AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Generate Portable Sequences from a Golden Specification

Write sequence once …

… run on any platform.

Welcome

Anupam Bakshi

PRESENTER

FOUNDER/CEO

AGNISYS

Christina Toole

HOST

MARKETING CONSULTANT

AGNISYS

AGENDA

1 INTRODUCTION

2 The Problem : Sequences Everywhere

3 ISequenceSpec

4 Link to PSS

5 How Agnisys can help

Introduction

What are sequences?

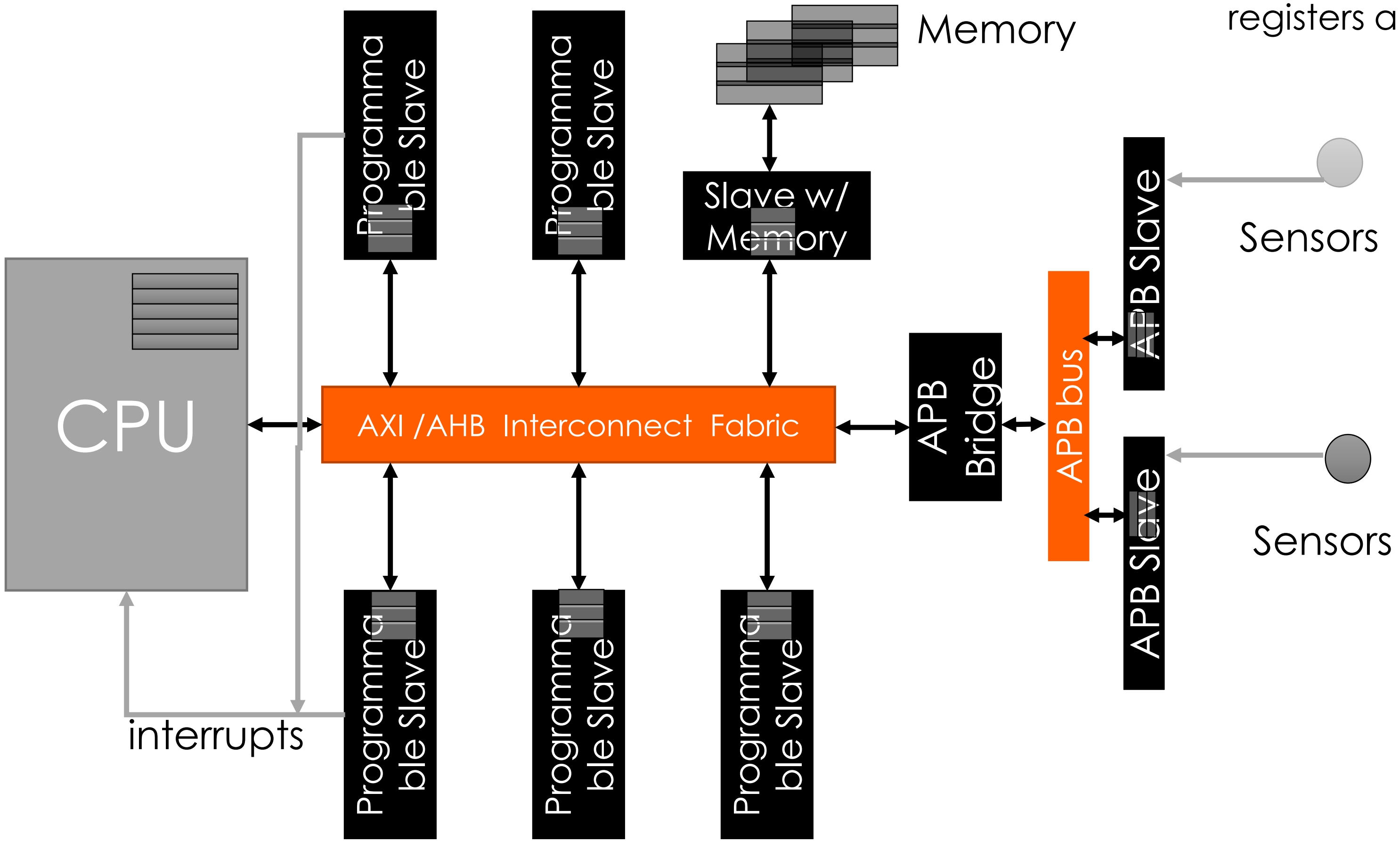
Sequences

- A set of steps to achieve a certain functionality in a device
  - write/reads
    - Field
    - Registers
    - Signals / Ports
  - Call other sequences

- Can be associated with Modeling, Verification, Firmware, Validation

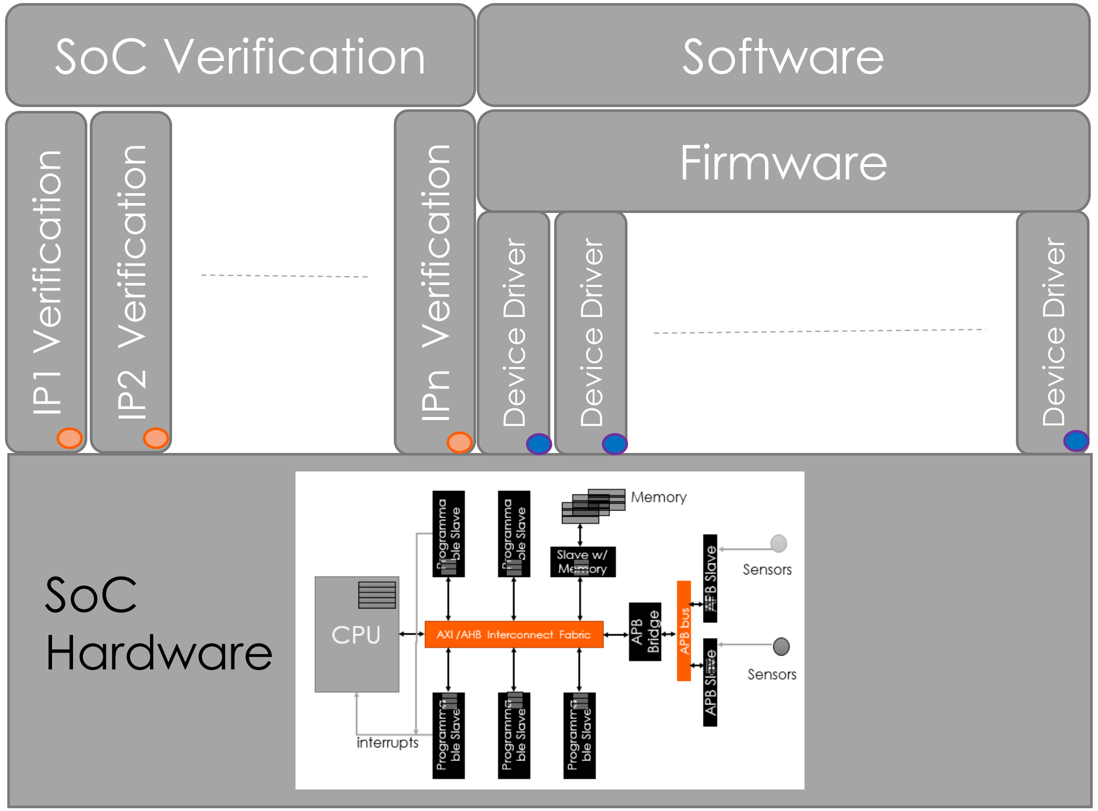
Typical SoC Hardware

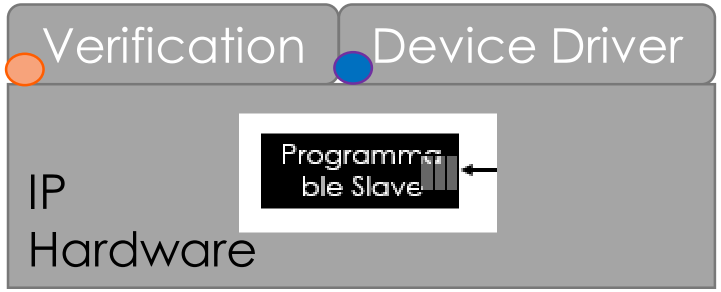
The slaves are programmed by reading/writing to the embedded registers and memory

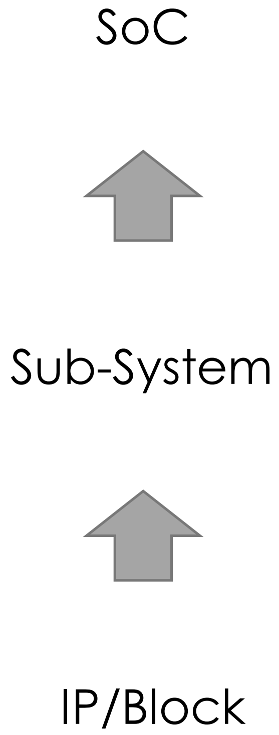


Hardware Software stack

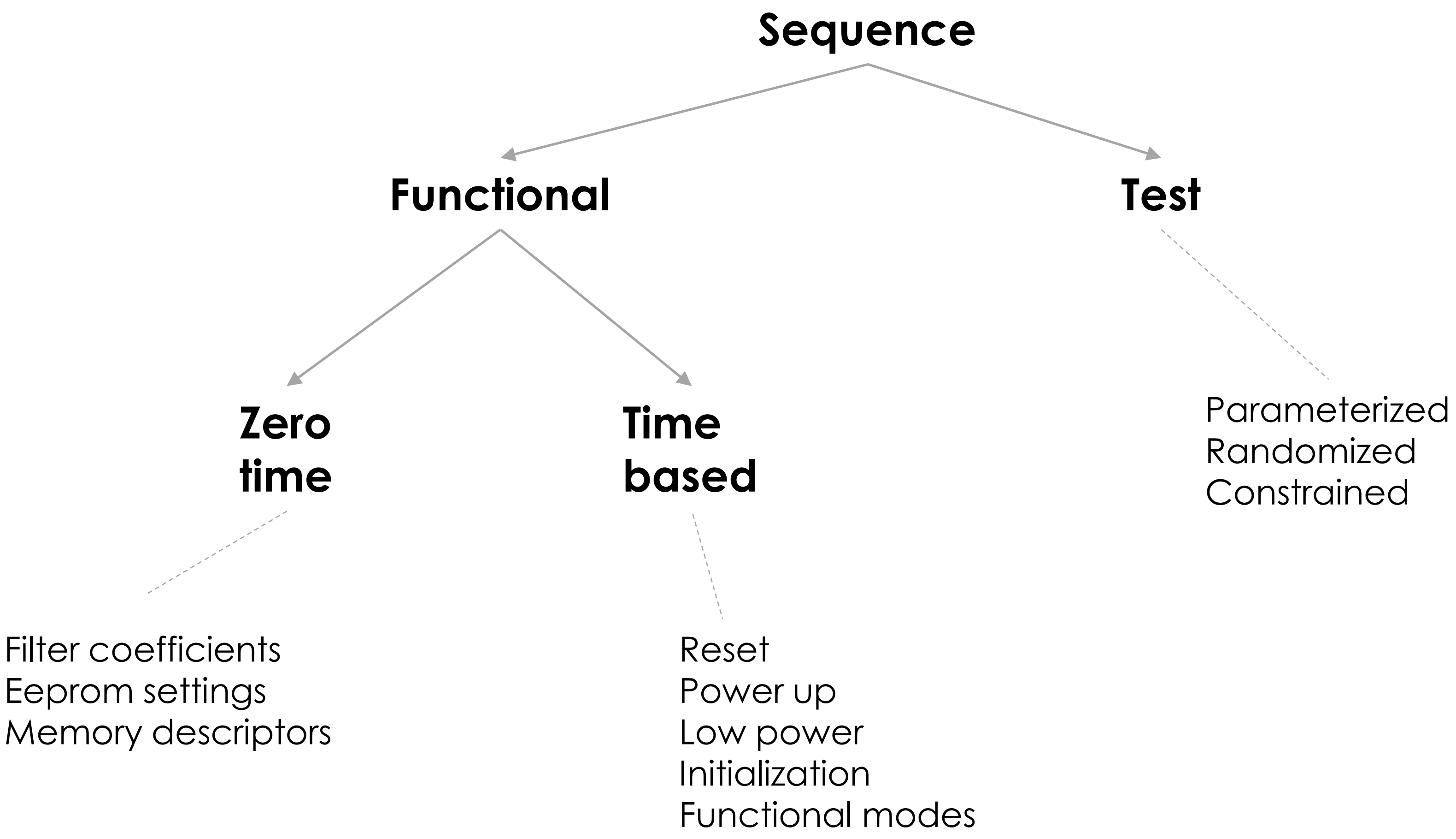
- Sequences are everywhere







Types of Sequence



Sequence

Functional

Zero time

Filter coefficients
Eeprom settings
Memory descriptors

Time based

Reset
Power up
Low power
Initialization
Functional modes

Test

Parameterized
Randomized
Constrained

The Problem

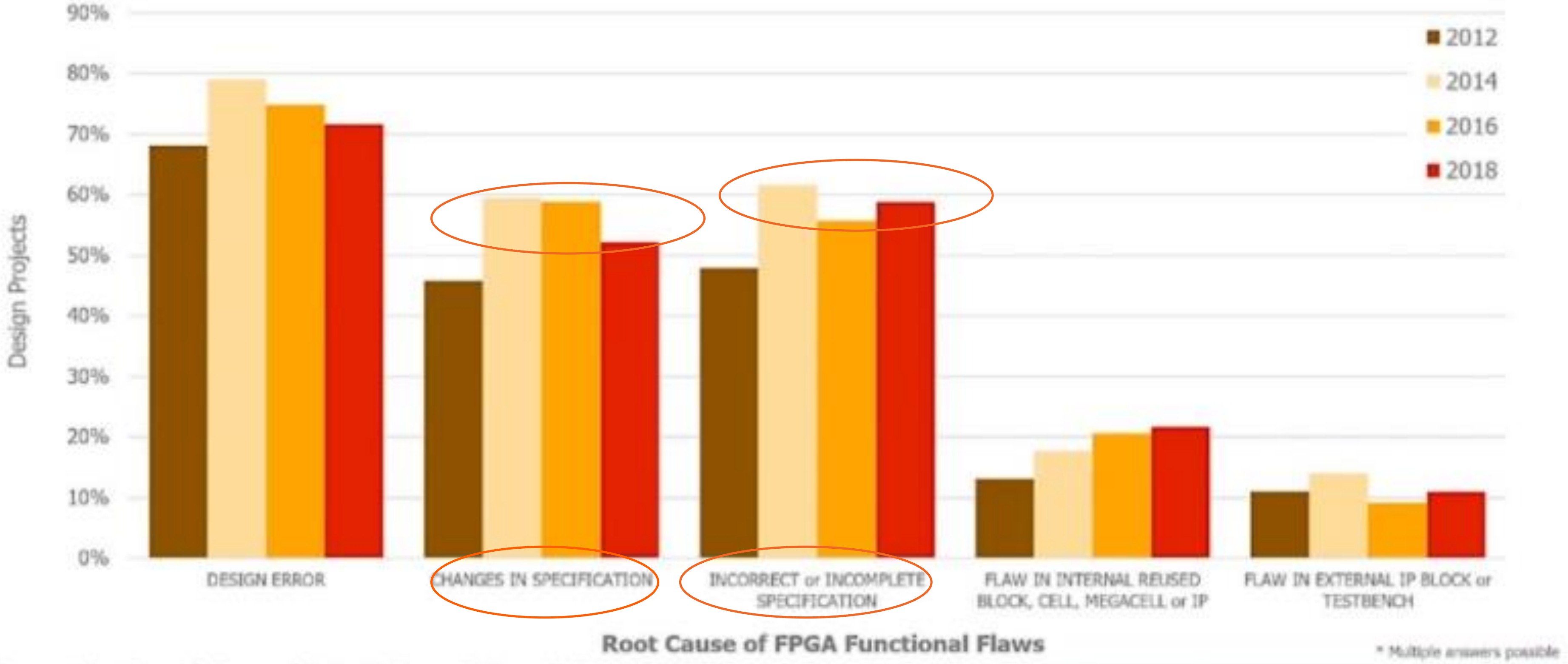
Golden Spec to Portable Sequence

Embedded Design Error statistics

- Average Development team spends 50% of schedule in debugging
- Average developer injects 5 to 10 errors per 100 lines of code but removes only 85%
- However, high quality and safety-critical teams inject less errors and remove more by following <u>better processes</u>

Root Causes of Functional Flaws in FPGAs

FPGA: Root Cause of Functional Flaws

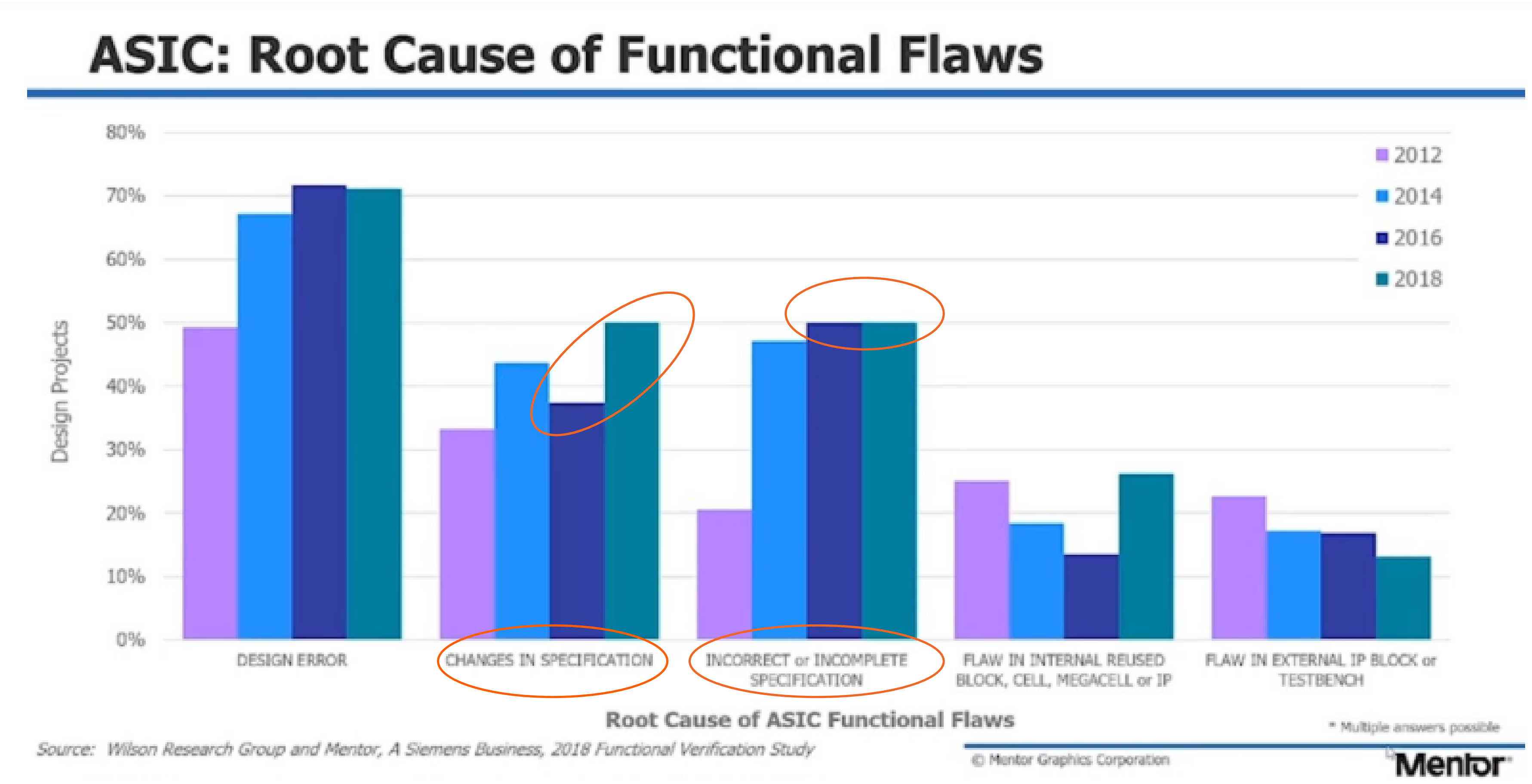


Source: Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study

© Mentor Graphics Corporation

Mentor

Root Causes of Functional Flaws in ASICs



Source: *Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study*

© Mentor Graphics Corporation

Challenges we face with Sequences

- Has this ever happened to you?
  - Sequence is not clear or well documented
  - A sequence works on one platform and not on other
  - No way to create the same debug environment on multiple platforms

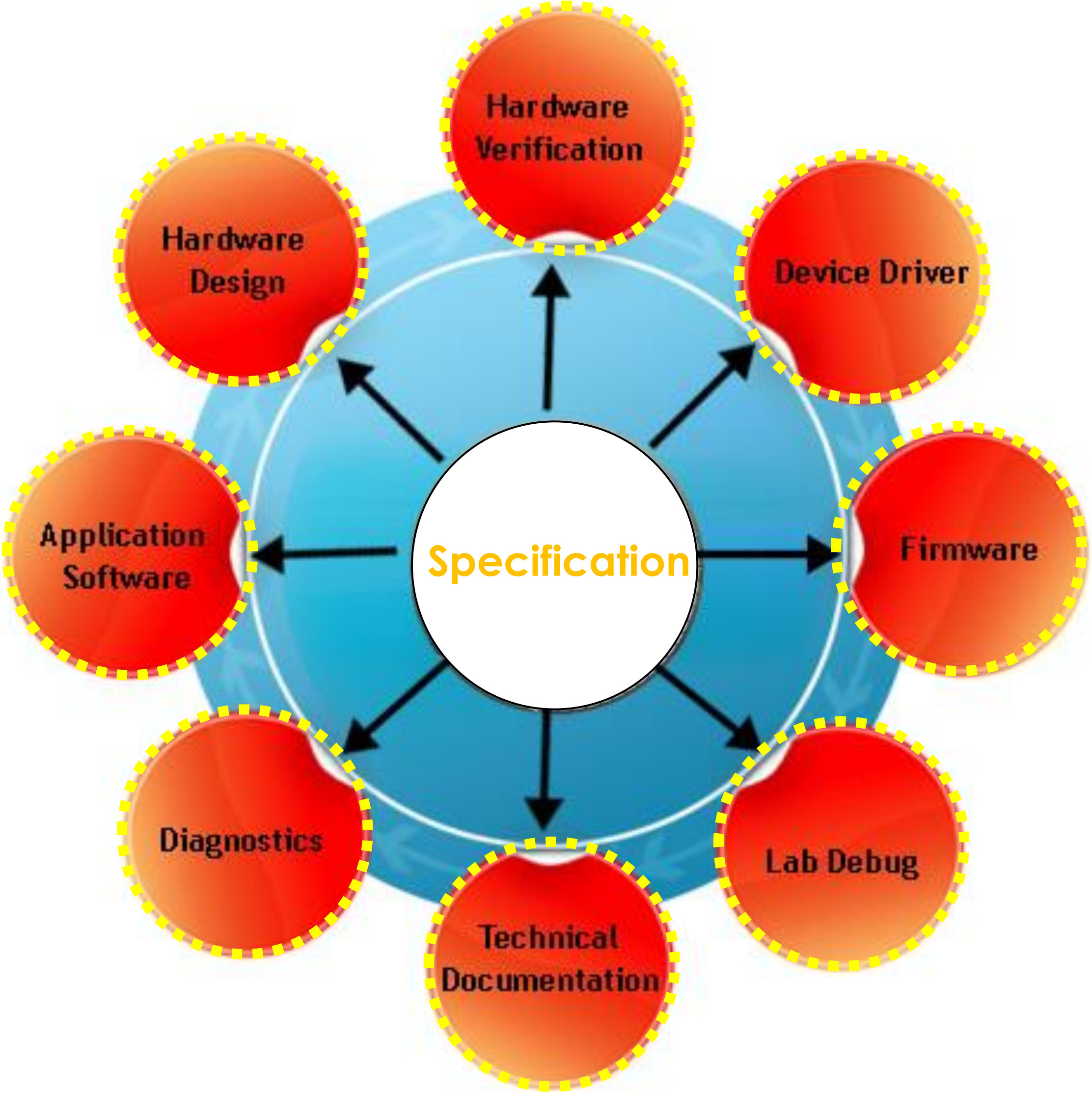
- Inconsistent definition of Sequences
  - In-exact definition
  - Inconsistent interpretation
  - Incorrect implementation

- Sequences contain Register data that can be in any format:
  - Industry Standards – IP-XACT, SystemRDL, RALF
  - Custom formats – CSV, Excel, XML
  - IDesignSpec formats – IDSWord, IDSExcel, XML

The Problem

- Sequences are everywhere
  - Architects/designers plan them
  - Design engineers encode Verilog functionality
  - Verification engineers write them in UVM or PSS
  - Firmware/System engineers write them in C/C++
  - Software engineers use the low level sequences to create higher level sequences
  - Validation and test engineers also use the sequences
- Complex Sequences need to be developed for a variety of platforms
  - Simulation (UVM-SystemVerilog)
  - Firmware and Device Driver (C/C++)
  - Prototyping and Emulation (SystemVerilog)
  - Post-Silicon Validation (C/C++)
  - Automatic Test Equipment (ASCII or CSV)

Consumers of Sequence information



Make changes to the Specification and have the change automatically permeate to all views

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

ISequenceSpec

Centralized sequence specification

&

Portable sequence generation

What does a common sequence specification need

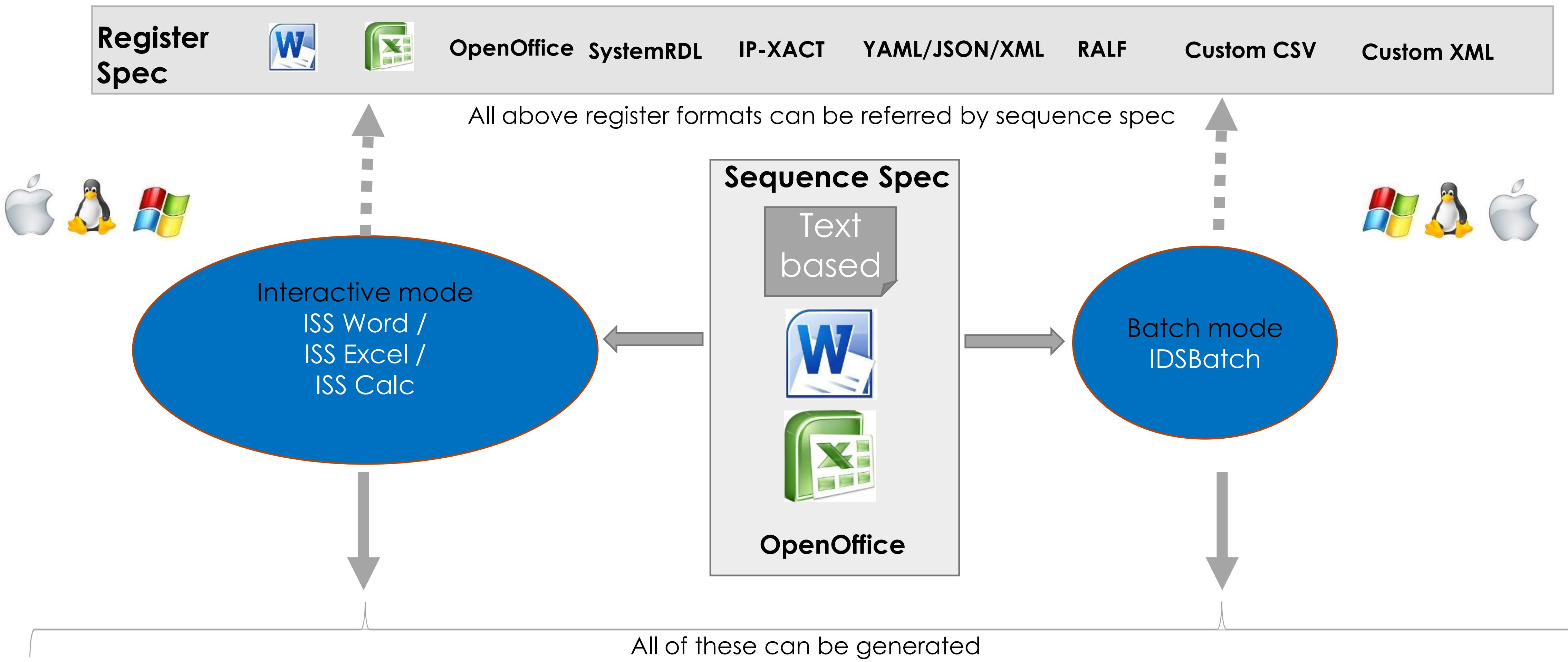
- Similar to pseudo code
- Control flow
- Register read/writes
- Signal or interface read/writes
- Ability to execute arbitrary transactions
- Deal with timing differently
  - A millisecond on the board takes a very long time to simulate
- Deal with hierarchy
  - Design hierarchy IP/SoC
  - Sequence calling other sequences
- Parallelism
  - Sub-system or SoC Level
  - Multiple interfaces at IP level
  - Between Environment and the Device

- Meta information
  - Arguments
  - Parameters
  - Variables
  - Enum
  - Define
  - Macros
  - Structures
  - Look up tables

What does a sequence generation need

- Create a variety of output formats
- Flexibility in how Read/Writes are generated
- Output specific
  - UVM : font door/back door / peekpoke
  - C/C++ : Consolidated read/write
  - Test/Validation : Multiple test sites – for testing multiple chips simultaneously
  - Target platform may not support hierarchy, loops, variables

ISequenceSpec suite



Register Spec

OpenOffice SystemRDL IP-XACT YAML/JSON/XML RALF Custom CSV Custom XML

All above register formats can be referred by sequence spec

Sequence Spec

Text based

OpenOffice

Interactive mode
ISS Word /
ISS Excel /
ISS Calc

Batch mode
IDSBatch

All of these can be generated

SV-UVM

SC-UVM

C/C++ Firmware

SystemC

Verilog for Validation/Emulation

Python, Tcl

Automatic Test Equipment Formats

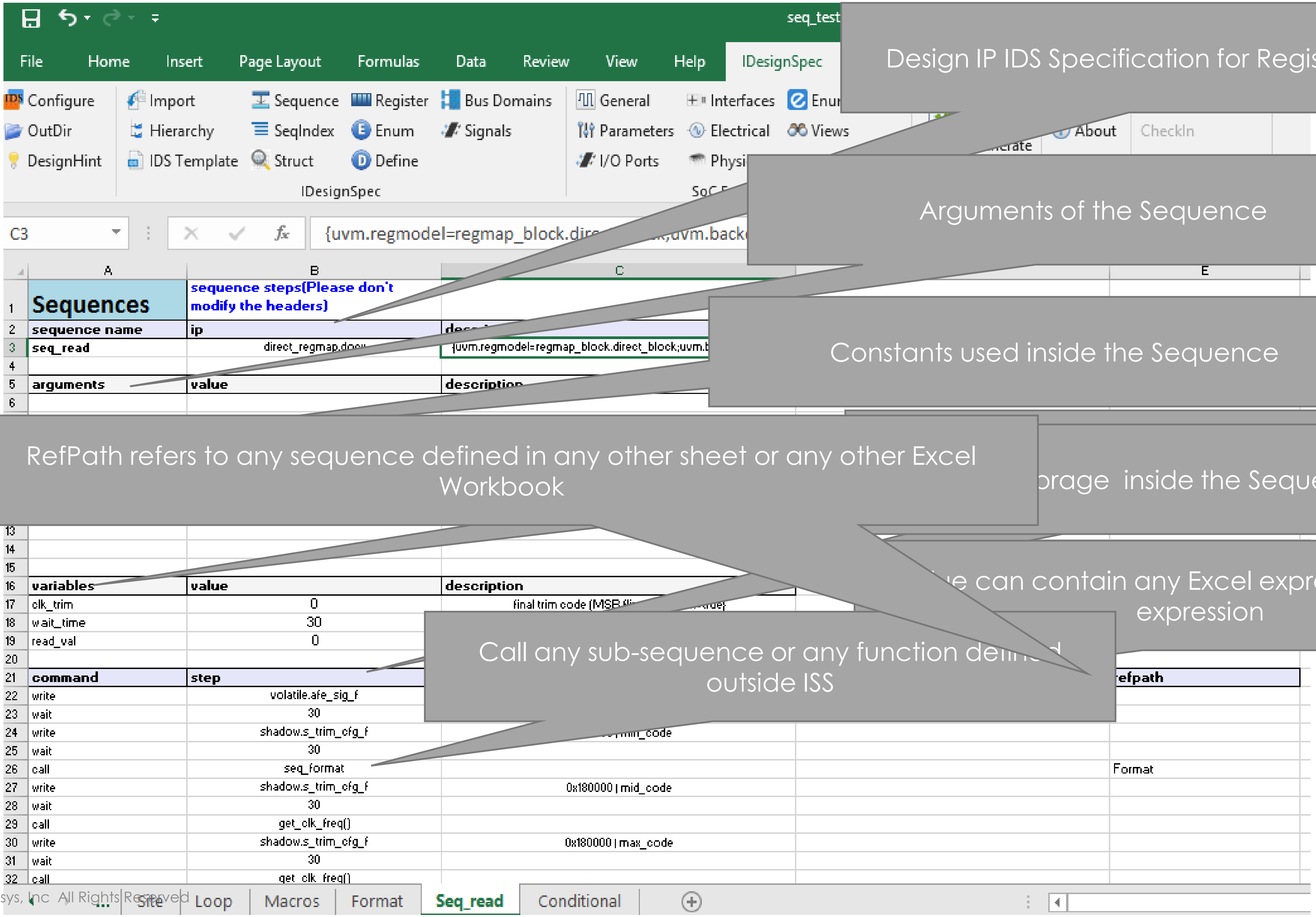
CSV

Matlab

PSS
Portable Stimulus Standard

HTML
PDF

ISS Template



{uvm.regmodel=regmap_block.dire ... ,uvm.back

	A	B	C	E
1	Sequences	sequence steps(Please don't modify the headers)		
2	sequence name	ip	des	
3	seq_read	direct_regmap.doc	{uvm.regmodel=regmap_block.direct_block;uvm.b	
4				
5	arguments	value	description	
6				
13				
14				
15				
16	variables	value	description	
17	clk_trim	0	final trim code (MSB fli	
18	wait_time	30		
19	read_val	0		
20				
21	command	step		efpath
22	write	volatile.afe_sig_f		
23	wait	30		
24	write	shadow.s_trim_cfg_f	min_code	
25	wait	30		
26	call	seq_format		Format
27	write	shadow.s_trim_cfg_f	0x180000 | mid_code	
28	wait	30		
29	call	get_clk_freq()		
30	write	shadow.s_trim_cfg_f	0x180000 | max_code	
31	wait	30		
32	call	get_clk_freq()		

Design IP IDS Specification for Register Map data

Arguments of the Sequence

Constants used inside the Sequence

RefPath refers to any sequence defined in any other sheet or any other Excel Workbook

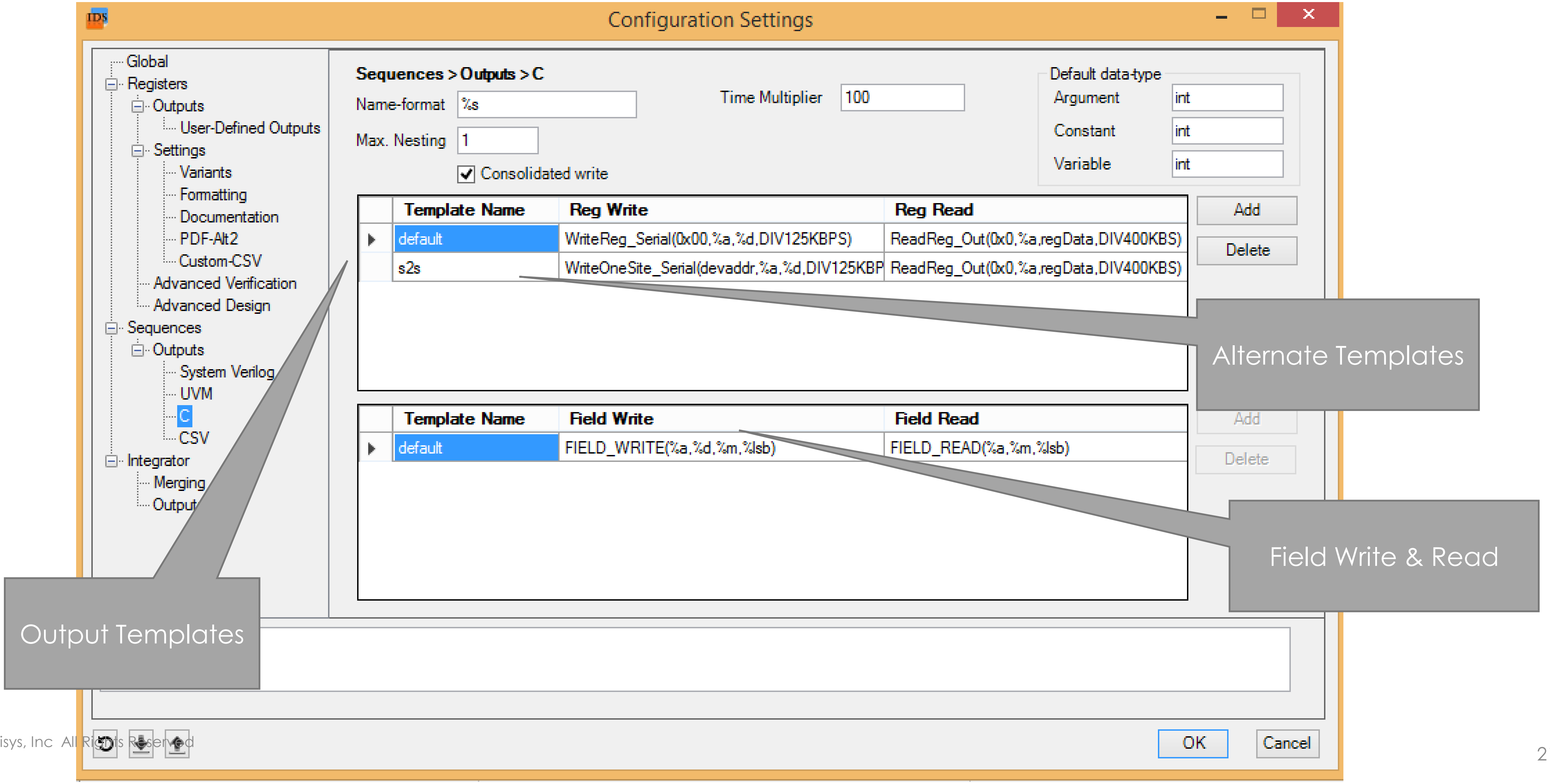
orage inside the Sequence

tion

e can contain any Excel expression OR static expression

Call any sub-sequence or any function defined outside ISS

User Defined Write/Read templates



Sequences > Outputs > C

Name-format: %s

Max. Nesting: 1

☑ Consolidated write

Time Multiplier: 100

Default data-type

Argument: int

Constant: int

Variable: int

Template Name	Reg Write	Reg Read
default	WriteReg_Serial(0x00,%a,%d,DIV125KBPS)	ReadReg_Out(0x0,%a,regData,DIV400KBS)
s2s	WriteOneSite_Serial(devaddr,%a,%d,DIV125KBP	ReadReg_Out(0x0,%a,regData,DIV400KBS)

Template Name	Field Write	Field Read
default	FIELD_WRITE(%a,%d,%m,%lsb)	FIELD_READ(%a,%m,%lsb)

Alternate Templates

Field Write & Read

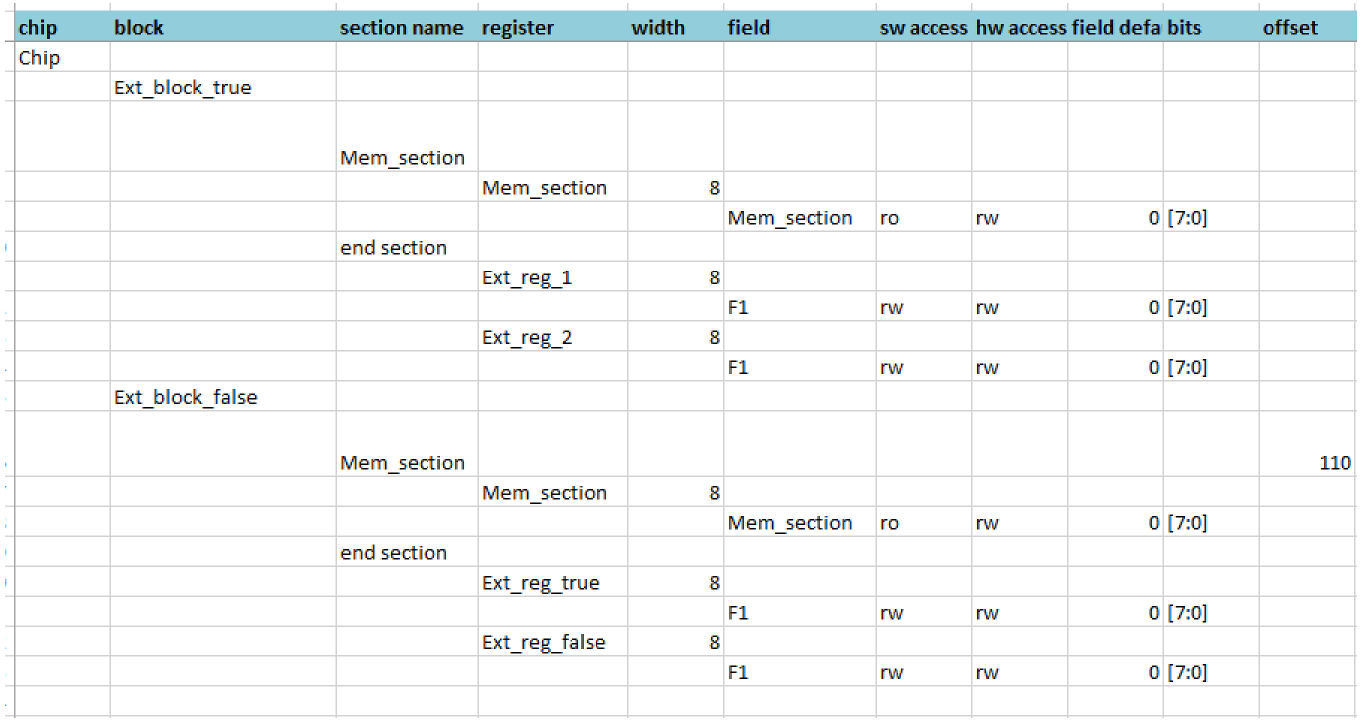
Output Templates

Issues that ISequenceSpec avoids

- Location of register in the register-map
- Location of field in a Register
- Format of the field
- Field access (ro/wo/rw/w1c/r1c/…)
- Consolidation of write / read
- Automatic Read-Modify-Write
- Indirect read/write
- Define and use structures
- Write to multiple chip sites in a tester

Links to IDesignSpec Register Specs

- Sequence is linked to the Design IP
- IDS is already being used to capture Register Spec
- Multiple formats are supported
  - Excel, Word, etc.
  - SystemRDL, IP-XACT
- Register Data can be created in ISS too.

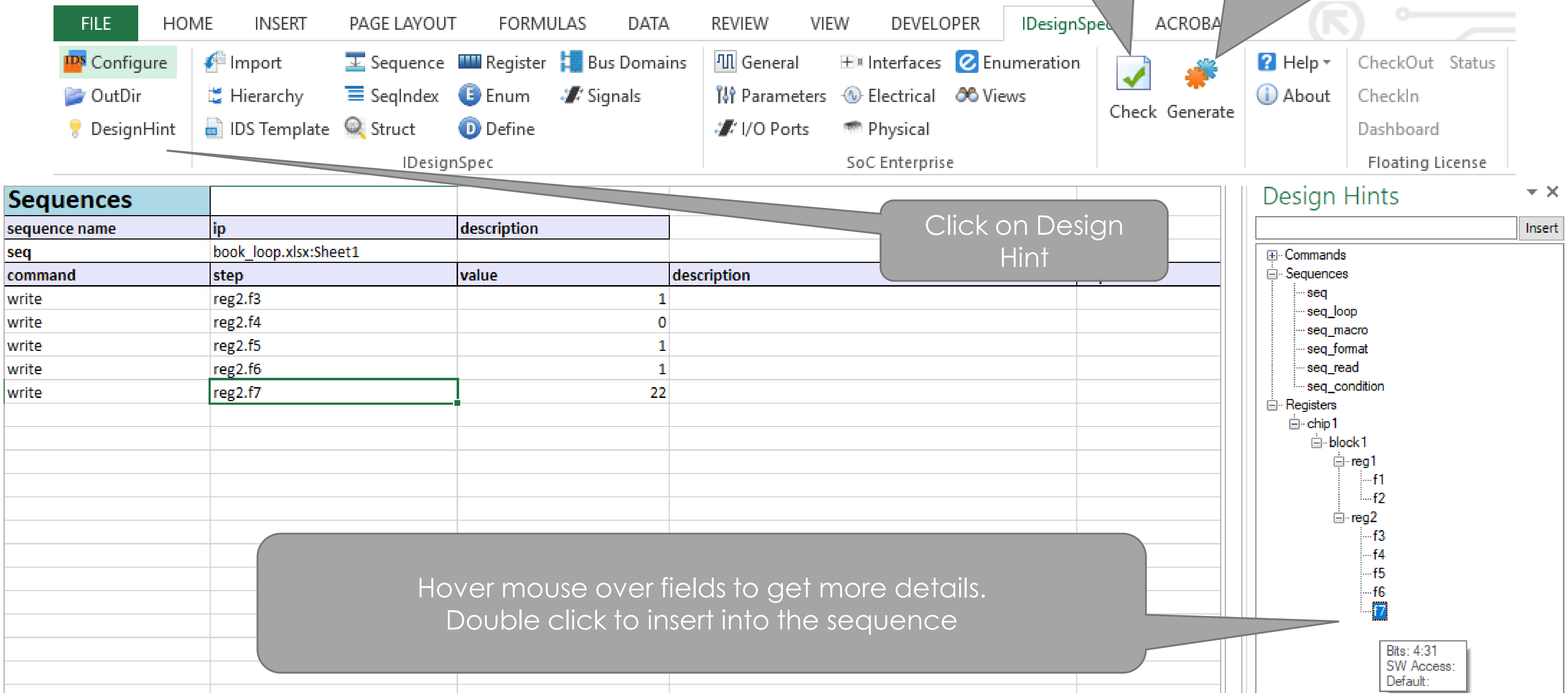


chip	block	section name	register	width	field	sw access	hw access	field defa	bits	offset
Chip										
	Ext_block_true									
		Mem_section								
			Mem_section	8						
					Mem_section	ro	rw	0	[7:0]	
		end section								
			Ext_reg_1	8						
					F1	rw	rw	0	[7:0]	
			Ext_reg_2	8						
					F1	rw	rw	0	[7:0]	
	Ext_block_false									
		Mem_section								110
			Mem_section	8						
					Mem_section	ro	rw	0	[7:0]	
		end section								
			Ext_reg_true	8						
					F1	rw	rw	0	[7:0]	
			Ext_reg_false	8						
					F1	rw	rw	0	[7:0]	

Simple Sequence

Click on Check to check for errors

Click on Generate to create outputs



Sequences			
sequence name	ip	description	
seq	book_loop.xlsx:Sheet1		
command	step	value	description
write	reg2.f3	1	
write	reg2.f4	0	
write	reg2.f5	1	
write	reg2.f6	1	
write	reg2.f7	22	

Click on Design Hint

Hover mouse over fields to get more details.
Double click to insert into the sequence

Simple Sequence Output

C OUTPUT

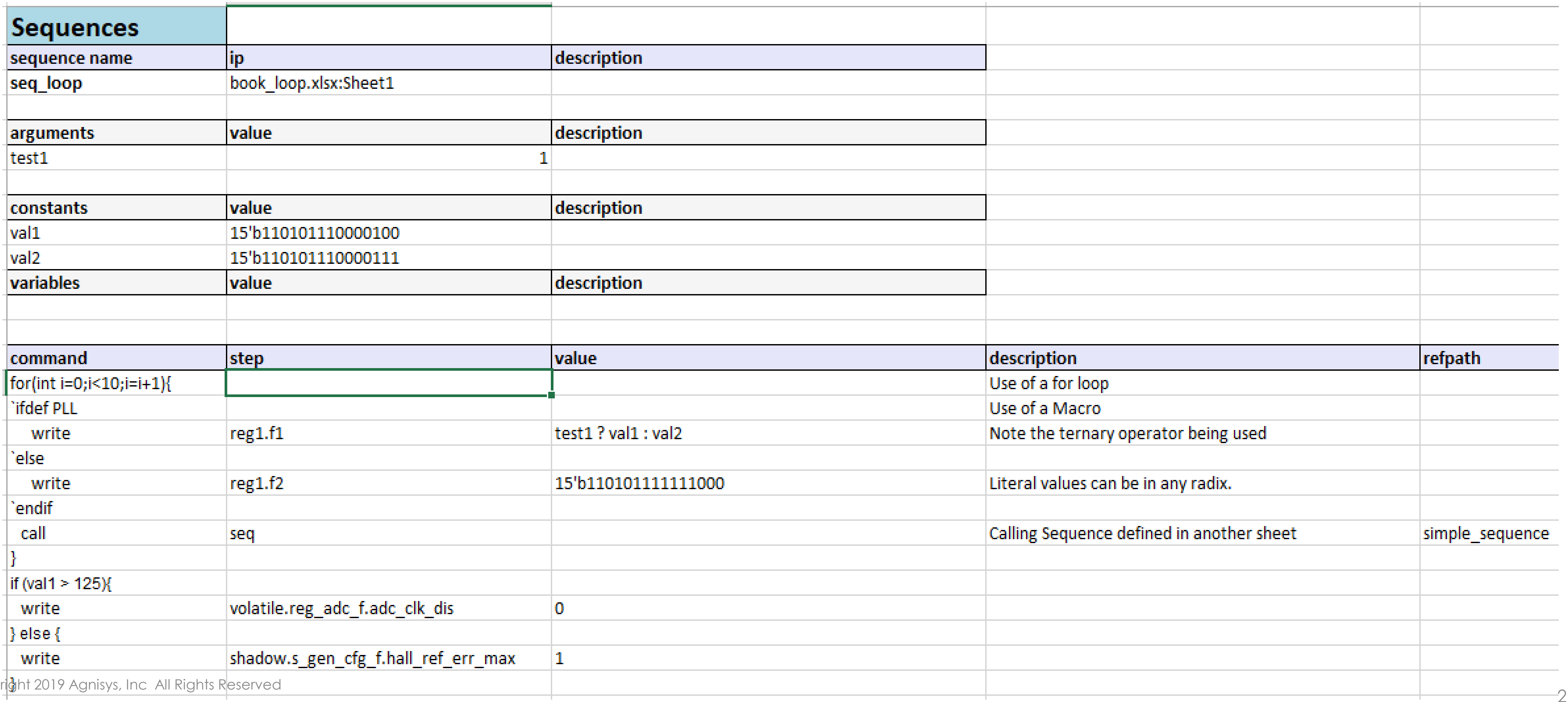
```
void seq( ) {

int consolidated_temp_value = 0;

consolidated_temp_value = ((0x00000001) & …
consolidated_temp_value = ((0x00000000) & …
consolidated_temp_value = ((0x00000004) & …
consolidated_temp_value = ((0x00000008) & …
consolidated_temp_value = ((0x00000160) & …


WriteReg_Serial(0x00,chip1_block1_reg2_ADDRESS,
consolidated_temp_value,DIV125KBPS);

}
```

Comes from the output template

UVM OUTPUT

```
class uvm_seq_seq extends
uvm_reg_sequence#(uvm_sequence#(uvm_reg_item));

`uvm_object_utils(uvm_seq_seq)    uvm_status_e
status;

chip1_block rm ;
:
task body;
:
rm.block1.reg2.f3.set('h1);
rm.block1.reg2.f4.set('h0);
rm.block1.reg2.f5.set('h1);
rm.block1.reg2.f6.set('h1);
rm.block1.reg2.f7.set('h16);
rm.block1.reg2.update(status);
endtask: body
endclass: uvm_seq_seq
```

Conditions, Loops, Sub Sequence, Macros, literal values …



Sequences

sequence name	ip	description
seq_loop	book_loop.xlsx:Sheet1	

arguments	value	description
test1	1	

constants	value	description
val1	15'b110101110000100	
val2	15'b110101110000111	

variables	value	description

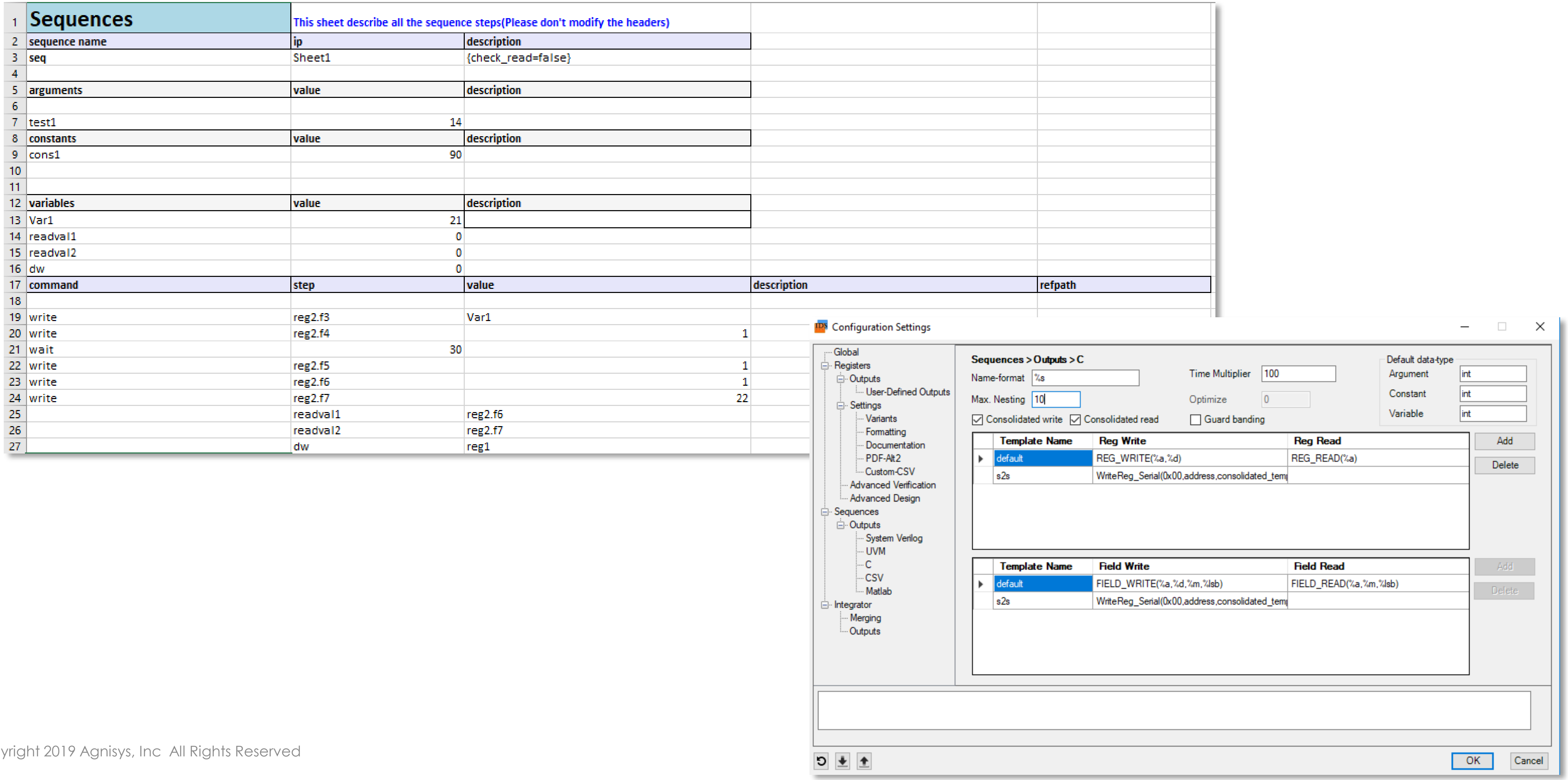
command	step	value	description	refpath
for(int i=0;i<10;i=i+1){			Use of a for loop	
`ifdef PLL			Use of a Macro	
write	reg1.f1	test1 ? val1 : val2	Note the ternary operator being used	
`else				
write	reg1.f2	15'b110101111111000	Literal values can be in any radix.	
`endif				
call	seq		Calling Sequence defined in another sheet	simple_sequence
}				
if (val1 > 125){				
write	volatile.reg_adc_f.adc_clk_dis	0		
} else {				
write	shadow.s_gen_cfg_f.hall_ref_err_max	1		
}				

Consolidated Write

- Several field writes can be consolidated into a Reg write
- Tool will determine locations of fields and registers
- This will save time on the tester
- Example :

```
        Register1.Field1 = 0x10
        Register1.Field2 = 0x20
        Register1.Field3 = 0xA0
    Is equivalent to
        Register1 = 0xA02010
```

Consolidated Read and Write



	A	B	C	D	E
1	Sequences	This sheet describe all the sequence steps(Please don't modify the headers)			
2	sequence name	ip	description		
3	seq	Sheet1	{check_read=false}		
4					
5	arguments	value	description		
6					
7	test1	14			
8	constants	value	description		
9	cons1	90			
10					
11					
12	variables	value	description		
13	Var1	21			
14	readval1	0			
15	readval2	0			
16	dw	0			
17	command	step	value	description	refpath
18					
19	write	reg2.f3	Var1		
20	write	reg2.f4	1		
21	wait	30			
22	write	reg2.f5	1		
23	write	reg2.f6	1		
24	write	reg2.f7	22		
25		readval1	reg2.f6		
26		readval2	reg2.f7		
27		dw	reg1		

Configuration Settings

Sequences > Outputs > C

Name-format: %s

Max. Nesting: 10

Time Multiplier: 100

Optimize: 0

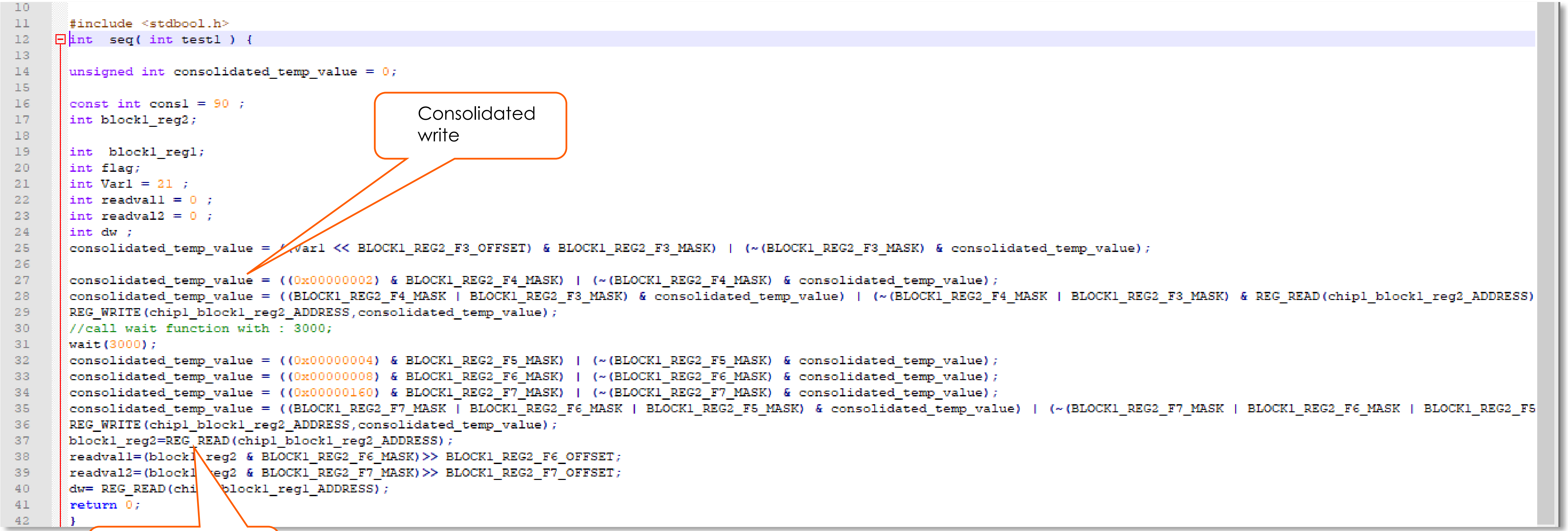
Default data-type
Argument: int
Constant: int
Variable: int

☑ Consolidated write ☑ Consolidated read ☐ Guard banding

Template Name	Reg Write	Reg Read
default	REG_WRITE(%a,%d)	REG_READ(%a)
s2s	WriteReg_Serial(0x00,address,consolidated_tem	

Template Name	Field Write	Field Read
default	FIELD_WRITE(%a,%d,%m,%lsb)	FIELD_READ(%a,%m,%lsb)
s2s	WriteReg_Serial(0x00,address,consolidated_tem	

C Output

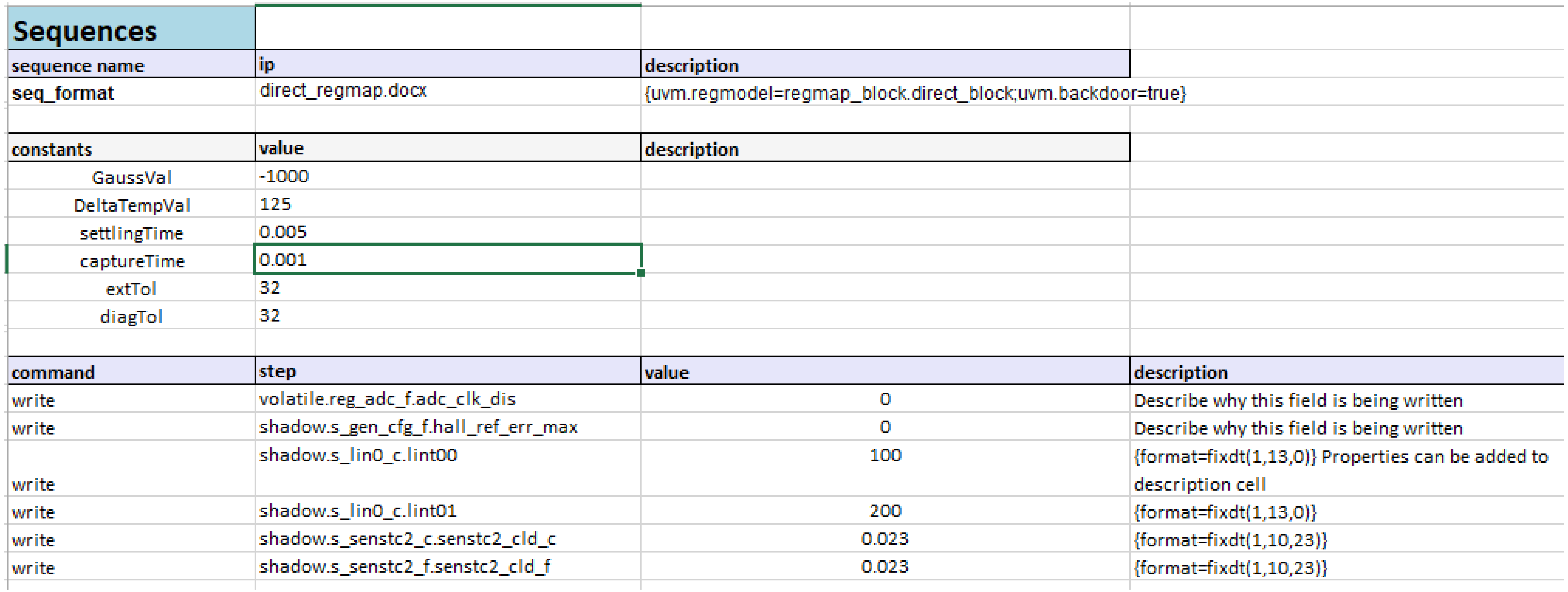


```c

#include <stdbool.h>
int  seq( int test1 ) {

unsigned int consolidated_temp_value = 0;

const int cons1 = 90 ;
int block1_reg2;

int  block1_reg1;
int flag;
int Var1 = 21 ;
int readval1 = 0 ;
int readval2 = 0 ;
int dw ;
consolidated_temp_value = ((var1 << BLOCK1_REG2_F3_OFFSET) & BLOCK1_REG2_F3_MASK) | (~(BLOCK1_REG2_F3_MASK) & consolidated_temp_value);

consolidated_temp_value = ((0x00000002) & BLOCK1_REG2_F4_MASK) | (~(BLOCK1_REG2_F4_MASK) & consolidated_temp_value);
consolidated_temp_value = ((BLOCK1_REG2_F4_MASK | BLOCK1_REG2_F3_MASK) & consolidated_temp_value) | (~(BLOCK1_REG2_F4_MASK | BLOCK1_REG2_F3_MASK) & REG_READ(chip1_block1_reg2_ADDRESS)
REG_WRITE(chip1_block1_reg2_ADDRESS,consolidated_temp_value);
//call wait function with : 3000;
wait(3000);
consolidated_temp_value = ((0x00000004) & BLOCK1_REG2_F5_MASK) | (~(BLOCK1_REG2_F5_MASK) & consolidated_temp_value);
consolidated_temp_value = ((0x00000008) & BLOCK1_REG2_F6_MASK) | (~(BLOCK1_REG2_F6_MASK) & consolidated_temp_value);
consolidated_temp_value = ((0x00000160) & BLOCK1_REG2_F7_MASK) | (~(BLOCK1_REG2_F7_MASK) & consolidated_temp_value);
consolidated_temp_value = ((BLOCK1_REG2_F7_MASK | BLOCK1_REG2_F6_MASK | BLOCK1_REG2_F5_MASK) & consolidated_temp_value) | (~(BLOCK1_REG2_F7_MASK | BLOCK1_REG2_F6_MASK | BLOCK1_REG2_F5
REG_WRITE(chip1_block1_reg2_ADDRESS,consolidated_temp_value);
block1_reg2=REG_READ(chip1_block1_reg2_ADDRESS);
readval1=(block1_reg2 & BLOCK1_REG2_F6_MASK)>> BLOCK1_REG2_F6_OFFSET;
readval2=(block1_reg2 & BLOCK1_REG2_F7_MASK)>> BLOCK1_REG2_F7_OFFSET;
dw= REG_READ(chip1_block1_reg1_ADDRESS);
return 0;
}
```

Consolidated write

Consolidated read

Formats of fields

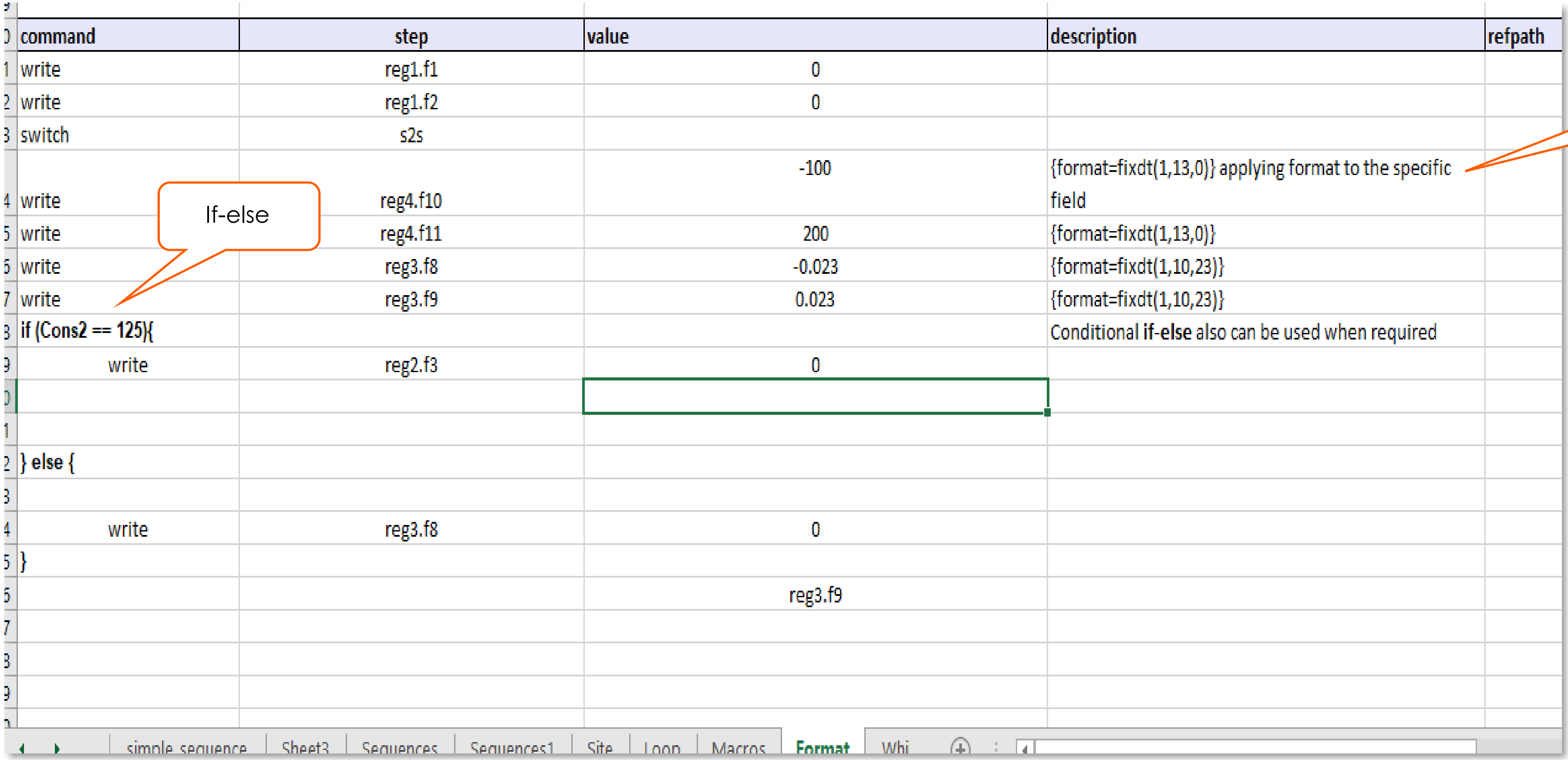


Sequences			
sequence name	**ip**	**description**	
seq_format	direct_regmap.docx	{uvm.regmodel=regmap_block.direct_block;uvm.backdoor=true}	
constants	**value**	**description**	
GaussVal	-1000		
DeltaTempVal	125		
settlingTime	0.005		
captureTime	0.001		
extTol	32		
diagTol	32		
command	**step**	**value**	**description**
write	volatile.reg_adc_f.adc_clk_dis	0	Describe why this field is being written
write	shadow.s_gen_cfg_f.hall_ref_err_max	0	Describe why this field is being written
write	shadow.s_lin0_c.lint00	100	{format=fixdt(1,13,0)} Properties can be added to description cell
write	shadow.s_lin0_c.lint01	200	{format=fixdt(1,13,0)}
write	shadow.s_senstc2_c.senstc2_cld_c	0.023	{format=fixdt(1,10,23)}
write	shadow.s_senstc2_f.senstc2_cld_f	0.023	{format=fixdt(1,10,23)}

Automatic Read-Modify-Write

- Do Read-Modify-Write only if required
- If only some of the writeable fields are written, then ISS will automatically do a Read-Modify-Write
  - So that the value is preserved for the fields that are not written.
- For read-only fields, it doesn’t have to preserve the value, because they cannot be modified by writing any value to it

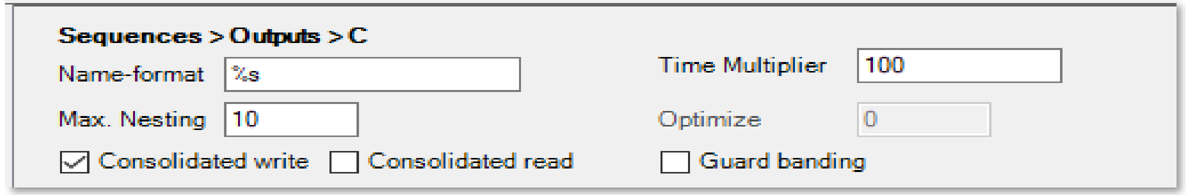
Format



command	step	value	description	refpath
write	reg1.f1	0		
write	reg1.f2	0		
switch	s2s			
write	reg4.f10	-100	{format=fixdt(1,13,0)} applying format to the specific field	
write	reg4.f11	200	{format=fixdt(1,13,0)}	
write	reg3.f8	-0.023	{format=fixdt(1,10,23)}	
write	reg3.f9	0.023	{format=fixdt(1,10,23)}	
if (Cons2 == 125){			Conditional **if-else** also can be used when required	
write	reg2.f3	0		
} else {				
write	reg3.f8	0		
}				
		reg3.f9		

If-else

Format



Sequences > Outputs > C

Name-format: %s

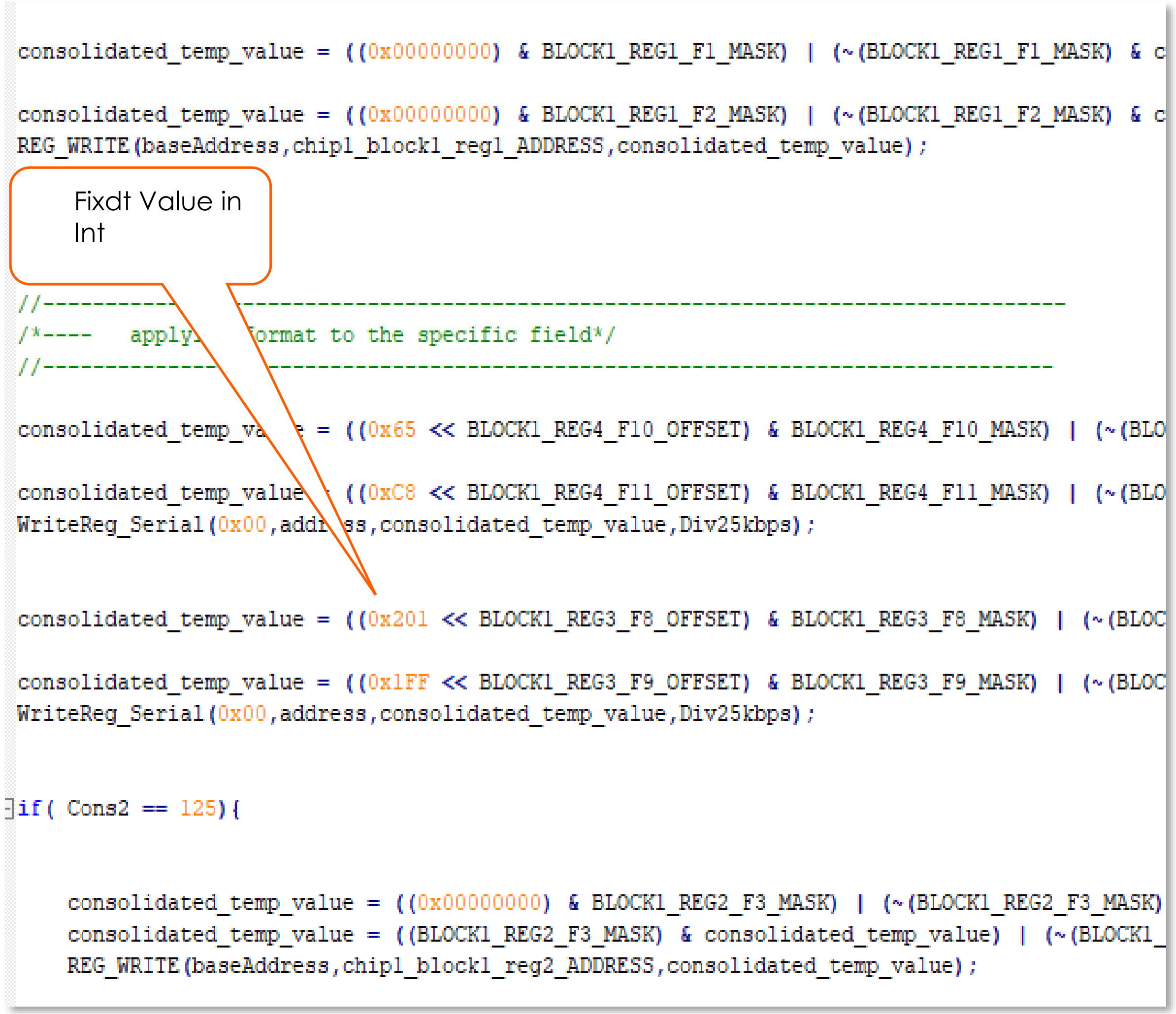
Time Multiplier: 100

Max. Nesting: 10

Optimize: 0

☑ Consolidated write ☐ Consolidated read ☐ Guard banding

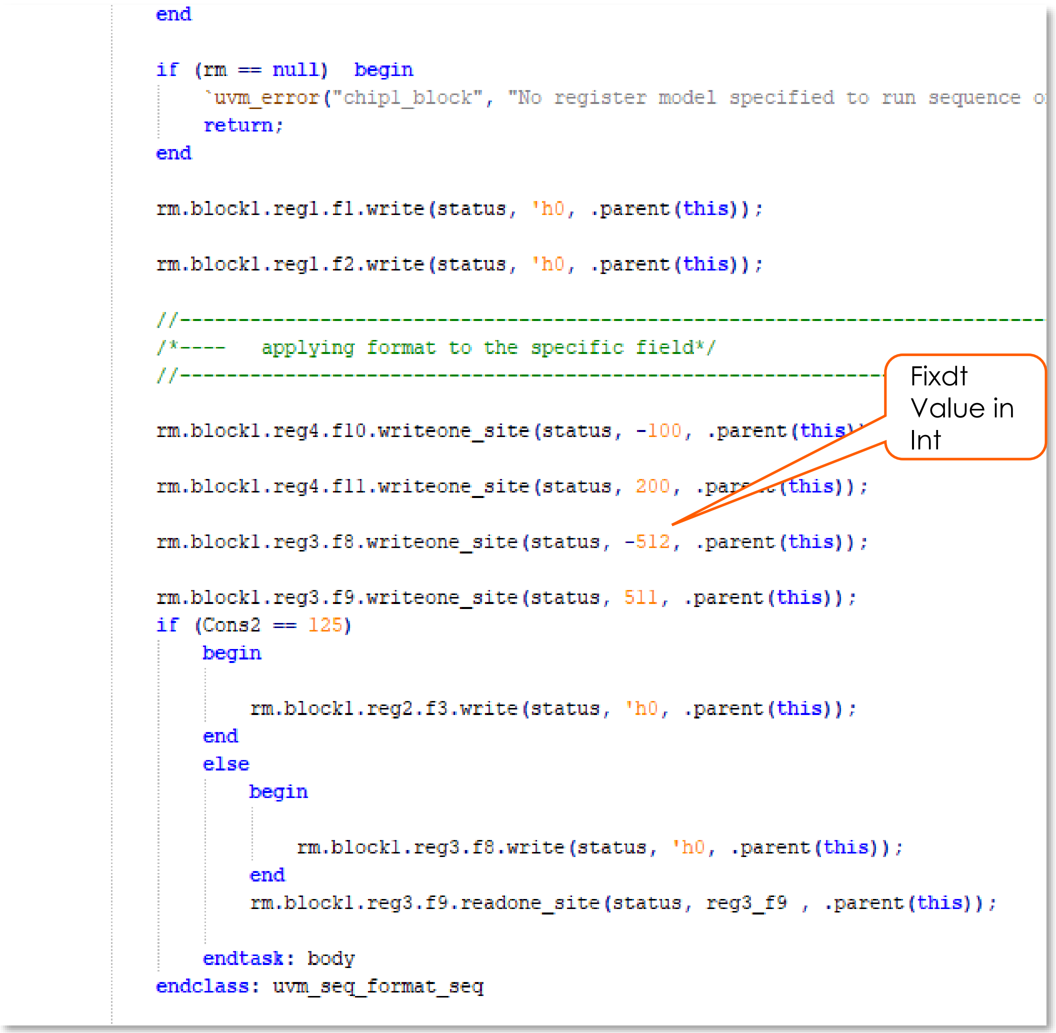
C Output



```
consolidated_temp_value = ((0x00000000) & BLOCK1_REG1_F1_MASK) | (~(BLOCK1_REG1_F1_MASK) & c

consolidated_temp_value = ((0x00000000) & BLOCK1_REG1_F2_MASK) | (~(BLOCK1_REG1_F2_MASK) & c
REG_WRITE(baseAddress,chip1_block1_reg1_ADDRESS,consolidated_temp_value);



//-----------------------------------------------------------------------------
/*----   apply   format to the specific field*/
//-----------------------------------------------------------------------------

consolidated_temp_va   e = ((0x65 << BLOCK1_REG4_F10_OFFSET) & BLOCK1_REG4_F10_MASK) | (~(BLO

consolidated_temp_value   ((0xC8 << BLOCK1_REG4_F11_OFFSET) & BLOCK1_REG4_F11_MASK) | (~(BLO
WriteReg_Serial(0x00,addr  ss,consolidated_temp_value,Div25kbps);


consolidated_temp_value = ((0x201 << BLOCK1_REG3_F8_OFFSET) & BLOCK1_REG3_F8_MASK) | (~(BLOC

consolidated_temp_value = ((0x1FF << BLOCK1_REG3_F9_OFFSET) & BLOCK1_REG3_F9_MASK) | (~(BLOC
WriteReg_Serial(0x00,address,consolidated_temp_value,Div25kbps);


if( Cons2 == 125){


    consolidated_temp_value = ((0x00000000) & BLOCK1_REG2_F3_MASK) | (~(BLOCK1_REG2_F3_MASK)
    consolidated_temp_value = ((BLOCK1_REG2_F3_MASK) & consolidated_temp_value) | (~(BLOCK1_
    REG_WRITE(baseAddress,chip1_block1_reg2_ADDRESS,consolidated_temp_value);
```

Fixdt Value in Int

UVM Output

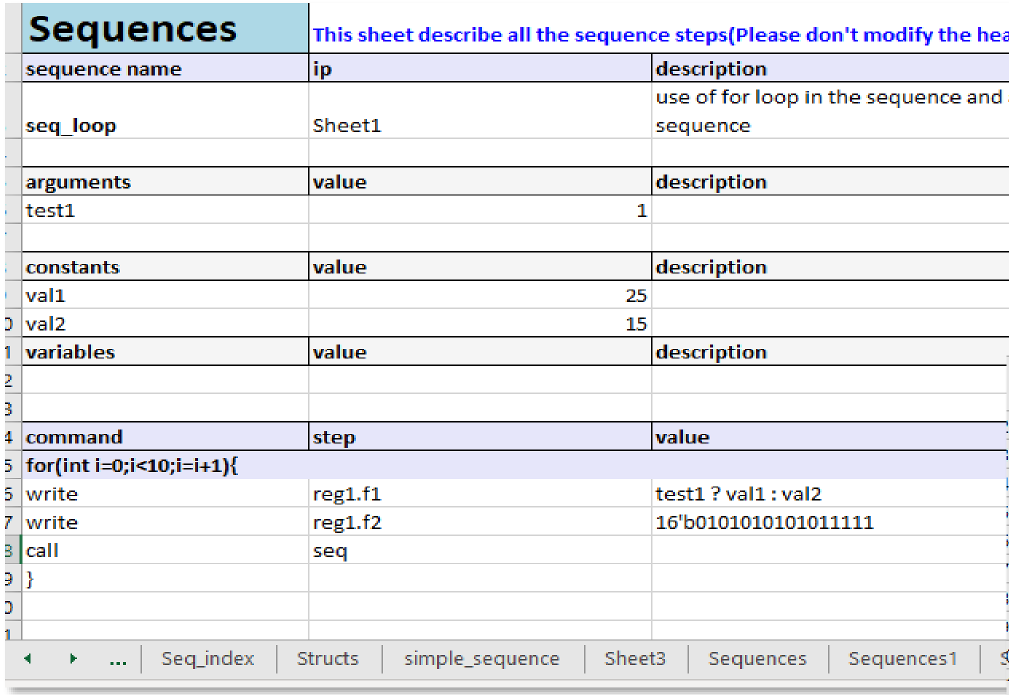


```
end

if (rm == null)  begin
    `uvm_error("chip1_block", "No register model specified to run sequence o
    return;
end

rm.block1.reg1.f1.write(status, 'h0, .parent(this));

rm.block1.reg1.f2.write(status, 'h0, .parent(this));

//----------------------------------------------------------------------------
/*----   applying format to the specific field*/
//---------------------------------------------------------------------

rm.block1.reg4.f10.writeone_site(status, -100, .parent(this)

rm.block1.reg4.f11.writeone_site(status, 200, .par   t(this));

rm.block1.reg3.f8.writeone_site(status, -512, .parent(this));

rm.block1.reg3.f9.writeone_site(status, 511, .parent(this));
if (Cons2 == 125)
    begin

        rm.block1.reg2.f3.write(status, 'h0, .parent(this));
    end
    else
        begin

            rm.block1.reg3.f8.write(status, 'h0, .parent(this));
        end
        rm.block1.reg3.f9.readone_site(status, reg3_f9 , .parent(this));

    endtask: body
endclass: uvm_seq_format_seq
```

Fixdt Value in Int

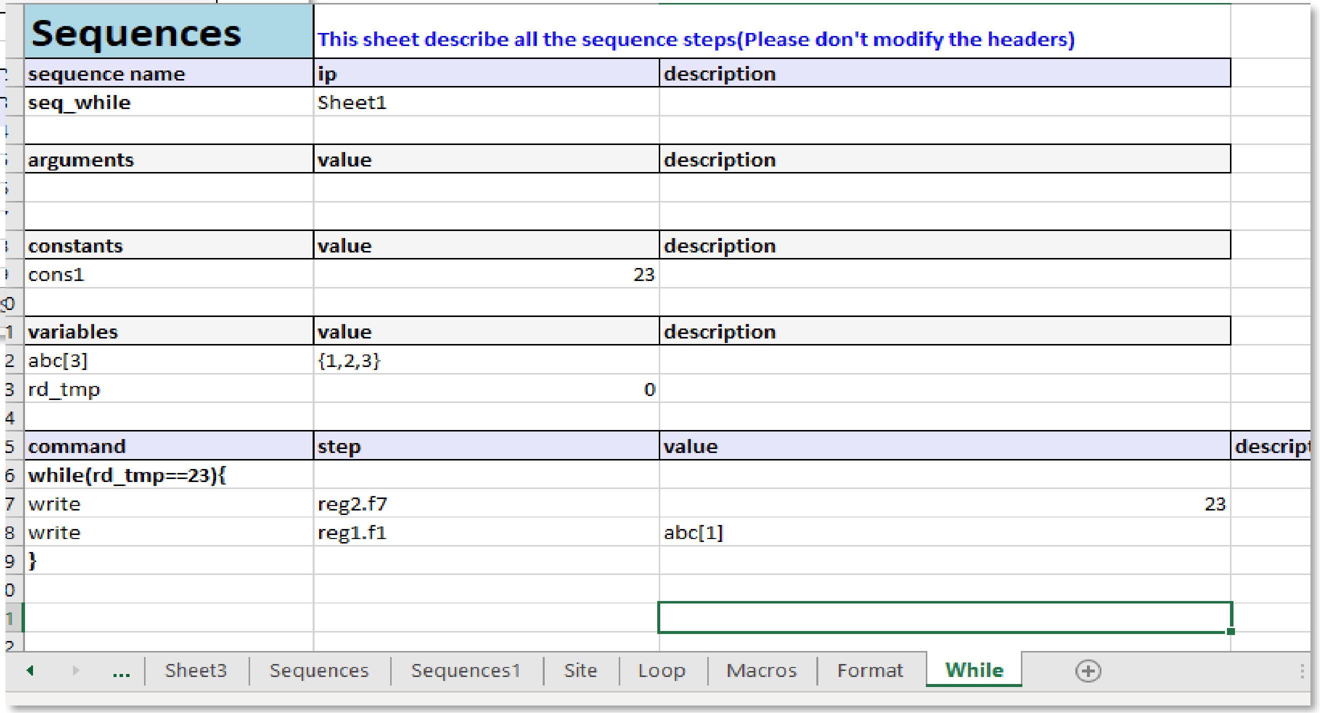
Loops “for” and “while”

For Loop



Sequences	This sheet describe all the sequence steps(Please don't modify the headers)	
sequence name	ip	description
seq_loop	Sheet1	use of for loop in the sequence and also calling another sequence
arguments	value	description
test1	1	
constants	value	description
val1	25	
val2	15	
variables	value	description
command	step	value
for(int i=0;i<10;i=i+1){		
write	reg1.f1	test1 ? val1 : val2
write	reg1.f2	16'b0101010101011111
call	seq	
}		

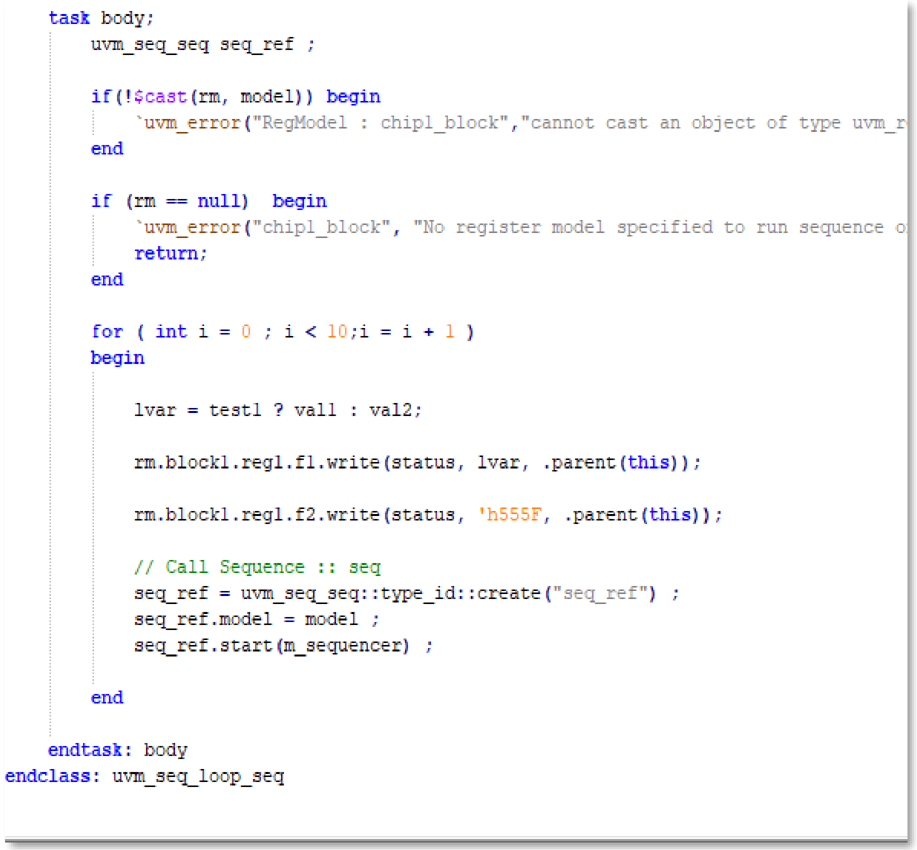
While Loop



Sequences	This sheet describe all the sequence steps(Please don't modify the headers)		
sequence name	ip	description	
seq_while	Sheet1		
arguments	value	description	
constants	value	description	
cons1	23		
variables	value	description	
abc[3]	{1,2,3}		
rd_tmp	0		
command	step	value	descript
while(rd_tmp==23){			
write	reg2.f7	23	
write	reg1.f1	abc[1]	
}			

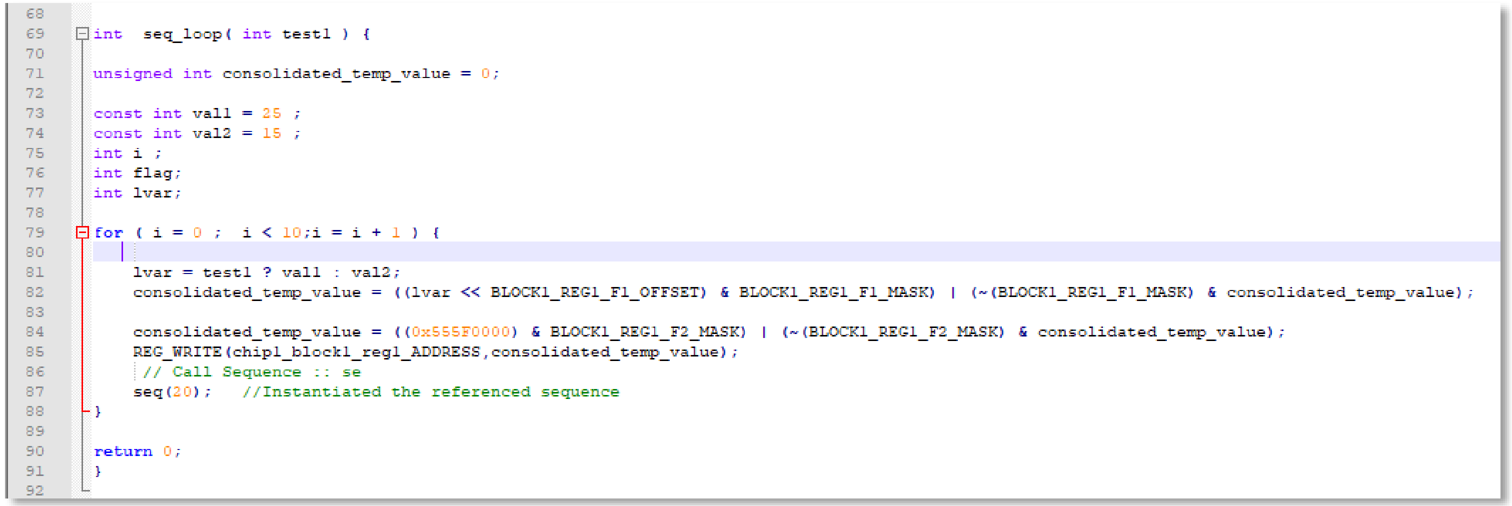
Outputs generated for “for” loop

UVM based output



```
    task body;
        uvm_seq_seq seq_ref ;

        if(!$cast(rm, model)) begin
            `uvm_error("RegModel : chip1_block","cannot cast an object of type uvm_r
        end

        if (rm == null)  begin
            `uvm_error("chip1_block", "No register model specified to run sequence o
            return;
        end

        for ( int i = 0 ; i < 10;i = i + 1 )
        begin

            lvar = test1 ? val1 : val2;

            rm.block1.reg1.f1.write(status, lvar, .parent(this));

            rm.block1.reg1.f2.write(status, 'h555F, .parent(this));

            // Call Sequence :: seq
            seq_ref = uvm_seq_seq::type_id::create("seq_ref") ;
            seq_ref.model = model ;
            seq_ref.start(m_sequencer) ;

        end

    endtask: body
endclass: uvm_seq_loop_seq
```

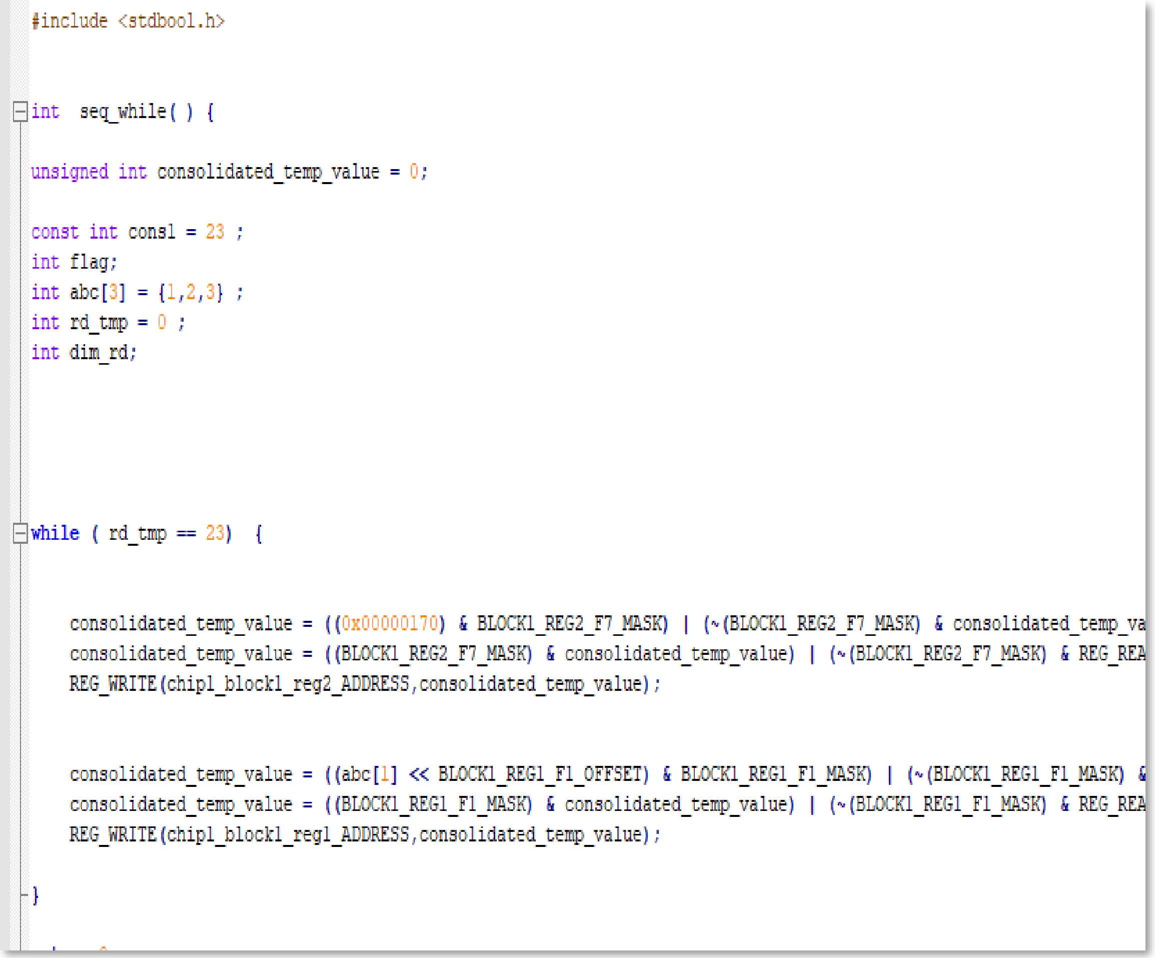
C based output



```
int  seq_loop( int test1 ) {

unsigned int consolidated_temp_value = 0;

const int val1 = 25 ;
const int val2 = 15 ;
int i ;
int flag;
int lvar;

for ( i = 0 ;  i < 10;i = i + 1 ) {

    lvar = test1 ? val1 : val2;
    consolidated_temp_value = ((lvar << BLOCK1_REG1_F1_OFFSET) & BLOCK1_REG1_F1_MASK) | (~(BLOCK1_REG1_F1_MASK) & consolidated_temp_value);

    consolidated_temp_value = ((0x555F0000) & BLOCK1_REG1_F2_MASK) | (~(BLOCK1_REG1_F2_MASK) & consolidated_temp_value);
    REG_WRITE(chip1_block1_reg1_ADDRESS,consolidated_temp_value);
     // Call Sequence :: se
    seq(20);   //Instantiated the referenced sequence
}

return 0;
}
```

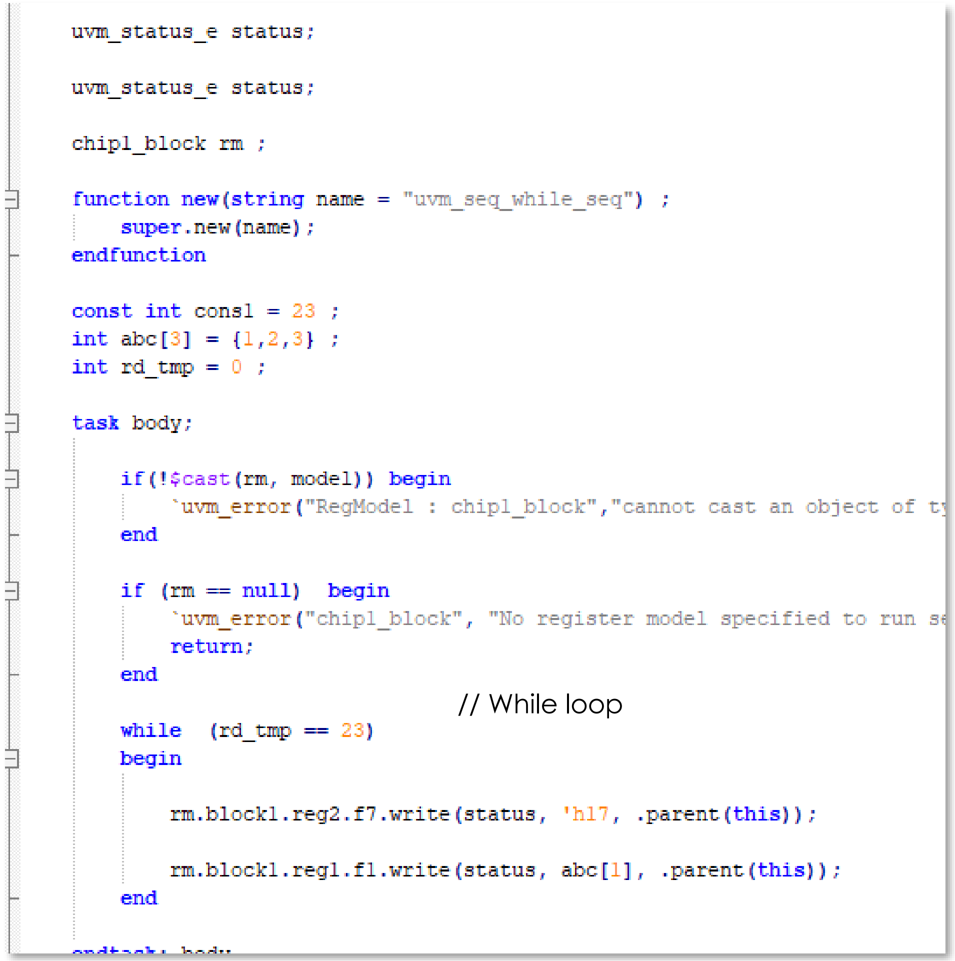
Outputs generated for “while” loop

C based output



```
#include <stdbool.h>


int  seq_while( ) {

unsigned int consolidated_temp_value = 0;

const int cons1 = 23 ;
int flag;
int abc[3] = {1,2,3} ;
int rd_tmp = 0 ;
int dim_rd;




while ( rd_tmp == 23)  {


    consolidated_temp_value = ((0x00000170) & BLOCK1_REG2_F7_MASK) | (~(BLOCK1_REG2_F7_MASK) & consolidated_temp_va
    consolidated_temp_value = ((BLOCK1_REG2_F7_MASK) & consolidated_temp_value) | (~(BLOCK1_REG2_F7_MASK) & REG_REA
    REG_WRITE(chip1_block1_reg2_ADDRESS,consolidated_temp_value);


    consolidated_temp_value = ((abc[1] << BLOCK1_REG1_F1_OFFSET) & BLOCK1_REG1_F1_MASK) | (~(BLOCK1_REG1_F1_MASK) &
    consolidated_temp_value = ((BLOCK1_REG1_F1_MASK) & consolidated_temp_value) | (~(BLOCK1_REG1_F1_MASK) & REG_REA
    REG_WRITE(chip1_block1_reg1_ADDRESS,consolidated_temp_value);

}
```

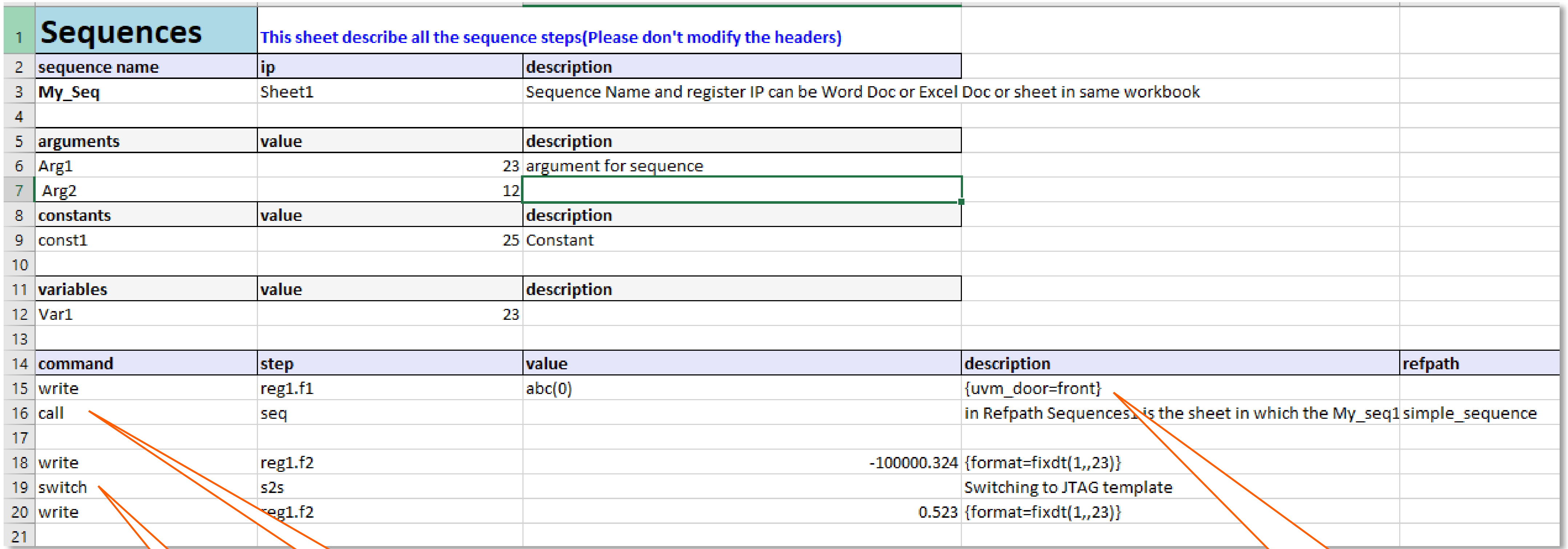
UVM based output



```
uvm_status_e status;

uvm_status_e status;

chip1_block rm ;

function new(string name = "uvm_seq_while_seq") ;
    super.new(name);
endfunction

const int cons1 = 23 ;
int abc[3] = {1,2,3} ;
int rd_tmp = 0 ;

task body;

    if(!$cast(rm, model)) begin
        `uvm_error("RegModel : chip1_block","cannot cast an object of ty
    end

    if (rm == null)  begin
        `uvm_error("chip1_block", "No register model specified to run se
        return;
    end

    while  (rd_tmp == 23)
    begin

        rm.block1.reg2.f7.write(status, 'h17, .parent(this));

        rm.block1.reg1.f1.write(status, abc[1], .parent(this));
    end

endtask: body
```

// While loop

Properties, Call and Switch



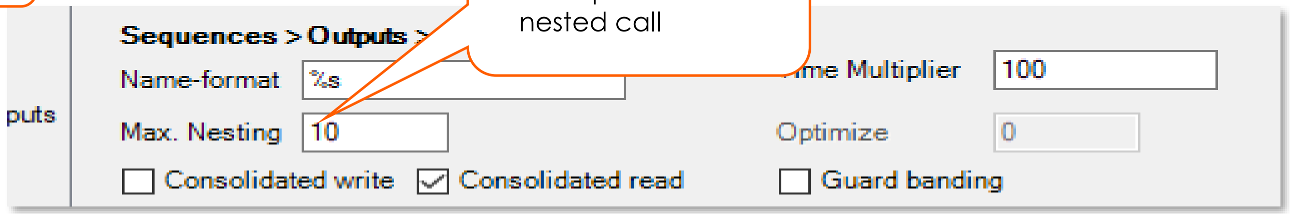
1	**Sequences**	This sheet describe all the sequence steps(Please don't modify the headers)			
2	**sequence name**	**ip**	**description**		
3	**My_Seq**	Sheet1	Sequence Name and register IP can be Word Doc or Excel Doc or sheet in same workbook		
4					
5	**arguments**	**value**	**description**		
6	Arg1	23	argument for sequence		
7	Arg2	12			
8	**constants**	**value**	**description**		
9	const1	25	Constant		
10					
11	**variables**	**value**	**description**		
12	Var1	23			
13					
14	**command**	**step**	**value**	**description**	**refpath**
15	write	reg1.f1	abc(0)	{uvm_door=front}	
16	call	seq		in Refpath Sequences1 is the sheet in which the My_seq1	simple_sequence
17					
18	write	reg1.f2	-100000.324	{format=fixdt(1,,23)}	
19	switch	s2s		Switching to JTAG template	
20	write	reg1.f2	0.523	{format=fixdt(1,,23)}	
21					

UVM property

Swiching to jtag

Call sequence

Maximum number of sequence for nested call



puts

Sequences > Outputs >

Name-format: %s

Max. Nesting: 10

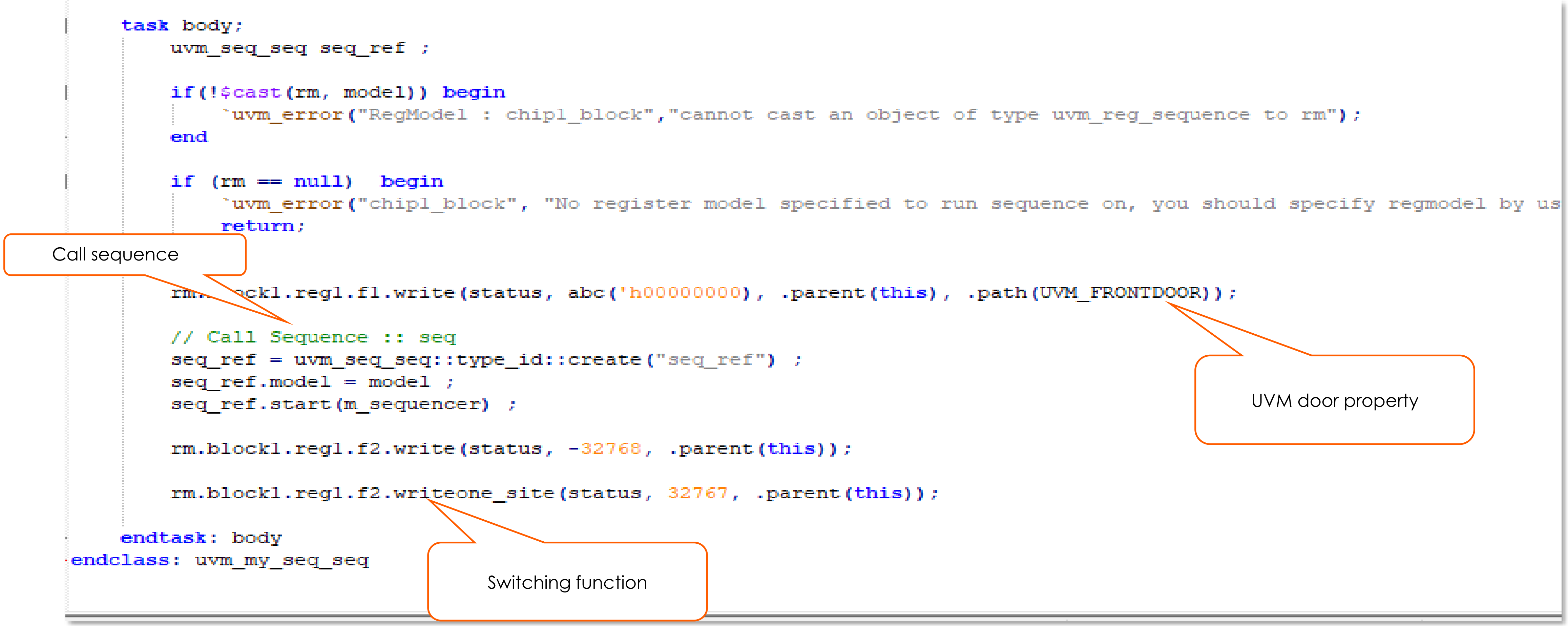
me Multiplier: 100

Optimize: 0

□ Consolidated write ☑ Consolidated read □ Guard banding

UVM Output

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY



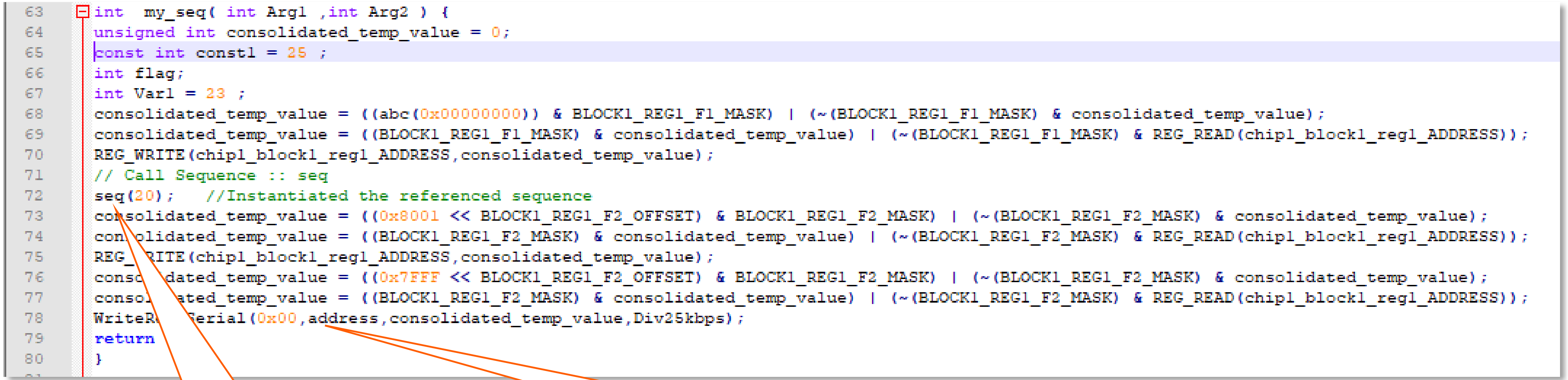
```
    task body;
        uvm_seq_seq seq_ref ;

        if(!$cast(rm, model)) begin
            `uvm_error("RegModel : chip1_block","cannot cast an object of type uvm_reg_sequence to rm");
        end

        if (rm == null)  begin
            `uvm_error("chip1_block", "No register model specified to run sequence on, you should specify regmodel by us
            return;


        rm.block1.reg1.f1.write(status, abc('h00000000), .parent(this), .path(UVM_FRONTDOOR));

        // Call Sequence :: seq
        seq_ref = uvm_seq_seq::type_id::create("seq_ref") ;
        seq_ref.model = model ;
        seq_ref.start(m_sequencer) ;

        rm.block1.reg1.f2.write(status, -32768, .parent(this));

        rm.block1.reg1.f2.writeone_site(status, 32767, .parent(this));

    endtask: body
endclass: uvm_my_seq_seq
```

Call sequence

UVM door property

Switching function

C Output



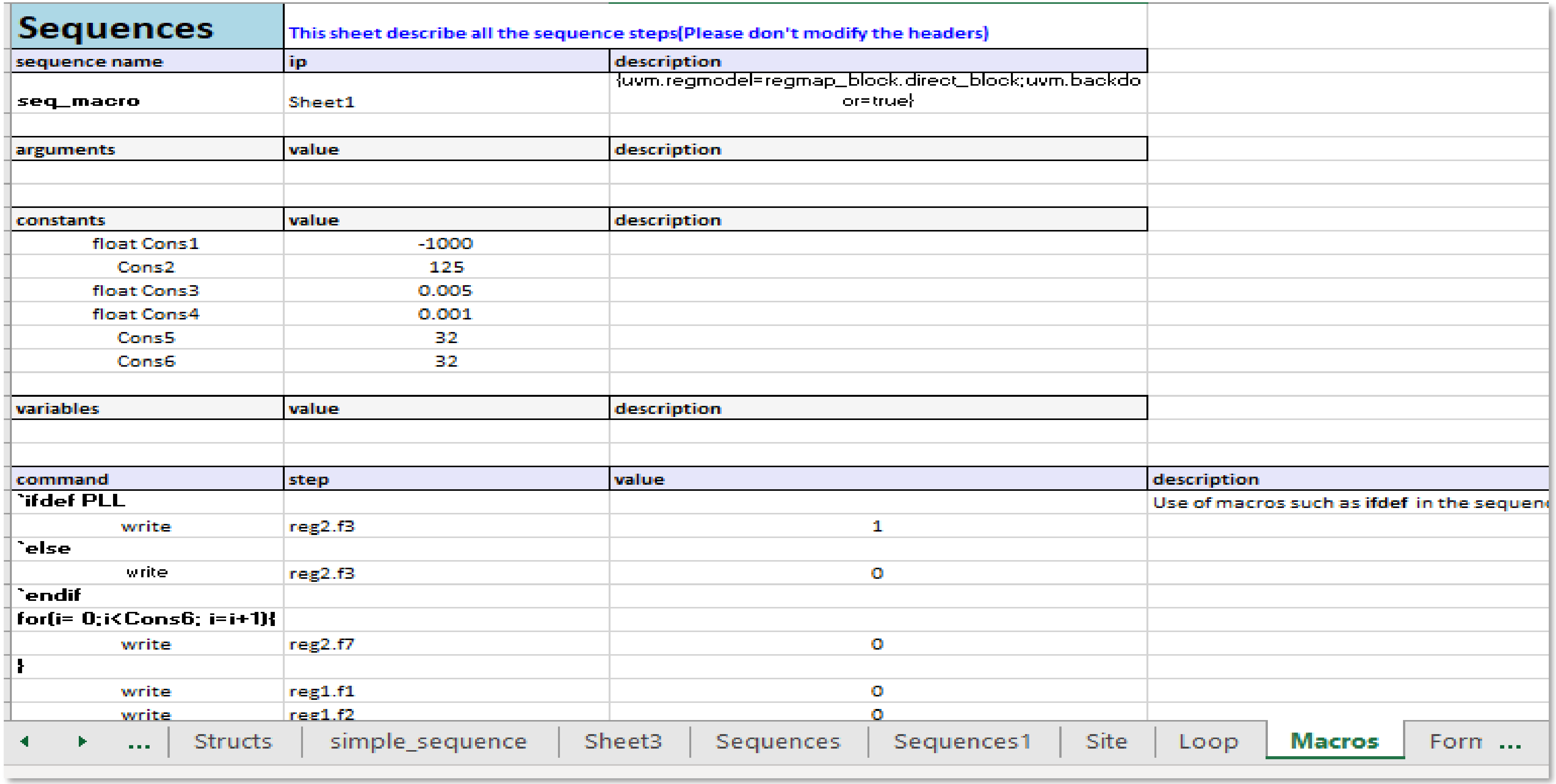
```c
int  my_seq( int Arg1 ,int Arg2 ) {
unsigned int consolidated_temp_value = 0;
const int const1 = 25 ;
int flag;
int Var1 = 23 ;
consolidated_temp_value = ((abc(0x00000000)) & BLOCK1_REG1_F1_MASK) | (~(BLOCK1_REG1_F1_MASK) & consolidated_temp_value);
consolidated_temp_value = ((BLOCK1_REG1_F1_MASK) & consolidated_temp_value) | (~(BLOCK1_REG1_F1_MASK) & REG_READ(chip1_block1_reg1_ADDRESS));
REG_WRITE(chip1_block1_reg1_ADDRESS,consolidated_temp_value);
// Call Sequence :: seq
seq(20);   //Instantiated the referenced sequence
consolidated_temp_value = ((0x8001 << BLOCK1_REG1_F2_OFFSET) & BLOCK1_REG1_F2_MASK) | (~(BLOCK1_REG1_F2_MASK) & consolidated_temp_value);
consolidated_temp_value = ((BLOCK1_REG1_F2_MASK) & consolidated_temp_value) | (~(BLOCK1_REG1_F2_MASK) & REG_READ(chip1_block1_reg1_ADDRESS));
REG_WRITE(chip1_block1_reg1_ADDRESS,consolidated_temp_value);
consolidated_temp_value = ((0x7FFF << BLOCK1_REG1_F2_OFFSET) & BLOCK1_REG1_F2_MASK) | (~(BLOCK1_REG1_F2_MASK) & consolidated_temp_value);
consolidated_temp_value = ((BLOCK1_REG1_F2_MASK) & consolidated_temp_value) | (~(BLOCK1_REG1_F2_MASK) & REG_READ(chip1_block1_reg1_ADDRESS));
WriteReg_Serial(0x00,address,consolidated_temp_value,Div25kbps);
return
}
```

Sequence call

After use of switch, function WriteReg_Serial in place of FIELD_WRITE

Use of Macros

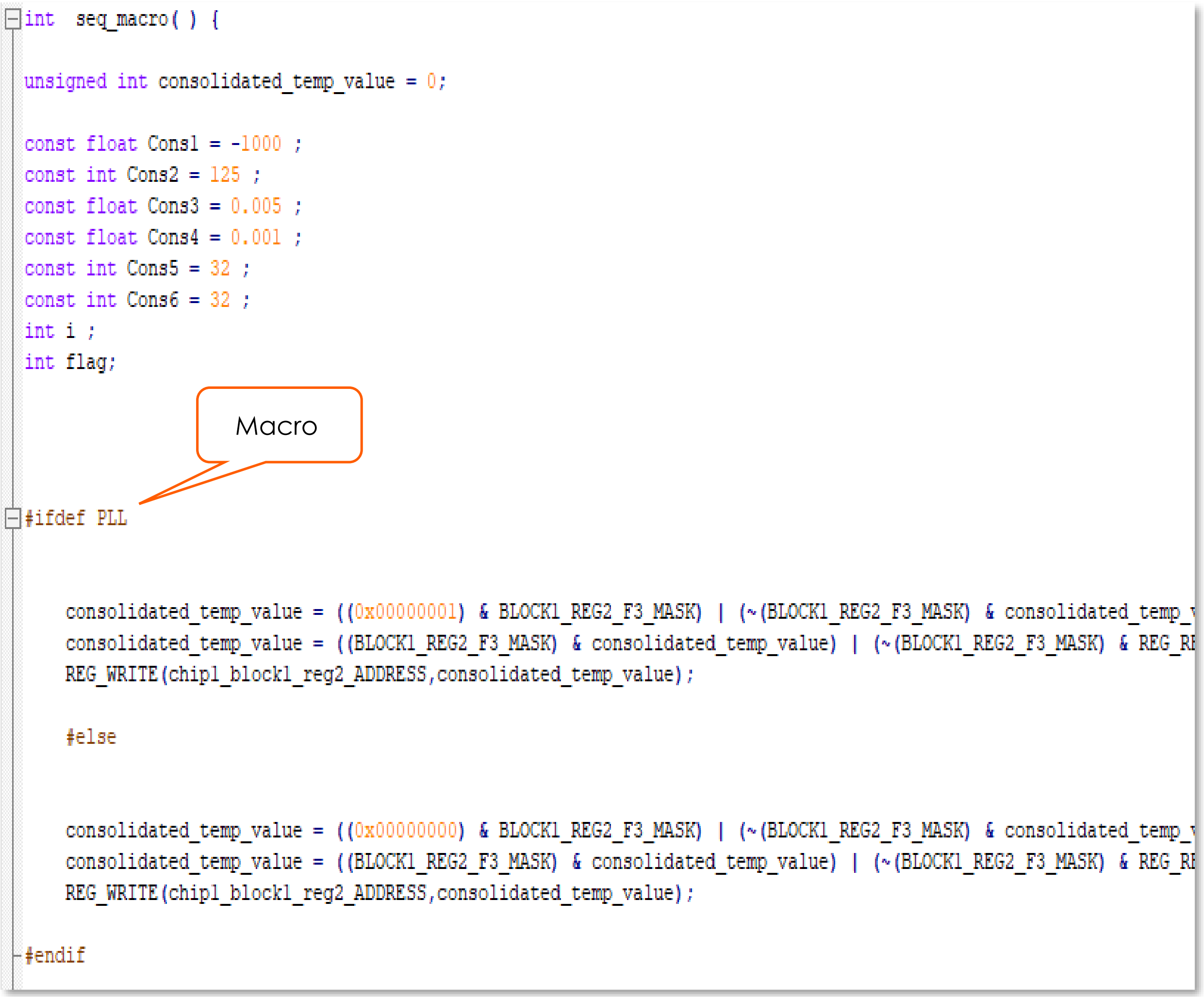
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY



Sequences	This sheet describe all the sequence steps(Please don't modify the headers)		
sequence name	ip	description	
seq_macro	Sheet1	{uvm.regmodel=regmap_block.direct_block;uvm.backdoor=true}	
arguments	value	description	
constants	value	description	
float Cons1	-1000		
Cons2	125		
float Cons3	0.005		
float Cons4	0.001		
Cons5	32		
Cons6	32		
variables	value	description	
command	step	value	description
`ifdef PLL			Use of macros such as ifdef in the sequen
write	reg2.f3	1	
`else			
write	reg2.f3	0	
`endif			
for(i= 0;i<Cons6; i=i+1){			
write	reg2.f7	0	
}			
write	reg1.f1	0	
write	reg1.f2	0	

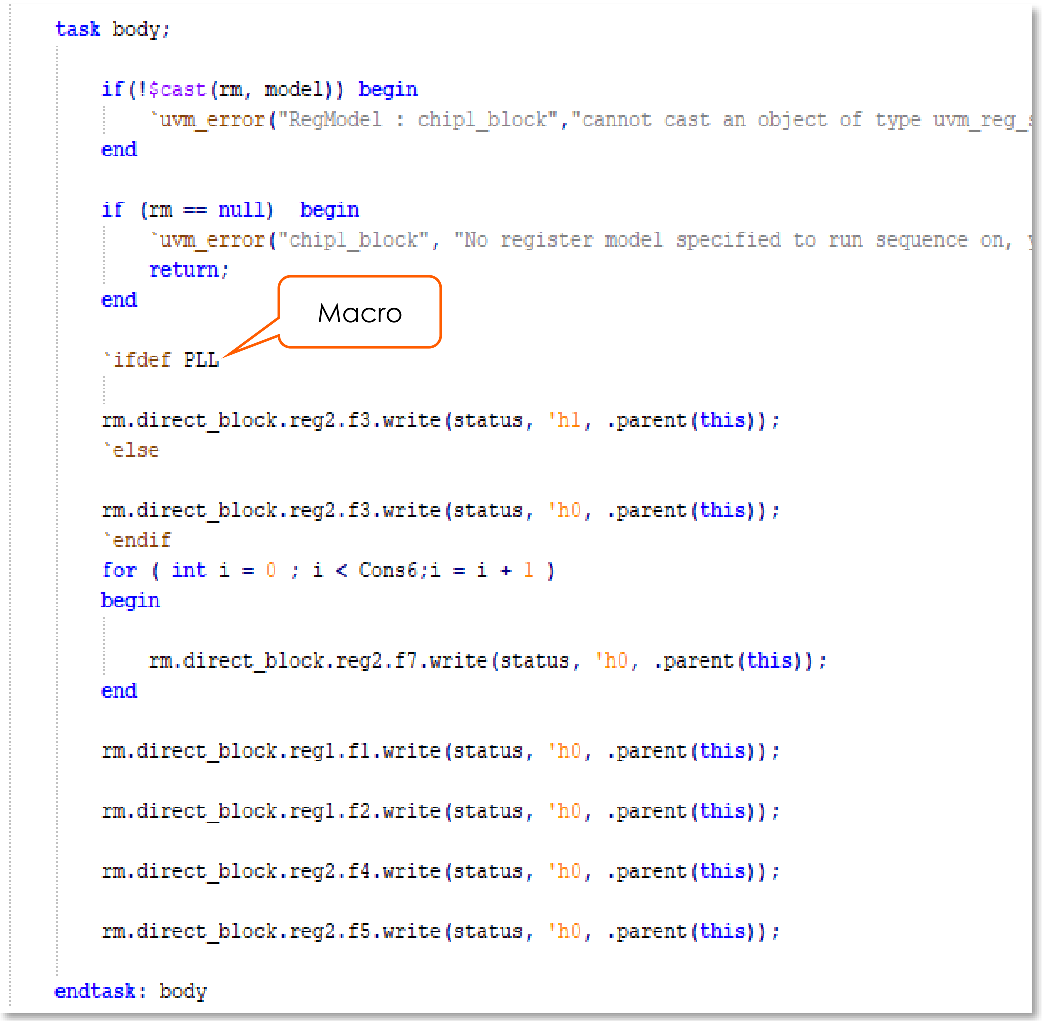
C Output

UVM Output



```
int  seq_macro( ) {

unsigned int consolidated_temp_value = 0;

const float Cons1 = -1000 ;
const int Cons2 = 125 ;
const float Cons3 = 0.005 ;
const float Cons4 = 0.001 ;
const int Cons5 = 32 ;
const int Cons6 = 32 ;
int i ;
int flag;



#ifdef PLL


    consolidated_temp_value = ((0x00000001) & BLOCK1_REG2_F3_MASK) | (~(BLOCK1_REG2_F3_MASK) & consolidated_temp_v
    consolidated_temp_value = ((BLOCK1_REG2_F3_MASK) & consolidated_temp_value) | (~(BLOCK1_REG2_F3_MASK) & REG_RE
    REG_WRITE(chip1_block1_reg2_ADDRESS,consolidated_temp_value);

    #else


    consolidated_temp_value = ((0x00000000) & BLOCK1_REG2_F3_MASK) | (~(BLOCK1_REG2_F3_MASK) & consolidated_temp_v
    consolidated_temp_value = ((BLOCK1_REG2_F3_MASK) & consolidated_temp_value) | (~(BLOCK1_REG2_F3_MASK) & REG_RE
    REG_WRITE(chip1_block1_reg2_ADDRESS,consolidated_temp_value);

#endif
```

Macro

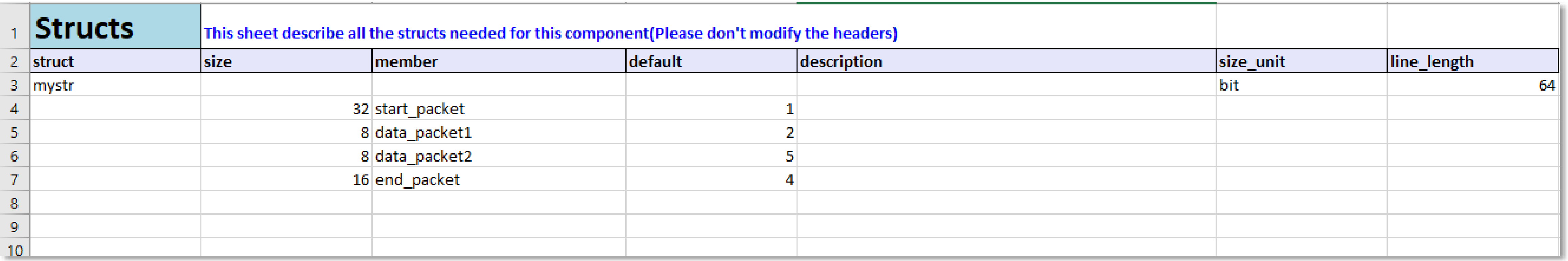


```
task body;

    if(!$cast(rm, model)) begin
        `uvm_error("RegModel : chip1_block","cannot cast an object of type uvm_reg_s
    end

    if (rm == null)  begin
        `uvm_error("chip1_block", "No register model specified to run sequence on, y
        return;
    end

    `ifdef PLL

    rm.direct_block.reg2.f3.write(status, 'h1, .parent(this));
    `else

    rm.direct_block.reg2.f3.write(status, 'h0, .parent(this));
    `endif
    for ( int i = 0 ; i < Cons6;i = i + 1 )
    begin

        rm.direct_block.reg2.f7.write(status, 'h0, .parent(this));
    end

    rm.direct_block.reg1.f1.write(status, 'h0, .parent(this));

    rm.direct_block.reg1.f2.write(status, 'h0, .parent(this));

    rm.direct_block.reg2.f4.write(status, 'h0, .parent(this));

    rm.direct_block.reg2.f5.write(status, 'h0, .parent(this));

endtask: body
```

Macro

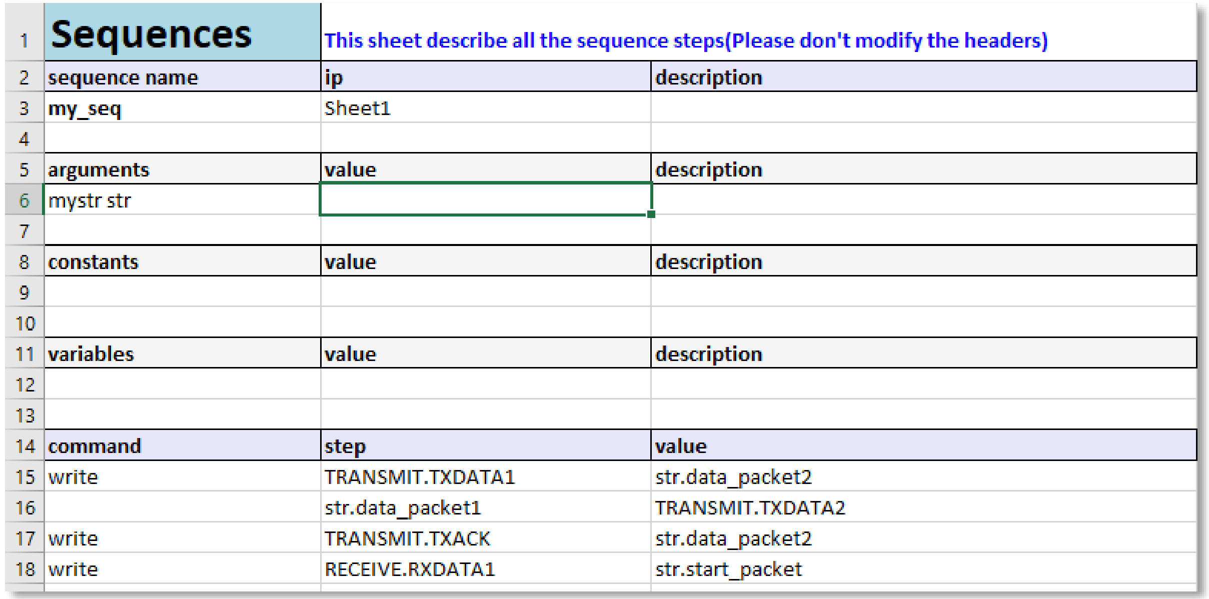
Structures in ISS

Initialization



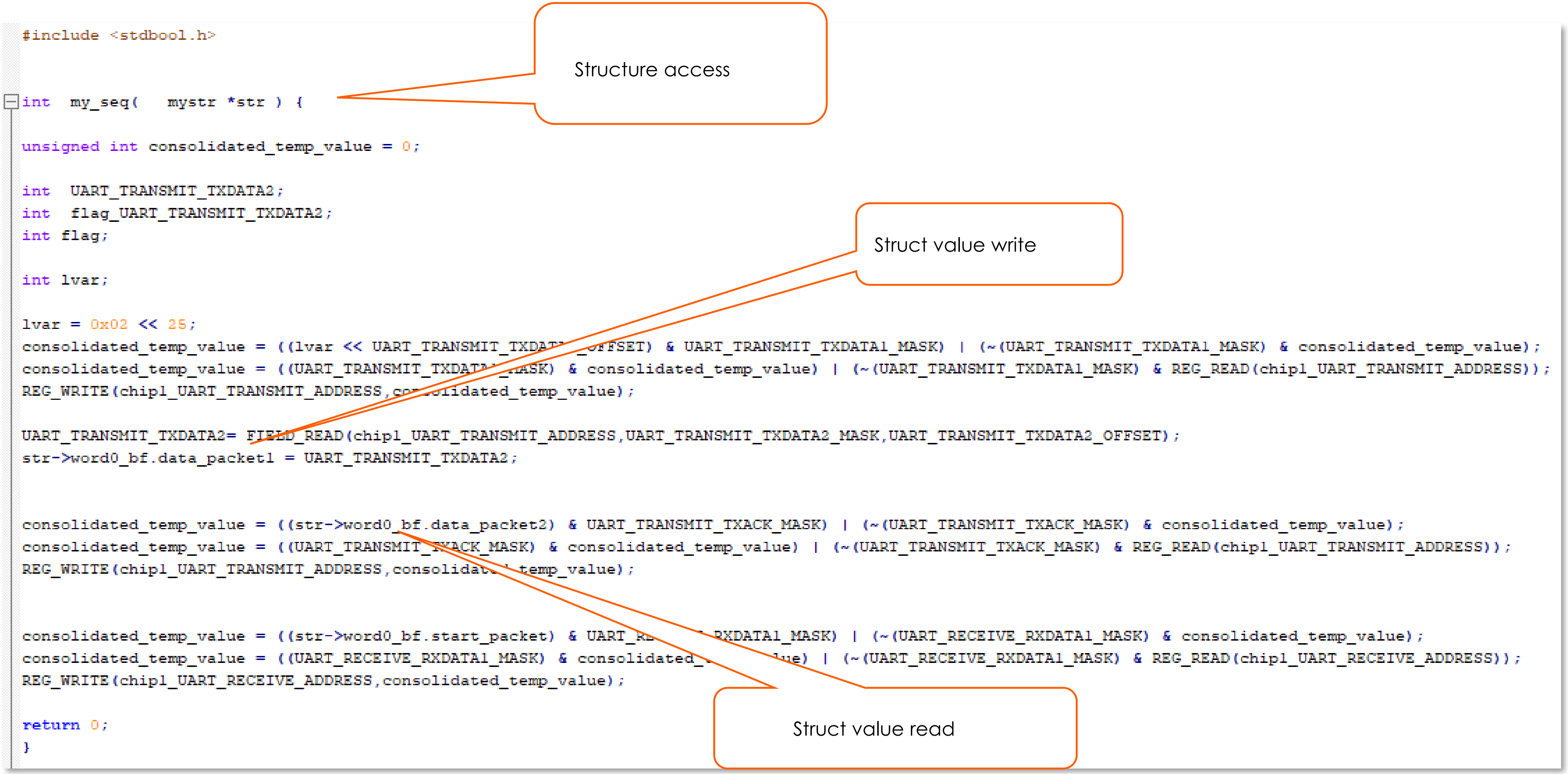
1	**Structs**	This sheet describe all the structs needed for this component(Please don't modify the headers)					
2	struct	size	member	default	description	size_unit	line_length
3	mystr					bit	64
4		32	start_packet	1			
5		8	data_packet1	2			
6		8	data_packet2	5			
7		16	end_packet	4			
8							
9							
10							

Read/Write using Struct



1	**Sequences**	This sheet describe all the sequence steps(Please don't modify the headers)	
2	sequence name	ip	description
3	my_seq	Sheet1	
4			
5	arguments	value	description
6	mystr str		
7			
8	constants	value	description
9			
10			
11	variables	value	description
12			
13			
14	command	step	value
15	write	TRANSMIT.TXDATA1	str.data_packet2
16		str.data_packet1	TRANSMIT.TXDATA2
17	write	TRANSMIT.TXACK	str.data_packet2
18	write	RECEIVE.RXDATA1	str.start_packet

C Output



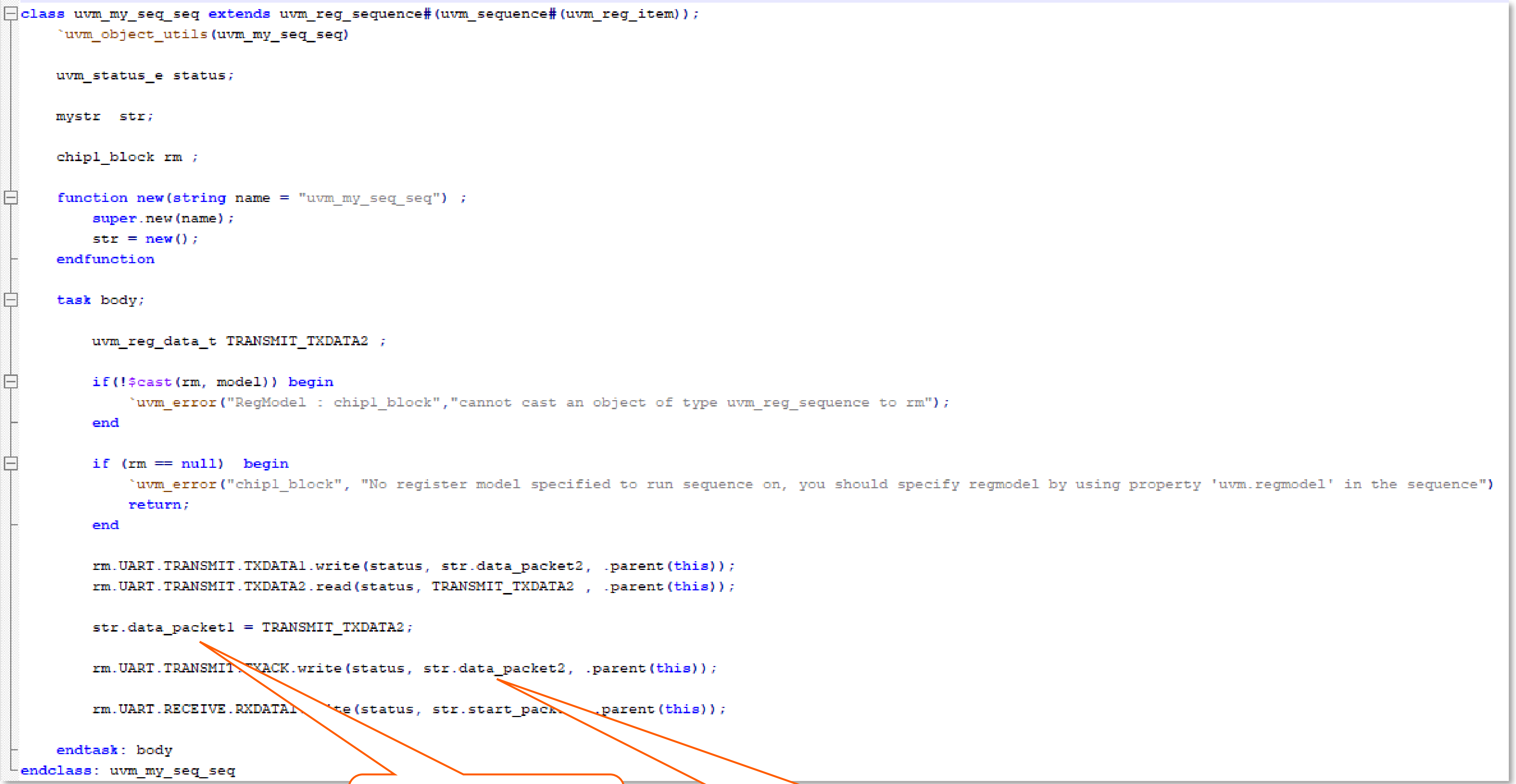
```
#include <stdbool.h>


int  my_seq(   mystr *str ) {

unsigned int consolidated_temp_value = 0;

int  UART_TRANSMIT_TXDATA2;
int  flag_UART_TRANSMIT_TXDATA2;
int flag;

int lvar;

lvar = 0x02 << 25;
consolidated_temp_value = ((lvar << UART_TRANSMIT_TXDATA1_OFFSET) & UART_TRANSMIT_TXDATA1_MASK) | (~(UART_TRANSMIT_TXDATA1_MASK) & consolidated_temp_value);
consolidated_temp_value = ((UART_TRANSMIT_TXDATA1_MASK) & consolidated_temp_value) | (~(UART_TRANSMIT_TXDATA1_MASK) & REG_READ(chip1_UART_TRANSMIT_ADDRESS));
REG_WRITE(chip1_UART_TRANSMIT_ADDRESS,consolidated_temp_value);

UART_TRANSMIT_TXDATA2= FIELD_READ(chip1_UART_TRANSMIT_ADDRESS,UART_TRANSMIT_TXDATA2_MASK,UART_TRANSMIT_TXDATA2_OFFSET);
str->word0_bf.data_packet1 = UART_TRANSMIT_TXDATA2;


consolidated_temp_value = ((str->word0_bf.data_packet2) & UART_TRANSMIT_TXACK_MASK) | (~(UART_TRANSMIT_TXACK_MASK) & consolidated_temp_value);
consolidated_temp_value = ((UART_TRANSMIT_TXACK_MASK) & consolidated_temp_value) | (~(UART_TRANSMIT_TXACK_MASK) & REG_READ(chip1_UART_TRANSMIT_ADDRESS));
REG_WRITE(chip1_UART_TRANSMIT_ADDRESS,consolidated_temp_value);


consolidated_temp_value = ((str->word0_bf.start_packet) & UART_RECEIVE_RXDATA1_MASK) | (~(UART_RECEIVE_RXDATA1_MASK) & consolidated_temp_value);
consolidated_temp_value = ((UART_RECEIVE_RXDATA1_MASK) & consolidated_temp_value) | (~(UART_RECEIVE_RXDATA1_MASK) & REG_READ(chip1_UART_RECEIVE_ADDRESS));
REG_WRITE(chip1_UART_RECEIVE_ADDRESS,consolidated_temp_value);

return 0;
}
```

Structure access

Struct value write

Struct value read

UVM Output



```
class uvm_my_seq_seq extends uvm_reg_sequence#(uvm_sequence#(uvm_reg_item));
    `uvm_object_utils(uvm_my_seq_seq)

    uvm_status_e status;

    mystr  str;

    chip1_block rm ;

    function new(string name = "uvm_my_seq_seq") ;
        super.new(name);
        str = new();
    endfunction

    task body;

        uvm_reg_data_t TRANSMIT_TXDATA2 ;

        if(!$cast(rm, model)) begin
            `uvm_error("RegModel : chip1_block","cannot cast an object of type uvm_reg_sequence to rm");
        end

        if (rm == null)  begin
            `uvm_error("chip1_block", "No register model specified to run sequence on, you should specify regmodel by using property 'uvm.regmodel' in the sequence")
            return;
        end

        rm.UART.TRANSMIT.TXDATA1.write(status, str.data_packet2, .parent(this));
        rm.UART.TRANSMIT.TXDATA2.read(status, TRANSMIT_TXDATA2 , .parent(this));

        str.data_packet1 = TRANSMIT_TXDATA2;

        rm.UART.TRANSMIT.TXACK.write(status, str.data_packet2, .parent(this));

        rm.UART.RECEIVE.RXDATA1.write(status, str.start_packet, .parent(this));

    endtask: body
endclass: uvm_my_seq_seq
```

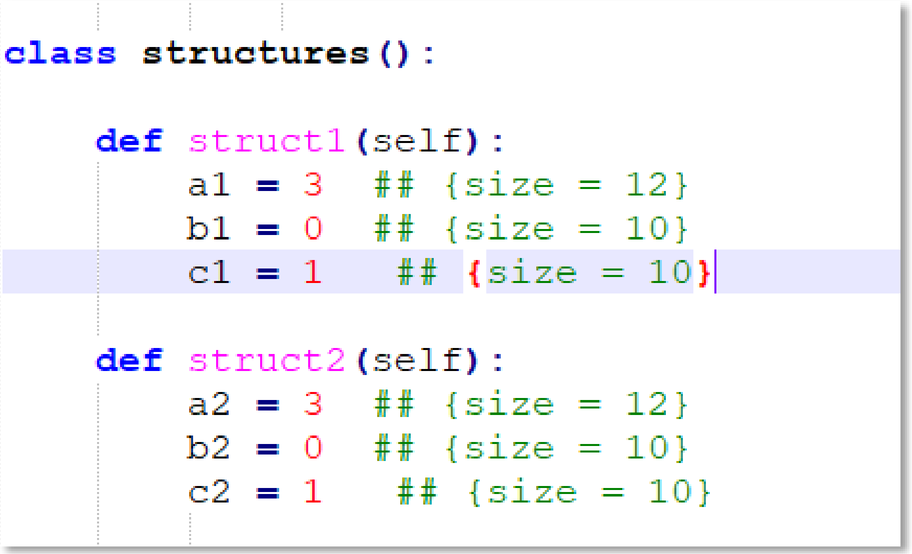
Struct value write

Struct value read

Text Based ISS

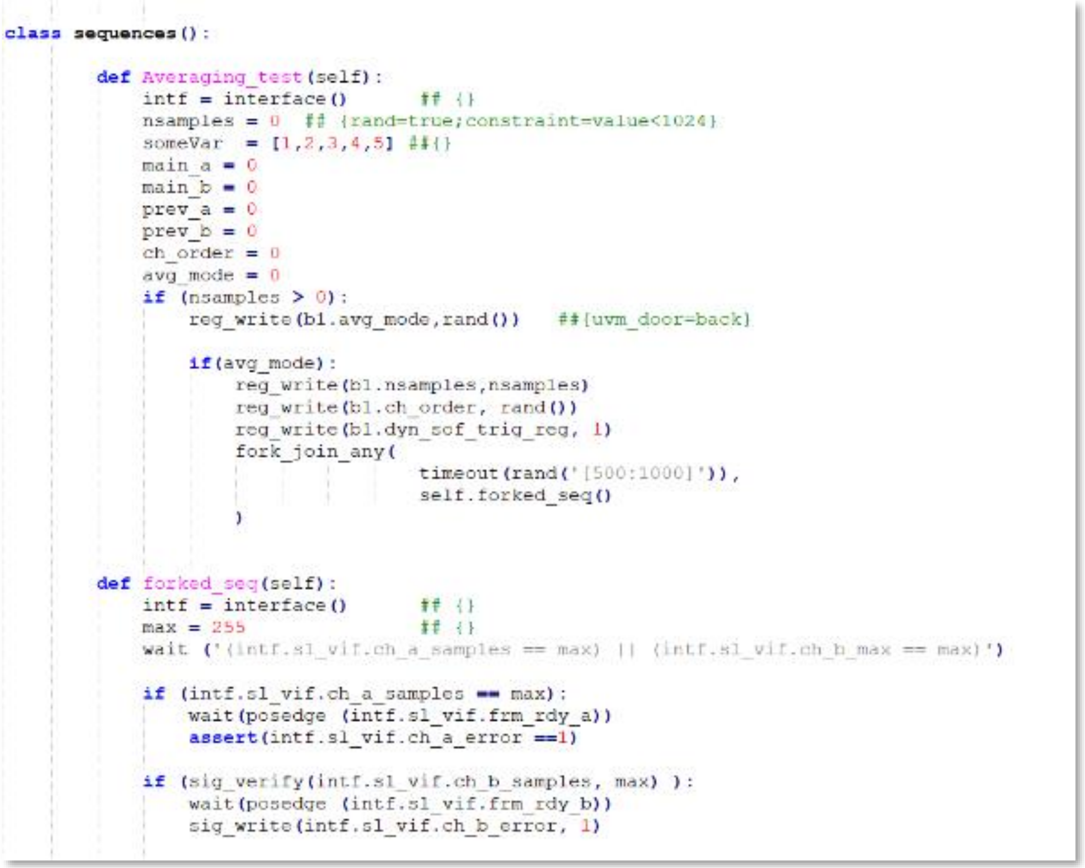
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Structures



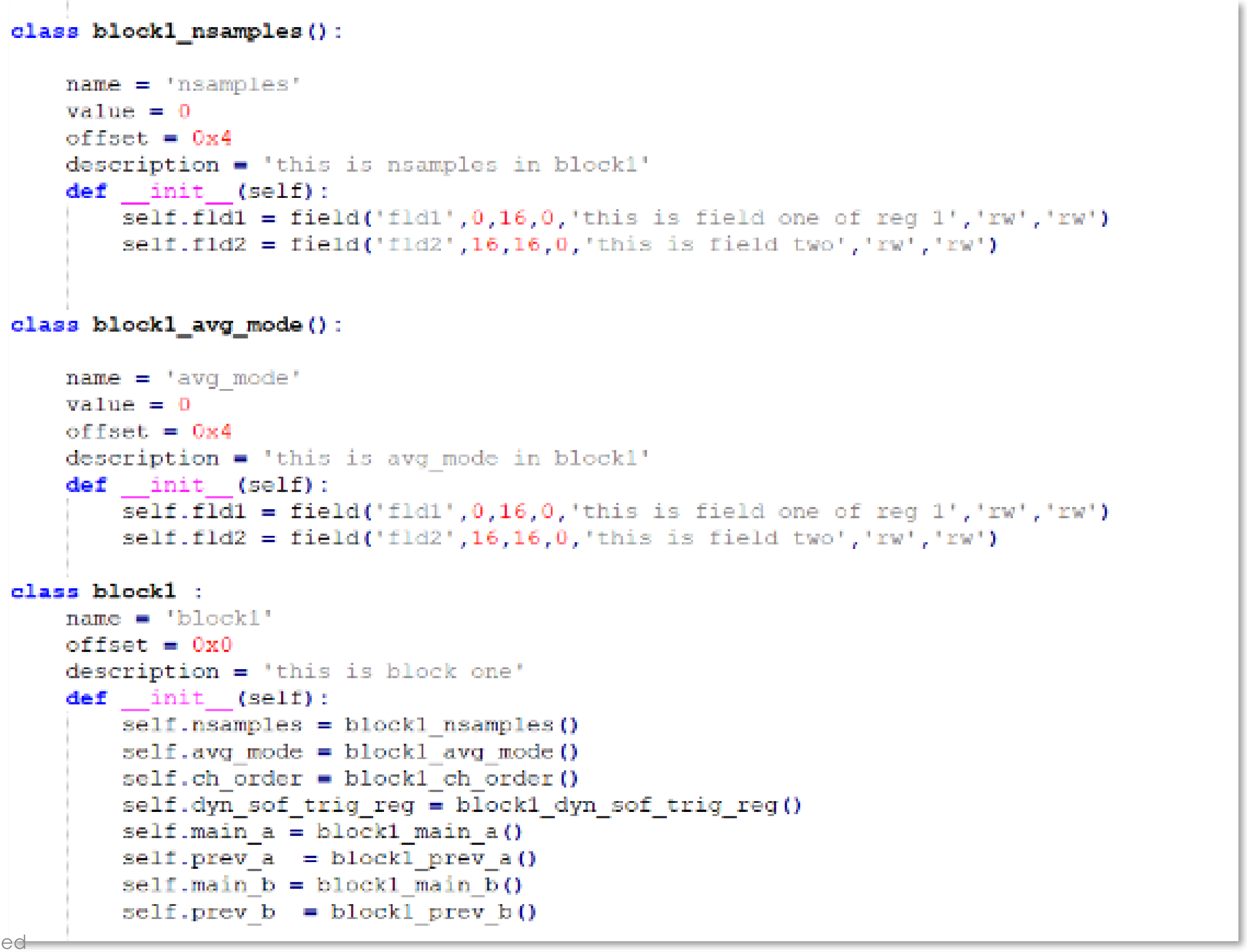
```
class structures():

    def struct1(self):
        a1 = 3   ## {size = 12}
        b1 = 0   ## {size = 10}
        c1 = 1    ## {size = 10}

    def struct2(self):
        a2 = 3   ## {size = 12}
        b2 = 0   ## {size = 10}
        c2 = 1    ## {size = 10}
```

Sequences



```
class sequences():

    def Averaging_test(self):
        intf = interface()       ## {}
        nsamples = 0  ## {rand=true;constraint=value<1024}
        someVar  = [1,2,3,4,5] ##{}
        main_a = 0
        main_b = 0
        prev_a = 0
        prev_b = 0
        ch_order = 0
        avg_mode = 0
        if (nsamples > 0):
            reg_write(bl.avg_mode,rand())   ##{uvm_door=back}

            if(avg_mode):
                reg_write(bl.nsamples,nsamples)
                reg_write(bl.ch_order, rand())
                reg_write(bl.dyn_sof_trig_reg, 1)
                fork_join_any(
                                timeout(rand('[500:1000]')),
                                self.forked_seq()
                )


    def forked_seq(self):
        intf = interface()       ## {}
        max = 255                ## {}
        wait ('(intf.sl_vif.ch_a_samples == max) || (intf.sl_vif.ch_b_max == max)')

        if (intf.sl_vif.ch_a_samples == max):
            wait(posedge (intf.sl_vif.frm_rdy_a))
            assert(intf.sl_vif.ch_a_error ==1)

        if (sig_verify(intf.sl_vif.ch_b_samples, max) ):
            wait(posedge (intf.sl_vif.frm_rdy_b))
            sig_write(intf.sl_vif.ch_b_error, 1)
```

Text Based ISS



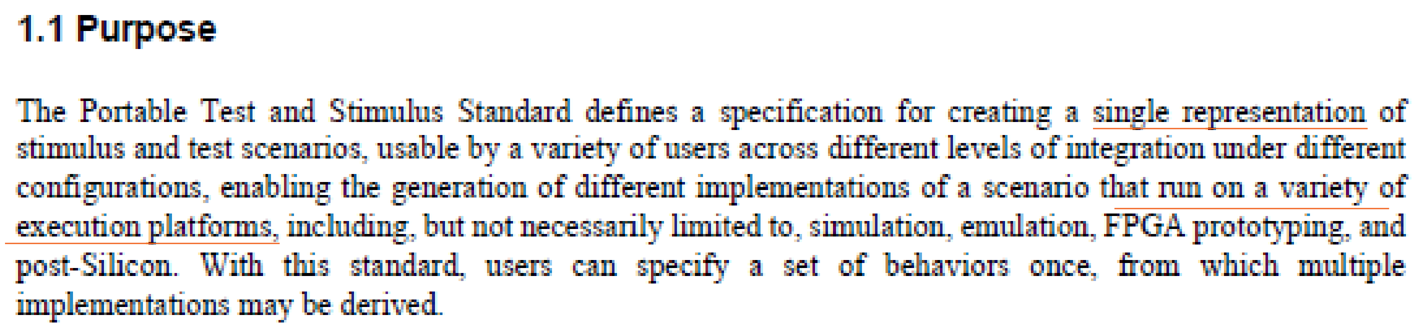
```
class block1_nsamples():

    name = 'nsamples'
    value = 0
    offset = 0x4
    description = 'this is nsamples in block1'
    def __init__(self):
        self.fld1 = field('fld1',0,16,0,'this is field one of reg 1','rw','rw')
        self.fld2 = field('fld2',16,16,0,'this is field two','rw','rw')


class block1_avg_mode():

    name = 'avg_mode'
    value = 0
    offset = 0x4
    description = 'this is avg_mode in block1'
    def __init__(self):
        self.fld1 = field('fld1',0,16,0,'this is field one of reg 1','rw','rw')
        self.fld2 = field('fld2',16,16,0,'this is field two','rw','rw')

class block1 :
    name = 'block1'
    offset = 0x0
    description = 'this is block one'
    def __init__(self):
        self.nsamples = block1_nsamples()
        self.avg_mode = block1_avg_mode()
        self.ch_order = block1_ch_order()
        self.dyn_sof_trig_reg = block1_dyn_sof_trig_reg()
        self.main_a = block1_main_a()
        self.prev_a  = block1_prev_a()
        self.main_b = block1_main_b()
        self.prev_b  = block1_prev_b()
```

Link to PSS

Portable Stimulus Standard

What is Portable Test and Stimulus Standard (PSS)?

- PSS 1.0 Standard … released in June 2018.



1.1 Purpose

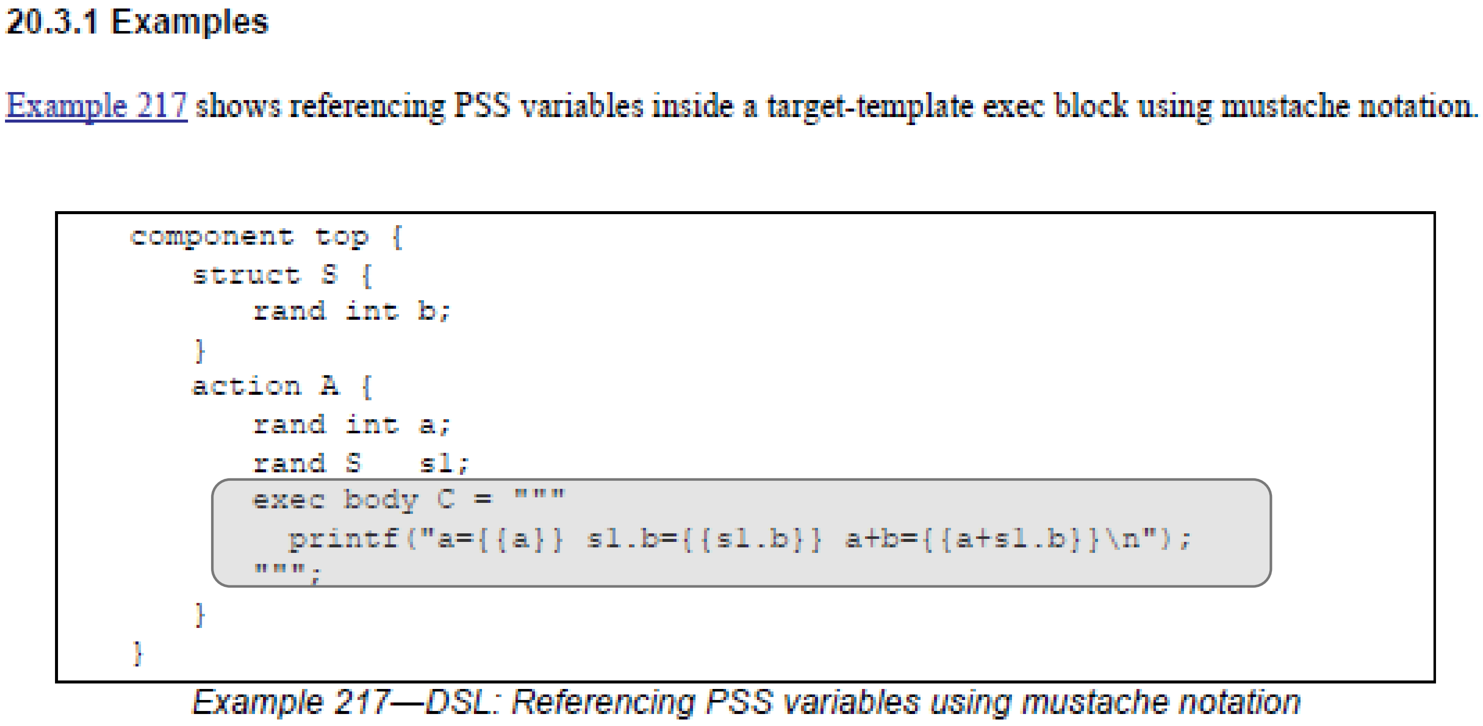
The Portable Test and Stimulus Standard defines a specification for creating a single representation of stimulus and test scenarios, usable by a variety of users across different levels of integration under different configurations, enabling the generation of different implementations of a scenario that run on a variety of execution platforms, including, but not necessarily limited to, simulation, emulation, FPGA prototyping, and post-Silicon. With this standard, users can specify a set of behaviors once, from which multiple implementations may be derived.

- PSS has constructs for
  - Modeling Data flow (Buffers, Streams, States, …)
  - Modeling Behavior (Actions, Activities, Components, Resource, Pooling, …)
  - Constraints, Randomization, Coverage, …

- PSS is useful for high SoC level test scenario creation
- The IP level details are handled using “exec blocks”

PSS Exec blocks

- Exec blocks are a mechanism for PSS tools to call code that they cannot generate from the PSS spec
- Example : Initialization Sequence with a specific order of register writes



20.3.1 Examples

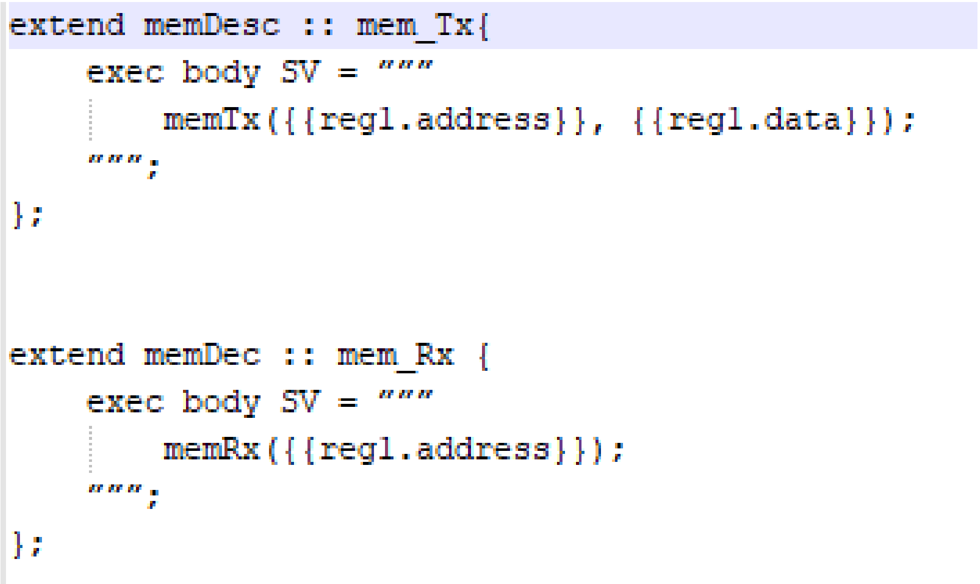
Example 217 shows referencing PSS variables inside a target-template exec block using mustache notation.

```
component top {
    struct S {
        rand int b;
    }
    action A {
        rand int a;
        rand S   s1;
        exec body C = """
          printf("a={{a}} s1.b={{s1.b}} a+b={{a+s1.b}}\n");
        """;
    }
}
```

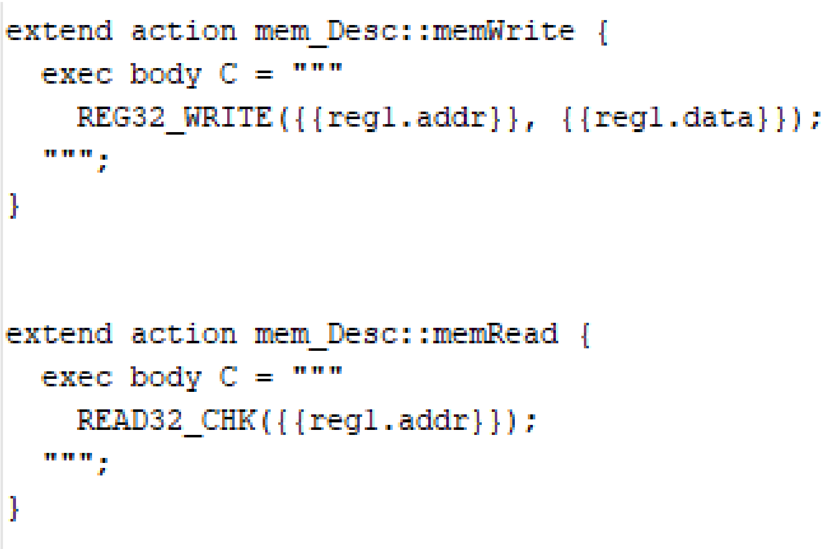
Example 217—DSL: Referencing PSS variables using mustache notation

ISeqeunceSpec can generate low level sequences

- User does not need to create the “exec block” implementation manually
- ISS will not only generate the PSS “header” for the exec blocks, but also various implementations
    - C
    - SV-UVM
    - Other platforms

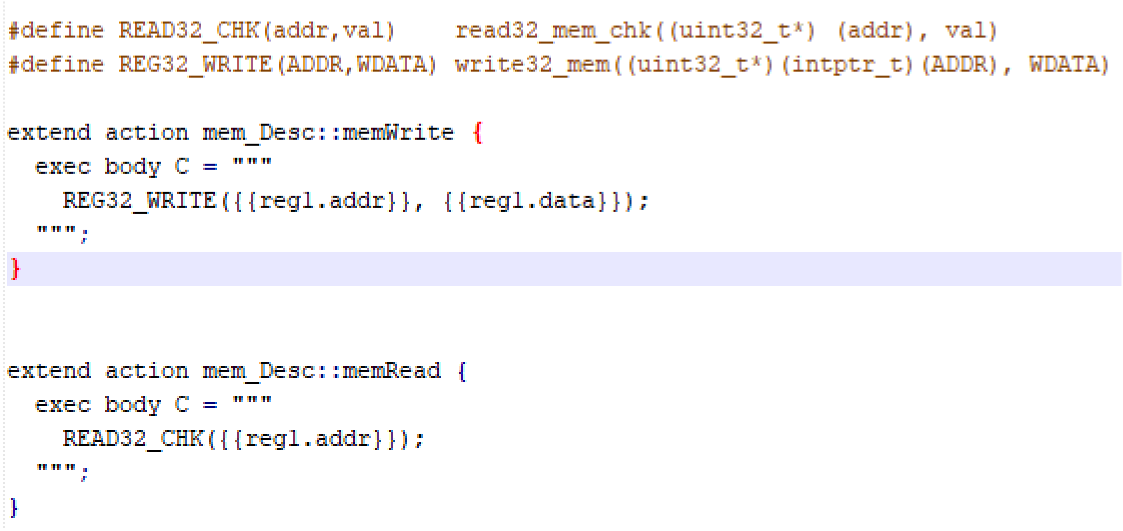


```
extend memDesc :: mem_Tx{
    exec body SV = """
        memTx({{regl.address}}, {{regl.data}});
    """;
};


extend memDec :: mem_Rx {
    exec body SV = """
        memRx({{regl.address}});
    """;
};
```

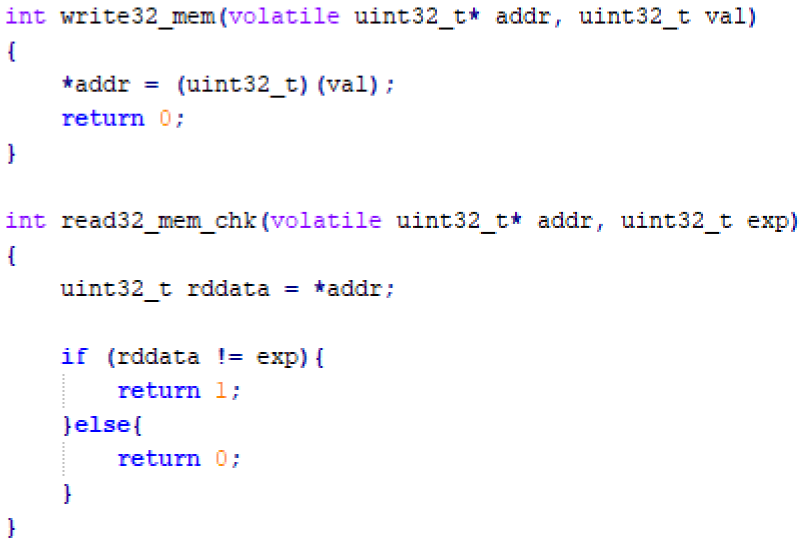


```
extend action mem_Desc::memWrite {
  exec body C = """
    REG32_WRITE({{regl.addr}}, {{regl.data}});
  """;
}


extend action mem_Desc::memRead {
  exec body C = """
    READ32_CHK({{regl.addr}});
  """;
}
```

C Output for Exec block

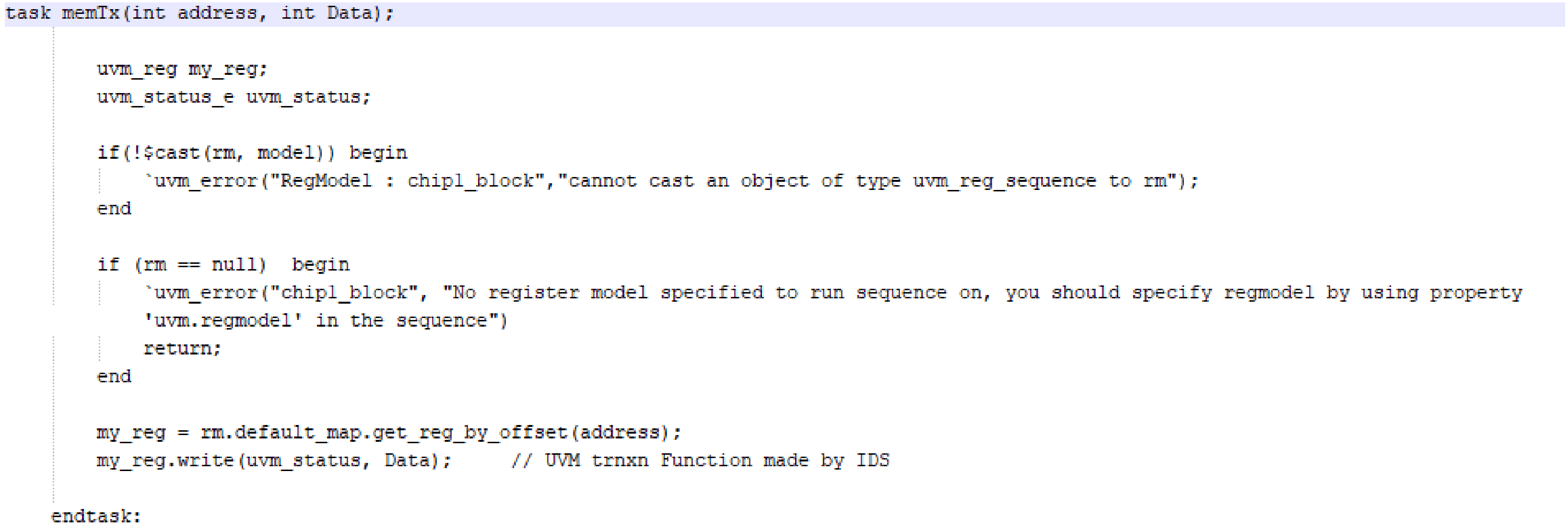


```
#define READ32_CHK(addr,val)    read32_mem_chk((uint32_t*) (addr), val)
#define REG32_WRITE(ADDR,WDATA) write32_mem((uint32_t*)(intptr_t)(ADDR), WDATA)

extend action mem_Desc::memWrite {
  exec body C = """
    REG32_WRITE({{regl.addr}}, {{regl.data}});
  """;
}


extend action mem_Desc::memRead {
  exec body C = """
    READ32_CHK({{regl.addr}});
  """;
}
```

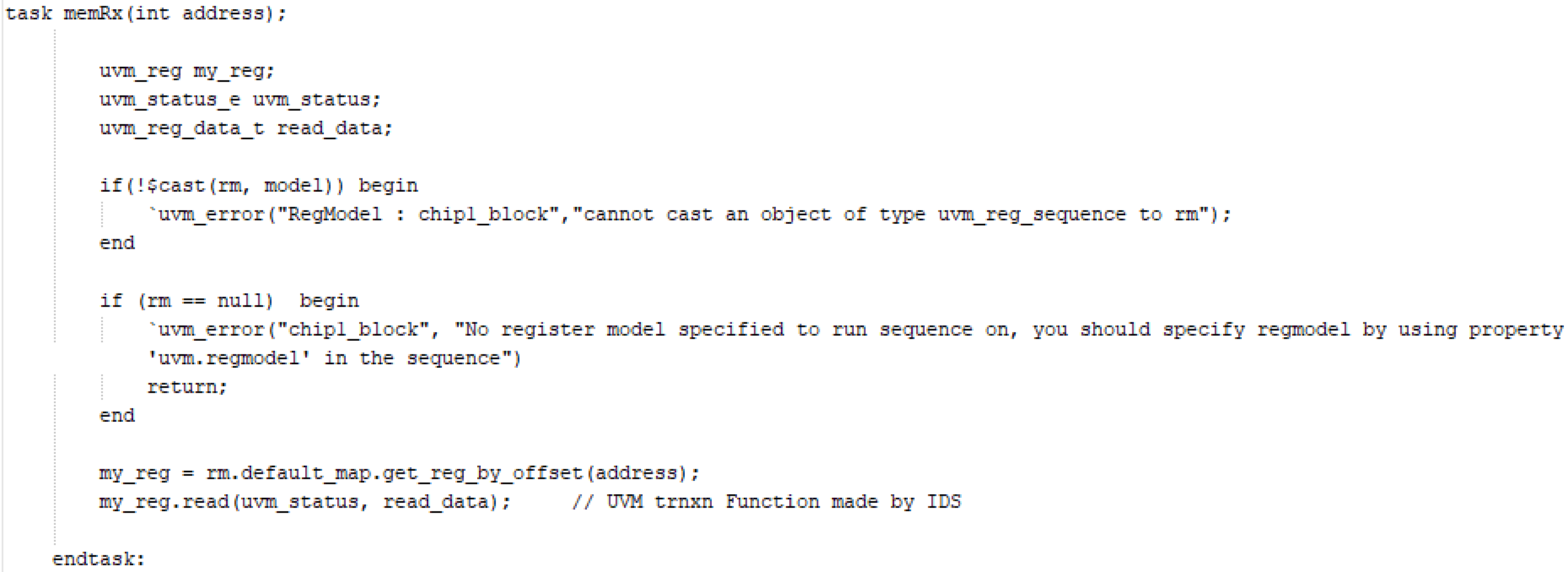


```
int write32_mem(volatile uint32_t* addr, uint32_t val)
{
    *addr = (uint32_t)(val);
    return 0;
}

int read32_mem_chk(volatile uint32_t* addr, uint32_t exp)
{
    uint32_t rddata = *addr;

    if (rddata != exp){
        return 1;
    }else{
        return 0;
    }
}
```

SV Output for Exec block



```
task memTx(int address, int Data);

    uvm_reg my_reg;
    uvm_status_e uvm_status;

    if(!$cast(rm, model)) begin
        `uvm_error("RegModel : chipl_block","cannot cast an object of type uvm_reg_sequence to rm");
    end

    if (rm == null)  begin
        `uvm_error("chipl_block", "No register model specified to run sequence on, you should specify regmodel by using property
        'uvm.regmodel' in the sequence")
        return;
    end

    my_reg = rm.default_map.get_reg_by_offset(address);
    my_reg.write(uvm_status, Data);     // UVM trnxn Function made by IDS

  endtask:
```



```
task memRx(int address);

    uvm_reg my_reg;
    uvm_status_e uvm_status;
    uvm_reg_data_t read_data;

    if(!$cast(rm, model)) begin
        `uvm_error("RegModel : chipl_block","cannot cast an object of type uvm_reg_sequence to rm");
    end

    if (rm == null)  begin
        `uvm_error("chipl_block", "No register model specified to run sequence on, you should specify regmodel by using property
        'uvm.regmodel' in the sequence")
        return;
    end

    my_reg = rm.default_map.get_reg_by_offset(address);
    my_reg.read(uvm_status, read_data);     // UVM trnxn Function made by IDS

  endtask:
```

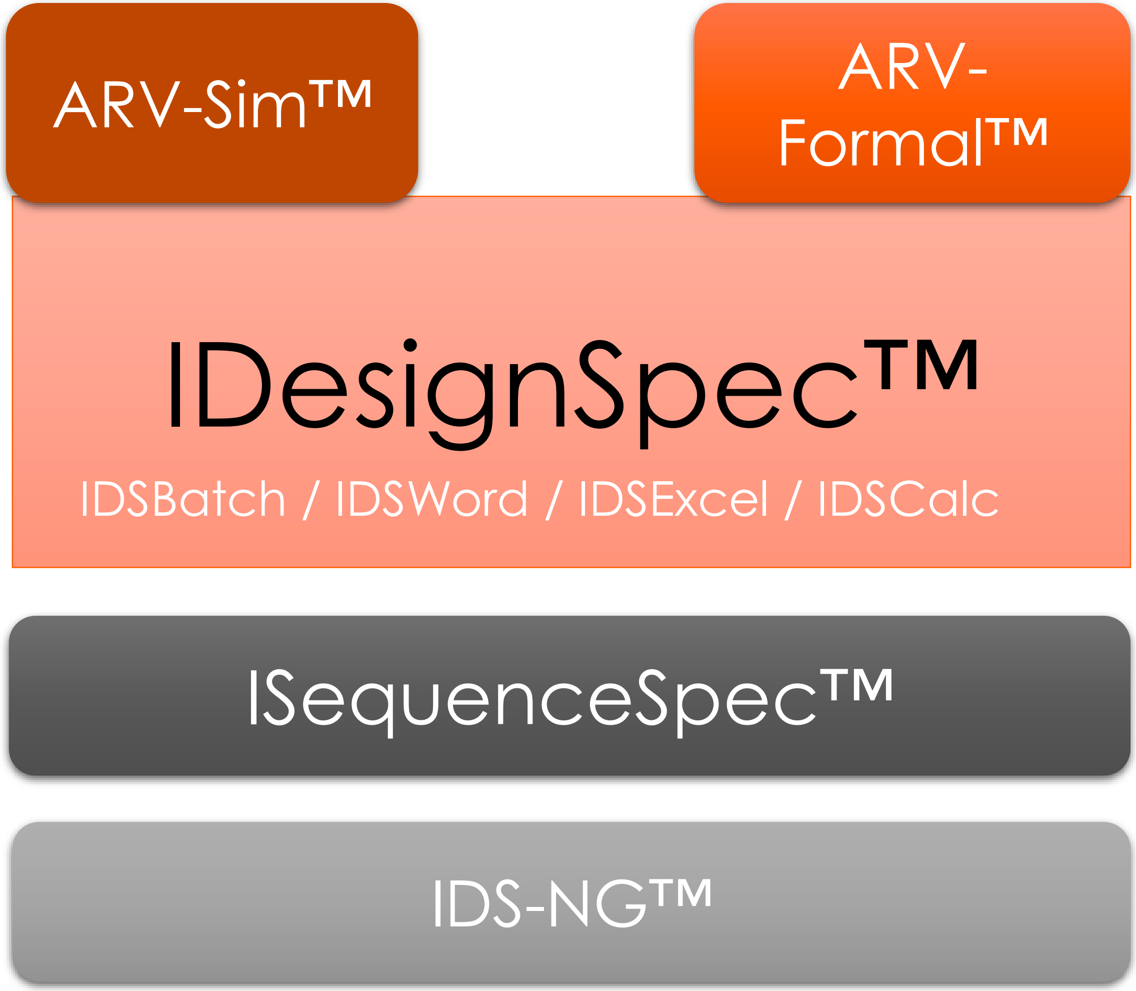
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

How Agnisys can Help

Tools and Services

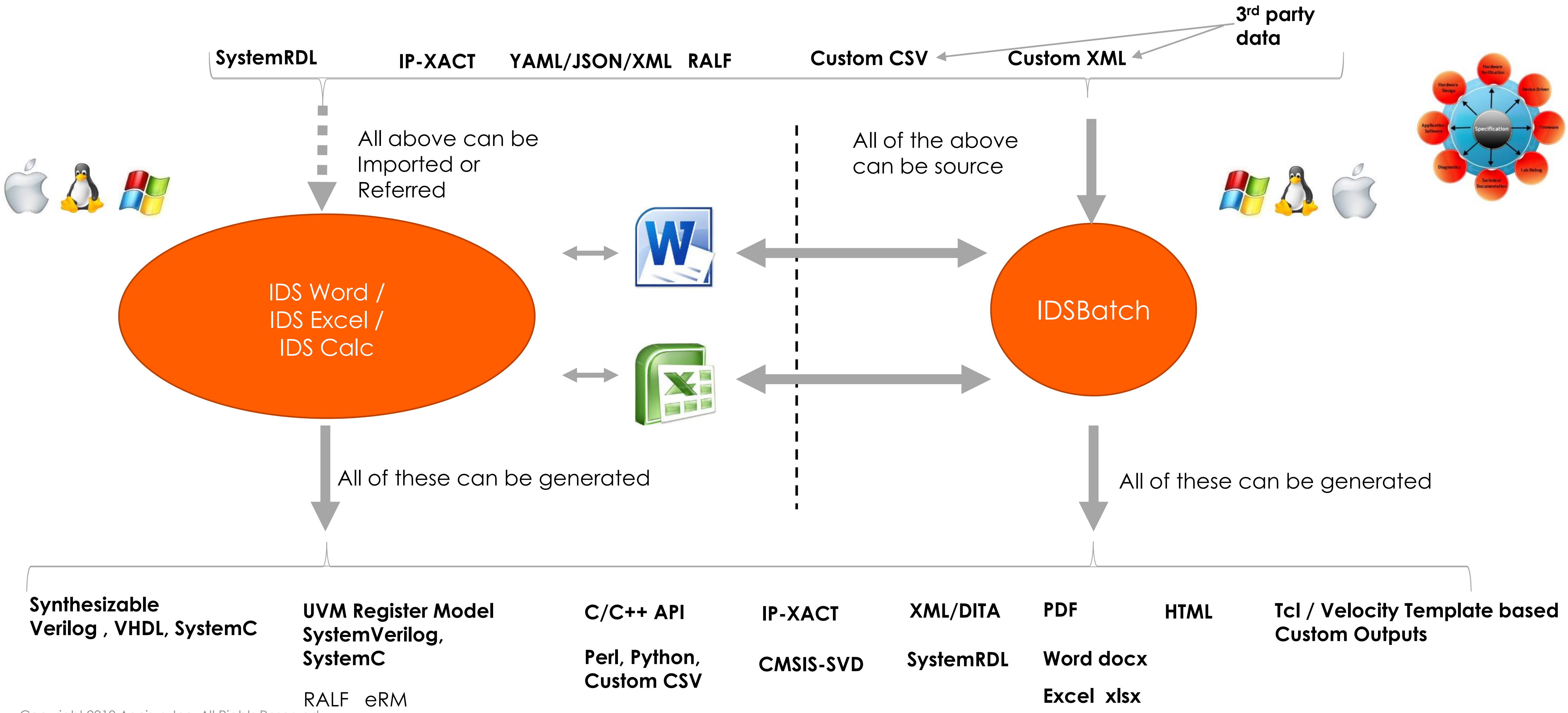
Agnisys' Solutions

- **IDesignSpec (IDS)**
  - Create Models
- **ARV-Sim**
  - Create Test Sequences & Environment
- **ARV-Formal**
  - Create Formal Properties and Assertions
- **ISequenceSpec (ISS)**
  - Create UVM sequences and Firmware routines
- **IDS-NextGen**
  - Cross-platform HSI Layer Specification
  - Single spec for both Registers and Sequences



IDesignSpec suite

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY



SystemRDL IP-XACT YAML/JSON/XML RALF Custom CSV Custom XML

3rd party data

All above can be Imported or Referred

All of the above can be source

IDS Word / IDS Excel / IDS Calc

IDSBatch

All of these can be generated

All of these can be generated

Synthesizable Verilog , VHDL, SystemC

UVM Register Model SystemVerilog, SystemC

RALF eRM

C/C++ API

Perl, Python, Custom CSV

IP-XACT

CMSIS-SVD

XML/DITA

SystemRDL

PDF

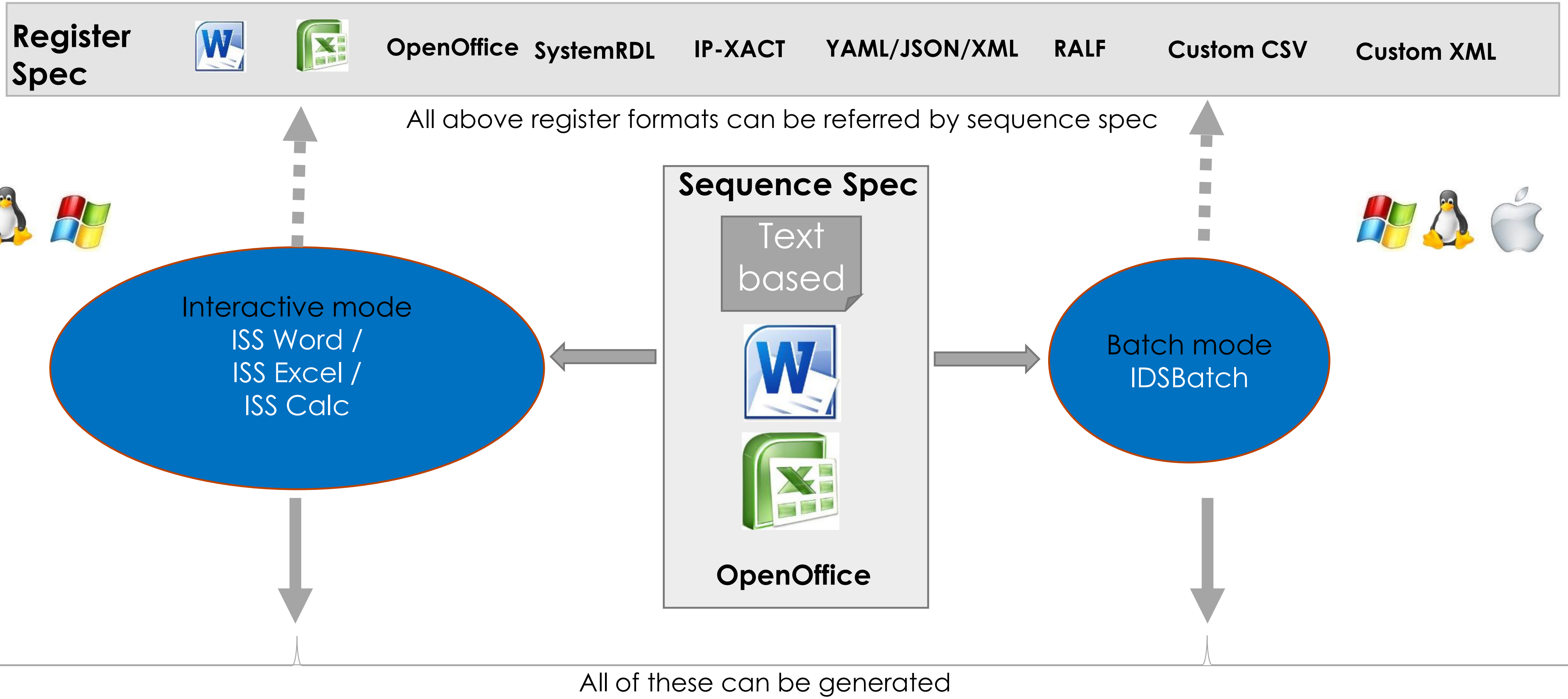
Word docx

Excel xlsx

HTML

Tcl / Velocity Template based Custom Outputs

ISequenceSpec suite



Register Spec

OpenOffice SystemRDL IP-XACT YAML/JSON/XML RALF Custom CSV Custom XML

All above register formats can be referred by sequence spec

Sequence Spec

Text based

OpenOffice

Interactive mode
ISS Word /
ISS Excel /
ISS Calc

Batch mode
IDSBatch

All of these can be generated

SV-UVM

SC-UVM

C/C++
Firmware

SystemC

Verilog for
Validation/Emulation

Python, Tcl

Automatic Test
Equipment Formats

CSV

Matlab

HTML
PDF

PSS
Portable Stimulus Standard

Benefits of ISequenceSpec

- Reduce Time and Cost of development
- Eliminate Duplication of sequence implementation
- Improve Quality
- Let ISS do the mundane work of ensuring correctness, while you focus on the algorithms

Resistance to change

- Some may say …
  - Why do something new when everything is working fine
    - It saves time across teams, reduces development and debug time
  - We love our UVM or C/C++ language – don't want to change
    - You can still write in your favorite language, just use the auto generated low level sequences
  - Reduces flexibility
    - But reduces work load as well

- We feel automating sequences is a low hanging fruit
  - Not much risk to it
  - And potential for high Return on Investment

Conclusion

- Today, sequences are at a point where registers were a decade ago
- There are a lot of challenges in dealing with Sequences …
- … Also a huge opportunity for automation
- Get it right to get huge productivity gains, get it wrong, and lose a lot of cycles debugging
- Agnisys is leading the drive to automate Sequences

www.agnisys.com

sales@agnisys.com

PH: 1 (855) VERIFY
1 (855) 837-4399

- IDesignSpec™
- ISequenceSpec™
- IDS NextGen™
- ARV™
- DVInsight-Pro™