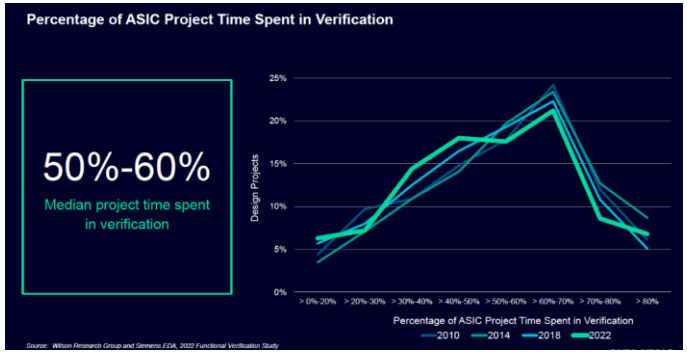
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

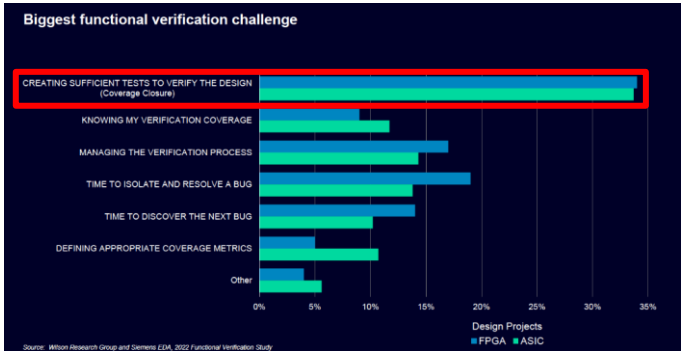
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

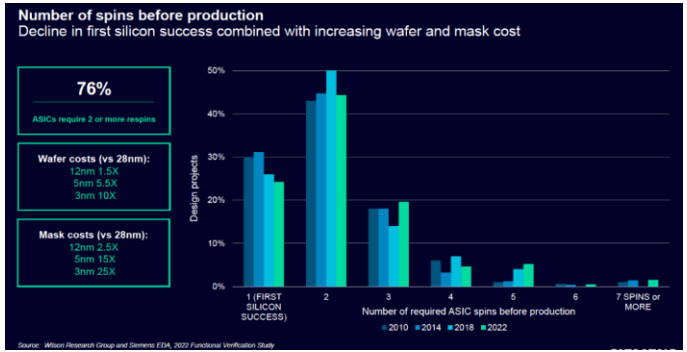
Automatic generation of Programmer Reference Manual and Device Driver from PSS

Freddy Nunez
AGNISYS

The State of Verification in 2023





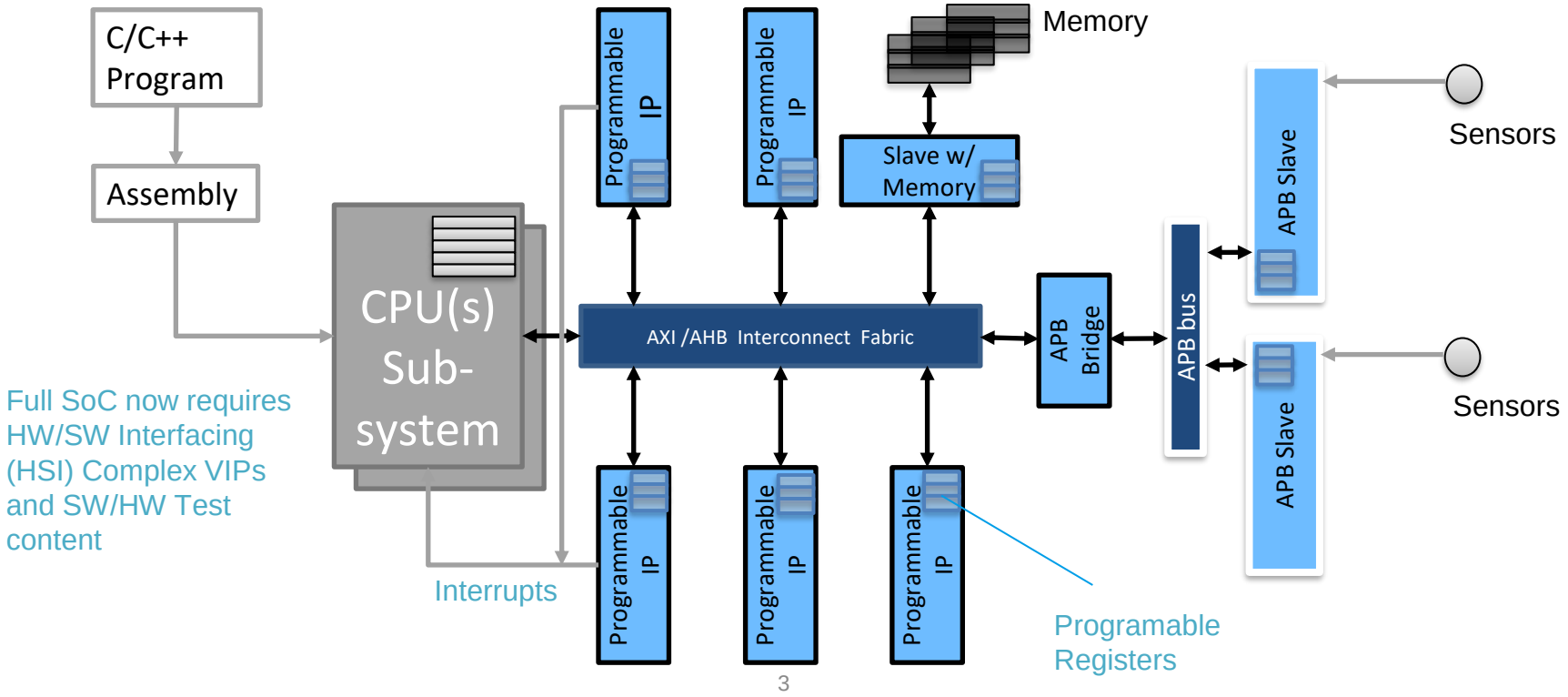


- Most development time spent in verification
- However, respins per project still increasing
- Greatest verification challenge (by far)…

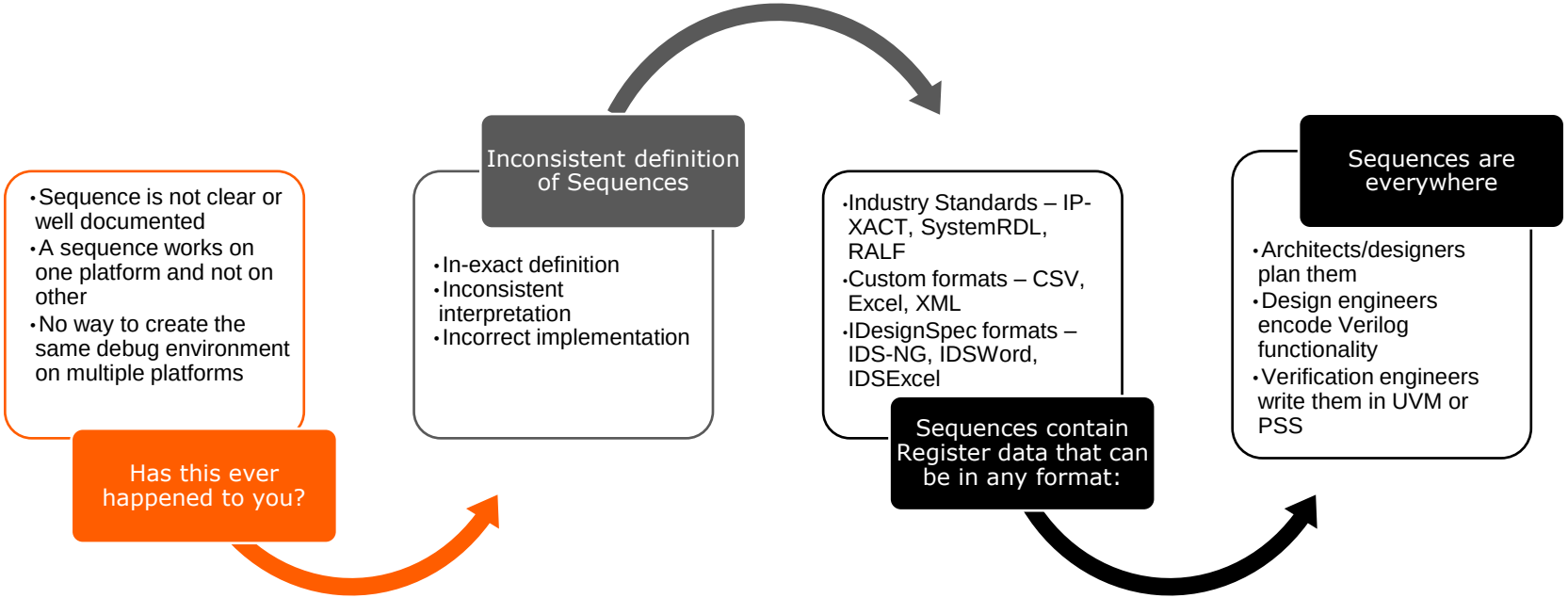
  creating sufficient tests

Source: Wilson Research Group 2022, courtesy Siemens EDA

HW/SW Interface of a Typical SoC



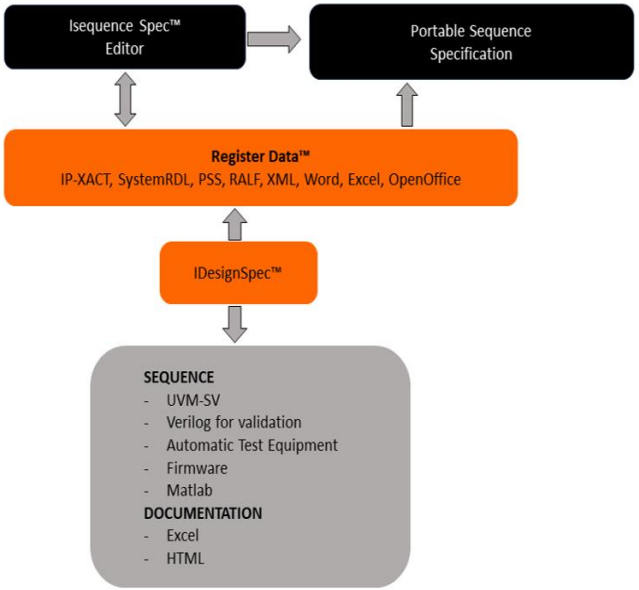
Challenges Development Teams Face with Sequences



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

An Ideal Solution

- Describe the programming and test sequences of a device and automatically generate sequences ready to use from an early design and verification stage to post silicon validation
- Centralize creation of sequences from a single specification and generate various output formats for multiple SoC teams
  - SV/UVM, **PSS**, C, CSV or MATLAB
  - PDF or HTML
- Specify portable sequences for multiple IPs at a higher level in-sync with the register specification
- Use register descriptions in standard formats such as IP-XACT, **SystemRDL**, RALF or leverage IDesignSpec™ integrated flow to use the register data
- Sequence constructs include loops, if-else, wait, arguments, constant, in-line functions

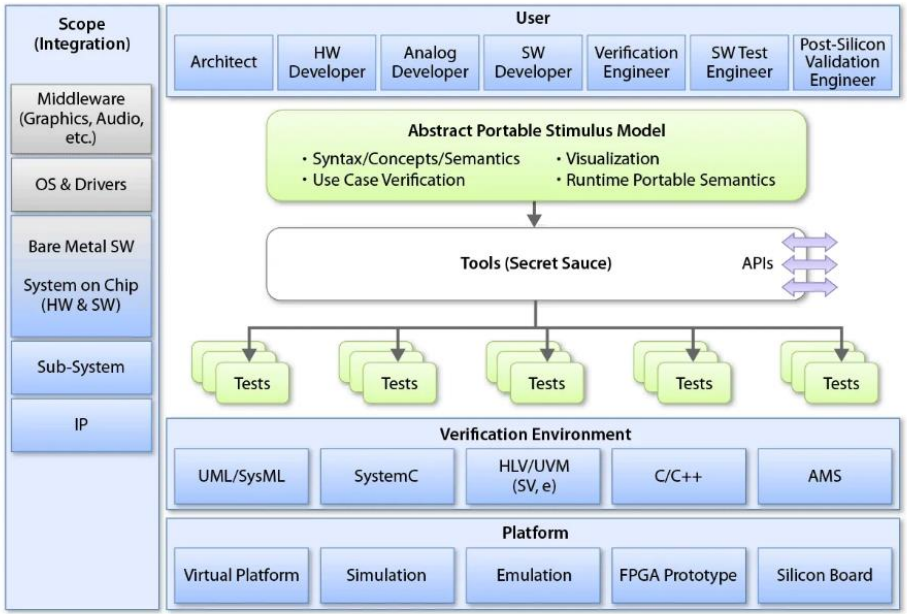


What does a common sequence specification need

- Like 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

The Accellera Portable Stimulus Standard



Proposed Portable Stimulus Specification (Courtesy: Accellera Systems Initiative)

Accellera's PSS committee was formed to drive a common standard for modeling stimulus that could be ported between simulation, emulation and fabricated silicon.

This stimulus methodology could drive block level simulation as well as embedded software tests for SoC designs.

For more detail of PSS, please visit Accellera PSWG page.

PSS (Portable Test and Stimulus Standard)

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. With this standard, users can specify a set of behaviours, from which multiple implementations may be derived.

- PSS has constructs for
    - Modelling Data flow (Buffers, Streams, States)
    - Modeling Behavior (Actions, Activities, Components, Resource, Pooling)
    - Constraints, Randomization, Coverage
- PSS is useful for SoC high-level test scenario creation

A concept of defining Registers and Sequences has been introduced in PSS2.0. Currently, three accesses are supported i.e., Read-Only, Read-Write, Write-Only.
IDS-Validate helps in generating the PSS register model through various inputs supported by IDS such as SystemRDL, IP-XACT, IDS-NG, Word, Custom CSV etc

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 / peek/poke
  - C/C++ : Consolidated read/write
  - Test/Validation : Multiple test sites – for testing multiple chips simultaneously
  - Target platform may not support hierarchy, loops, variables

SystemRDL (System Register Description Language)

- accellera – Standardized by the SystemRDL Working Group.
  https://www.accellera.org/activities/working-groups/systemrdl/

- “Excerpt from “Introduction”
  The SystemRDL language was designed specifically for describing and implementing registers and memory. SystemRDL allows developers to automatically generate and synchronize register specifications in hardware design, software development, verification, and documentation.

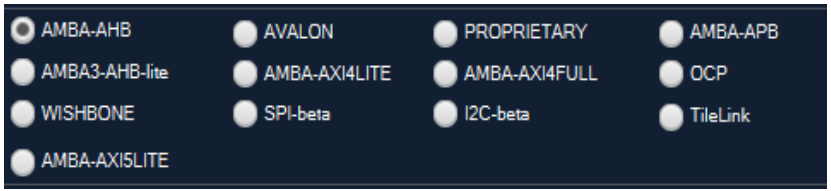
  The purpose behind language standardization is to significantly shorten the development cycle for hardware designers, hardware verification engineers, software developers, and document developers.

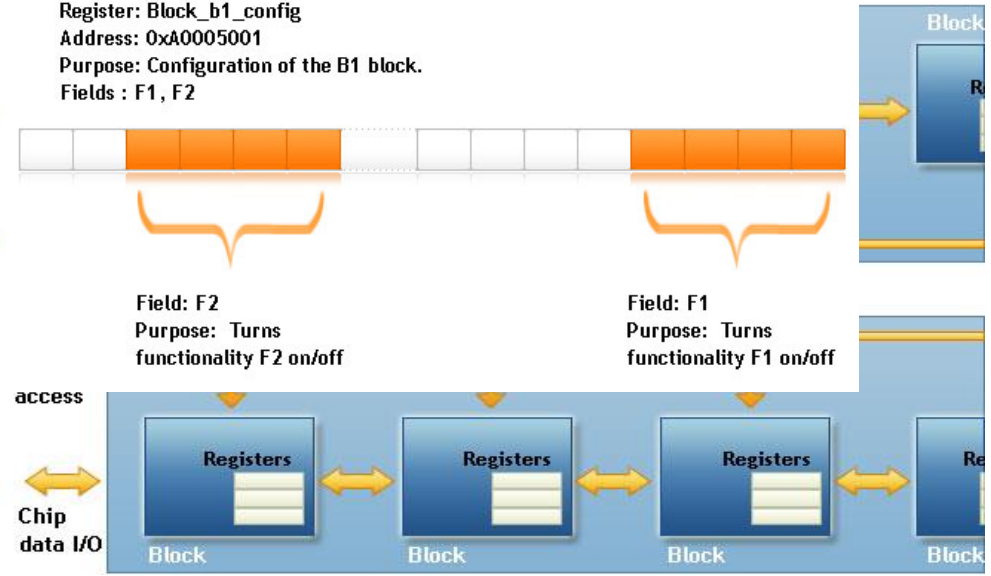
  intended to be applied for the following purposes
  — RTL generation & Validation
  — Document
  — Pass information to other tools such as debuggers
  — Software development (Register info.)

```
addrmap block1 {
 reg myReg #(longint unsigned  SIZE = 32, longint unsigned $P1 = 1)
   regwidth = SIZE; //documentation level parameter
   ispresent= $P1; //output level parameter
   field {
   } data[SIZE-1]; //parameter used in expressions
 };
  myReg reg32;
  myReg #(.SIZE(16)) reg16;  //Parameter overriding
};
struct my_struct {        //structures
  string foo ;
  string desc1;
} ;
```

Register Implementation in Hardware Design

- Characterized by a large number of control and status registers.
- Registers are important for making the chip/IP configurable.
- A configurable chip/IP is more versatile, and generates larger ROI.
- Supported Register Buses :

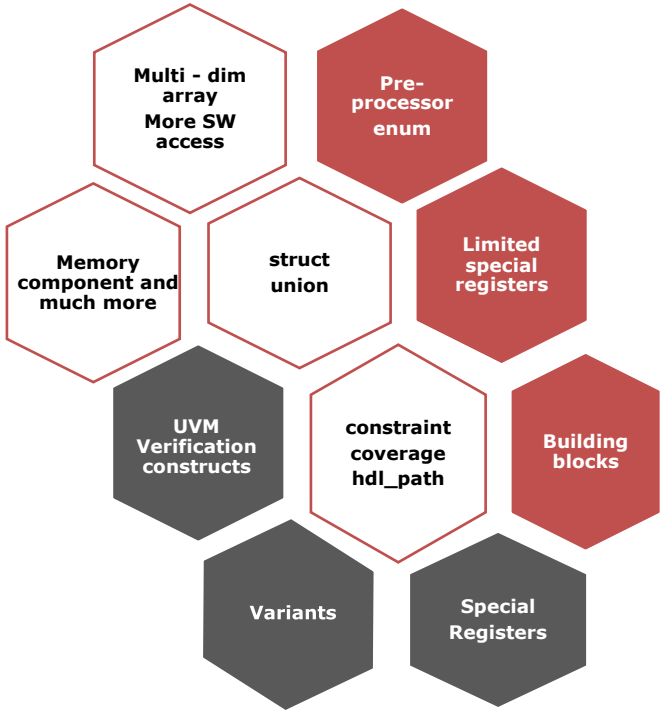




SystemRDL & Agnisys Innovations

A wide range of **special registers** are only supported by **AGNISYS**

AGNISYS	• Agnisys Enhancements • Special features for use by customers
SystemRDL 2.0	• Constructs given by Accellera SystemRDL 2.0 committee. • Has many constructs so that user can create whole spec in less time.
SystemRDL 1.0	• Some of the old construct that are already been used in the industry. • Includes preprocessor, components, limited special registers.



SystemRDL Register Model

In a SystemRDL (Register Description Language) specification, you can specify various information about registers. Here are the typical pieces of register information you can specify in SystemRDL:

- Register Name: Give a unique name to each register.
- Register Address: Define the memory-mapped address where the register is located.
- Register Access Type: Indicate whether the register is READ-ONLY, READ-WRITE, or WRITE-ONLY.
- Register Description: Provide a textual description or comment to describe the purpose and functionality of the register.
- Register Width: Specify the number of bits that the register contains.
- Register Fields: Define individual bit fields within the register, including their names, bit offsets, and bit widths.
- Field Descriptions: Describe the functionality and purpose of each individual field within the register.
- Reset Values: Specify the default values for the register and its fields after a reset or power-up.

SystemRDL Register Model

- Access Permissions: Define the permissions or access rights for the register and its fields, specifying who can read or write to them.
- Interrupt Information: If the register is related to interrupts, you can specify interrupt-related information such as interrupt enable bits, clear-on-read flags, etc.
- Reserved Bits: Indicate whether any bits in the register are reserved and should not be modified.
- Test and Debug Features: Specify any test and debug-related features associated with the register.
- Register Dependencies: Describe any dependencies or interactions between this register and other registers in the system.
- Address Regions: If applicable, specify the address regions or memory-mapped spaces where the register resides.
- Aliases: Define any alias names or alternative names for the register.
- Synchronization and Timing: Describe any synchronization or timing requirements for reading or writing to the register.

PSS Register Model

In PSS the Register model we can have limited register info as we can only defined these five info in PSS register Model

- Register Access -- READ-ONLY , READWRITE, WRITE-ONLY
- READ value and WRITE value MASK
- Register width
- Register offset value
- Address Region/Memory Region
- Reset value and Reset Mask

IDS-Validate (PSS Support)

- PSS 2.0 is a new* industry standard created by Accellera
- Agnisys is a working group member & contributed to standardization

Expertise in creating the Realization Layer

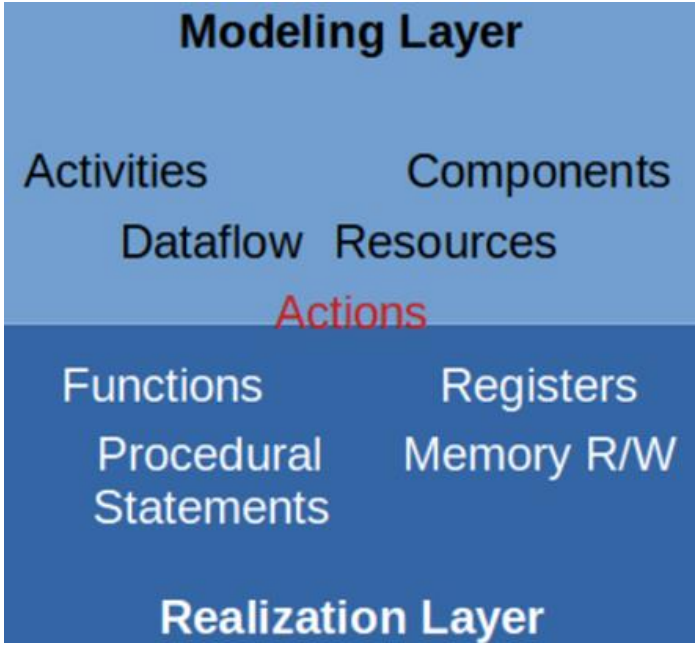
- Widest / Most comprehensive Register/Memory definition
- Pioneer in Sequence/Functions for IP/SoC

Agnisys offers

- Use PSS (or Excel, Python, GUI (NG)) to create Golden Spec for Sequences
- Generate C functions and UVM Sequences

Key Benefits

- Single Golden Source for Registers and Sequences reduces Time to Market, improves quality

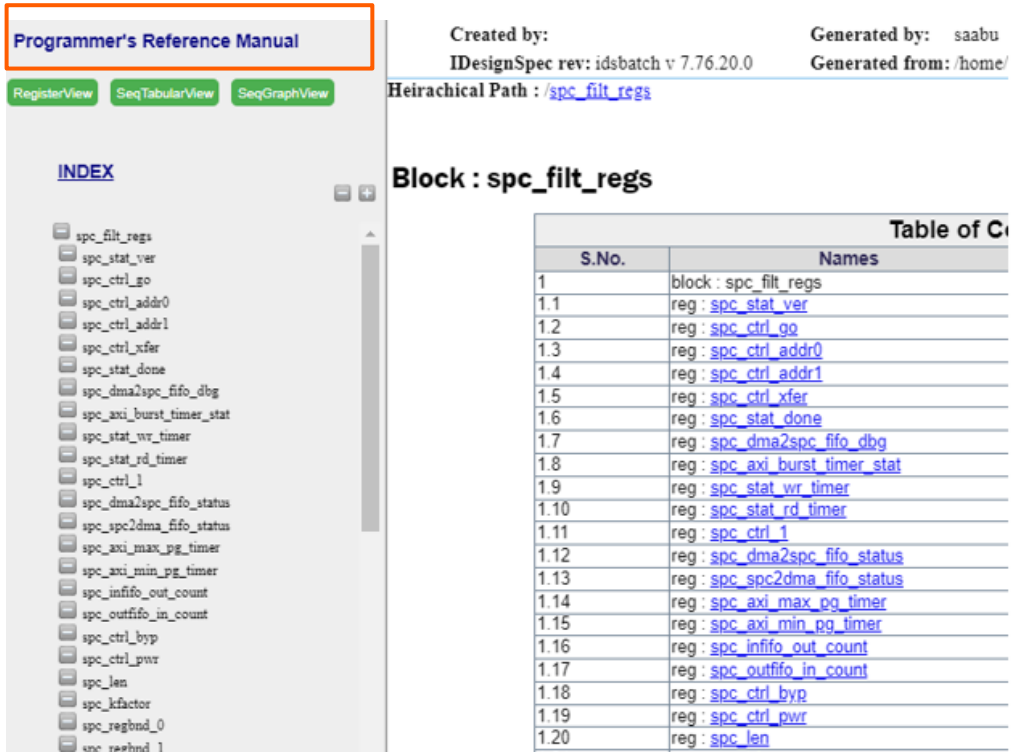


PRM (PROGRAMMER'S REFERENCE MANUAL)

Agnisys has developed a Programmer's Reference Manual (PRM) that serves as a comprehensive documentation resource for programming sequences and hardware architecture. It is intended to be a key reference for programmers, developers, and individuals seeking a detailed understanding of the intricacies of specific technologies

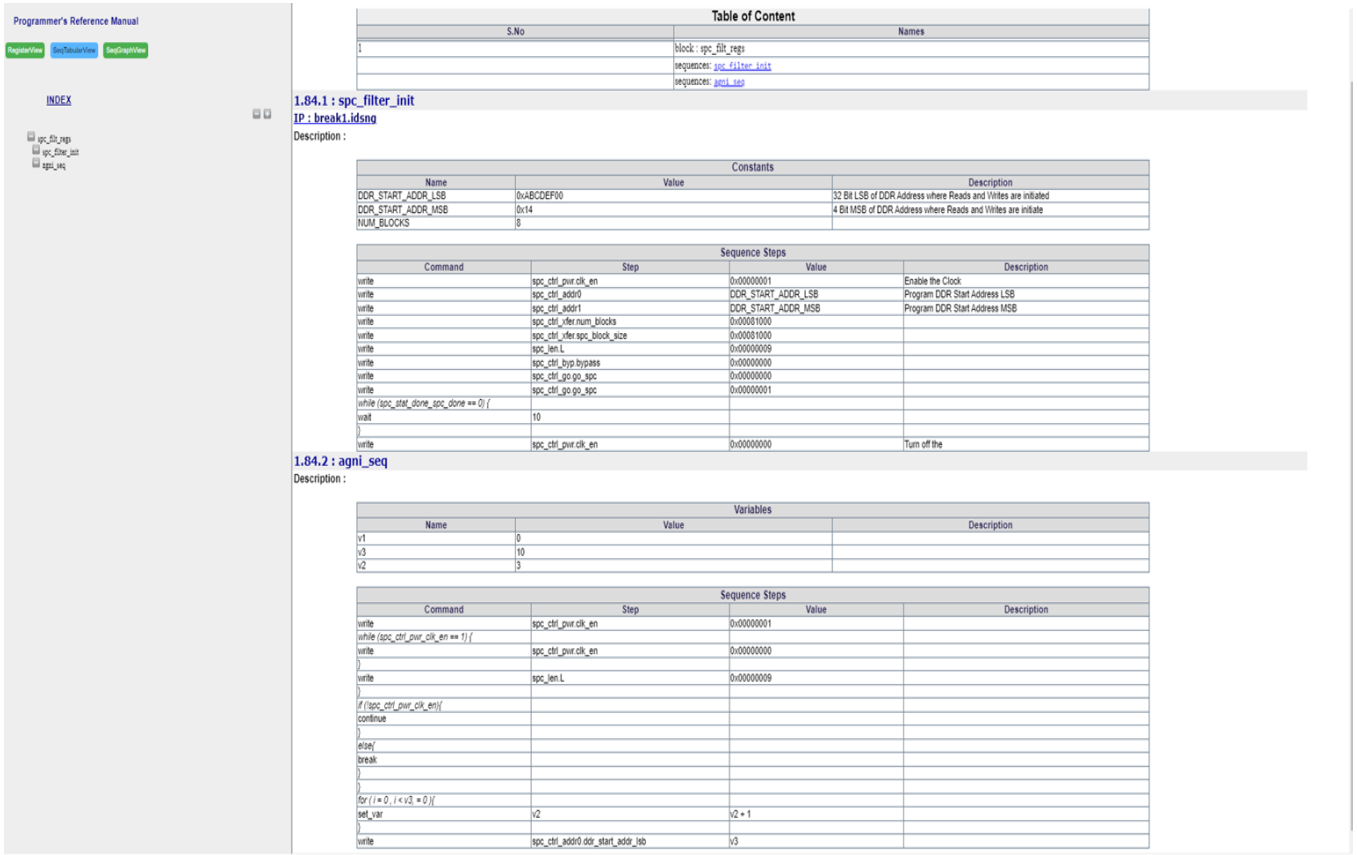
Agnisys' PRM comprises three views that are beneficial for users:

1. **Register View:** This view provides complete information on register and memory data.
2. **Sequences Tabular View**: This section presents detailed information about sequences in a tabular format. Users can conveniently track and access information related to each sequence.
3. **Flowchart View:** This view includes a graphical representation or flowchart illustrating the sequences, offering a visual understanding of the information flow.

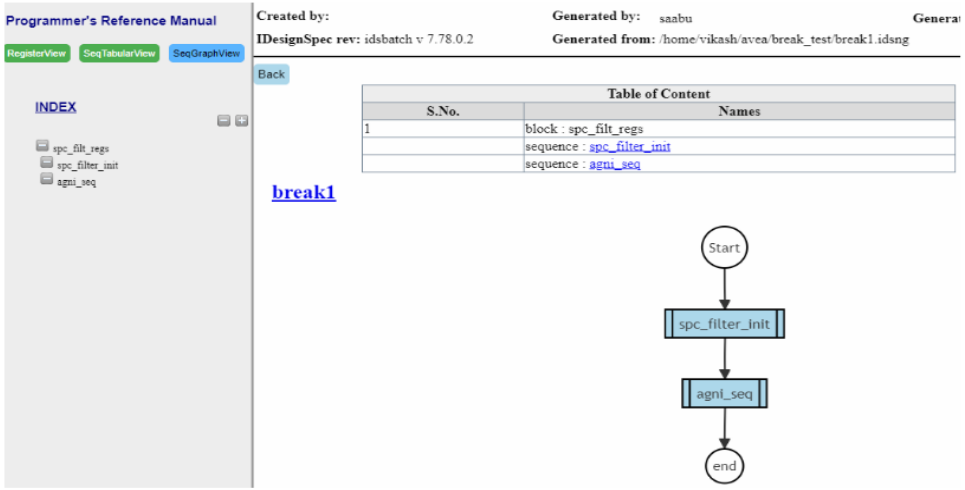


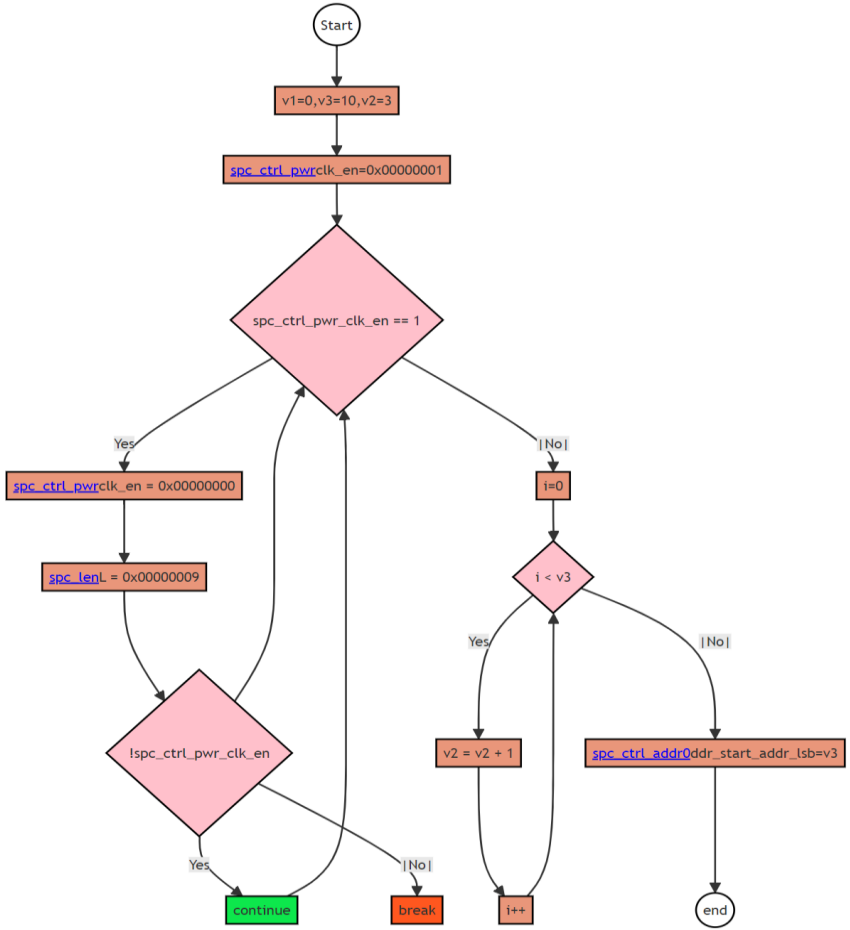
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Sequence Tabular view



Flowchart view





Device Driver

Device drivers act as intermediaries, translating high level commands from software into instructions that the hardware components, particularly semiconductor devices, can understand and execute. They provide a standardised interface for software applications, shielding them from the intricacies of the underlying hardware.

Functions of Device Drivers in VLSI:

a. Abstraction and Interface: Device drivers abstract the complexity of semiconductor devices, presenting a uniform interface to higher-level software. This abstraction shields software developers from the low level details of hardware implementation.

b. Initialization and Configuration: Drivers are responsible for initializing and configuring semiconductor devices during system startup. This involves setting parameters, establishing communication channels, and preparing the hardware for operation.

c. Data Transfer Management: Efficient data transfer between software and semiconductor devices is a critical function of device drivers. They manage the flow of data, handle buffering, and ensure the integrity of information exchanged between the software and hardware components.

An Example

```
action Status_control {

        MACHINE_STATUS_REGISTER_reg_s status_reg ;

        exec post_solve {
           status_reg.SLEEP_MODE_STATUS=1;
           status_reg.FREQUENCY_STATUS=1;
           status_reg.POWER_ON_RESET_STATUS=1;
           status_reg.VOLTAGE_CONTROL_STATUS=1;
           }

        exec body {
           message(" Checking for status registers " );
           while(status_reg.MACHINE_STATUS_REGISTER.read().POWER_ON_RESET_STATUS==0) {
                 message("Checking wheather status reg is set on not");
                 comp.regs.MACHINE_STATUS_REGISTER.write.VOLTAGE_CONTROL_STATUS(status_reg);
           }

           if( MACHINE_STATUS_REGISTER.read().SLEEP_MODE_STATUS == 0 && MACHINE_STATUS_REGISTER.read()
                 comp.regs.MACHINE_STATUS_REGISTER.write(contr_reg);
                 message(" ALL STATUS REGISTER ARE RESET " );
           }
    }

}
```

PSS SEQUENCES

C - FIRMWARE OUTPUT

```
int status_control( ) {

  int  MACHINE_CONTROL_REGISTER_ENABLE;
  int  MACHINE_STATUS_REGISTER_SLEEP_MODE_STATUS;
  int  MACHINE_STATUS_REGISTER_POWER_ON_RESET_STATUS;
  int  MACHINE_STATUS_REGISTER_FREQUENCY_STATUS;


  FIELD_WRITE(Machine_power_controller_MACHINE_STATUS_REGISTER_ADDRESS,0x00000002,
  MACHINE_POWER_CONTROLLER_MACHINE_STATUS_REGISTER_SLEEP_MODE_STATUS_OFFSET);


  FIELD_WRITE(Machine_power_controller_MACHINE_STATUS_REGISTER_ADDRESS,0x00000008,
  MACHINE_POWER_CONTROLLER_MACHINE_STATUS_REGISTER_FREQUENCY_STATUS_OFFSET);


  FIELD_WRITE(Machine_power_controller_MACHINE_STATUS_REGISTER_ADDRESS,0x00000020,
  MACHINE_POWER_CONTROLLER_MACHINE_STATUS_REGISTER_POWER_ON_RESET_STATUS_OFFSET);


 FIELD_WRITE(Machine_power_controller_MACHINE_STATUS_REGISTER_ADDRESS,0x00000040,
 MACHINE_POWER_CONTROLLER_MACHINE_STATUS_REGISTER_VOLTAGE_CONTROL_STATUS_OFFSET);


  // Call firmware print method

  printf("Checking for status registers",);
```

UVM OUTPUT

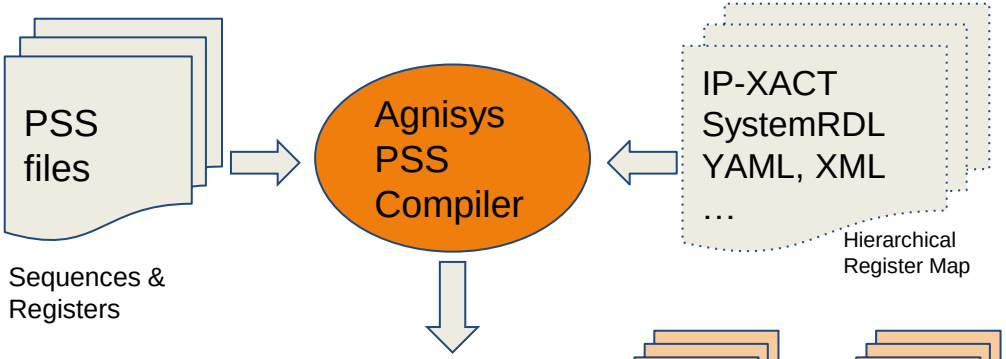
```
Class       : Machine_power_controller_REQUEST_CONTROL_REG
DESCRIPTION:-
------------------------------------------------------------------------------*/
`ifndef CLASS_Machine_power_controller_REQUEST_CONTROL_REG
`define CLASS_Machine_power_controller_REQUEST_CONTROL_REG
class Machine_power_controller_REQUEST_CONTROL_REG extends uvm_reg;
    `uvm_object_utils(Machine_power_controller_REQUEST_CONTROL_REG)

    rand uvm_reg_field REQ_CONTR;/**/

    // Function : new
    function new(string name = "Machine_power_controller_REQUEST_CONTROL_REG");
        super.new(name, 32, build_coverage(UVM_NO_COVERAGE));
        add_coverage(build_coverage(UVM_NO_COVERAGE));
    endfunction

    // Function : build
    virtual function void build();
        this.REQ_CONTR = uvm_reg_field::type_id::create("REQ_CONTR");
        this.REQ_CONTR.configure(.parent(this), .size(1), .lsb_pos(1), .access("RW"),
    endfunction
endclass
`endif

/*------------------------------------------------------------------------------
Class       : Machine_power_controller_ACKNOWLEDGE_REG
DESCRIPTION:-
------------------------------------------------------------------------------*/
`ifndef CLASS_Machine_power_controller_ACKNOWLEDGE_REG
`define CLASS_Machine_power_controller_ACKNOWLEDGE_REG
class Machine_power_controller_ACKNOWLEDGE_REG extends uvm_reg;
    `uvm_object_utils(Machine_power_controller_ACKNOWLEDGE_REG)

    rand uvm_reg_field UNDERVOLTAGE_OR_OVERVOLTAGE;/**/
    rand uvm_reg_field SHORT_CIRCUIT;/**/
    rand uvm_reg_field POWER_FAIL;/**/
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Agnisys® PSS Compiler

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Possible Outputs From PSS Files: Tests



1. **Models**	RTL Registers	UVM RAL Reg Abstraction Layer	C/C++ Header Registers	HTML PDF Registers	Flow Chart View Sequences	
2. **Run Env**	Verification (SV/UVM based)		Validation (C/C++ based)			
3. **Tests**	1. Automatic Reg tests 2. UVM based PSS tests		1. Automatic Reg tests 2. **C based PSS tests**			
4. **Misc**	Custom outputs (Velocity Template based)		*Makefile, *main.c			

PSS Editor

This new addition enables you to work with PSS files, create and edit portable stimulus models and tests with ease and ensures a seamless experience for engineers and testers.

Key Features:

1. PSS File Management: Create new PSS files, open existing ones, and organize your project resources in a user-friendly interface.

2. Syntax Highlighting: Syntax highlighting and code formatting

3. Code Navigation: Features like code folding, context-aware code suggestions, and jump-to-definition functionality.

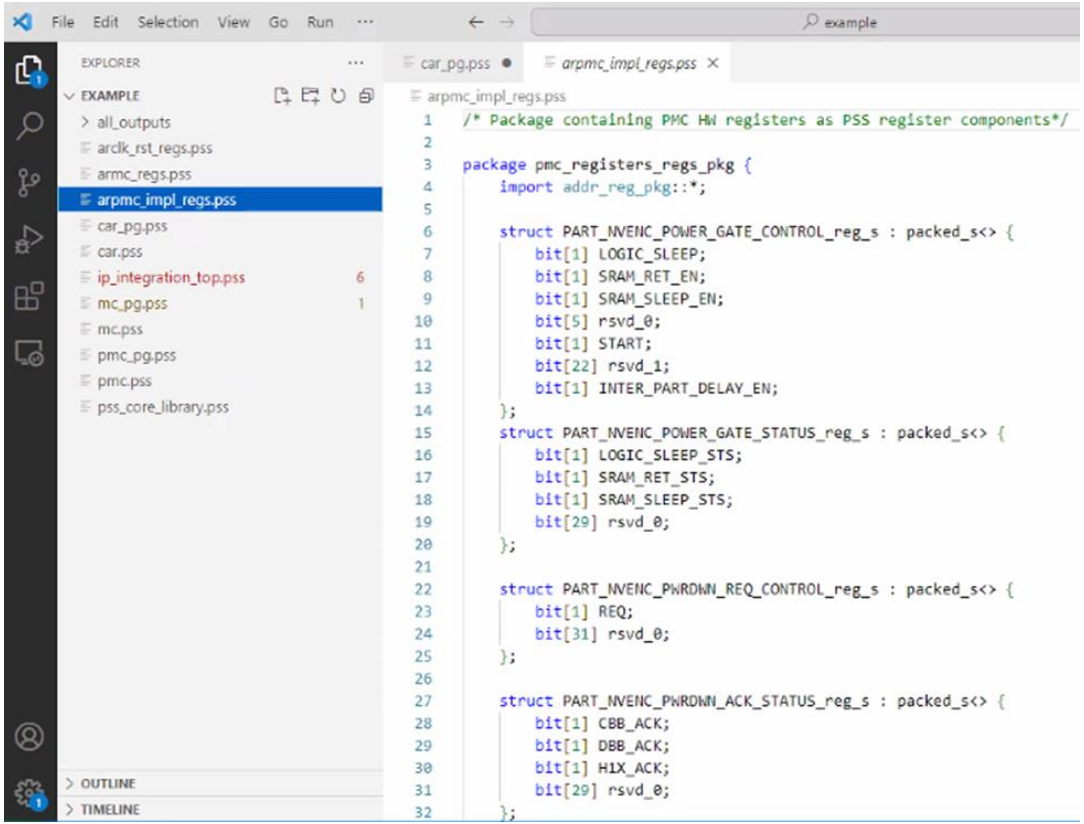
4. Validation and Semantic checks: Utilise built-in validation and debugging tools to ensure your PSS models and tests adhere to industry standards and functional requirements.

5. Search and Replace: Quickly find and replace elements within your PSS code

To install and use the tool, download and install it directly from the Visual Studio Code (VS Code) Marketplace and Follows the instruction given in README.md

https://marketplace.visualstudio.com/items?itemName=AgnisysInc.agnisysPSS

PSS Editor



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Conclusion

The SoC specification defining the registers and memory can be written in SystemRDL format as well as in PSS 2.0 format released by Accellera recently.

Both SystemRDL and PSS powerful compilers have been written to generate various outputs such RTL, UVM, Headers and documentation. There should be a way to generate custom tests for boards as well as UVM and UVM-C based environments through a common specification. This provides a solution for firmware engineers to write and debug their device drivers and application software. Therefore, PSS helps in the solution for SOC/IP teams who aim to cut down the verification and validation time, through automatic generation of UVM and sequences which enables exhaustive testing of memories and register maps.

This approach also unifies the creation of portable sequences from a golden specification. Sequences can be captured in PSS, python, spreadsheet format, or GUI(NG) and Register models has been capture in system RDL and generate multiple output formats for a variety of domains:

- UVM sequences for verification
- SystemVerilog sequences for validation
- C code for firmware and device driver development
- Specialized formats for automated test equipment (ATE)
- Hooks to the latest Portable Stimulus Standard (PSS)
- Programmer Reference Mannual (PRM)

Thank You

Agnisys, Inc.

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY