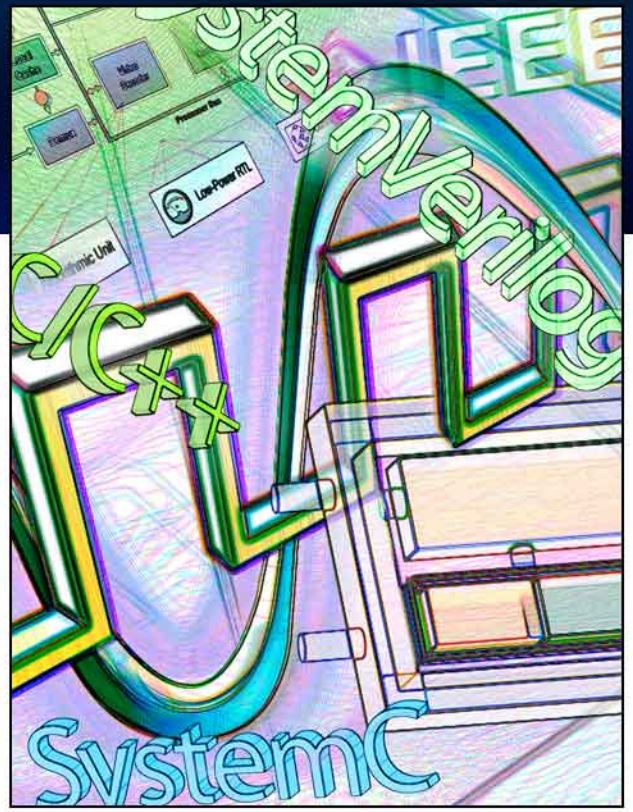


UVM Cookbook
Recipe of the Month:
C-Based Stimulus for UVM

Tom Fitzpatrick

Verification Evangelist

Design Verification Technology

October, 2012

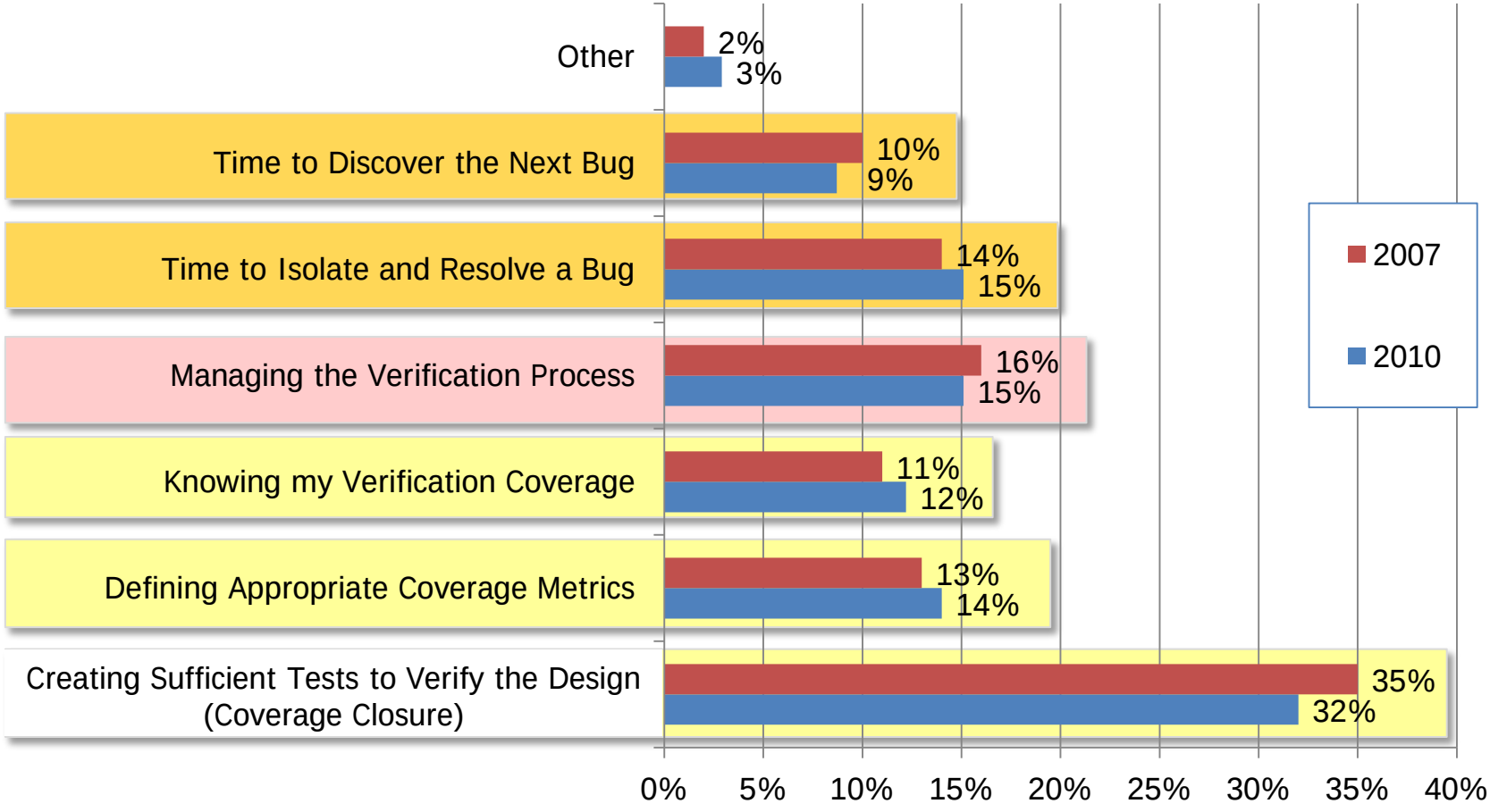
Mentor Graphics®

The Idea Behind The Methodology

- OVM & UVM underpin best practices
  - It's all about people...
  - Team Development
- Peopleware is most important
  - Develop Skill Set
  - Common language
  - Strategy and cohesion
  - Clarity and transparency
- A Guiding Methodology
  - Provides Freedom From Choice
  - Avoids Chaos and Repetition
  - Ease of Use APIs
  - Not just for Super-heroes!

Verification Challenges

Coverage , Process, & Debugging Still Identified as Top Priorities



Non-FPGA Designs

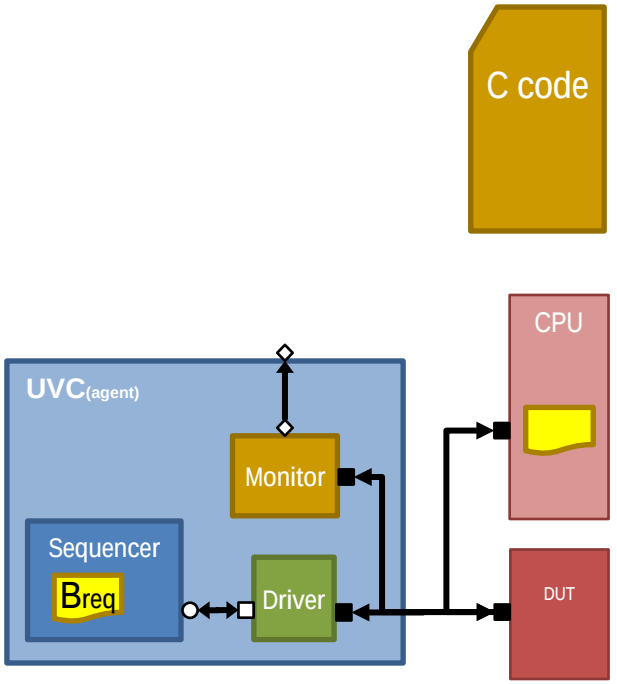
Source: Wilson Research Group and Mentor Graphics 2010 Functional Verification Study

How Do We Create More Tests?

- UVM Sequences
  - Run with multiple seeds
  - Use factory to swap sequences
- inFact Intelligent Testbench Automation
  - Graph-based stimulus
  - Run as UVM Sequence
- Why Not C Code?
  - SW Engineers are available to write tests
  - Devlop device driver code early
  - Reuse tests at higher levels of integration
    - Run on target device

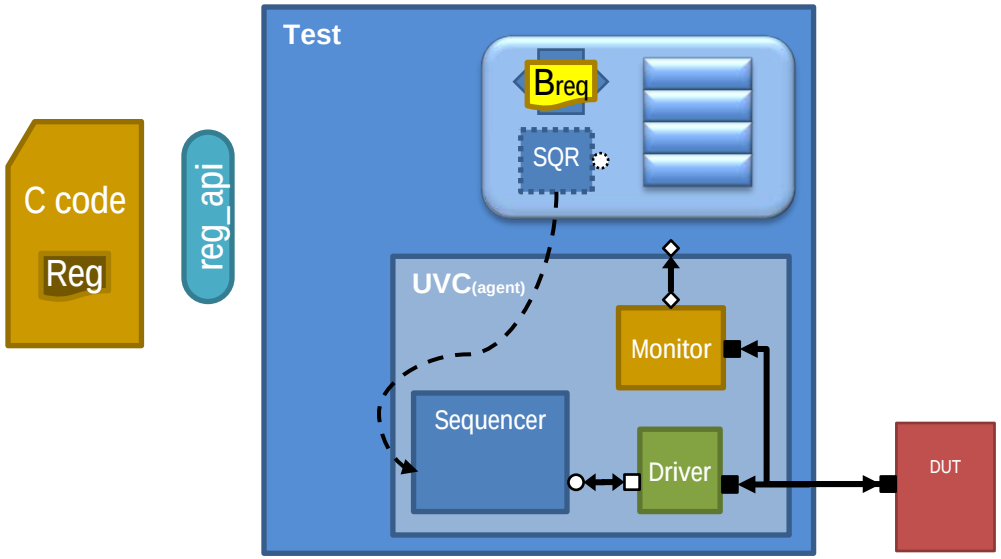
How To Use C Code?
Option 1: Use CPU Model

- Compile C code to run on processor model
- Instantiate processor along with DUT
- Use UVM to drive additional stimulus
  - Background traffic
- Pros
  - Visibility into CPU internals
- Cons
  - Setup overhead
  - Performance overhead



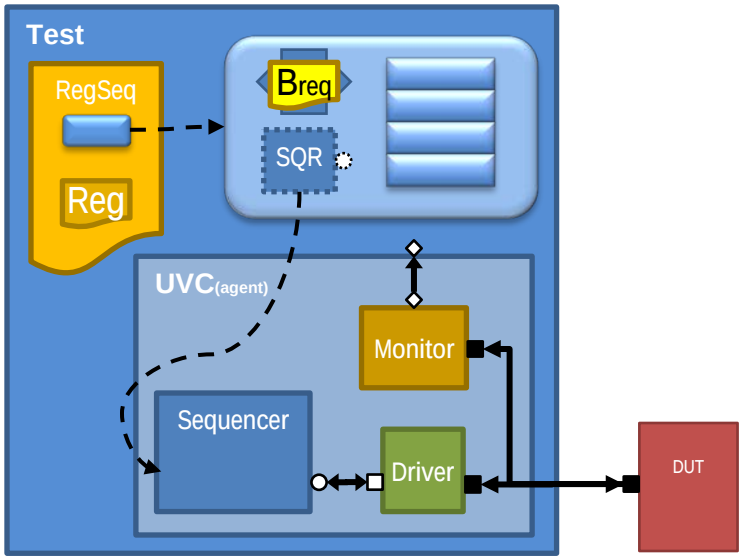
How To Use C Code?
Option 2: Register R/W API + UVM Registers

- A C register read/write API
  - Use DPI to call SV tasks
  - SV tasks make UVM register accesses
  - C code runs on host workstation
- Pros
  - Lightweight
  - Simple extension to existing UVM environment
- Cons
  - Lack of CPU visibility



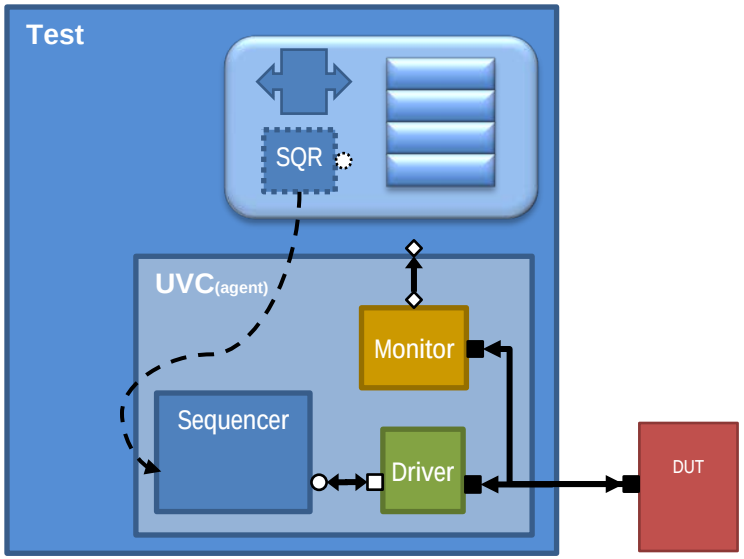
Quick Review: UVM Registers

- Registers contain fields
  - Fields contain bits
- Register block contains all registers that pertain to a DUT block
  - May also contain sub-blocks
- Register map specifies register offsets
  - Defines target sequencer
  - Defines register-to-bus adapter
- Register sequence executes register transactions



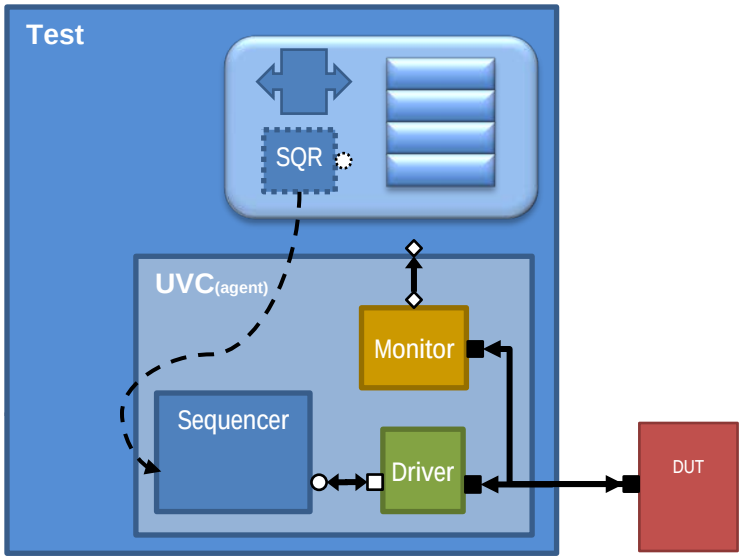
Quick Review: UVM Registers – The Env

```
class spi_env extends uvm_env;
  `uvm_component_utils(spi_env)
  apb_agent m_apb_agent;
  spi_env_config m_cfg;
  reg2apb_adapter reg2apb;
  uvm_reg_predictor #(apb_seq_item) apb2reg_predictor;

  function void build_phase(uvm_phase phase);
    uvm_config_db #(apb_agent_config)::set(this, "m_apb_agent*", "apb_agent_config",
       m_cfg.m_apb_agent_cfg);
    m_apb_agent = apb_agent::type_id::create("m_apb_agent", this);
    apb2reg_predictor =
        uvm_reg_predictor #(apb_seq_item)::
        type_id::create("apb2reg_predictor", this);
  endfunction:build_phase

  function void connect_phase(uvm_phase phase);
    reg2apb = reg2apb_adapter::
                        type_id::create("reg2apb");
    if(m_cfg.spi_rm.get_parent() == null) begin
      m_cfg.spi_rm.APB_map.set_sequencer(
                      m_apb_agent.m_sequencer, reg2apb);
    end
    apb2reg_predictor.map = m_cfg.spi_rm.APB_map;
    apb2reg_predictor.adapter = reg2apb;
    m_cfg.spi_rm.APB_map.set_auto_predict(0);
    m_apb_agent.ap.connect(apb2reg_predictor.bus_in);
  endfunction
```



Quick Review: UVM Registers – Reg Block

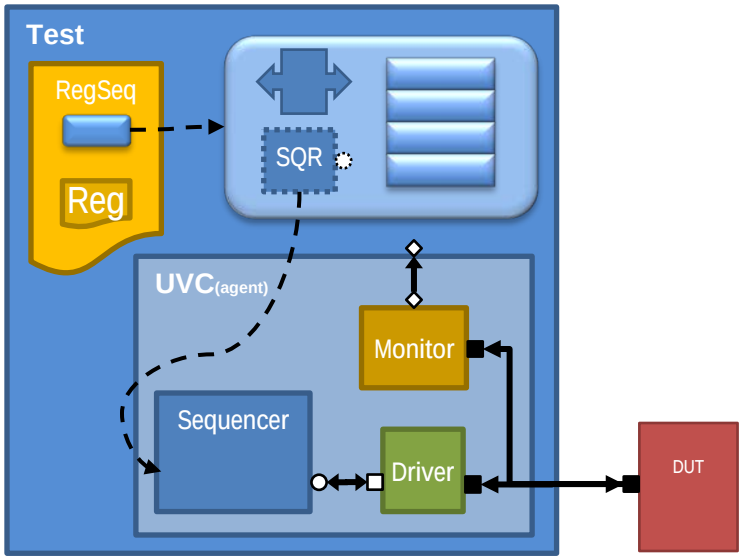
```
class spi_reg_block extends uvm_reg_block;
  `uvm_object_utils(spi_reg_block)

  …
  rand ctrl ctrl_reg;
  rand divider divider_reg;

  uvm_reg_map APB_map; // Block map

  virtual function void build();
    …
    ctrl_reg = ctrl::type_id::create("ctrl");
    ctrl_reg.configure(this, null, "");
    ctrl_reg.build();
    ctrl_reg.add_hdl_path_slice("ctrl", 0, 14);
    divider_reg=divider::type_id::create("divider");
    divider_reg.configure(this, null, "");
    divider_reg.build();
    divider_reg.add_hdl_path_slice("divider", 0, 16);
    …
    APB_map = create_map("APB_map", 'h0, 4,
                         UVM_LITTLE_ENDIAN);
    APB_map.add_reg(ctrl_reg, 32'h00000010, "RW");
    APB_map.add_reg(divider_reg, 32'h00000014, "RW");
    add_hdl_path("top_tb.DUT", "RTL");
    …
  endfunction
```



Quick Review: UVM Registers – Sequence

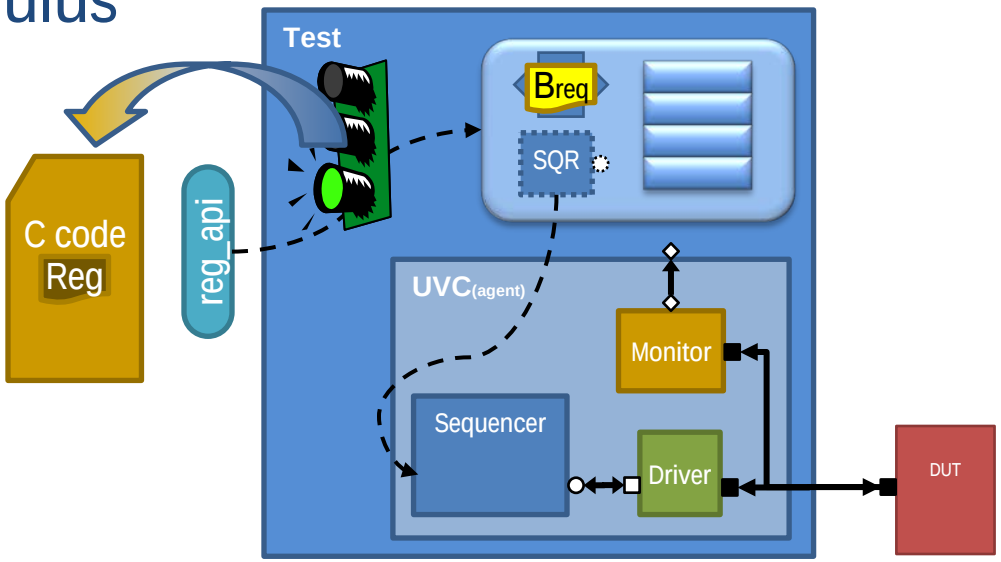
```
class send_10_chars_seq extends spi_bus_base_seq;
  `uvm_object_utils(send_10_chars_seq)

  uvm_reg_data_t control_config = 32'h2c30 ;
  uvm_reg_data_t data_0 = 32'hDEAD_BEEF,
                 data_1 = 32'hBAAD_CAFE;

  task body;
    super.body();
    spi_rm.divider_reg.write(status, 2, .parent(this));
    spi_rm.ctrl_reg.write(status, control_config, .parent(this));
    spi_rm.ss_reg.write(status, 1, .parent(this));
    repeat(10) begin
      spi_rm.rxtx0_reg.write(status, data_0,
                             .parent(this));
      spi_rm.rxtx1_reg.write(status, data_1,
                             .parent(this));
      spi_rm.ctrl_reg.write(status,
          (control_config + 32'h100), .parent(this));
      spi_rm.ctrl_reg.read(status, data,
                           .parent(this));
      while(data[8] == 1) begin
        spi_rm.ctrl_reg.read(status, data,
                             .parent(this));
      end
      data_0++;
      data_1++;
    end
  endtask: body
```

```
class spi_bus_base_seq
        extends uvm_sequence #(uvm_sequence_item);
  `uvm_object_utils(spi_bus_base_seq)
  spi_reg_block spi_rm;
  spi_env_config m_cfg;
  task body;
    if(!uvm_config_db #(spi_env_config)::get(null,
        get_full_name(), "spi_env_config", m_cfg)) begin
      `uvm_error("body", "Could not find spi_env_config")
    end
    spi_rm = m_cfg.spi_rm;
  endtask: body
```



C Stimulus Package Overview

- Requires a register model
- C code accesses registers via (addr,data)
  - Normal C code
  - `include reg_api.h
- Package uses DPI to convert to register reads/writes
- UVM test starts the C stimulus



An (Incredibly Simple) Example

simple_test.c

```
#include "spi_regs.h"
#include "reg_api.h"

int simple_test_routine() {
  int control = 0x2c30;
  int status;

  reg_write(CTRL, control);
  status = reg_read(CTRL);
}

int start_c_code() {
  simple_test_routine();
  return 0;
}
```

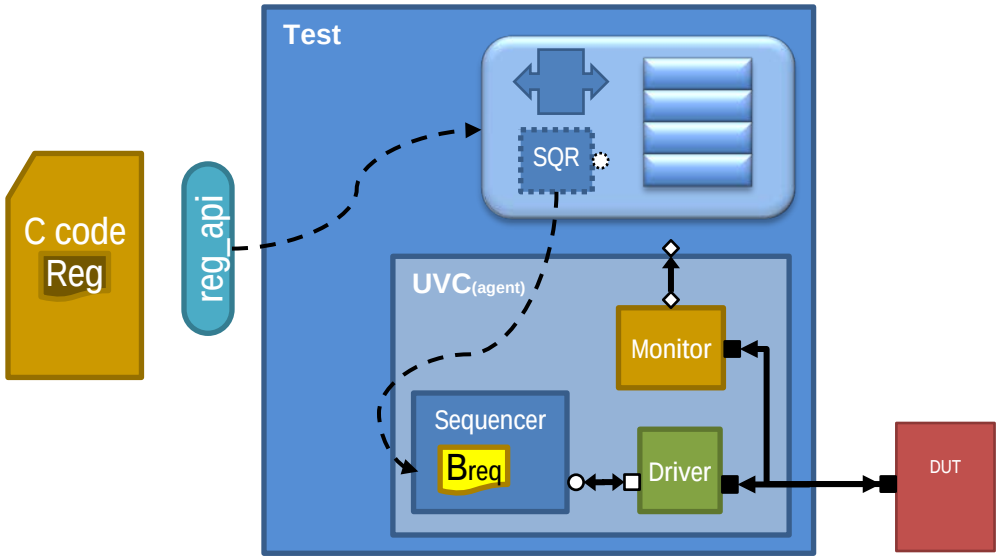
c_stimulus_pkg.sv

```
import "DPI-C" context task start_c_code();
```

Imported C routine called by SV

simple_test.svh

```
task simple_test::run_phase(uvm_phase phase);
  phase.raise_objection(this, "Test Started");
  set_c_stimulus_register_block(spi_rm);

  fork
    start_c_code();
    bckgrnd_seq.start(env.spi.seqr);
  join_any
  phase.drop_objection(this, "Test Ended");
endtask
```

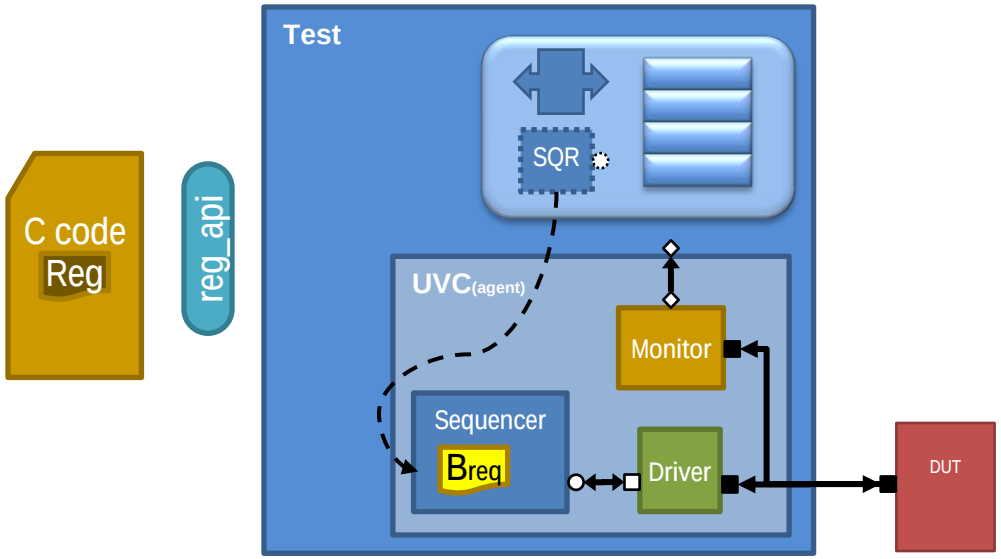


How It Works

c_stimulus_pkg.sv

```
task automatic c_reg_read(input int address, output int data);
  …
endtask
task automatic c_reg_write(input int address, input int data);
  uvm_reg_data_t reg_data;
  uvm_status_e status;
  uvm_reg write_reg;

  write_reg = get_register_from_address(address);
  if(write_reg == null) begin
    `uvm_error("c_reg_write", $sformatf("Register not found at address: %0h", address))
    return;
  end
  reg_data = data;
  if(interrupt_in_progress == 1) begin
    wait(interrupt_in_progress == 0);
  end
  write_reg.write(status, reg_data);
endtask: c_reg_write


export "DPI-C" task c_reg_write;
export "DPI-C" task c_reg_read;
export "DPI-C" task wait_1ns;
import "DPI-C" context task start_c_code();
```

Exported SV tasks called by C



Back to the Example

reg_api.c

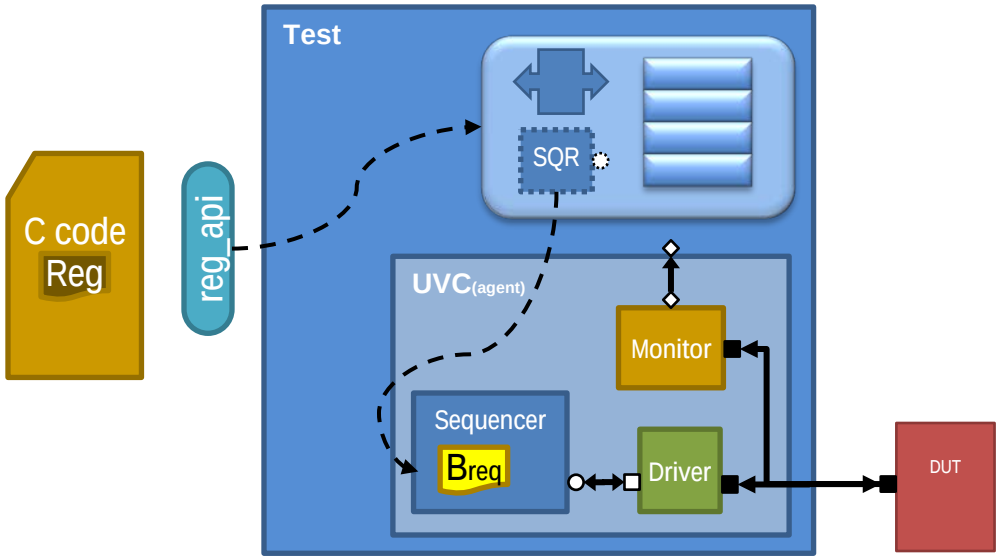
```
#include "reg_api.h"
int reg_read(int address) {
  int data;
  c_reg_read(address, &data);
  return data;
}

void reg_write(int address, int data) {
  c_reg_write(address, data);
}

void hw_wait_1ns(int n) {
  wait_1ns(n);
}

void register_thread() {
  svSetScope(svGetScopeFromName(
                    "c_stimulus_pkg"));
}
```

c_stimulus_pkg.sv

```
export "DPI-C" task c_reg_write;
export "DPI-C" task c_reg_read;
export "DPI-C" task wait_1ns;
import "DPI-C" context task start_c_code();
```

simple_test.c

```
#include "spi_regs.h"
#include "reg_api.h"

int simple_test_routine() {
  int control = 0x2c30;
  int status;

  reg_write(CTRL, control);
  status = reg_read(CTRL);
}
```



Side-by-Side

UVM Sequence

```
task body;
  super.body();
  repeat(10) begin
    spi_rm.rxtx0_reg.write(status, data_0,
                           .parent(this));
    spi_rm.rxtx1_reg.write(status, data_1,
                           .parent(this));
    spi_rm.ctrl_reg.write(status,
                          (control_config + 32'h100),
                          .parent(this));
    spi_rm.ctrl_reg.read(status, data, .parent(this));
    while(data[8] == 1) begin
      spi_rm.ctrl_reg.read(status, data,
                           .parent(this));
    end

    data_0++;
    data_1++;
  end

endtask
```

C Code

```
int send_10_test_routine() {

  while(i < 10) {
    reg_write(TX0, data_0);

    reg_write(TX1, data_1);

    reg_write(CTRL, (control + 0x100));


    status = reg_read(CTRL);
    while((status & 0x100) == 0x100) {
      status = reg_read(CTRL);

    }
    i++;
    data_0++;
    data_1++;
  }
  return 0;
}
```

Adding Additional C Test(s)

new_test.c

```
#include "spi_regs.h"
#include "reg_api.h"

int new_test_routine() {
  int control = 0x2c30;
  int status;

  register_thread();

  status = reg_read(CTRL);
  …
}
```

new_test.svh

```
task new_test::run_phase(uvm_phase phase);
  phase.raise_objection(this, "Test Started");
  set_c_stimulus_register_block(spi_rm);

  fork
    new_test_routine();
    bckgrnd_seq.start(env.spi.seqr);
  join_any
  phase.drop_objection(this, "Test Ended");
endtask
```

Call new test explicitly

Non-default tests must call register_thread()

c_test_pkg.sv

```
`include "new_test.c"
import "DPI-C" context task new_test_routine();
```

Import new C test with **context**

Handling Interrupts – isr_pkg

int_test.c

```
#include "spi_regs.h“
#include "reg_api.h"

int int_test_routine() {
  int control = 0x2c30;
  int status;

  reg_write(CTRL, control);
  status = reg_read(CTRL);
}

int spi_isr() {
  …
  status = reg_read(CTRL);
  reg_write(SS, 0x0);
  …
}

int start_c_code () {
  int_test_routine();
  return 0;
}

int start_isr() {
  spi_isr();
  return 0;
}
```

Does **NOT** call register_thread()

int_test.svh

```
task simple_test::run_phase(uvm_phase phase);
  phase.raise_objection(this, "Test Started");
  set_c_stimulus_register_block(spi_rm);

  fork
    start_c_code();
    bckgrnd_seq.start(env.spi.seqr);
    begin
      forever begin
        m_env_cfg.wait_for_interrupt();
        interrupt_service_routine();
      end
    end
  join_any
  phase.drop_objection(this, “Test Ended”);
endtask
```

isr_pkg.sv

```
import c_stimulus_pkg::*;
task interrupt_service_routine;
  interrupt_in_progress = 1;
  start_isr();
  interrupt_in_progress = 0;
endtask: interrupt_service_routine

import "DPI-C" context task start_isr();
```

Compilation and Simulation

- Compile the c_stimulus_pkg.sv file, and if required, the isr_pkg.sv

```
vlog $(C_PKG_HOME)/c_stimulus_pkg.sv -dpiheader sv_dpi.h
vlog $(C_PKG_HOME)/isr_pkg.sv -dpiheader sv_dpi.h
```

- Compile non-default C test routines in a test package:

```
vlog +incdir+$(TEST_PKG_HOME)
    $(TEST_PKG_HOME)/test_pkg.sv -dpiheader sv_dpi.h
```

- Compile the reg_api.c file

```
vlog +incdir+$(C_PKG_HOME) $(C_PKG_HOME)/reg_api.c
```

- Compile the C application c code

```
vlog +incdir+$(C_CODE_HOME) $(C_CODE_HOME)/my_c_code.c
     -ccflags -I$(C_PKG_HOME)
```

- Simulate

```
vsim top_tb +UVM_TESTNAME=spi_c_int_test
```

UVM C Stimulus Summary

- Provides a light-weight solution to allow C code to interact with the DUT
  - At the register level
- Allows for development of C code earlier in the project
- Lets software team contribute to test development
- Different from UVM Connect
  - Targeted only at allowing C code to communicate with Registers
    - Ideally, C code will be application-level
  - UVM Connect lets mixed-language components communicate
  - UVM Connect supports command-and-control across languages

Verification Academy—Provide the necessary skills that enable you to take advantage of today's latest advanced functional verification techniques.

Wide Variety of Topics

- *Verification Planning*
- *FPGA Verification*
- ***Metrics for SoC Verification***
- *ABV*
- *Basic and Advanced UVM/OVM*
- *CDC*
- *UVM Express*
- ***UVM Connect***
- ***ADMS***
- *Intelligent Testbench Automation*
- *SystemVerilog Testbench Acceleration*

Verification Academy—Provide the skills that enable you to take advanta latest advanced functional verificatior

Wide Variety of Topics

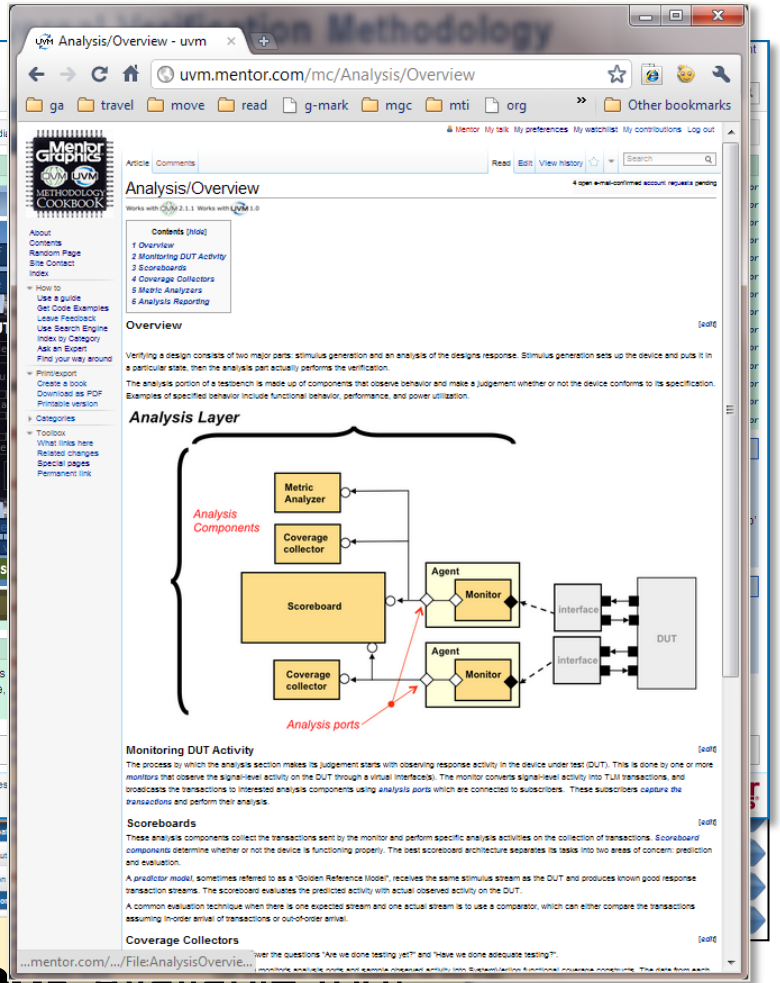
- *Verification Planning* •*FPGA Verificatior*
- ***Metrics for SoC Verification*** •*ABV*
- *Basic and Advanced UVM/OVM* •*CDC*
- *UVM Express* •***UVM Connect*** •***ADMS***
- *Intelligent Testbench Automation*
- *SystemVerilog Testbench Acceleration*

UVM/OVM Online Methodology Cookbook Available too!

Verification Academy
skills that enable you to
latest advanced functio

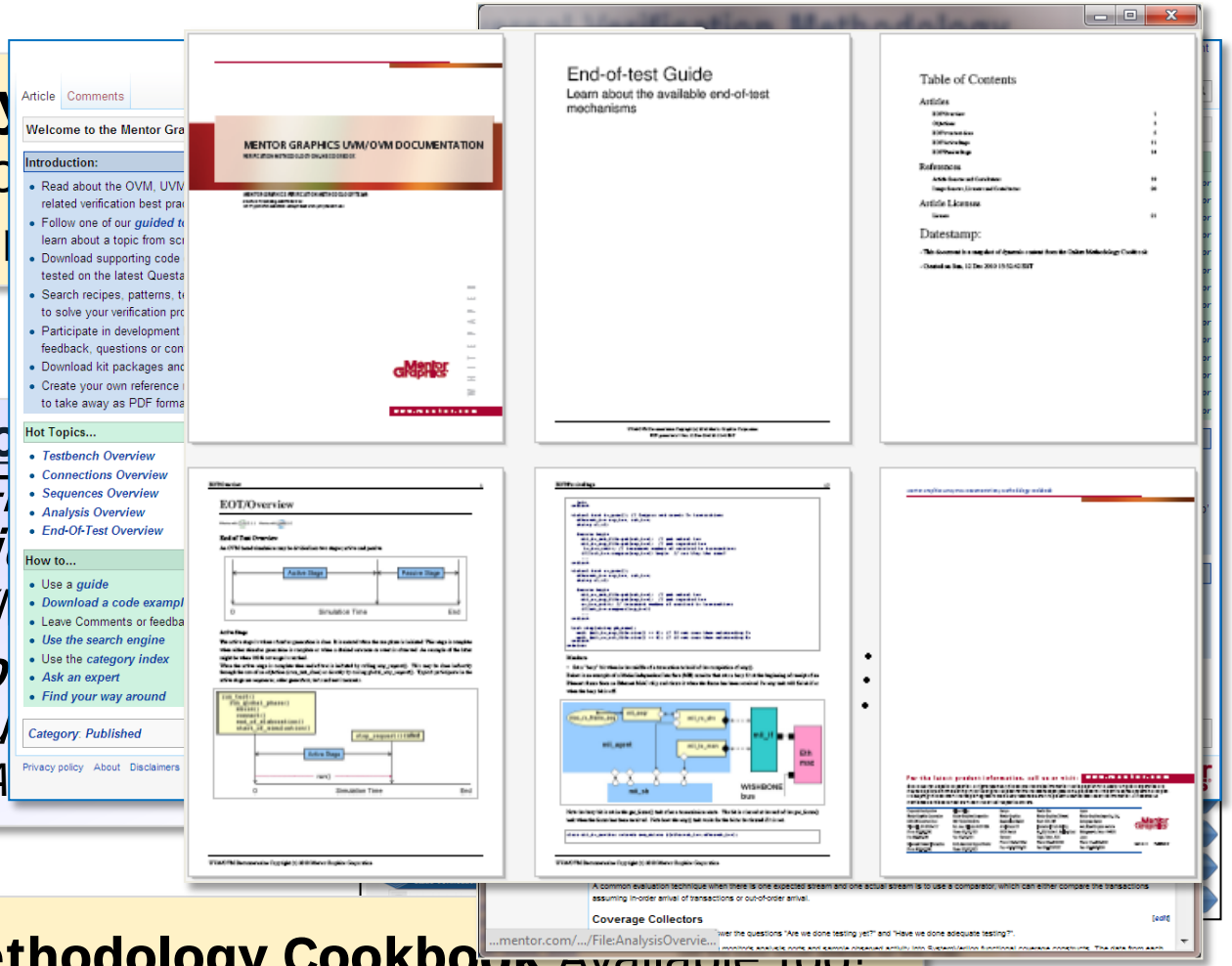
Wide Variety c

- *Verification Planning* • *F*
- ***Metrics for SoC Verifi***
- *Basic and Advanced UVI*
- *UVM Express* • ***UVM Co***
- *Intelligent Testbench Au*
- *SystemVerilog Testbench A*



UVM/OVM Online Methodology Cookbook Available too!

Verification Academy
skills that enable you to
latest advanced functio



Wide Variety c

- *Verification Planning* • *F*
- ***Metrics for SoC Verifi***
- *Basic and Advanced UVI*
- *UVM Express* • ***UVM Co***
- *Intelligent Testbench Au*
- *SystemVerilog Testbench A*

UVM/OVM Online Methodology Cookbook Available too!

www.mentor.com