AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Automatic Verification Using Specta-AV
(A Boost to Verification Productivity)

Neena Chandawale
(Host)

Nikhil Arora
(Presenter)

Agenda

- Introduction to Design Verification and Challenges
- Specta-AV : A Complete Verification System
- Overview of Specta-AV Verification Environment
  - Register RTL Block
  - Register Model
  - Verification Components (Model Updater, Functional Checkers, Coverage Collector…)
  - Register Sequences
  - Custom Sequences
  - VIP Sequences
  - Specta-AV Library
- A Simple Example (Multiplier and Accumulator)
- Benefits of Specta-AV
- Q&A

Introduction to Design Verification

- Intent is to verify that the design meets the system requirements and specifications.

- Approaches to design verification consists of:
  - Logic simulation : Detailed functionality and timing of design is checked
  - Functional verification : Functional models describing the functionality of the design are developed to check against the behavioral specification
  - Formal verification : Functionality is checked against a golden model

- Verification done to check the correctness of protocols and interfaces

- Comprehensive tests are used for increasing test coverage

- Verification is one of the biggest challenges in the design of modern system on chips (SoCs) and reusable IP blocks

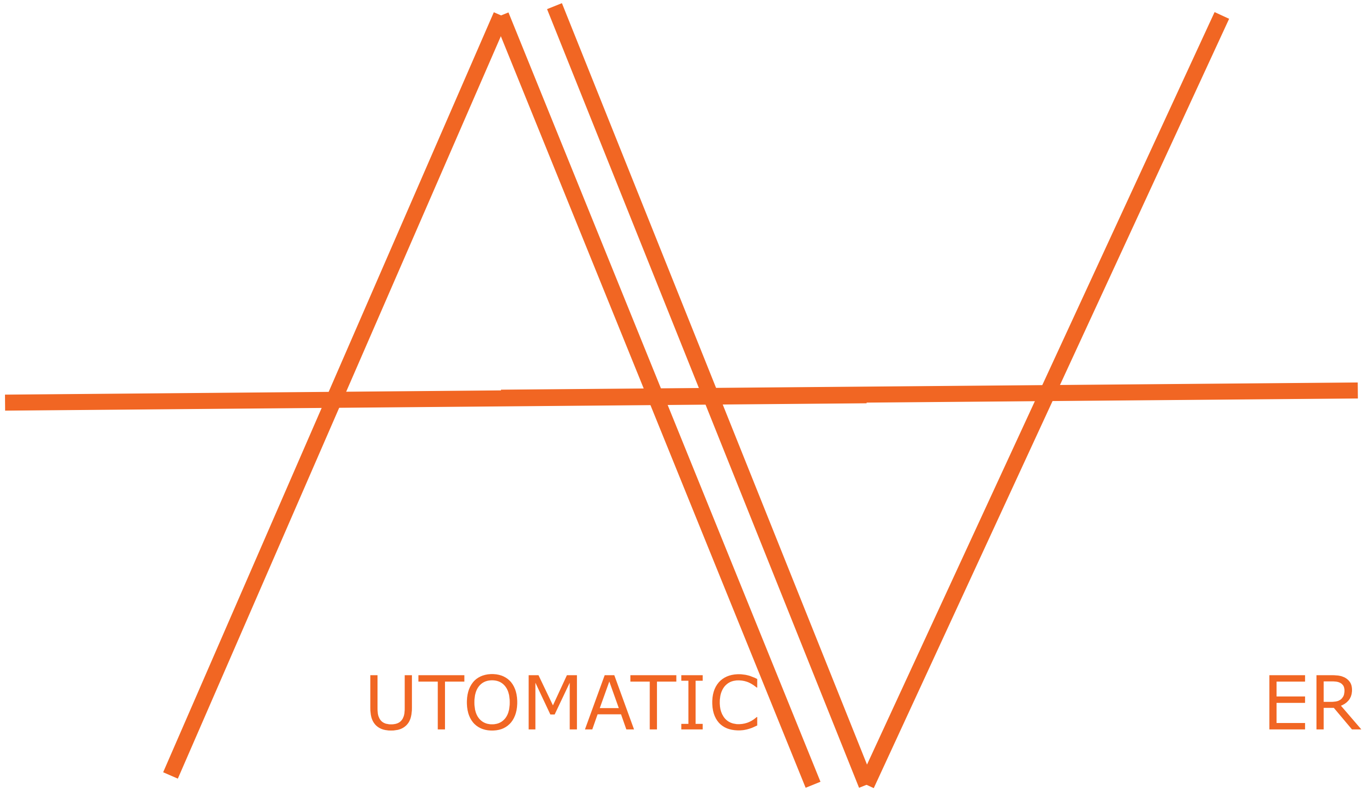
The Verification Challenge

- **The high cost of verification**
  - 70% of development effort is verification
  - Tools, simulation licenses, cost of compute power
- **The difficulty of verification**
  - Lack of qualified, experienced verification resources
  - Lot of manual work translating designer intent into test code – error prone, tedious, not good use of time
- **Horizontal reuse**
  - Verification is not the end, there is also firmware, prototype, and validation that is needed
  - Typically these teams don't share code
  - Different environment, language, focus, …
- **Vertical reuse**
  - Test sequences, register specification created at block or IP level can be run from subsystem or system level
    - Changes in bus protocol
    - Differences in configuration
    - Differences in the way transaction are carried out

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Specta-AV : A Complete Automated Verification System

SPECTA



UTOMATIC ERIFICATION

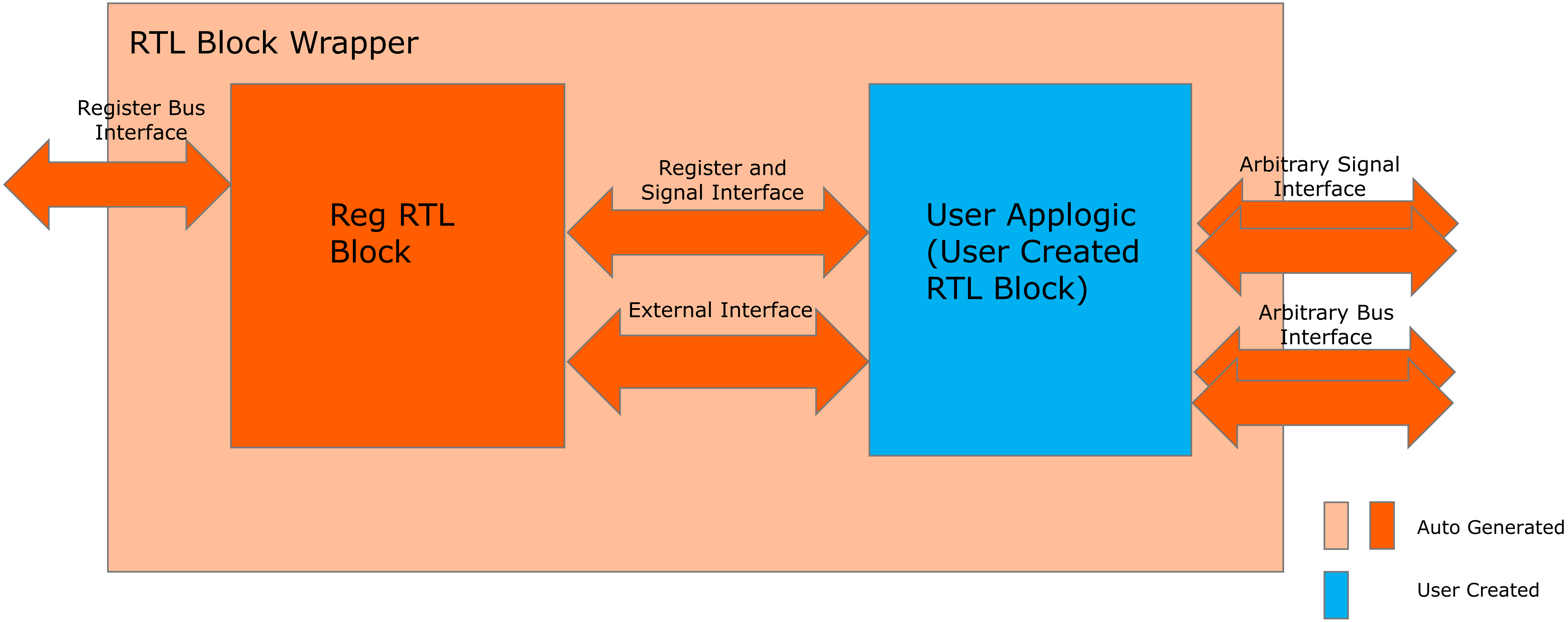
What Specta-AV Brings to the Table

- **Generates the design components**
  - IP register logic design
  - IP wrapper module
- **Generates UVM verification Environment**
  - Complete UVM infrastructure (Regmodel, agents, coverage collector, …)
  - Support for multiple buses such as AXI4Full, AXILite, AHB, APB, Avalon
  - Run standard UVM tests
  - Auto mirroring of registers
- **Generates sequences for registers**
  - Covers simple registers
  - Additional sequences for special registers
- **Generates custom sequences**
  - From Excel or Python spec based on IP functionality
- **Generates custom checks**
  - From Excel or Python spec based on IP functionality

Typical Block in an SoC

The Canonical Block

- Typical Block in an SoC



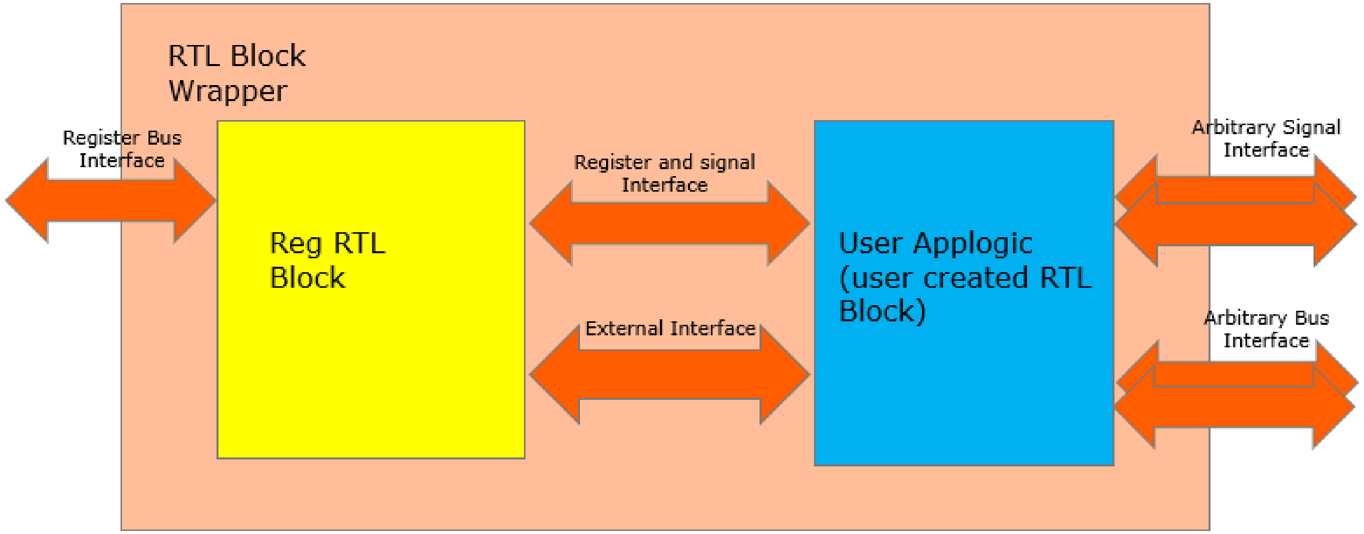
Reg RTL Block

- At least 25% of time is spent on addressable register related tasks

- Register seems so simple and yet a typical chip may have hundreds or even thousands of registers

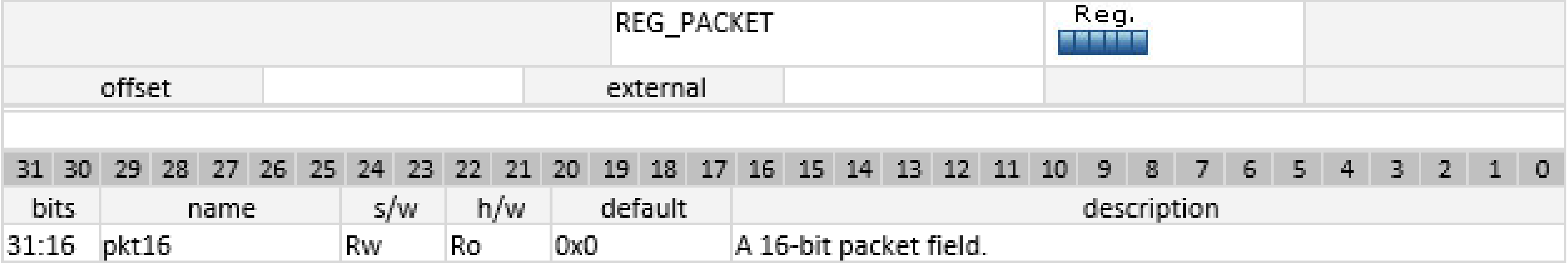
- A single register may be:
  - described in Word
  - documented in HTML
  - designed in Verilog
  - verified in UVM

- A single change must be translated in all formats


RTL Block Wrapper
Register Bus Interface
Reg RTL Block
Register and signal Interface
External Interface
User Applogic (user created RTL Block)
Arbitrary Signal Interface
Arbitrary Bus Interface

Specifying Registers in Specta-AV

Define your register's settings



		REG_PACKET		Reg.	
offset		external			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

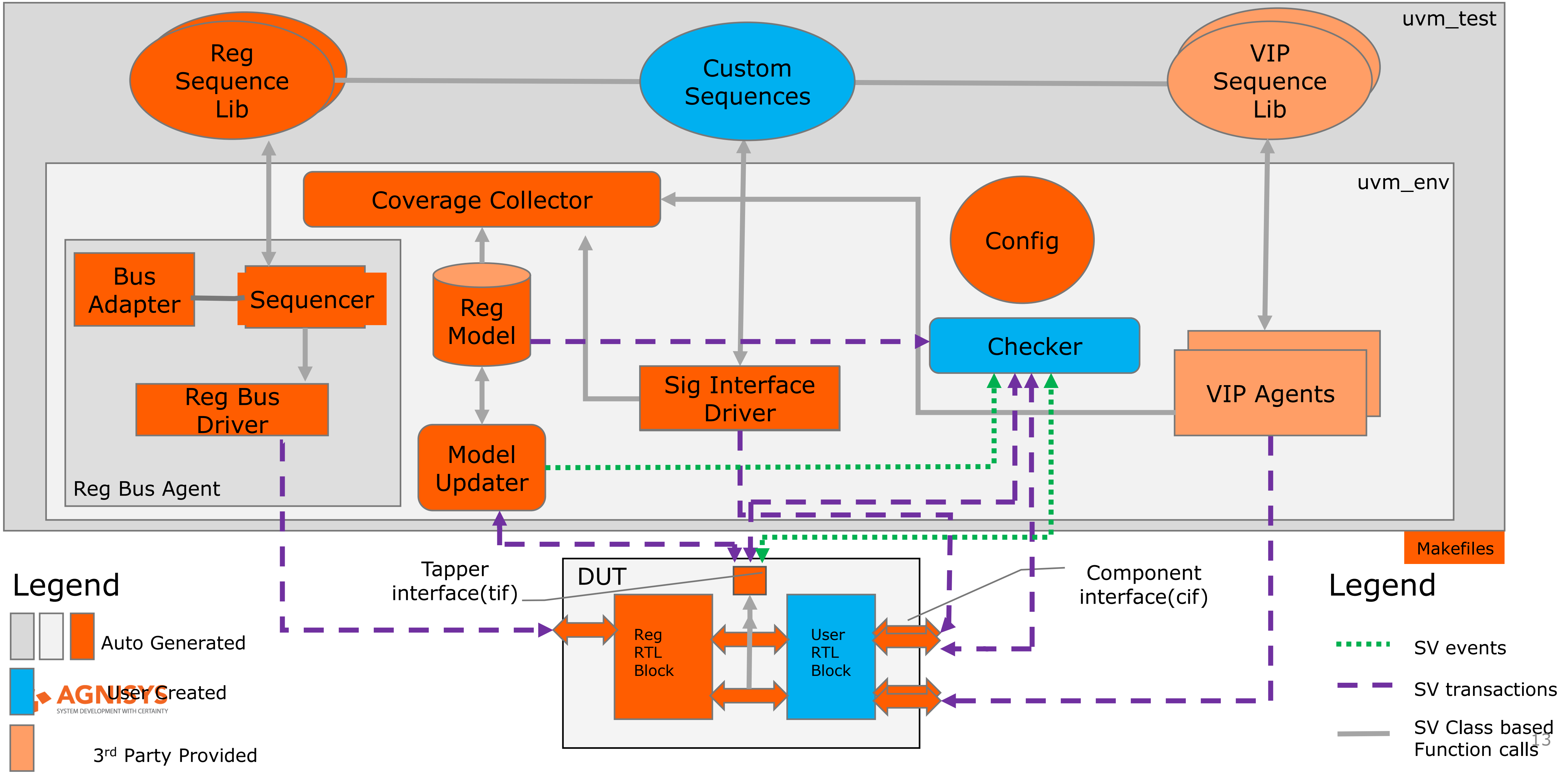
bits	name	s/w	h/w	default	description
31:16	pkt16	Rw	Ro	0x0	A 16-bit packet field.

Verilog, UVM, SystemVerilog, HTML, IP-Xact, SystemRDL, Register Sequences…

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Overview of Specta-AV Verification Environment

Specta-AV Generated UVM Testbench



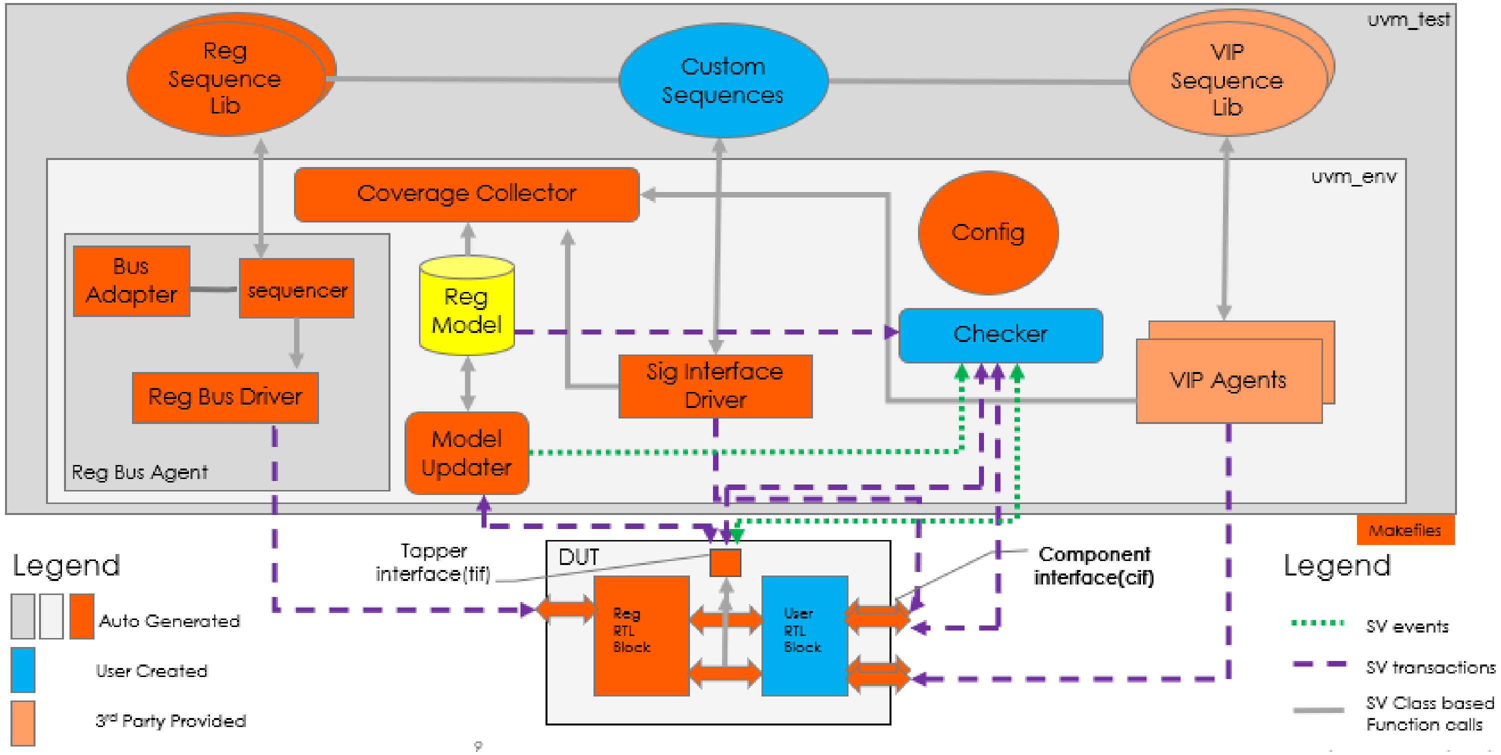
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Specta-AV RegModel

- UVM provides the best framework to achieve coverage-driven verification

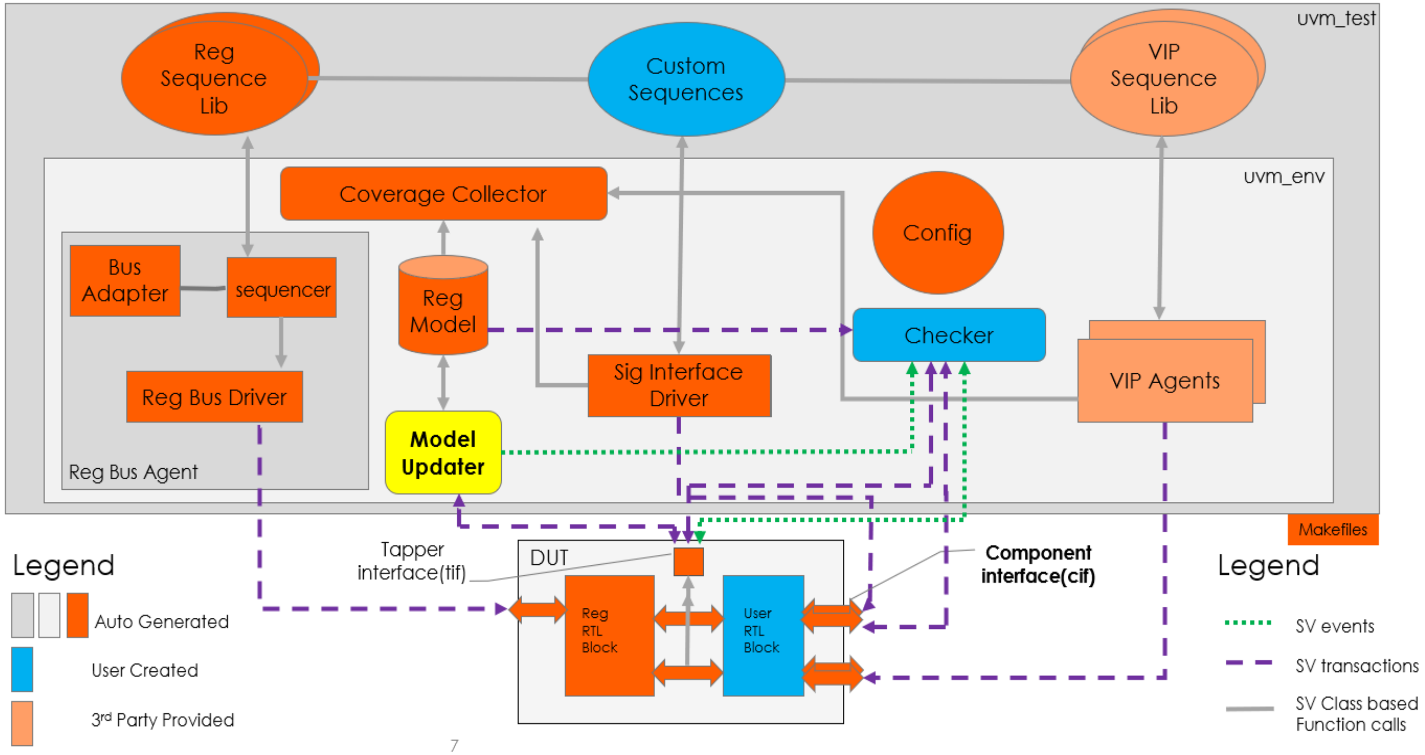
- A register model is an instance of a register block, which may contain any number of registers, register files, memories, and other blocks

- Specta-AV through Register Specification generates Regmodel automatically



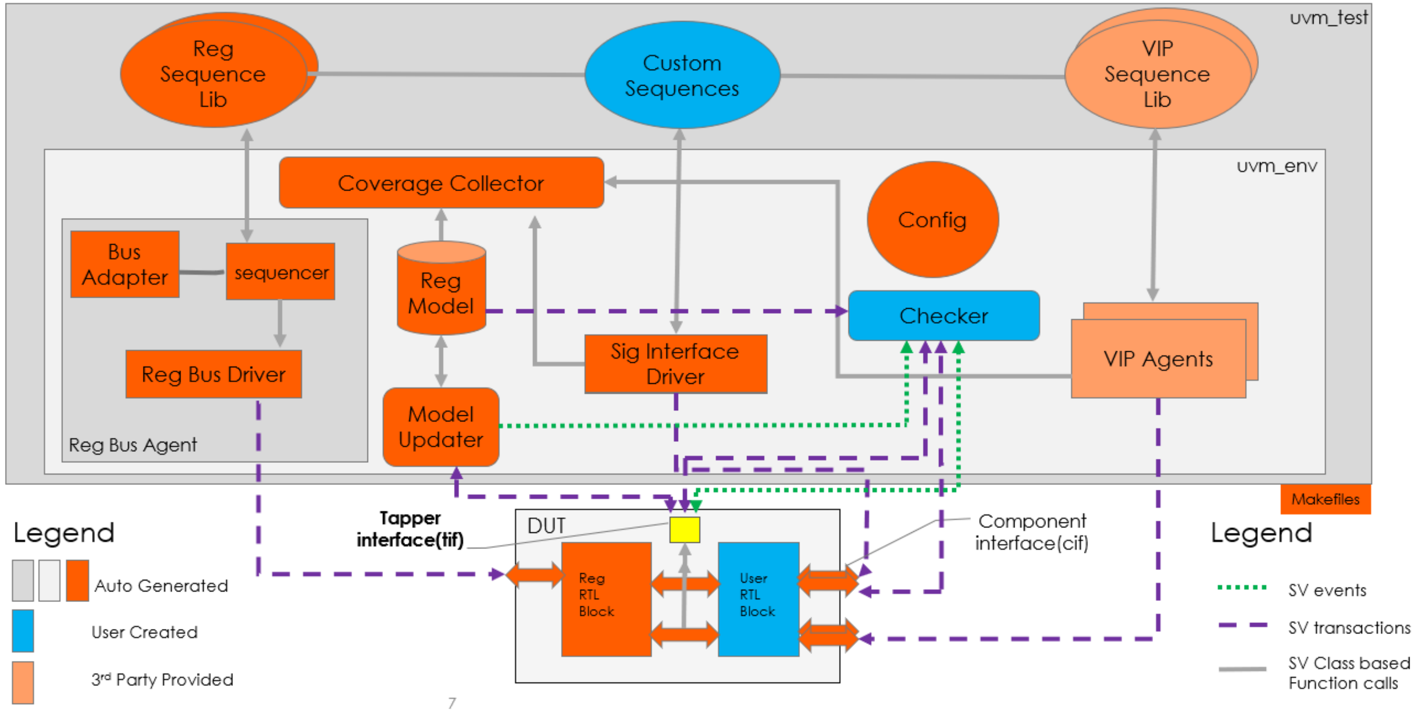
Model Updater

- Updates the UVM Register Field with the value when a hardware event occurs on it
- The required value is tapped from inside the HW Register interface through a tapper interface
- Also used to generate the HW write events



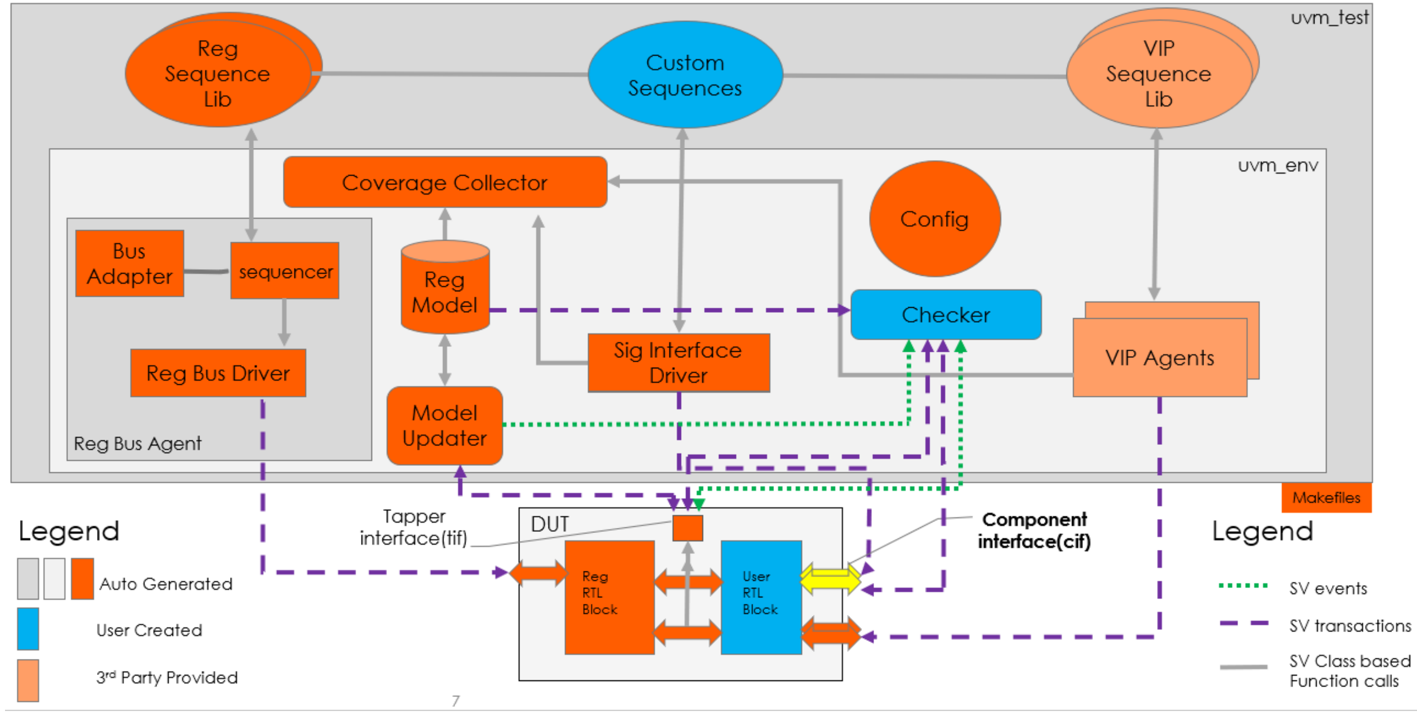
Tapper Interface

- Taps the required value from inside the HW register interface
- Model Updater gets the HW interface value from tapper interface to update the Regmodel on HW write
- Based on tapped values, model updater generates HW write events
- Tapped values are also used inside checkers to apply appropriate checks



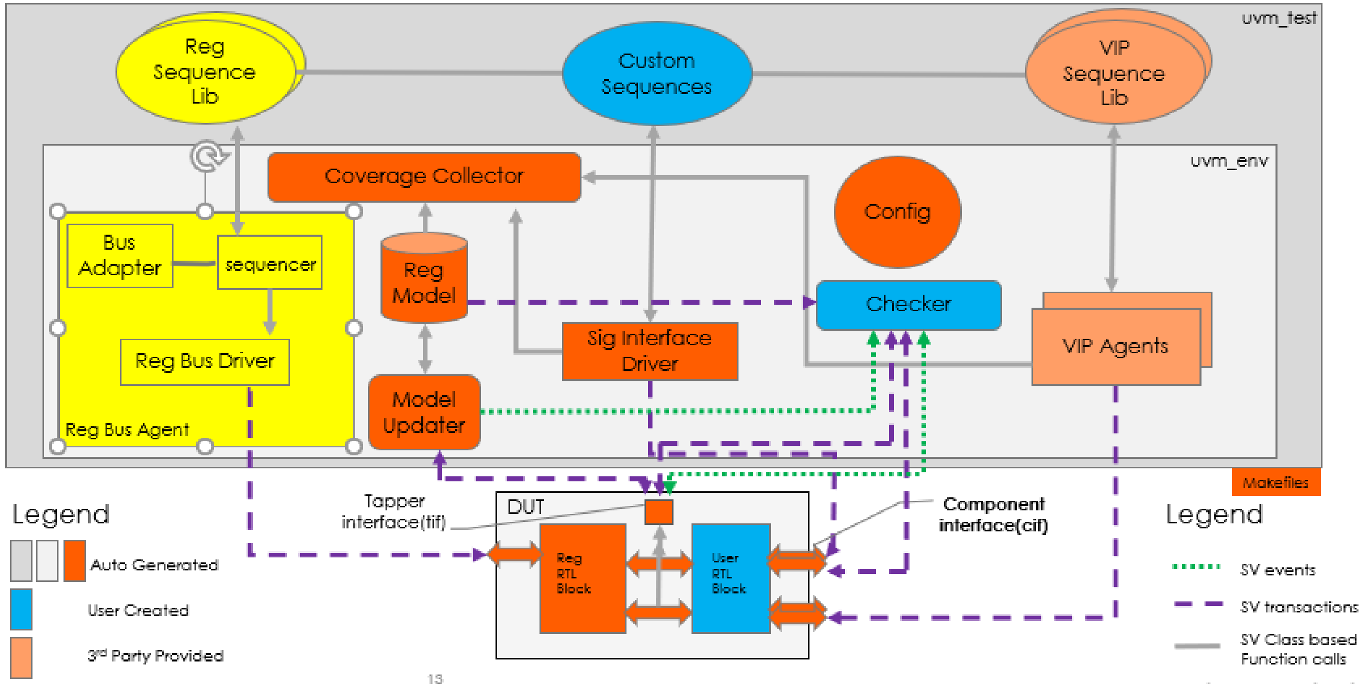
Component Interface

- HW interface between user defined logic and the wrapper block
- Contains pins with which wrapper block communicates with the outside world
- Pins can be used inside checker to monitor correct data flow
- Pins be driven through custom sequences



Automated Register Verification Using Specta-AV

- Automation that provides 100% coverage is key to verification success as IPs and SoCs grow in complexity
- Based on the register specification, Specta-AV generates the complete UVM testbench: bus agents, drivers, adaptors, sequencers and sequences, as well as the Makefiles for all major simulators.
- The generated UVM testbench is fully connected to the UVM Regmodel and DUT, providing you with a push-button verification
- Generates 100% functional coverage out of the box with register-focused cover groups
- Significantly reduces verification cycles
- Generates sequences for special registers: Lock, Shadow, Alias and Interrupts Registers

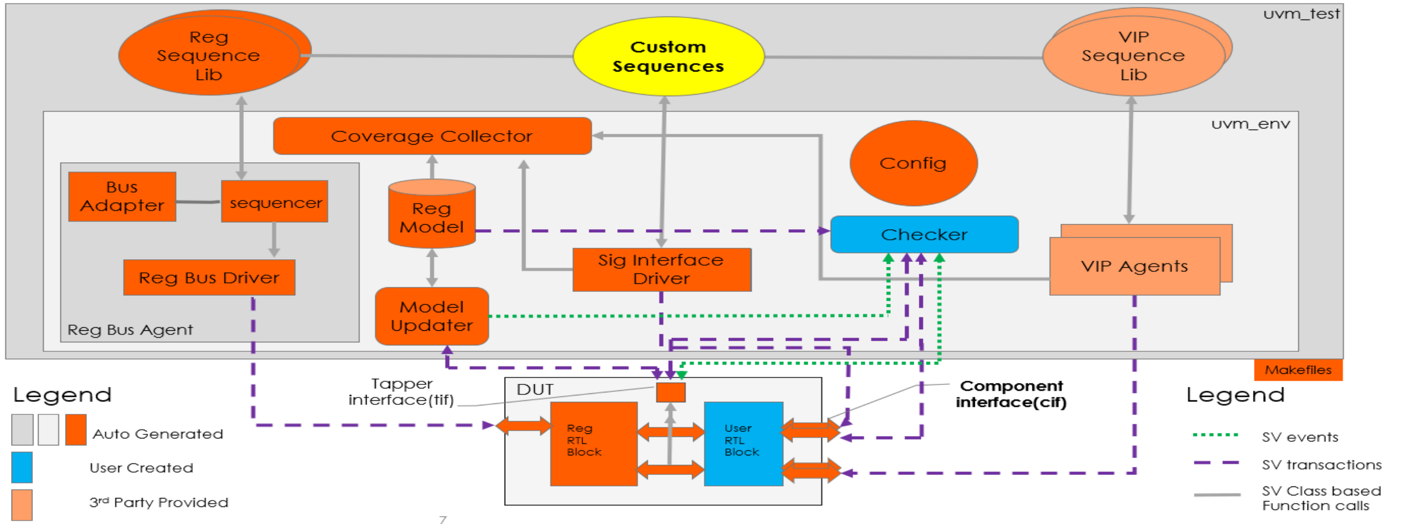


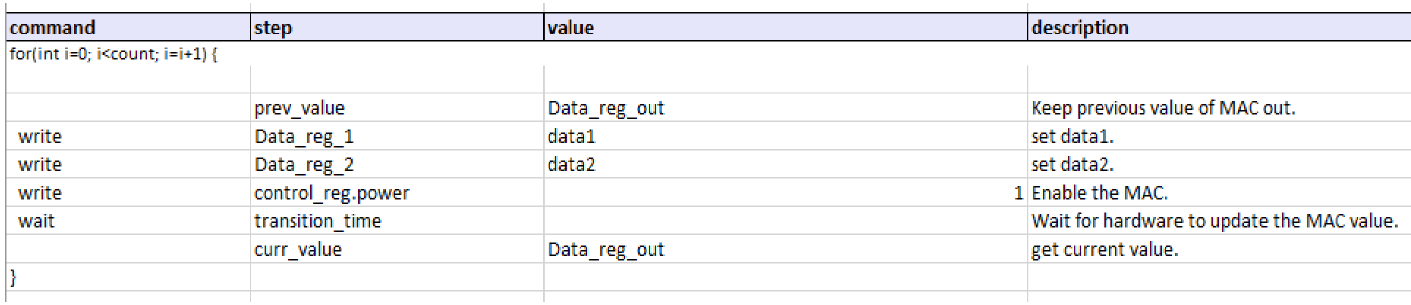
Specta-AV Generated Register Sequences

- Positive and negative test for register access
  - RO, WO, RW
- Positive and negative tests for all field level access with additional sequences for side-effects
  - RO, WO, RW, RC, RS, WS, WC, W1C, W0C, W1S, W0S, W1T, W0T, WRS, WRC, WSRC, WCRS, W1SRC, W1CRS, W0SRC, W0CRS
- Special register tests
  - Lock and unlock register sequences
  - Sequence to write to a register and read from all its aliases
  - Writing to register and reading from its shadow
  - Sequence for indirect accessing an array of registers through a set of data-index register
  - Constraints on register fields
  - FIFO
- Tests for memory
  - Quadrant based testing, includes access tests
- Multiple bus domains
- Check for holes in the address map
- Buses supported : ARM AXI-Full, AXI-Lite, AHB, AHB-Lite, APB, Avalon, Proprietary

Custom Sequences

- Abstract register read/writes with automatic read-modify-write
- Waiting on time, field values, or signal events
- Access to Component Interface (cif) that enables driving/sampling values on it
- Creating hierarchical sequences
- Creation and use of structures
- Parallelism using fork-join constructs
- Supports looping and conditional constructs such as if-else, for. and while
- Randomization and constraints



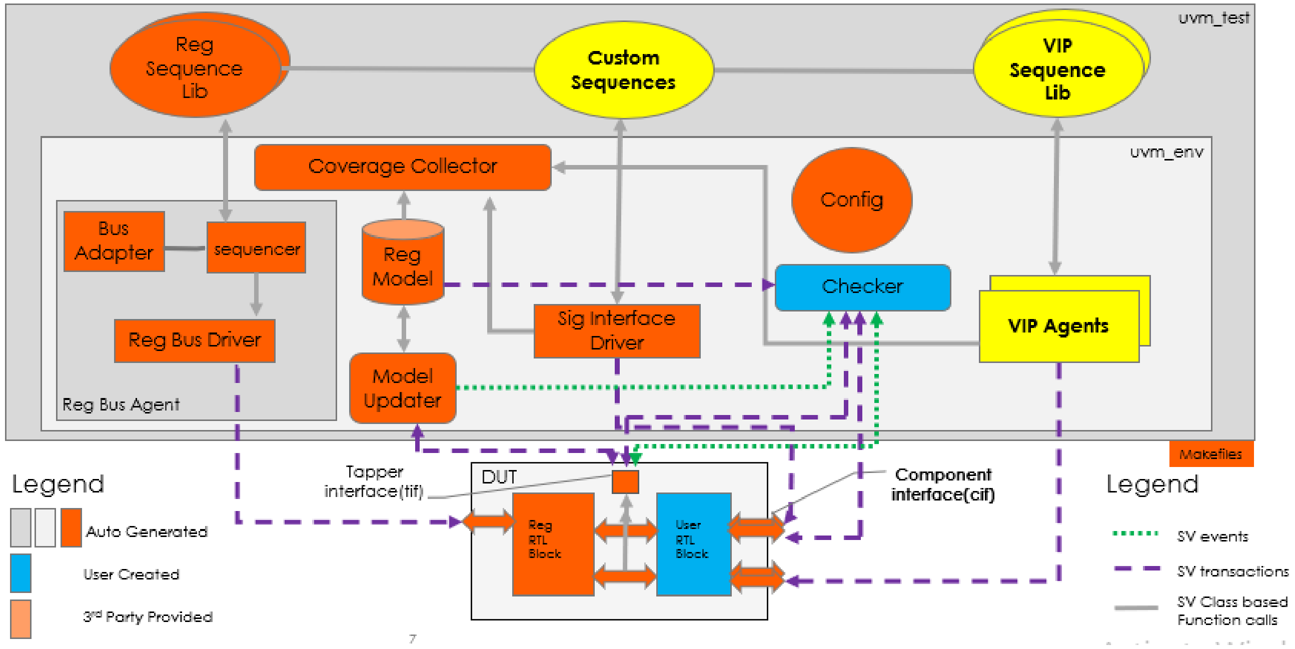


command	step	value	description
for(int i=0; i<count; i=i+1) {			
	prev_value	Data_reg_out	Keep previous value of MAC out.
write	Data_reg_1	data1	set data1.
write	Data_reg_2	data2	set data2.
write	control_reg.power	1	Enable the MAC.
wait	transition_time		Wait for hardware to update the MAC value.
	curr_value	Data_reg_out	get current value.
}			

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Custom Sequences For VIP

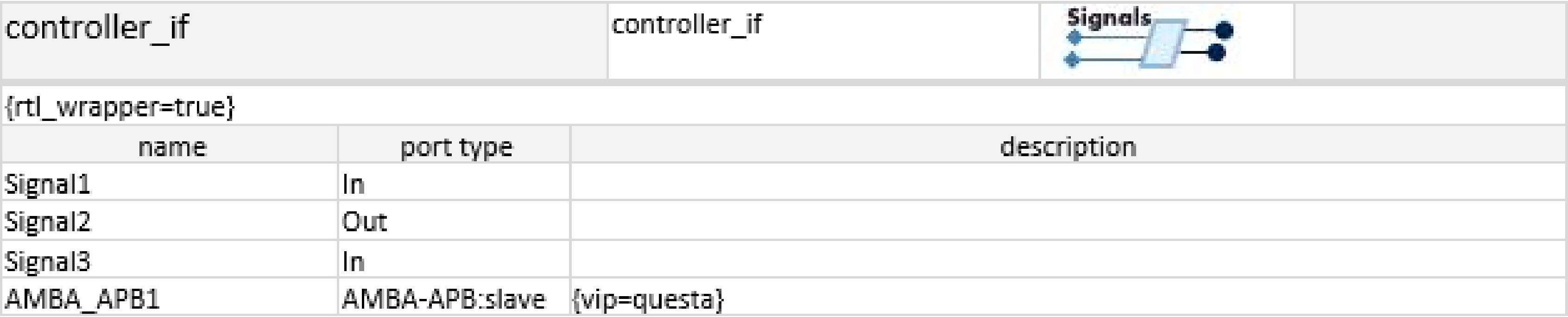
- Abstract register read/write APIs
- Waiting on time, field values or signal events
- Creating hierarchical sequences
- Parallelism using fork-join constructs
- Supports looping and conditional constructs such as if-else, for, and while
- Randomization and constraints



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

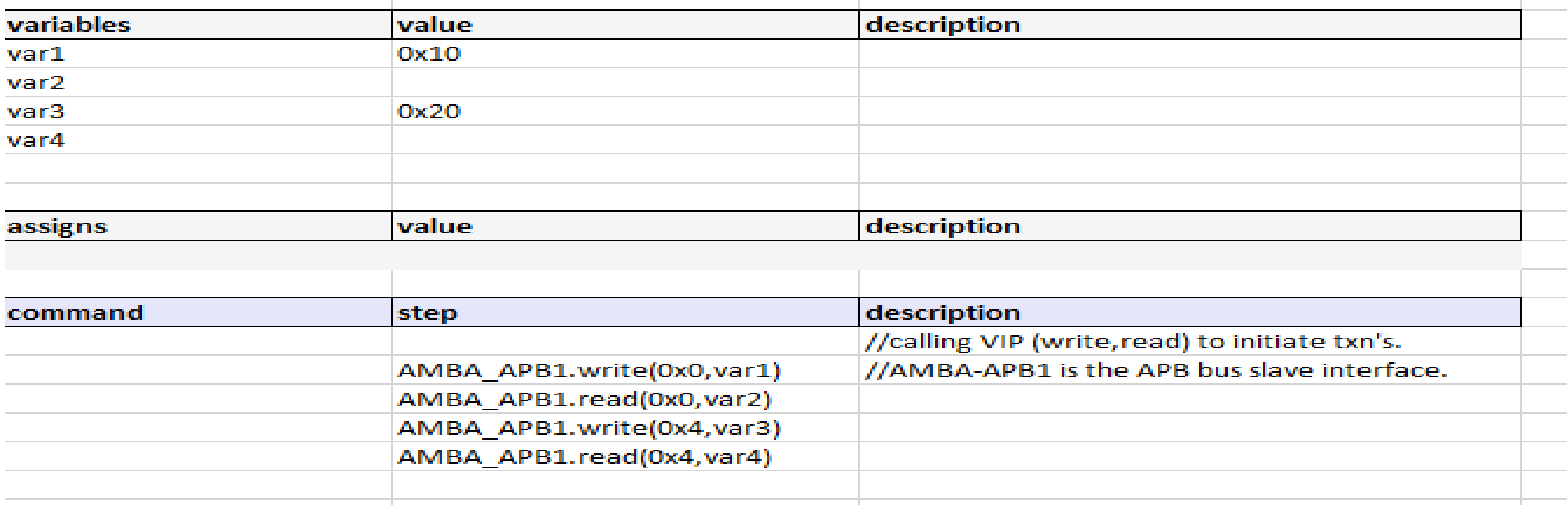
Specification

- In the register specification, user can specify the VIP along with the slave interface



controller_if	controller_if	Signals	
{rtl_wrapper=true}			
name	port type	description	
Signal1	In		
Signal2	Out		
Signal3	In		
AMBA_APB1	AMBA-APB:slave	{vip=questa}	

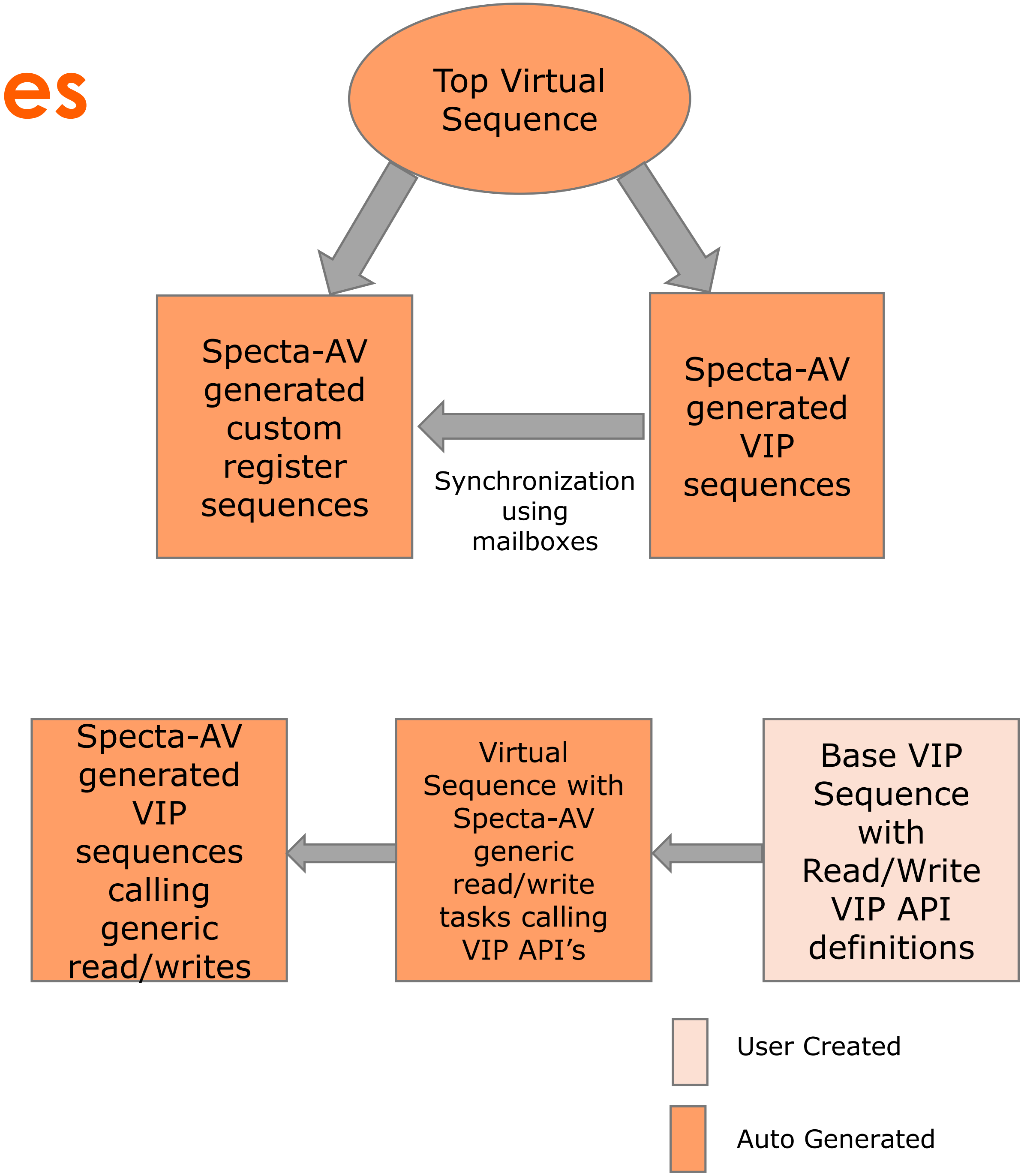
- In the sequence specification, then user can write custom sequences by calling VIP read/write APIs



variables	value	description
var1	0x10	
var2		
var3	0x20	
var4		
assigns	**value**	**description**
command	**step**	**description**
		//calling VIP (write,read) to initiate txn's.
	AMBA_APB1.write(0x0,var1)	//AMBA-APB1 is the APB bus slave interface.
	AMBA_APB1.read(0x0,var2)	
	AMBA_APB1.write(0x4,var3)	
	AMBA_APB1.read(0x4,var4)	

Layered Virtual Sequences

- Allows abstraction of read/write APIs
- Provides a generic API for VIPs from multiple vendors
- Parallelism supports inside top level virtual sequence
- Enables better synchronizations between multiple sequences
- UVM event pool based or mailbox-based synchronization



Top Virtual Sequence

Specta-AV generated custom register sequences

Specta-AV generated VIP sequences

Synchronization using mailboxes

Specta-AV generated VIP sequences calling generic read/writes

Virtual Sequence with Specta-AV generic read/write tasks calling VIP API's

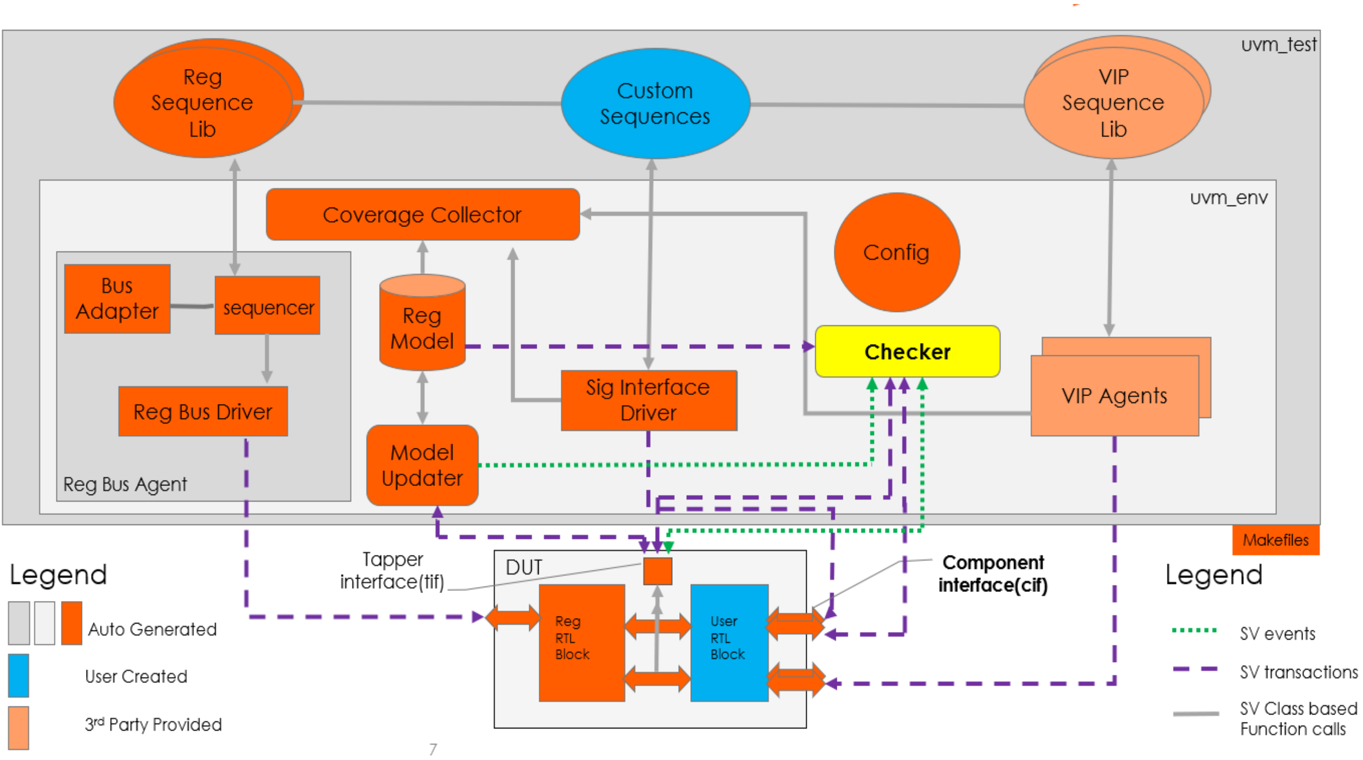
Base VIP Sequence with Read/Write VIP API definitions

□ User Created

■ Auto Generated

Functional Checkers

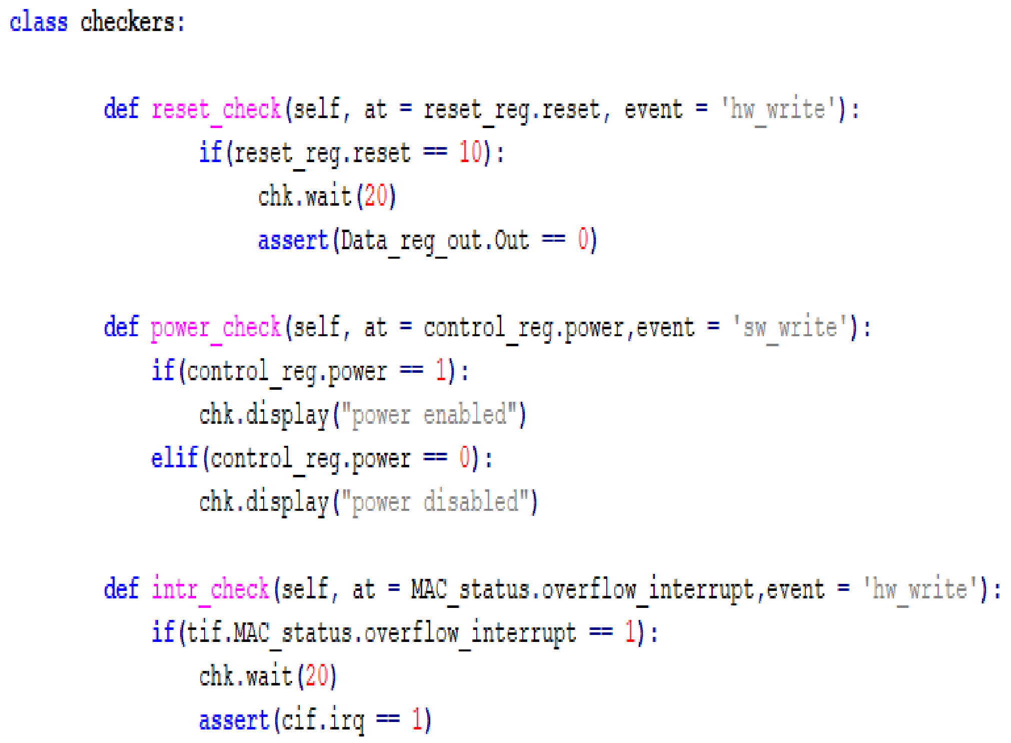
- Access to field callbacks, tapper interface (tif), and component interface (cif) to enable:
  - Bus write/read events
    - @sw_write @sw_read
  - Field hardware update events
    - @hw_write
  - Special RTL events
    - @pulse (rtl.hw_w1p)
    - @intr   (intr / halt)
    - @underflow @overflow (counter)
  - Events on Component interface (cif) signals
- Waiting on time and events
- Asserts construct to check functional correctness
- Parallelism using fork-join constructs
- Supports conditional constructs such as if-else
- Checkers can be captured using MS-Excel or they can be text based in Python



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

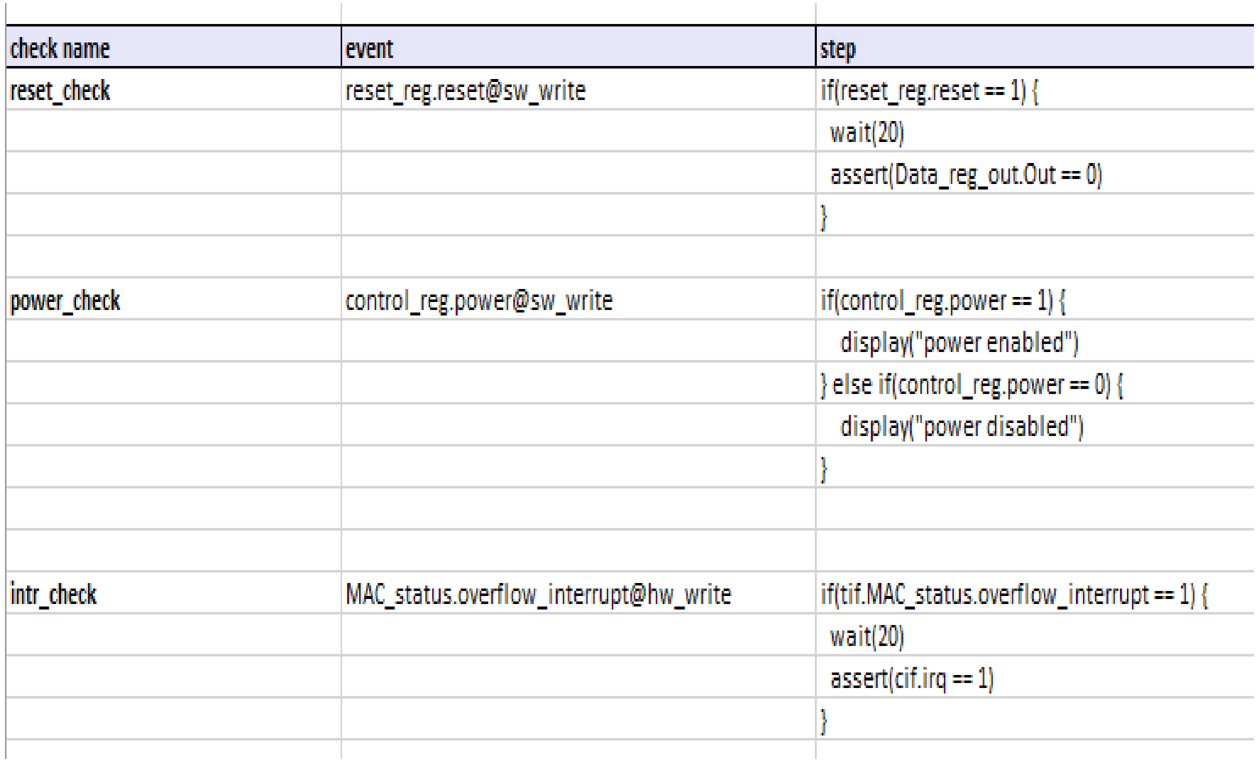
Functional Checkers

- **Python Checker**



```
class checkers:

        def reset_check(self, at = reset_reg.reset, event = 'hw_write'):
                if(reset_reg.reset == 10):
                     chk.wait(20)
                     assert(Data_reg_out.Out == 0)

        def power_check(self, at = control_reg.power,event = 'sw_write'):
            if(control_reg.power == 1):
                chk.display("power enabled")
            elif(control_reg.power == 0):
                chk.display("power disabled")

        def intr_check(self, at = MAC_status.overflow_interrupt,event = 'hw_write'):
            if(tif.MAC_status.overflow_interrupt == 1):
                chk.wait(20)
                assert(cif.irq == 1)
```

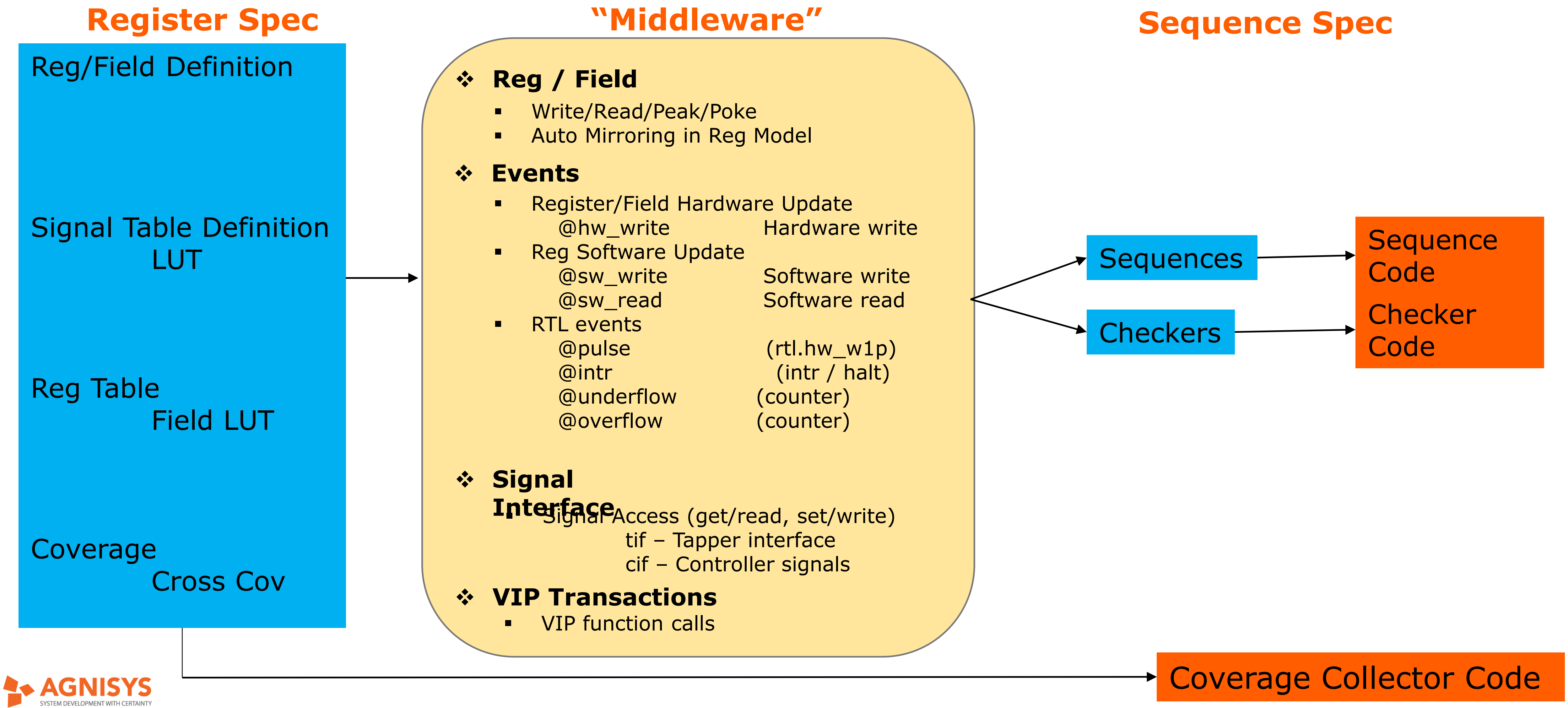
- **Excel Checker**



check name	event	step
reset_check	reset_reg.reset@sw_write	if(reset_reg.reset == 1) {
		wait(20)
		assert(Data_reg_out.Out == 0)
		}
power_check	control_reg.power@sw_write	if(control_reg.power == 1) {
		display("power enabled")
		} else if(control_reg.power == 0) {
		display("power disabled")
		}
intr_check	MAC_status.overflow_interrupt@hw_write	if(tif.MAC_status.overflow_interrupt == 1) {
		wait(20)
		assert(cif.irq == 1)
		}

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Using “Middleware” to Create Sequences and Checkers



Register Spec

Reg/Field Definition

Signal Table Definition
LUT

Reg Table
Field LUT

Coverage
Cross Cov

“Middleware”

- **Reg / Field**
  - Write/Read/Peak/Poke
  - Auto Mirroring in Reg Model
- **Events**
  - Register/Field Hardware Update
    - @hw_write Hardware write
  - Reg Software Update
    - @sw_write Software write
    - @sw_read Software read
  - RTL events
    - @pulse (rtl.hw_w1p)
    - @intr (intr / halt)
    - @underflow (counter)
    - @overflow (counter)
- **Signal Interface**
  - Signal Access (get/read, set/write)
    - tif – Tapper interface
    - cif – Controller signals
- **VIP Transactions**
  - VIP function calls

Sequence Spec

Sequences

Checkers

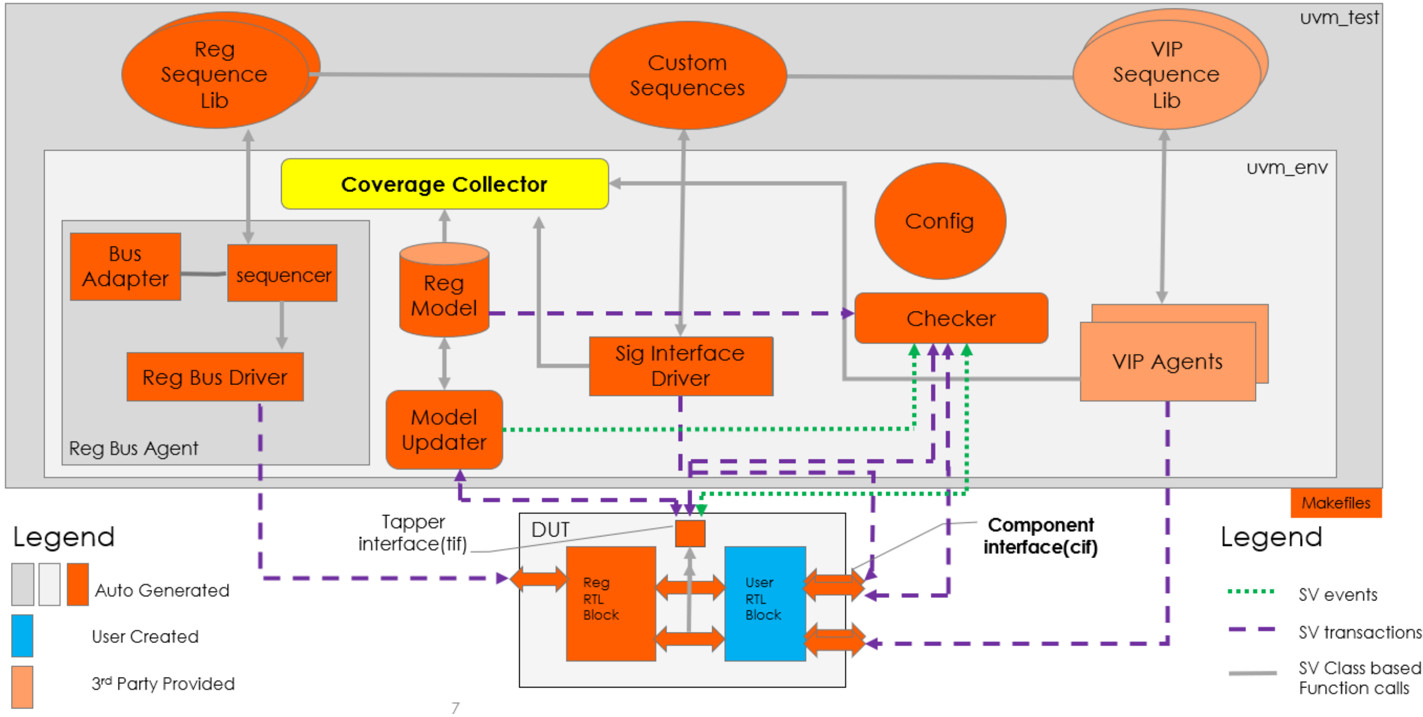
Sequence Code

Checker Code

Coverage Collector Code

Coverage Collector

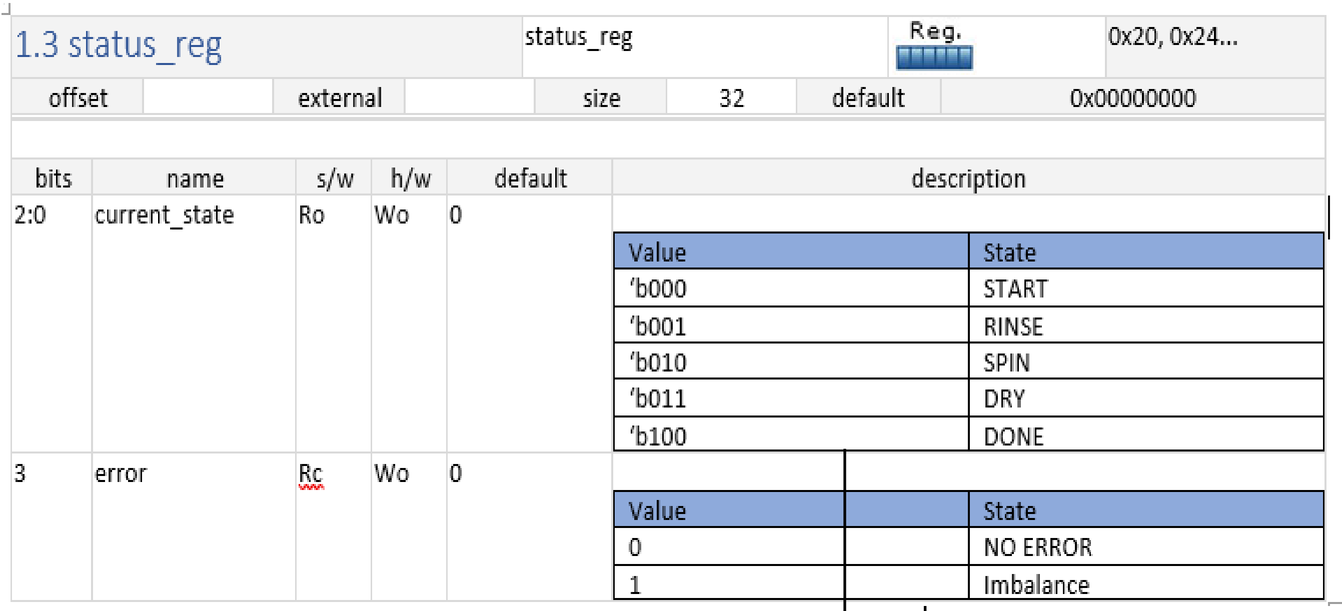
- Specta-AV automatically translates the LUT provided in the Register Specification into cover points
- These implicit cover points are specified in the Coverage Collector
- As LUTs contains important functional details of the design, creating cover points for their values help ascertain that all possible values are covered in the verification run
- Cross coverage is also supported in Specta-AV using “cross” property



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Coverage Collector Using LUTs

Coverage Code



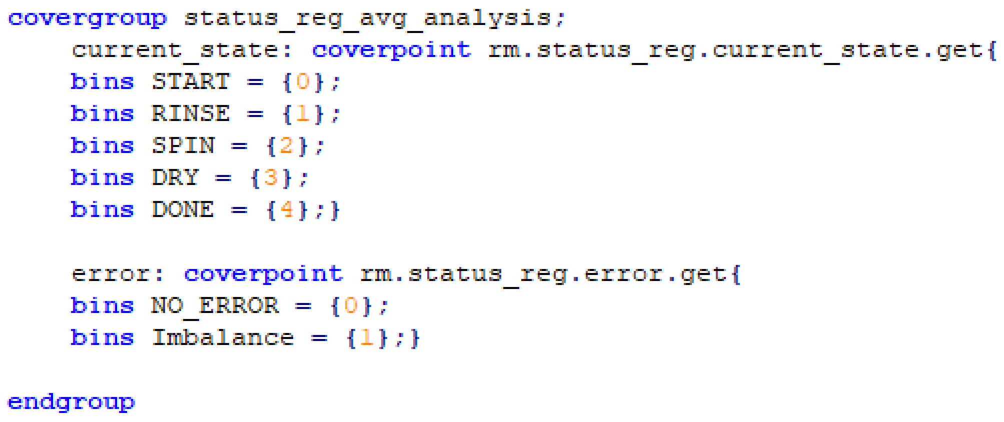
1.3 status_reg	status_reg	Reg.	0x20, 0x24...

offset		external		size	32	default	0x00000000

bits	name	s/w	h/w	default	description
2:0	current_state	Ro	Wo	0	
3	error	Rc	Wo	0	

Value	State
‘b000	START
‘b001	RINSE
‘b010	SPIN
‘b011	DRY
‘b100	DONE

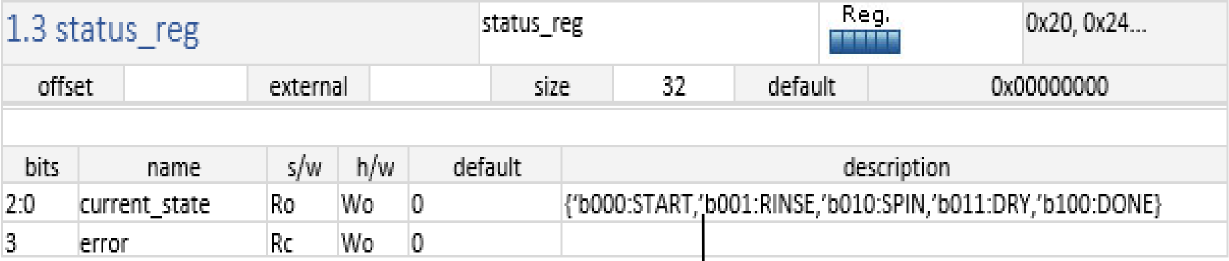
Value	State
0	NO ERROR
1	Imbalance



```
covergroup status_reg_avg_analysis;
    current_state: coverpoint rm.status_reg.current_state.get{
    bins START = {0};
    bins RINSE = {1};
    bins SPIN = {2};
    bins DRY = {3};
    bins DONE = {4};}

    error: coverpoint rm.status_reg.error.get{
    bins NO_ERROR = {0};
    bins Imbalance = {1};}

endgroup
```

Look Up Tables

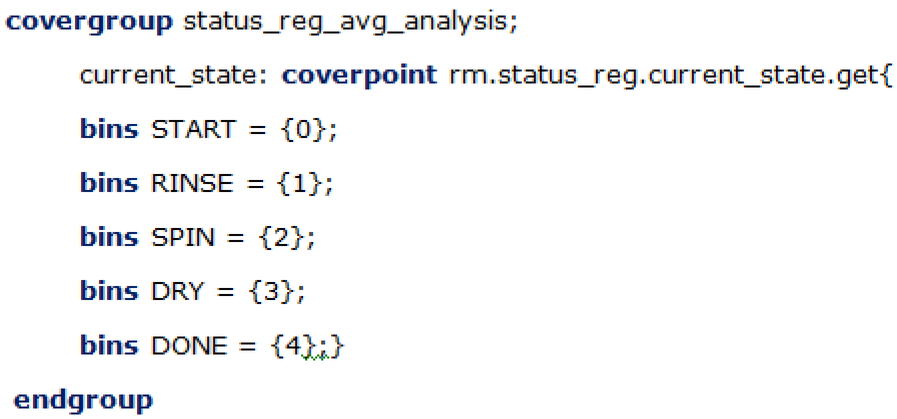
Coverage Collector Using Inline LUTs



1.3 status_reg	status_reg	Reg.	0x20, 0x24...

offset		external		size	32	default	0x00000000

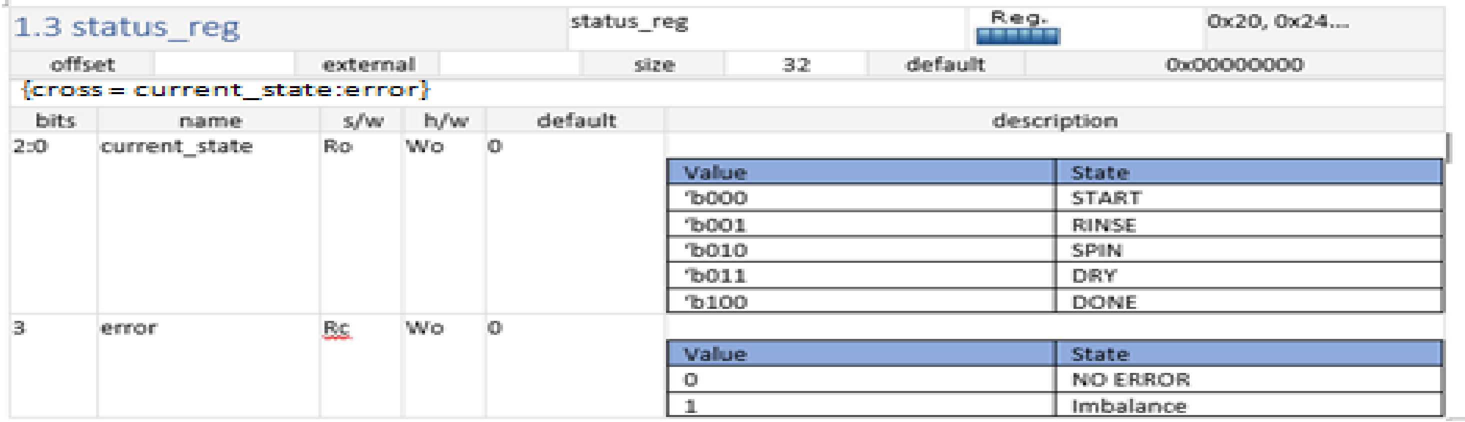
bits	name	s/w	h/w	default	description
2:0	current_state	Ro	Wo	0	{'b000:START,'b001:RINSE,'b010:SPIN,'b011:DRY,'b100:DONE}
3	error	Rc	Wo	0	



```
covergroup status_reg_avg_analysis;
      current_state: coverpoint rm.status_reg.current_state.get{
      bins START = {0};
      bins RINSE = {1};
      bins SPIN = {2};
      bins DRY = {3};
      bins DONE = {4};}
 endgroup
```

Inline Look Up Tables

Cross Coverage between LUTs

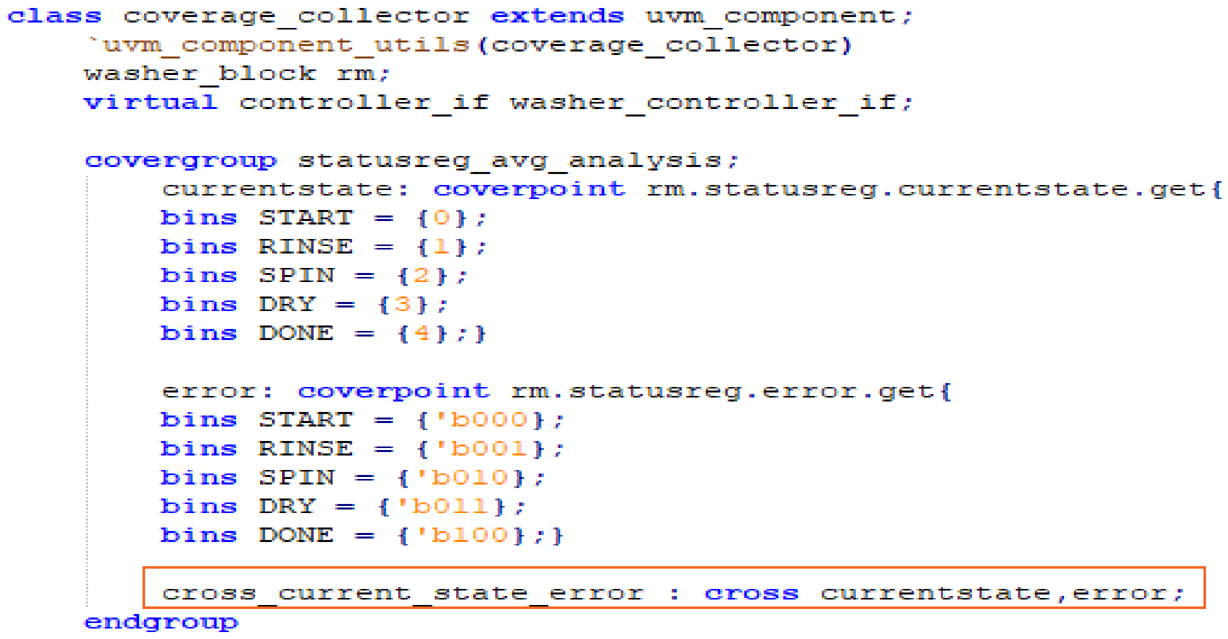


1.3 status_reg				status_reg		Reg.	0x20, 0x24...
offset		external		size	32	default	0x00000000
{cross = current_state:error}							

bits	name	s/w	h/w	default	description
2:0	current_state	Ro	Wo	0	
3	error	Rc	Wo	0	

Value	State
'b000	START
'b001	RINSE
'b010	SPIN
'b011	DRY
'b100	DONE

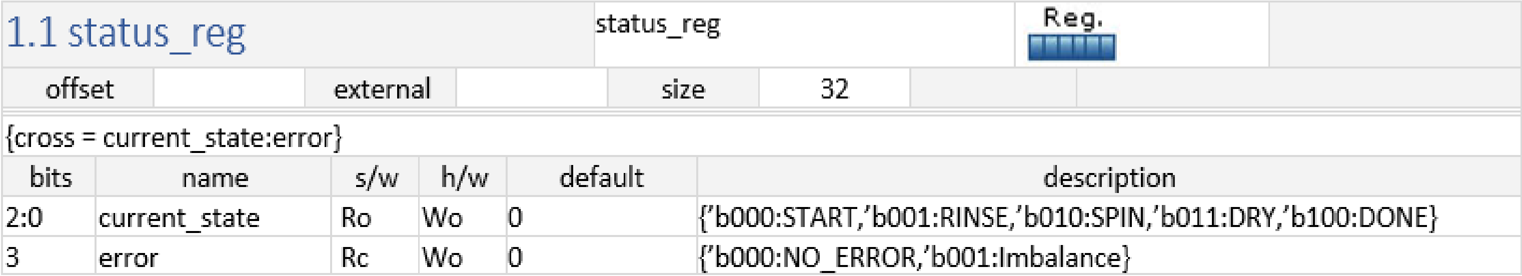
Value	State
0	NO ERROR
1	Imbalance



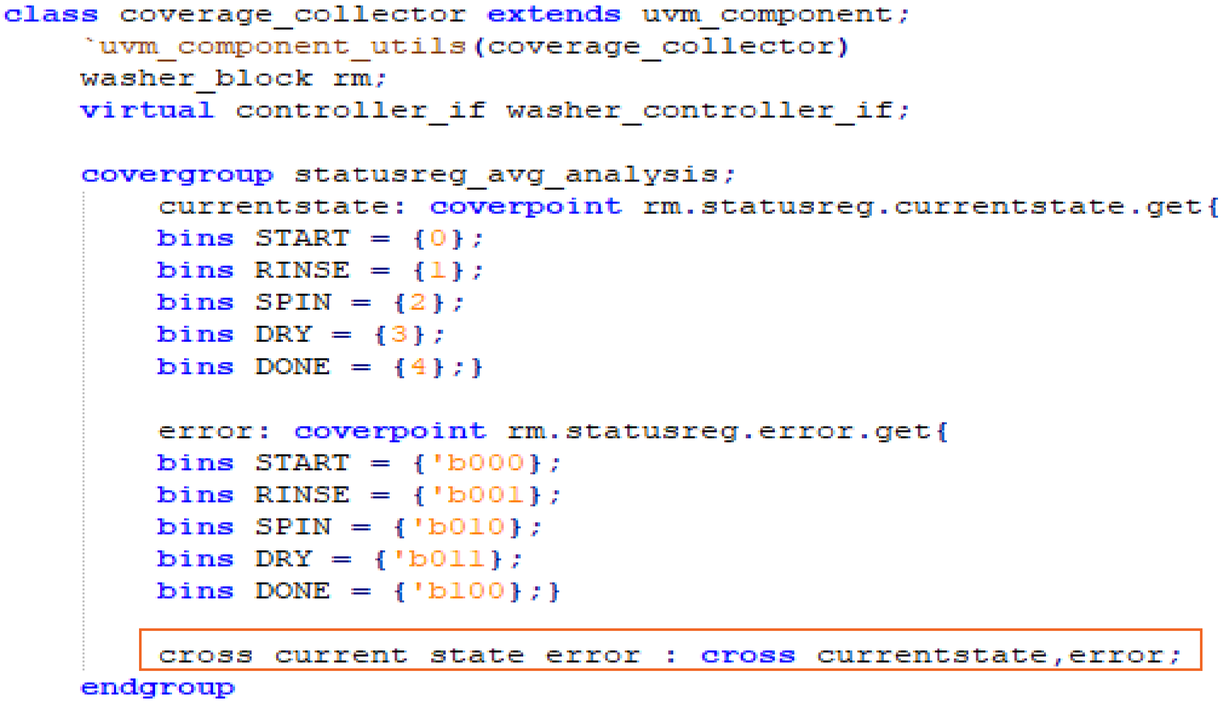
```
class coverage_collector extends uvm_component;
    `uvm_component_utils(coverage_collector)
    washer_block rm;
    virtual controller_if washer_controller_if;

    covergroup statusreg_avg_analysis;
        currentstate: coverpoint rm.statusreg.currentstate.get{
        bins START = {0};
        bins RINSE = {1};
        bins SPIN = {2};
        bins DRY = {3};
        bins DONE = {4};}

        error: coverpoint rm.statusreg.error.get{
        bins START = {'b000};
        bins RINSE = {'b001};
        bins SPIN = {'b010};
        bins DRY = {'b011};
        bins DONE = {'b100};}

        cross_current_state_error : cross currentstate,error;
    endgroup
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Cross Coverage between Inline LUTs



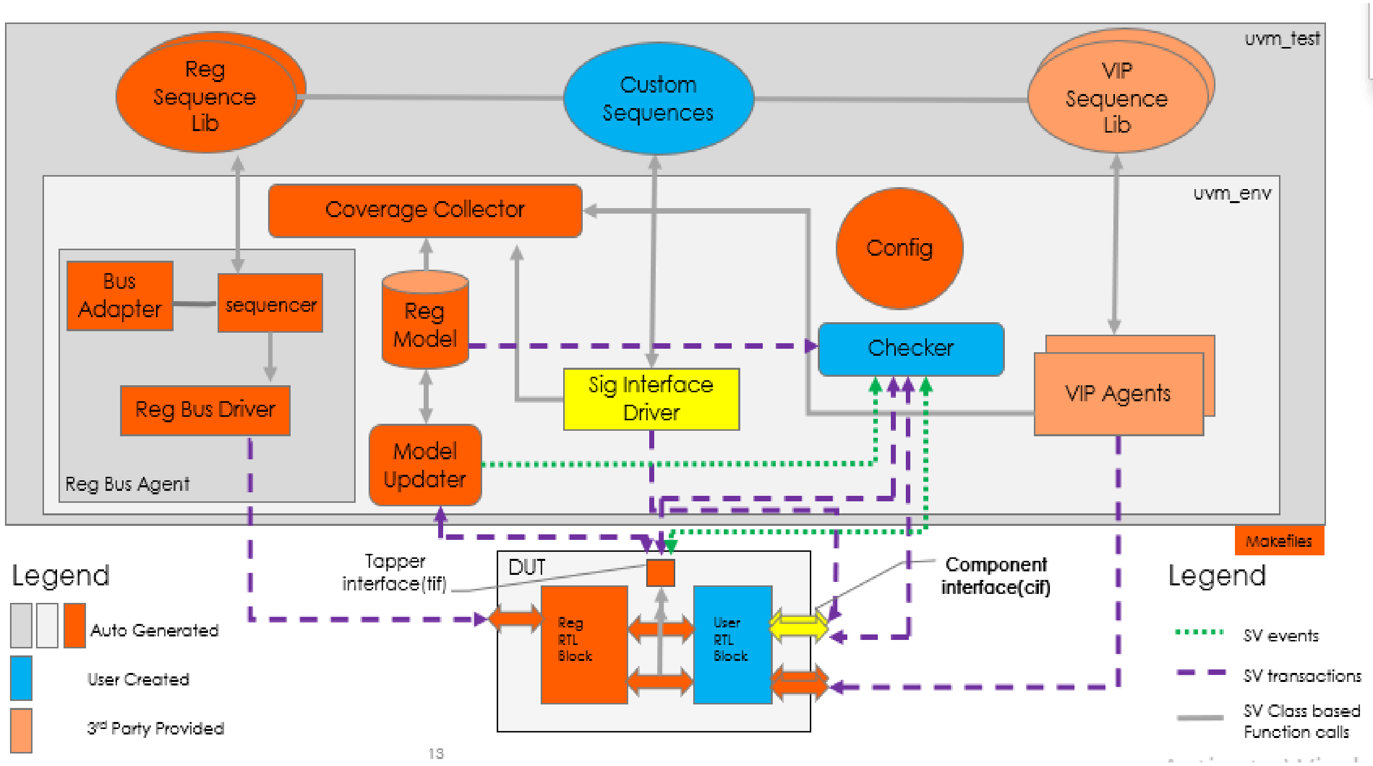
1.1 status_reg		status_reg		Reg.	
offset		external		size	32
{cross = current_state:error}					
bits	name	s/w	h/w	default	description
2:0	current_state	Ro	Wo	0	{'b000:START,'b001:RINSE,'b010:SPIN,'b011:DRY,'b100:DONE}
3	error	Rc	Wo	0	{'b000:NO_ERROR,'b001:Imbalance}



```
class coverage_collector extends uvm_component;
    `uvm_component_utils(coverage_collector)
    washer_block rm;
    virtual controller_if washer_controller_if;

    covergroup statusreg_avg_analysis;
        currentstate: coverpoint rm.statusreg.currentstate.get{
        bins START = {0};
        bins RINSE = {1};
        bins SPIN = {2};
        bins DRY = {3};
        bins DONE = {4};}

        error: coverpoint rm.statusreg.error.get{
        bins START = {'b000};
        bins RINSE = {'b001};
        bins SPIN = {'b010};
        bins DRY = {'b011};
        bins DONE = {'b100};}

        cross current state error : cross currentstate,error;
    endgroup
```

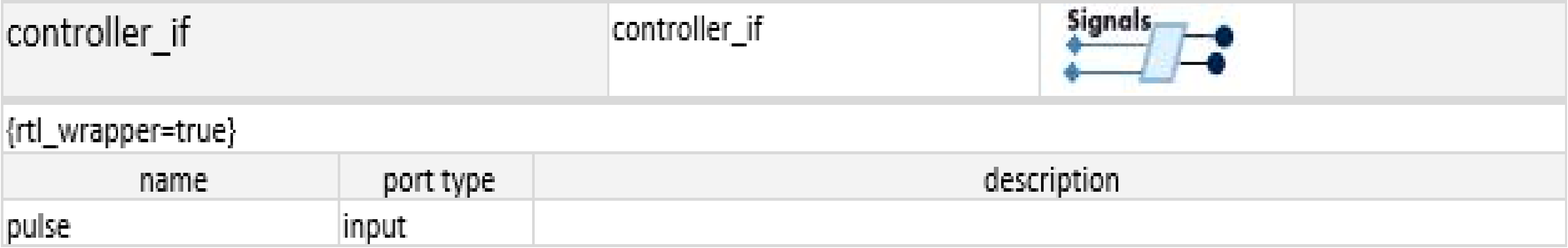
Assign-Lib

- A predefined library containing SV tasks that can be used to drive arbitrary pulses at a required interface
- Allows pulse width modulation
- The assignments of pulses are continuous throughout
- Allows dynamic customization of required wave by randomizing duty cycle and period over a required instance
- Vectors can be driven too
- Can be synchronized over clock edges



Specification

- In the register specification, user can specify the the signal that needs to be driven

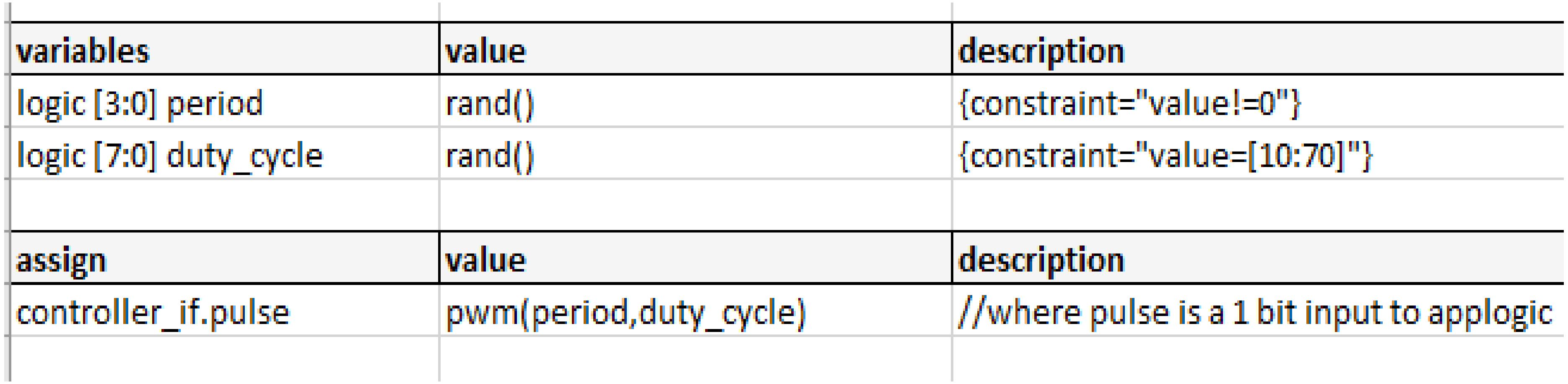


controller_if	controller_if	Signals	

{rtl_wrapper=true}

name	port type	description
pulse	input	

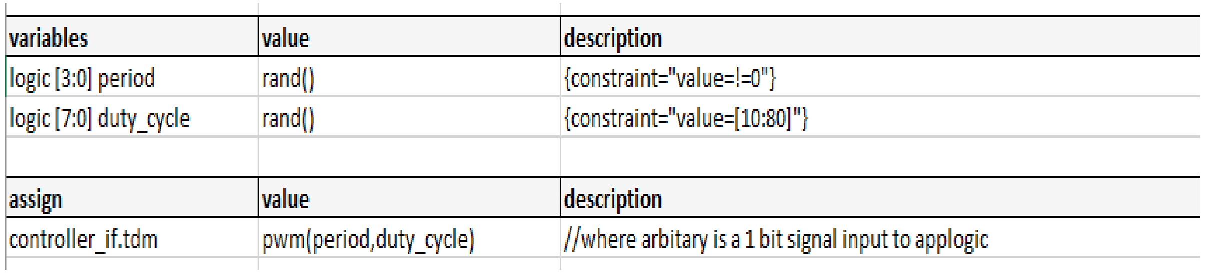
- In the sequence specification, user can specify the magnitude of pulse you want to generate



variables	value	description
logic [3:0] period	rand()	{constraint="value!=0"}
logic [7:0] duty_cycle	rand()	{constraint="value=[10:70]"}

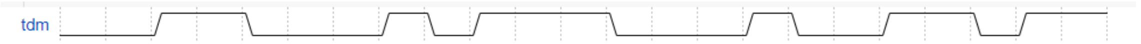
assign	value	description
controller_if.pulse	pwm(period,duty_cycle)	//where pulse is a 1 bit input to applogic

Random Stimulus Generated

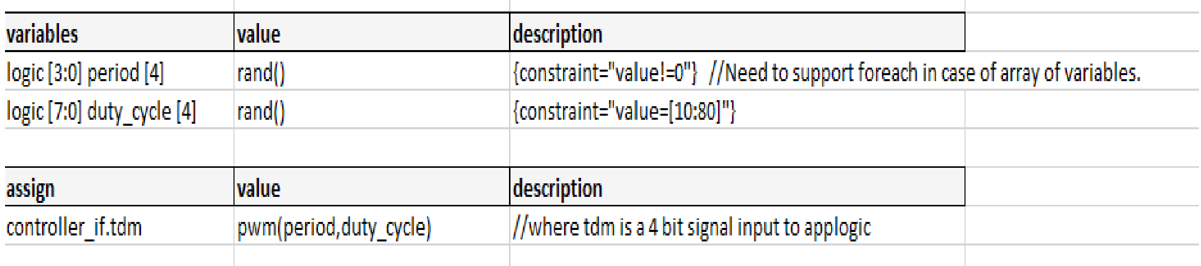


variables	value	description
logic [3:0] period	rand()	{constraint="value=!=0"}
logic [7:0] duty_cycle	rand()	{constraint="value=[10:80]"}

assign	value	description
controller_if.tdm	pwm(period,duty_cycle)	//where arbitary is a 1 bit signal input to applogic

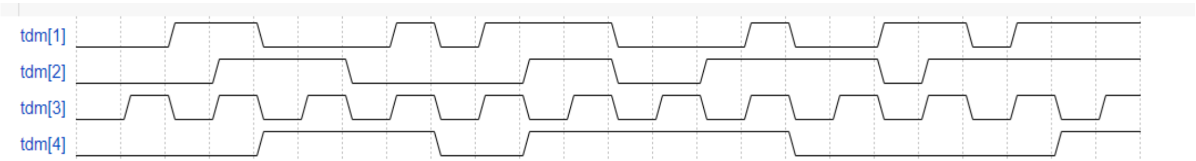


PWM Wave for Single Bit Input

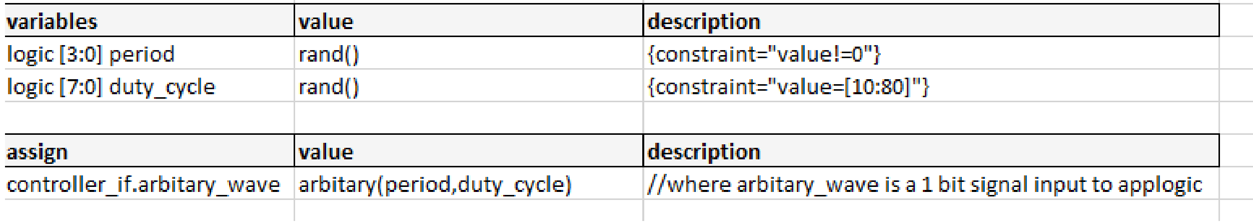


variables	value	description
logic [3:0] period [4]	rand()	{constraint="value!=0"} //Need to support foreach in case of array of variables.
logic [7:0] duty_cycle [4]	rand()	{constraint="value=[10:80]"}

assign	value	description
controller_if.tdm	pwm(period,duty_cycle)	//where tdm is a 4 bit signal input to applogic

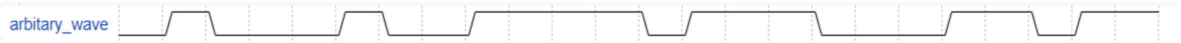


PWM Wave for Vector Input



variables	value	description
logic [3:0] period	rand()	{constraint="value!=0"}
logic [7:0] duty_cycle	rand()	{constraint="value=[10:80]"}

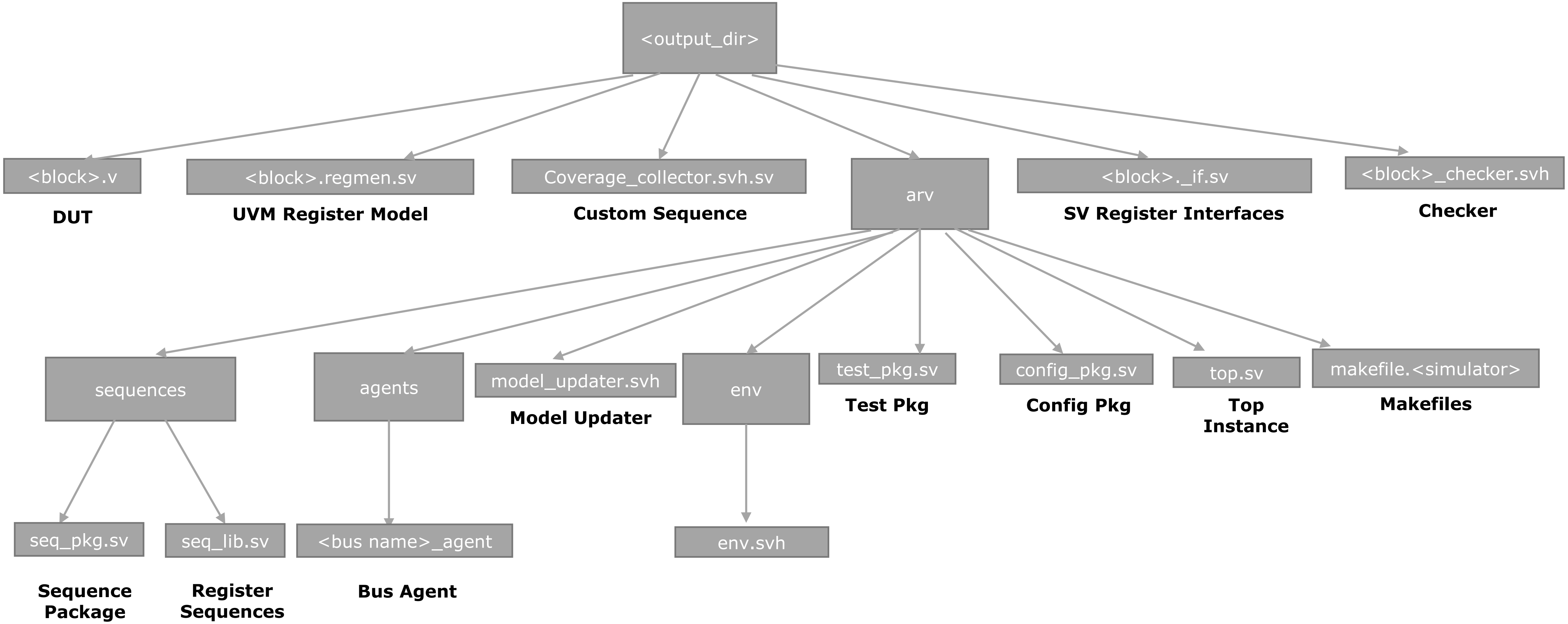
assign	value	description
controller_if.arbitary_wave	arbitary(period,duty_cycle)	//where arbitary_wave is a 1 bit signal input to applogic



Arbitary Wave Generator

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Specta-AV Directory Structure



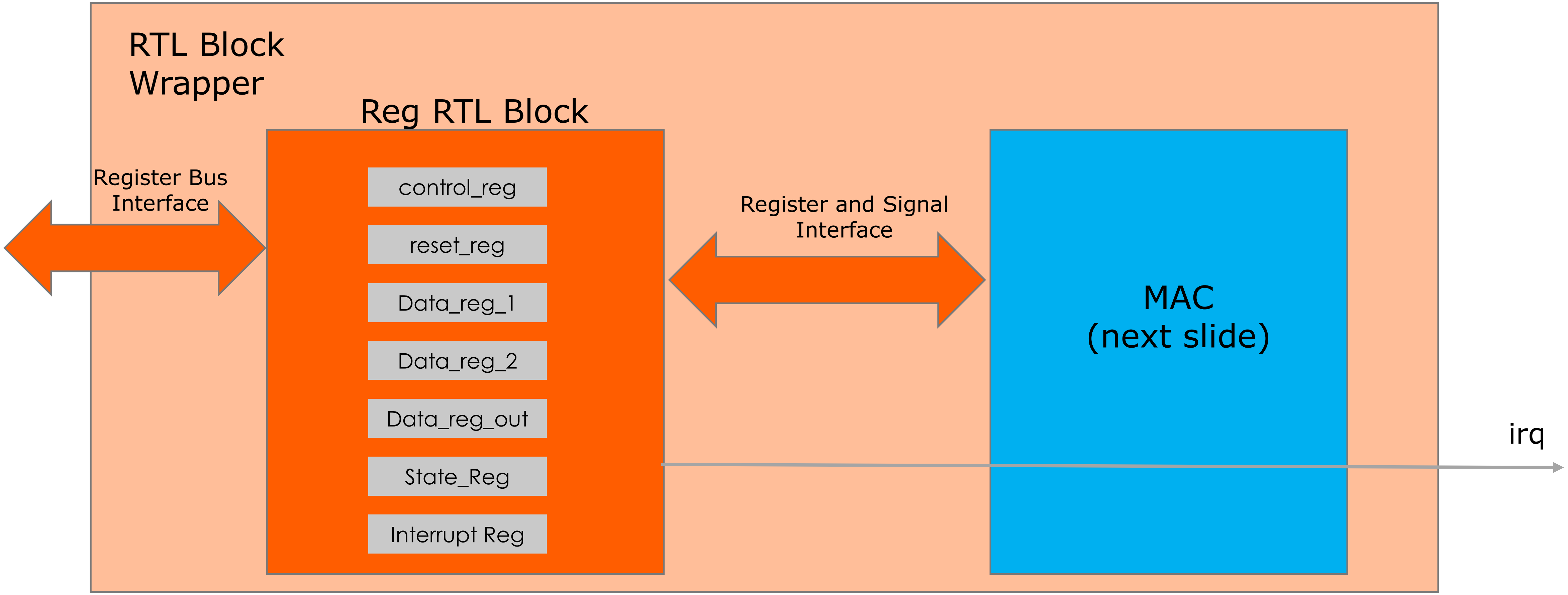
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

A Simple Example

Multiplier & Accumulator (MAC)

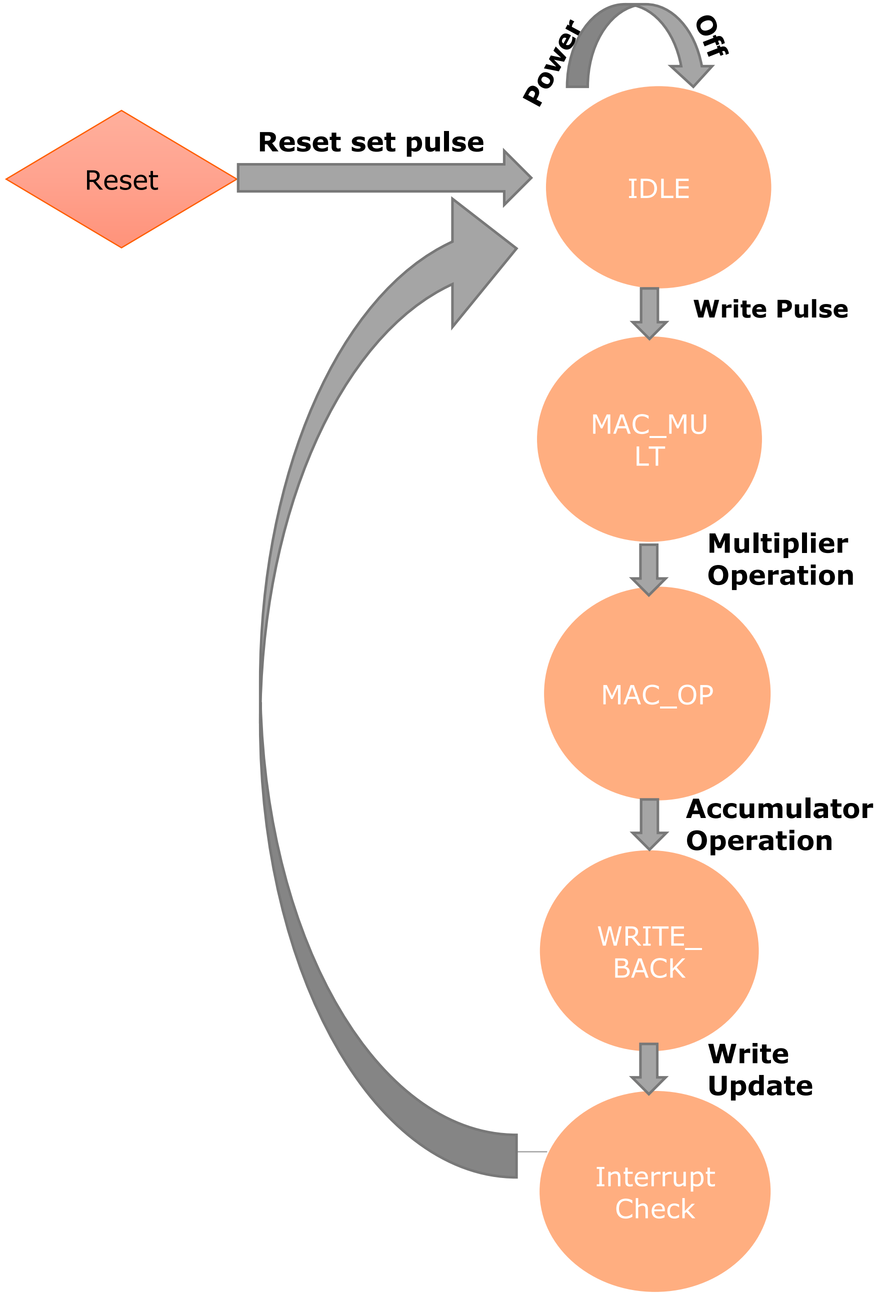
The Design : Multiplier & Accumulator

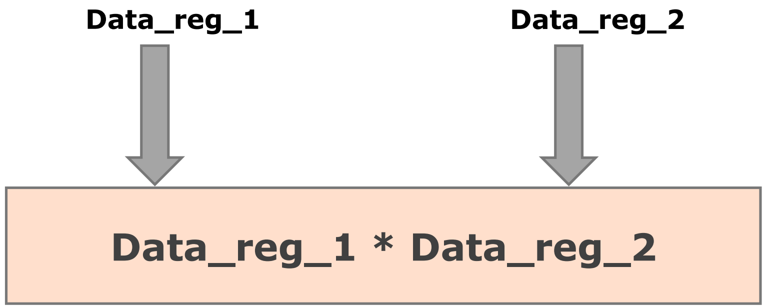
Data_reg_out = Data_reg_out + Data_reg_1 * Data_reg_2



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

MAC State Machine





Data_reg_1 * Data_reg_2

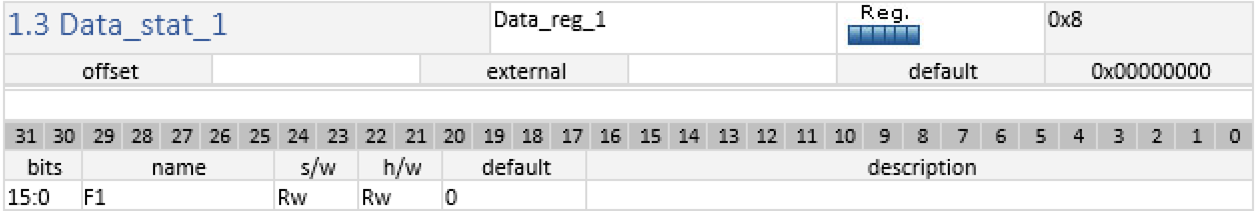
Data_reg_out

+Data_reg_1*Data_reg_2

Data_reg_out (Storing the Output)

Interrupt Reg (Checking for Interrupt)

Register Specification

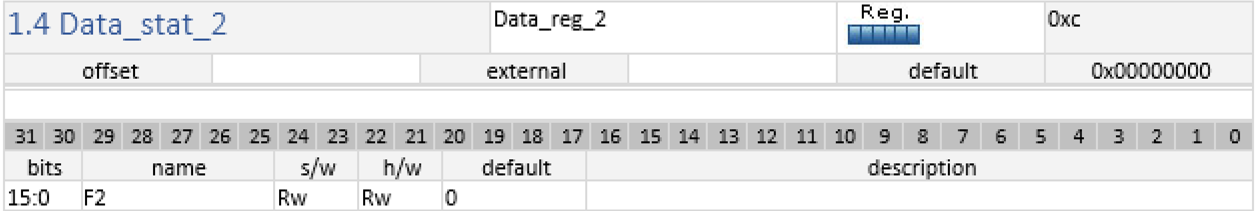


1.3 Data_stat_1	Data_reg_1	Reg.	0x8

offset		external		default	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	description
15:0	F1	Rw	Rw	0	

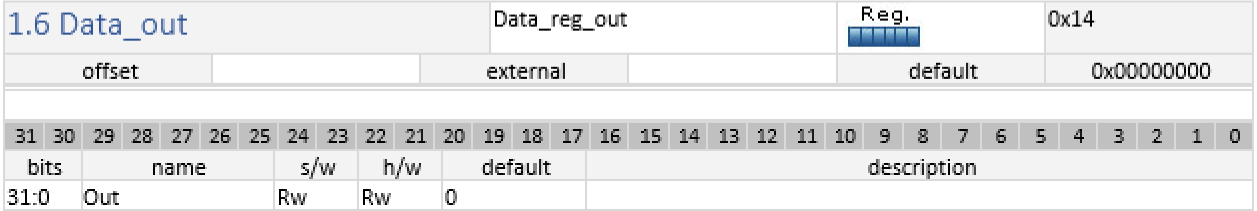


1.4 Data_stat_2	Data_reg_2	Reg.	0xc

offset		external		default	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	description
15:0	F2	Rw	Rw	0	



1.6 Data_out	Data_reg_out	Reg.	0x14

offset		external		default	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

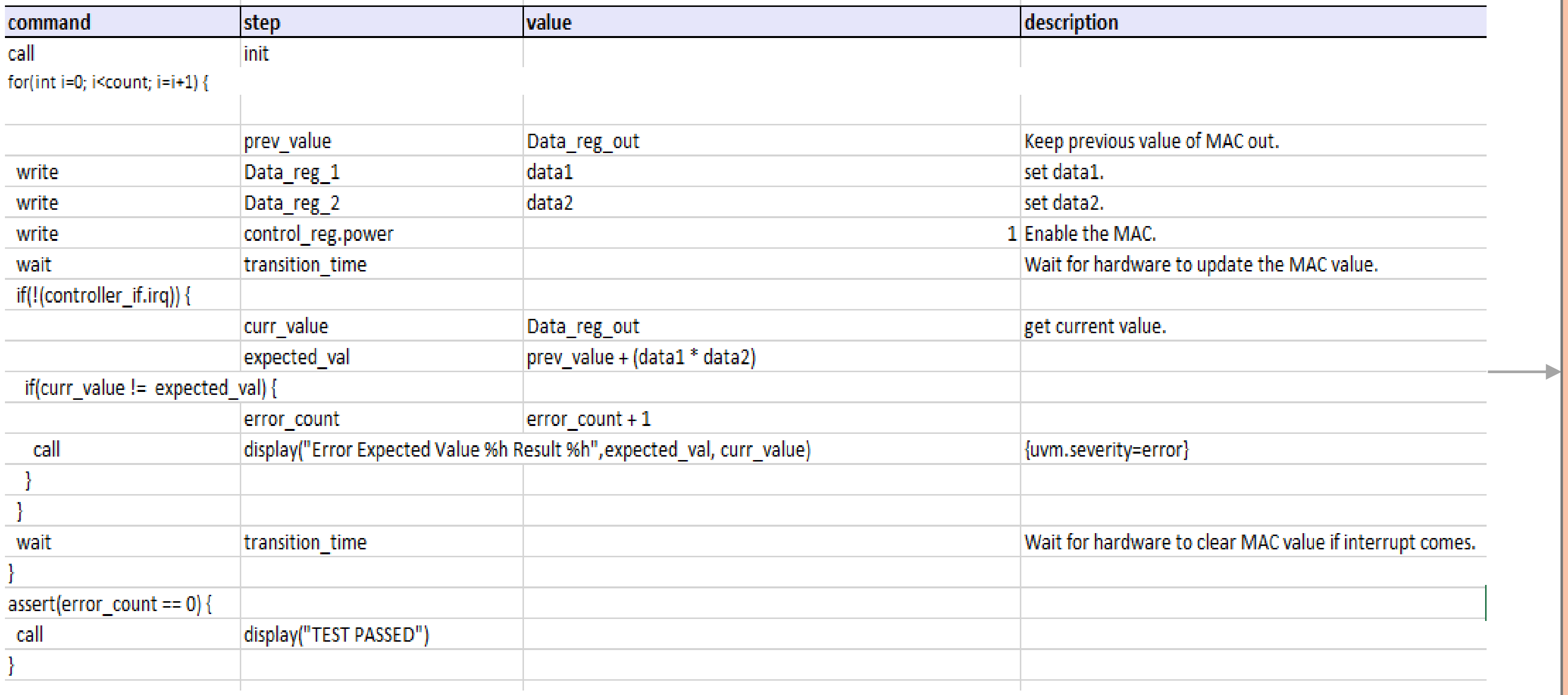
bits	name	s/w	h/w	default	description
31:0	Out	Rw	Rw	0	

MAC Block Register Components

```
always @(posedge clk)
…
  if (Data_reg_1_F1_in_enb)   // F1 : HW Write
    begin
      Data_reg_1_F1_q <= Data_reg_1_F1_in;
    end
  else
    begin
       if   (Data_reg_1_wr_valid)   // F1 : SW Write
         begin
            …
         end
     end

always @(posedge clk)
 …
 if (Data_reg_2_F2_in_enb)   // F2 : HW Write
   begin
     Data_reg_2_F2_q <= Data_reg_2_F2_in;
    end
 else
  begin
    if (Data_reg_2_wr_valid)   // F2 : SW Write
     begin
       …
     end
  end

always @(posedge clk)
 …
 if (Data_reg_out_Out_in_enb)   // OUT : HW Write
   begin
     Data_reg_out_Out_q <= Data_reg_out_Out_in;
   end
  else
    begin
 if (Data_reg_out_wr_valid)   // OUT : SW Write
     begin
      …
      end
  end  // sw_write_close
 end
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Custom Sequence



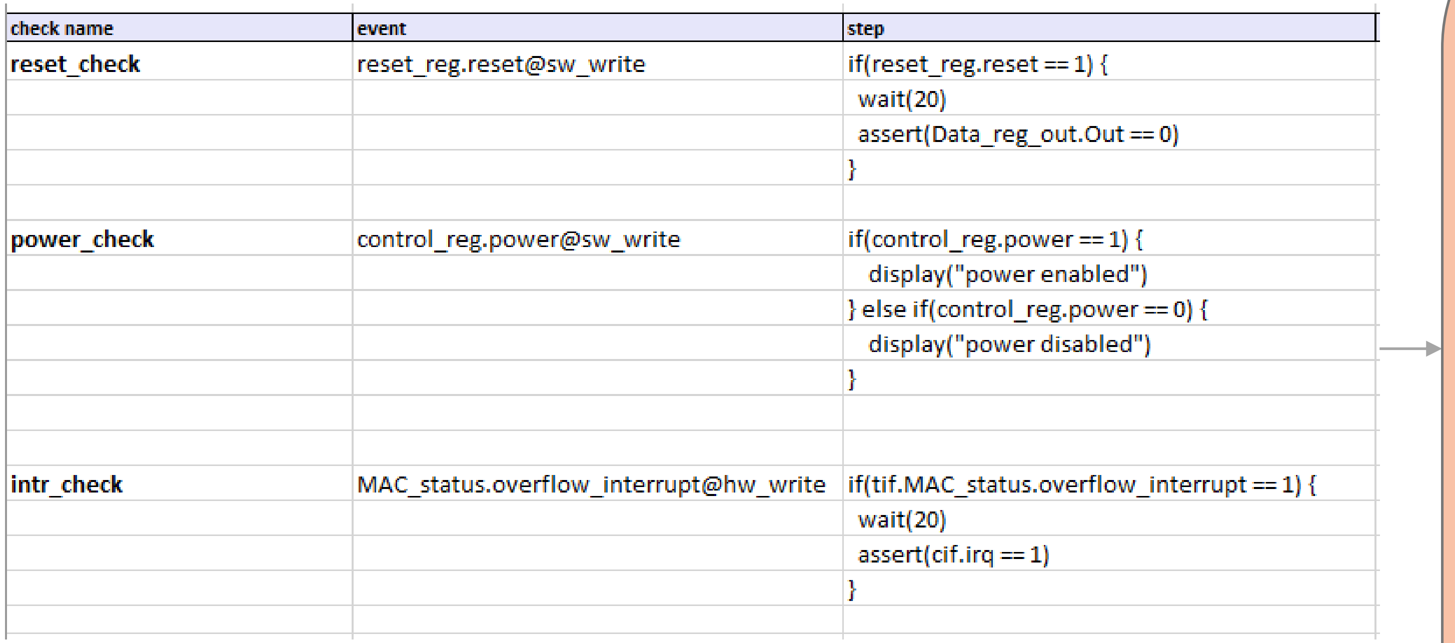
command	step	value	description
call	init		
for(int i=0; i<count; i=i+1) {			
	prev_value	Data_reg_out	Keep previous value of MAC out.
write	Data_reg_1	data1	set data1.
write	Data_reg_2	data2	set data2.
write	control_reg.power	1	Enable the MAC.
wait	transition_time		Wait for hardware to update the MAC value.
if(!(controller_if.irq)) {			
	curr_value	Data_reg_out	get current value.
	expected_val	prev_value + (data1 * data2)	
if(curr_value != expected_val) {			
	error_count	error_count + 1	
call	display("Error Expected Value %h Result %h",expected_val, curr_value)		{uvm.severity=error}
}			
}			
wait	transition_time		Wait for hardware to clear MAC value if interrupt comes.
}			
assert(error_count == 0) {			
call	display("TEST PASSED")		
}			

MAC Block Custom Sequences

```
 for ( int i = 0 ; i < count;i = i + 1 )
 begin
   rm.Data_reg_out.read(status, Data_reg_out,
.parent(this));

   void'(this.randomize());
    rm.Data_reg_1.write(status, data1, .parent(this));

    rm.Data_reg_2.write(status, data2, .parent(this));

    rm.control_reg.power.write(status, 'h1,
.parent(this));
    #transition_time;
    if (!(controller_if_if.irq)) begin
       rm.Data_reg_out.read(status, Data_reg_out,
.parent(this));

       expected_val =  prev_value + (data1 * data2);
       if (curr_value != expected_val) begin
          lvar = error_count + 1;
          error_count=lvar;
          `uvm_error("ISS", $sformatf("Error Expected
Value %0h
           Result %0h  ", expected_val, curr_value));
        end
      end
   #transition_time;
  end

assert (error_count == 0)begin
`uvm_info("ISS", $sformatf("TEST
PASSED"),UVM_LOW);

end
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Checker Specification

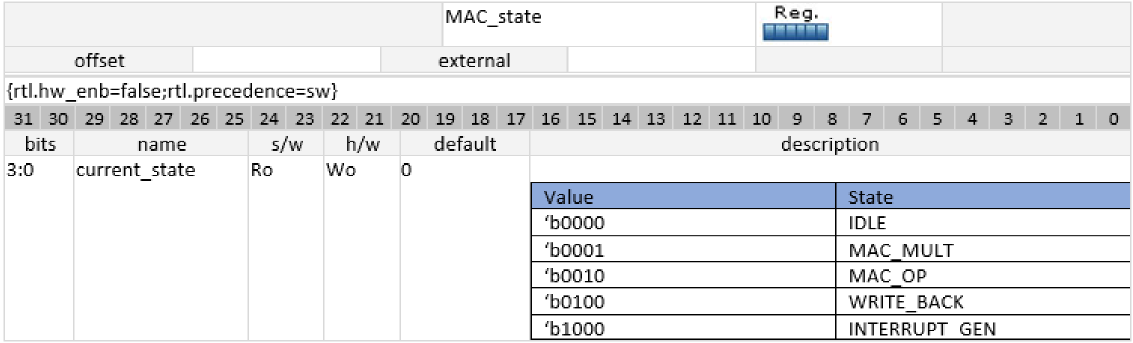


check name	event	step
reset_check	reset_reg.reset@sw_write	if(reset_reg.reset == 1) {
		wait(20)
		assert(Data_reg_out.Out == 0)
		}
power_check	control_reg.power@sw_write	if(control_reg.power == 1) {
		display("power enabled")
		} else if(control_reg.power == 0) {
		display("power disabled")
		}
intr_check	MAC_status.overflow_interrupt@hw_write	if(tif.MAC_status.overflow_interrupt == 1) {
		wait(20)
		assert(cif.irq == 1)
		}

MAC Block Checker

```
 forever
  begin
     reset_reg_reset_sw_write_event.wait_ptrigger();
     if (rm.reset_reg.reset.get() == 1 ) begin
        #20;
         assert(rm.Data_reg_out.Out.get() == 0)
          …
          ..
     end

  begin : power_check
   forever  begin
      control_reg_power_sw_write_event.wait_ptrigger();
      if (rm.control_reg.power.get() == 1 ) begin
         `uvm_info("FROM CHECKER","power
enabled",UVM_LOW);
            …
            ..

  begin : intr_check
   forever  begin
       if (tif.MAC_status_overflow_interrupt_in == 1 ) begin
           #20;
           assert(cif.irq == 1) else
            …
            ..
       end
```

Coverage Collector



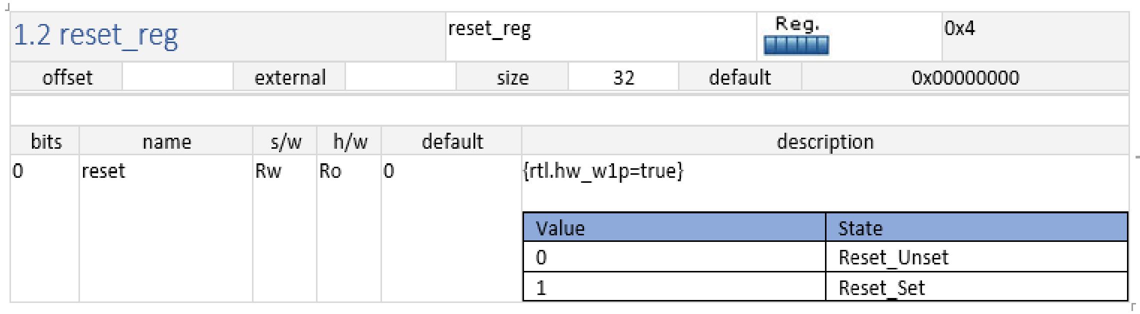
	MAC_state	Reg.	
offset		external	

{rtl.hw_enb=false;rtl.precedence=sw}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	description
3:0	current_state	Ro	Wo	0	

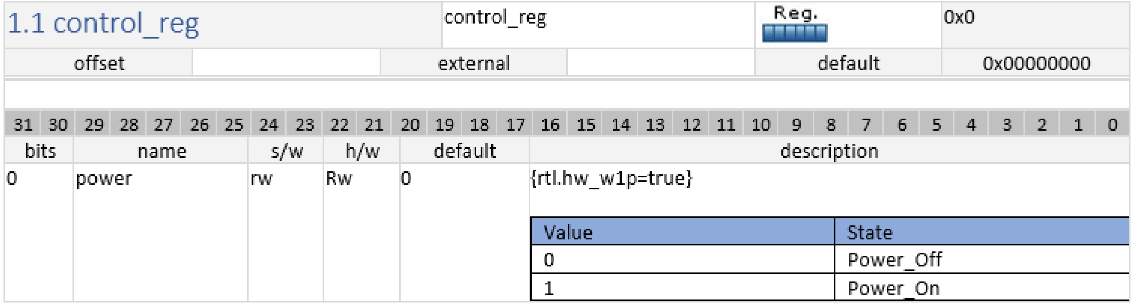
Value	State
‘b0000	IDLE
‘b0001	MAC_MULT
‘b0010	MAC_OP
‘b0100	WRITE_BACK
‘b1000	INTERRUPT_GEN



1.2 reset_reg	reset_reg	Reg.	0x4
offset	external	size 32	default 0x00000000

bits	name	s/w	h/w	default	description
0	reset	Rw	Ro	0	{rtl.hw_w1p=true}

Value	State
0	Reset_Unset
1	Reset_Set



1.1 control_reg	control_reg	Reg.	0x0
offset	external	default	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	description
0	power	rw	Rw	0	{rtl.hw_w1p=true}

Value	State
0	Power_Off
1	Power_On

```
covergroup MAC_state_avg_analysis;

    current_state: coverpoint
rm.MAC_state.current_state.get{
        bins IDLE = {0};
        bins MAC_MULT = {1};
        bins MAC_OP = {2};
        bins WRITE_BACK = {4};
        bins INTERRUPT_GEN = {8};}
endgroup


covergroup reset_reg_avg_analysis;

      reset: coverpoint rm.reset_reg.reset.get{
          bins Reset_Unset = {0};
          bins Reset_Set = {1};}
 endgroup


covergroup control_reg_avg_analysis;
      power: coverpoint rm.control_reg.power.get{
           bins Power_Off = {0};
           bins Power_On = {1};}
 endgroup
```

Code Generated by Specta-AV

Word	: 2 Pages
Excel	: 5 Pages
User Applogic Code	: 80 lines

What is generated	Typical Lines of code
• DUT	~ 1130
• Test Environment	
– Reg Model	~ 1907
– UVM plumbing	~ 1805
◦ Sequencers	
◦ Agents	
◦ uvm_env, …	
• Tests	~ 5217
– Register Sequences	~ 581
– Custom Sequences	~ 431
– Coverage	~ 374
– Model Updater	~ 495
– Checker	
	~10,000 loc

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Specta-AV gives a Boost to Verification Productivity

Specta-AV Benefits

- Automatically generate code from specification
- Reduce required manual changes
- Improves verification productivity

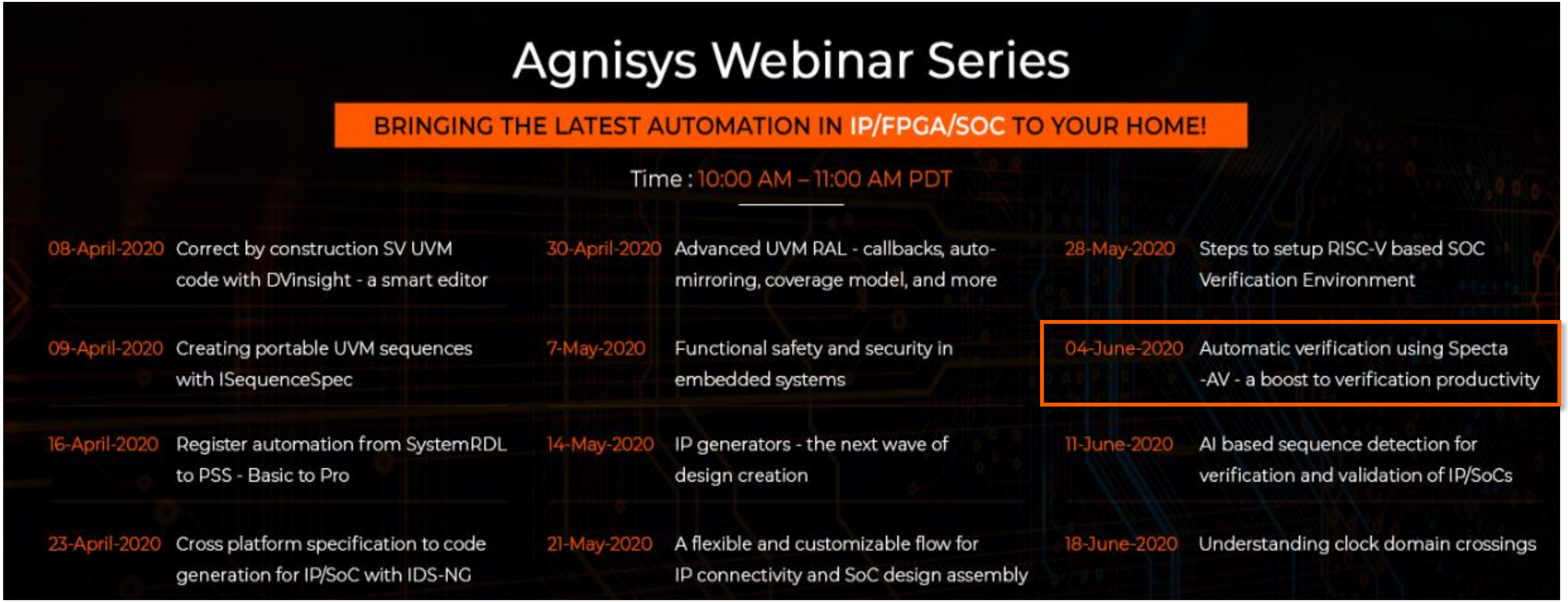
Design Change	Manual Coding	Specta-AV
Register Access Change	• Need to Update ~ Register Map RTL (Verilog,SystemVerilog) ~ RAL ~ Register Access Sequences ~ Environment (Top, Interface, etc.) ~ Checks ~ Coverage code ~ Headers	• Need to Update ~ Specification
Register Address Change	• Need to Update ~ Register Map RTL ~ RAL Model ~ Sequences ~ Headers ~ Checks ~ Address Map (Coverage Collector)	• Need to Update ~ Specification
Signal Addition	• Need to Update ~ RTL ~ Interfaces ~ Sequences ~ Checks	• Need to Update ~ Specification

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Specta-AV Benefits Cont'd.

- If you are new to UVM
  - Create a complete UVM based verification environment
  - No verification/UVM expertise needed

- If you are an advanced verification user
  - No need to manually create environment
  - More time to work on complex areas of the chip



Agnisys Webinar Series

BRINGING THE LATEST AUTOMATION IN IP/FPGA/SOC TO YOUR HOME!

Time : 10:00 AM – 11:00 AM PDT

08-April-2020	Correct by construction SV UVM code with DVinsight - a smart editor	30-April-2020	Advanced UVM RAL - callbacks, auto-mirroring, coverage model, and more	28-May-2020	Steps to setup RISC-V based SOC Verification Environment
09-April-2020	Creating portable UVM sequences with ISequenceSpec	7-May-2020	Functional safety and security in embedded systems	04-June-2020	Automatic verification using Specta -AV - a boost to verification productivity
16-April-2020	Register automation from SystemRDL to PSS - Basic to Pro	14-May-2020	IP generators - the next wave of design creation	11-June-2020	AI based sequence detection for verification and validation of IP/SoCs
23-April-2020	Cross platform specification to code generation for IP/SoC with IDS-NG	21-May-2020	A flexible and customizable flow for IP connectivity and SoC design assembly	18-June-2020	Understanding clock domain crossings

About Agnisys

- The EDA leader in solving complex design and verification problems associated with HW/SW interface
- Formed in 2007
- Privately held and profitable
- Headquarters in Boston, MA
  - ~1000 users worldwide
  - ~50 customer companies
- Customer retention rate ~90%
- R&D centers (US and India)
- Support centers - committed to ensure comprehensive support
  - Email : support@agnisys.com
  - Phone : 1-855-VERIFYY
  - Response time within one day; within hours in many cases
  - Multiple time zones (Boston MA, San Jose CA and Noida India)

IDESIGNSPEC™ (IDS) EXECUTABLE REGISTER SPECIFICATION

Automatically generate UVM models, Verilog/VHDL models, coverage model, software model, etc.

AUTOMATIC REGISTER VERIFICATION (ARV)

ARV-Sim™ : Create UVM test environment, sequences, and verification plans, and instantly know the status of the verification project

ARV-Formal™ : Create formal properties and assertions, and coverage model from the specification

ISEQUENCESPEC™ (ISS)

Create UVM sequences and firmware routines from the specification

DVinsight™ (DVi)

Smart editor for SystemVerilog and UVM projects

IDS – Next Generation™ (IDS-NG)

Comprehensive SoC/IP spec creation and code generation tool

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY