<unsigned> transmit_delay;
    ...

    crv_constraint c_read_write {inside(read_write(),
        std::set<ubus_rw_enum> {
            ubus_rw_enum::READ, ubus_rw_enum::WRITE
        });
    }

    crv_constraint c_size {inside(size(),
        std::set<int> { 1, 2, 4, 8 }
    });

    crv_constraint c_data_wait_size {
        data().size() == size(),
        wait_state().size() == size()
    };

SystemVerilog  |
---|
SystemC
### UBus – Randomized sequence

```systemverilog
class write_double_word_seq extends ubus_base_sequence;
...
    rand bit [15:0] start_addr;
    rand bit [7:0] data0;    rand bit [7:0] data1;    rand bit [7:0] data2;
    rand bit [7:0] data3;    rand bit [7:0] data4;    rand bit [7:0] data5;
    rand bit [7:0] data6;    rand bit [7:0] data7;
    rand int unsigned transmit_del = 0;
    constraint transmit_del_ct { (transmit_del <= 10); }

    virtual task body();
...
```

```systemc
template<typename REQ = ubus_transfer, typename RSP = REQ>
class write_double_word_seq : public ubus_base_sequence<REQ, RSP> {
public:
    crv_variable<sc_bv<16>> start_addr;
    crv_variable<sc_bv<8>> data0, data1, ... data7;
    crv_variable<unsigned int> transmit_del;
    crv_constraint transmit_del_ct { transmit_del() <= 10 };  

    void body() {
...
```
UBus – Randomized sequence

```systemverilog
`uvm_do_with(req,
    { req.addr == start_addr; req.read_write == WRITE; req.size == 8;
      req.data[0] == data0;
      req.data[1] == data1;
      ...
      req.data[7] == data7;
      req.error_pos == 1000;
      req.transmit_delay == transmit_del; }
)
```

```
UVM_DO_WITH(req, (req.addr() == start_addr(),
    req.read_write() == ubus_read_write_enum::WRITE,
    req.size() == 8,
    req.data()[0] == data0(),
    req.data()[1] == data1(),
    ...
    req.data()[7] == data7(),
    req.error_pos() == 1000,
    req.transmit_delay() == transmit_del()
));
```
How to get CRAVE

- [www.systemc-verification.org/crave](www.systemc-verification.org/crave)
- Download / clone from GitHub!
- Follow installation instructions given in INSTALL.txt
- CRAVE is freely available under the MIT license.
- For more information (crave@informatik.uni-bremen.de)
Closing verification gap

Manuel Soto
Closing verification gap

• Verification of sensor models
  – Accelerometer
  – Pressure sensor and temperature sensor
  – Gyroscope
  – Support of I2C and SPI protocol
Closing verification gap

• Steps
  – Step-1
    • Define verification plan

<table>
<thead>
<tr>
<th>Testlevel</th>
<th>Component</th>
<th>Subcomponent</th>
<th>Block</th>
<th>Scenario</th>
<th>Testenvname</th>
<th>Testosename</th>
<th>Subdocuments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor Configuration</td>
<td>Systemlevel</td>
<td>useop_toplevel_stim_test_level2_sensor_configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor Calibration</td>
<td>Systemlevel</td>
<td>useop_toplevel_stim_test_level2_sensor_calibration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check SPI write transaction</td>
<td>Systemlevel</td>
<td>vip_spi_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check SPI read transaction</td>
<td>Systemlevel</td>
<td>vip_spi_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check SPI write burst transaction</td>
<td>Systemlevel</td>
<td>vip_spi_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check SPI read bursts transaction</td>
<td>Systemlevel</td>
<td>vip_spi_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, do a read of the acceleration value on X</td>
<td>Systemlevel</td>
<td>vip_xyz_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, do a read of the acceleration value on Y</td>
<td>Systemlevel</td>
<td>vip_xyz_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, do a read of the acceleration value on Z</td>
<td>Systemlevel</td>
<td>vip_xyz_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check I2C write transaction</td>
<td>Systemlevel</td>
<td>vip_i2c_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check I2C read transaction</td>
<td>Systemlevel</td>
<td>vip_i2c_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check I2C write burst transaction</td>
<td>Systemlevel</td>
<td>vip_i2c_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td>Level 2 Sensors</td>
<td>BM4456</td>
<td>Sensor data transfer, check I2C read bursts transaction</td>
<td>Systemlevel</td>
<td>vip_i2c_test</td>
<td>useop_tests/sensor/vye</td>
<td></td>
</tr>
</tbody>
</table>
Closing verification gap

• Steps
  – Step-2
    • Define environment
      – Sequences
      – Driver
      – Monitor
      – Coverage
      – GM
Closing verification gap
Closing verification gap

• Support for coverage
  – Particular model
    • Coverage_collector
  – Fc4sc used
    • Define coverpoints

```cpp
#include "fc4sc.hpp"
#include <tlm.h>

class spi_cvg: public covergroup{
public:
    int m_value;
    CG_CONS(spi_cvg){
        m_value = 0;
    }
    ...
    COVERPOINT(int, spi_opration_done, m_value) {
        bin<int>("SPI_READ", 0),
        bin<int>("SPI_WRITE", 1),
        ignore_bin<int>("SPI_DUMMY", 2),
        bin<int>("SPI_WRITE_BURST", 3),
        bin<int>("SPI_READ_BURST", 4)
    };
};
```
Closing verification gap

• Intermediate coverage results

<table>
<thead>
<tr>
<th>my_cg</th>
<th>new_table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>spi_operation_done &quot;m_value&quot;</td>
</tr>
<tr>
<td>SPI_READ</td>
<td>0</td>
</tr>
<tr>
<td>SPI_WRITE</td>
<td>0</td>
</tr>
<tr>
<td>SPI_WRITE_BURST</td>
<td>3</td>
</tr>
<tr>
<td>SPI_READ_BURST</td>
<td>4</td>
</tr>
</tbody>
</table>
Closing verification gap

- Intermediate coverage results

<table>
<thead>
<tr>
<th>spi_cvg</th>
<th>my_cg</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100.00%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>spi_operation_done &quot;m_value&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_READ 2 1 ✓</td>
</tr>
<tr>
<td>SPI_WRITE 3 1 ✓</td>
</tr>
<tr>
<td>SPI_WRITE_BURST 5 1 ✓</td>
</tr>
<tr>
<td>SPI_READ_BURST 6 1 ✓</td>
</tr>
</tbody>
</table>
## Closing verification gap

- **Register access control**

---

### registers_access_cvg

<table>
<thead>
<tr>
<th>my_addr_cvog</th>
<th>register_name &quot;aux_addr&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACC.X_LSB</td>
</tr>
<tr>
<td></td>
<td>ACC.X_MSB</td>
</tr>
<tr>
<td></td>
<td>ACC.Y_LSB</td>
</tr>
<tr>
<td></td>
<td>ACC.Y_MSB</td>
</tr>
<tr>
<td></td>
<td>ACC.Z_LSB</td>
</tr>
<tr>
<td></td>
<td>ACC.Z_MSB</td>
</tr>
<tr>
<td></td>
<td>not valid address</td>
</tr>
</tbody>
</table>

---

- **Simulation results**
Closing verification gap

• Support for transaction recording
  – Using scv library
    • `scv_stream`
    • `scv_tr_generator`
    • `scv_tr_handle`

```cpp
#include <scv.h>
...
//declaration
scv_tr_stream* driver_spi_item_tr;
scv_tr_generator<double, double>* driver_spi_gen_seq;
scv_tr_handle spi_seq_h, spi_item_h, spi_sub_seq_h;
...
//new
driver_spi_seq = new scv_tr_stream("driver_spi_seq", "seq_SPI");
...
//begin
spi_seq_h = driver_spi_gen_seq->begin_transaction(sc_time_stamp().to_seconds(),
sc_time_stamp());
    spi_seq_h.record_attribute<std::string>("name", p.name);
    spi_seq_h.record_attribute("addr", p.vip_spi_seq_addr.to_uint());
    spi_seq_h.record_attribute<std::string>("data", p.convert2string_h());
...
//end transaction
driver_spi_gen_seq->end_transaction(spi_seq_h, sc_core::sc_time_stamp().to_seconds(),
sc_time_stamp());
```
Closing verification gap
Closing verification gap
Closing verification gap
Closing verification gap

• Results
  – First time use of coverage library
    • Still open timing sequences
  – Recording transactions, help for debugging
  – Reuse of the environment for
    • Pressure sensor and temperature sensor
    • Gyroscope
Closing verification gap

• Open points
  – Back annotation
  – Automatization for regressions
Randomizing SystemC AMS DUTs

Thilo Voertler
Randomizing SystemC AMS DUTs

- Self checking UVM based test environment

Diagram:
- Sequences/Drivers
- DUT
- Monitors + Signal Checkers
- Scoreboard

Use Constraints to modify input stimulus
Use parallel simulations for speed up
Use constraints to modify DUT and model parameter variations

© Accellera Systems Initiative
Randomizing SystemC AMS DUTs

• SystemC AMS can be used to model analog components at the system level
• Possible to modify DUT before instantiation using constraints
• Not possible to use constraints in SystemVerilog module construction (Constraints can be used in classes only)
• UVM-SystemC allows access to SystemC phases from a UVM test
Randomizing SystemC AMS DUTs

- Parameter Constraints are used before DUT construction is finalized

- Runtime constraints are solved after simulation has started ($t_{sim} > 0$)
Randomizing SystemC AMS DUTs

Build phase of UVM test can be used to modify DUT
Randomizing SystemC AMS DUTs

• Simple example: Setting stuck faults in ADC behavioral model

  • Set one output bit of adc output to "1" → Stuck at fault
  • Several bits can be also set to "1"
  • Modification of DUT just from UVM test possible
void top_test_base::build_phase(uvm::uvm_phase& phase) {
    // connect to SystemC DUT error location probe
    dut_connector<int> stuck_bit_int;
    stuck_bit_int.bind("*i_conv_tdf2de1.tdf_i");

    // define bit error functor to model fault
    auto stuck_at_1_error =
        [&](int error_mask, double time, const int& oldval) { return int(error_mask | oldval); };
    rand_one_hot rand_bits;
    rand_bits.randomize();

    auto error_class = std::bind(stuck_at_1_error, rand_bits.bits, std::placeholders::_1,
                                 std::placeholders::_2);
    stuck_bit_int.change_dynamic(error_class);

    uvm::uvm_test::build_phase(phase);
    // Instantiate the environment used for this test
    m_top_env = top_env::type_id::create("m_top_env", this);
}
Randomizing SystemC AMS DUTs

Ramp - no error

Random stuck at 1 error (bit 5)
DC-DC Converter Example

Digital Inputs

Analog outputs
DC-DC Converter Example

Randomly select value of resistor

Controls output voltage settling time
DC-DC Converter Example – Results
UVM-SystemC Wrap-Up

Stephan Gerth

BOSCH
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

• 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