Automatic Generation of Implementation Layer for Embedded System using PSS and SystemRDL

SUDHIR BISHT
R & D ENGINEER
AGNISYS

accellera SYSTEMS INITIATIVE™

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

About us

Pioneer in providing design & verification solutions for hardware/software interface

Founded in 2007 in Boston, MA

Privately held, US owned business.

Profitable since founded!

Worldwide offices

- 50+ Engineers Worldwide
- 24x7 Worldwide support
- Sales Offices - USA, Europe, Japan, S Korea, China, Taiwan & India
- R&D centers in USA and India

Trusted by 40+ customers

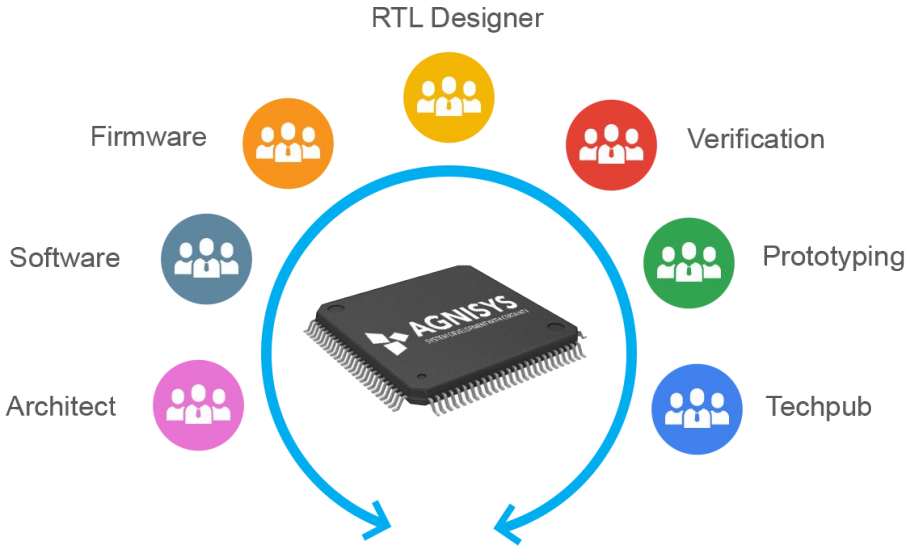
1000+ users worldwide

Over 80% Customer retention rate

accellera SYSTEMS INITIATIVE™

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

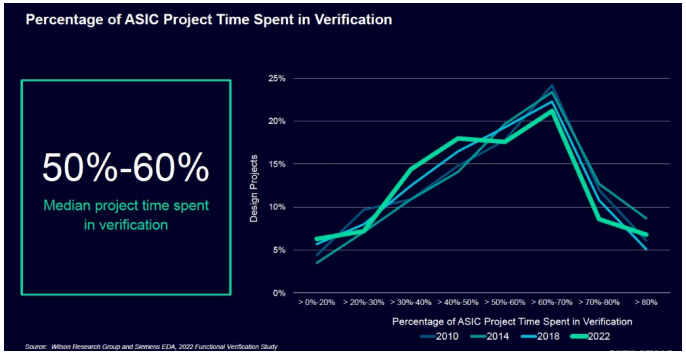
Agnisys Maximizes Productivity & Eliminates Errors

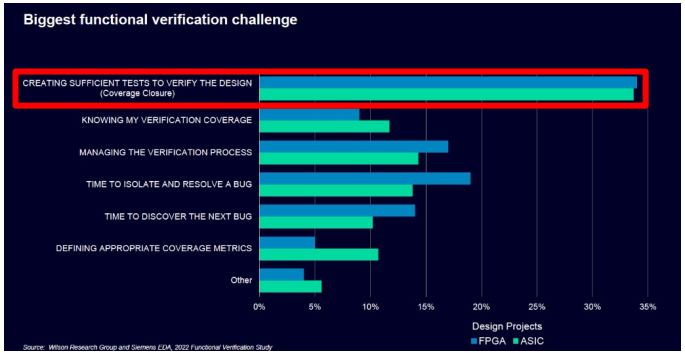


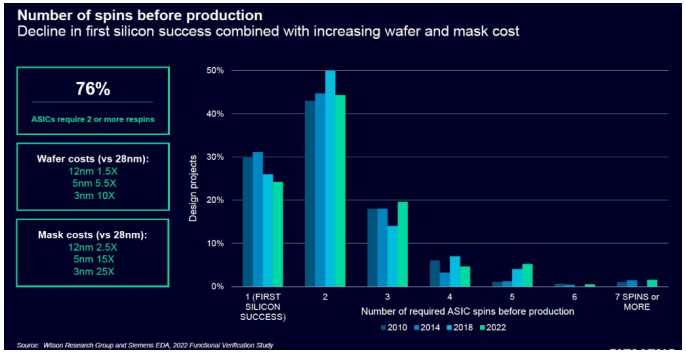
- Automatic Generation of SoC HSI:
  - Synthesizable RTL
  - UVM Register Abstraction Layer (RAL)
  - Software models
  - Documentation
- Single Golden Specification across Teams
- Correct-by-Construction Output for Teams

Peace of Mind: No Errors introduced in HSI portion of SoC

The State of Verification in 2023







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

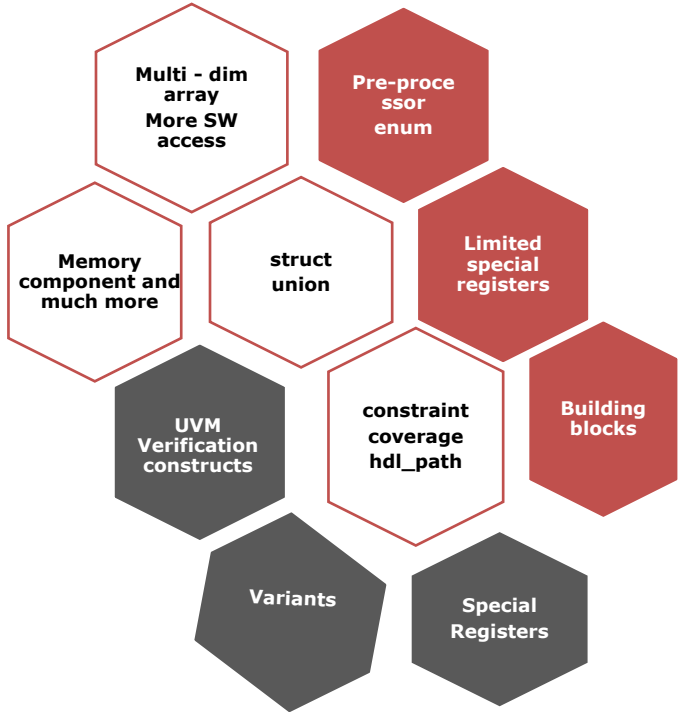
  creating sufficient tests

Source: Wilson Research Group 2022, courtesy Siemens EDA

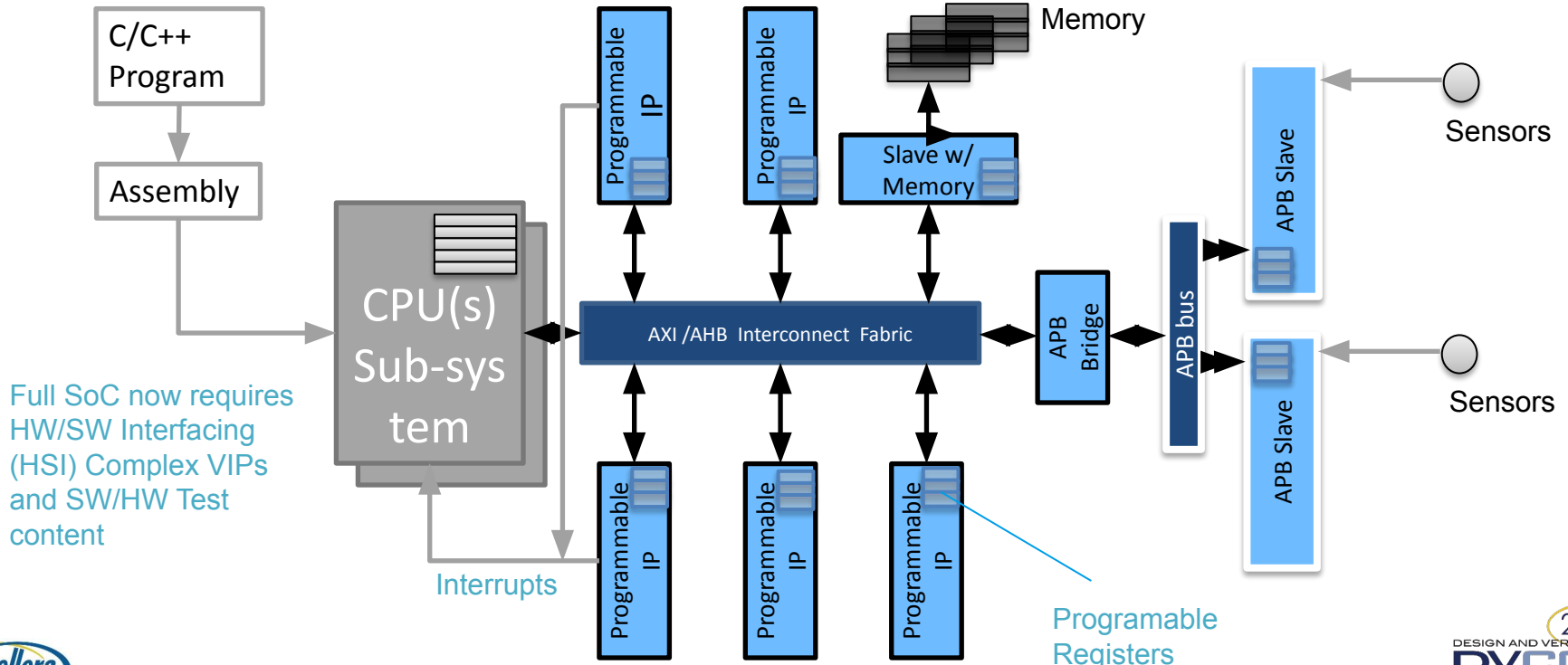
SystemRDL & Agnisys Innovations

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

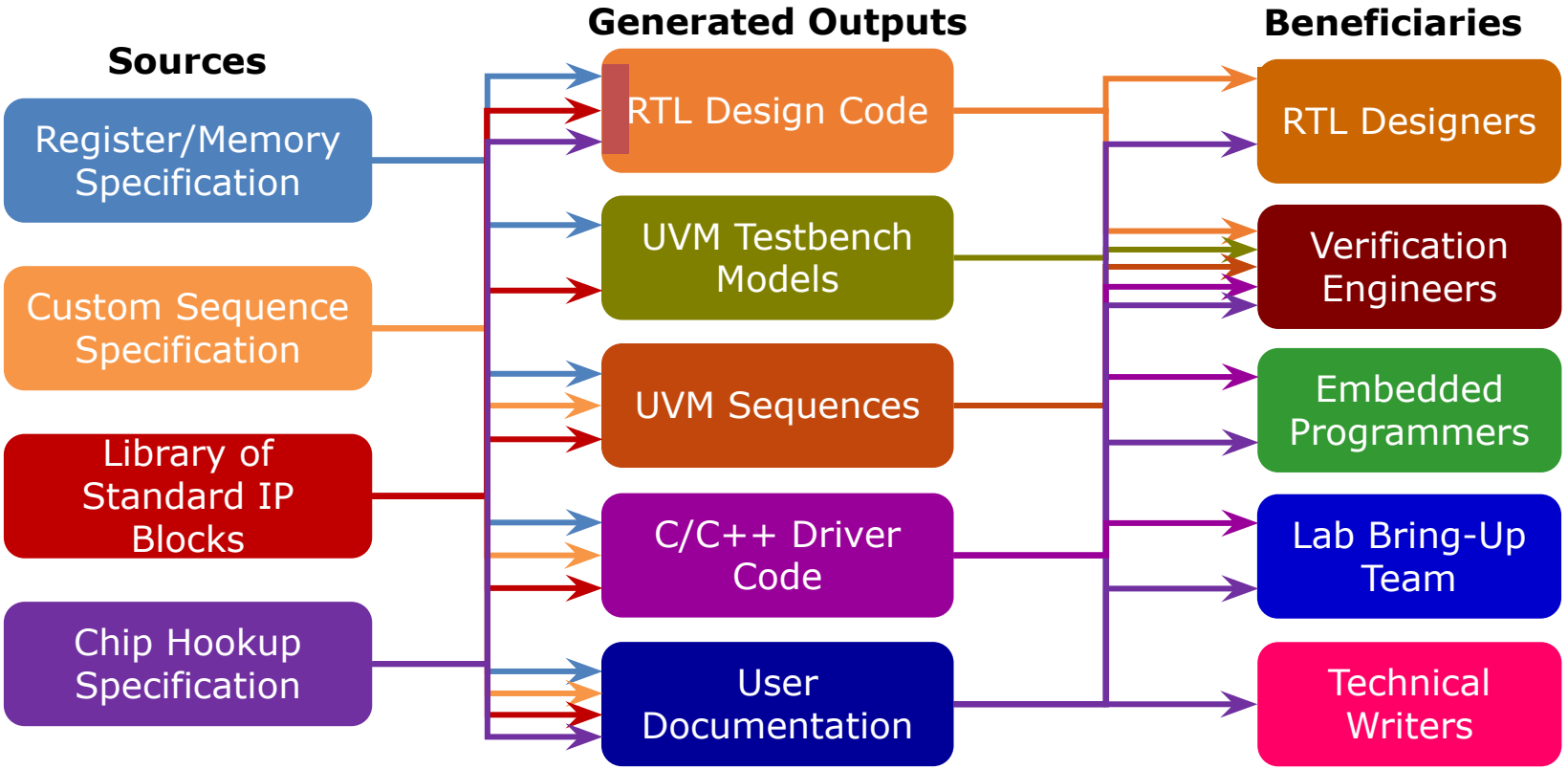
AGNISYS	• Agnisys Enhancements • Special features for use by customers
SystemRDL 2.0	• Constructs given by Accellera SystemRDL 2.0 committee. • Has many constructs so that user can create whole spec in less time.
SystemRDL 1.0	•Some of the old construct that are already been used in the industry. • Includes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



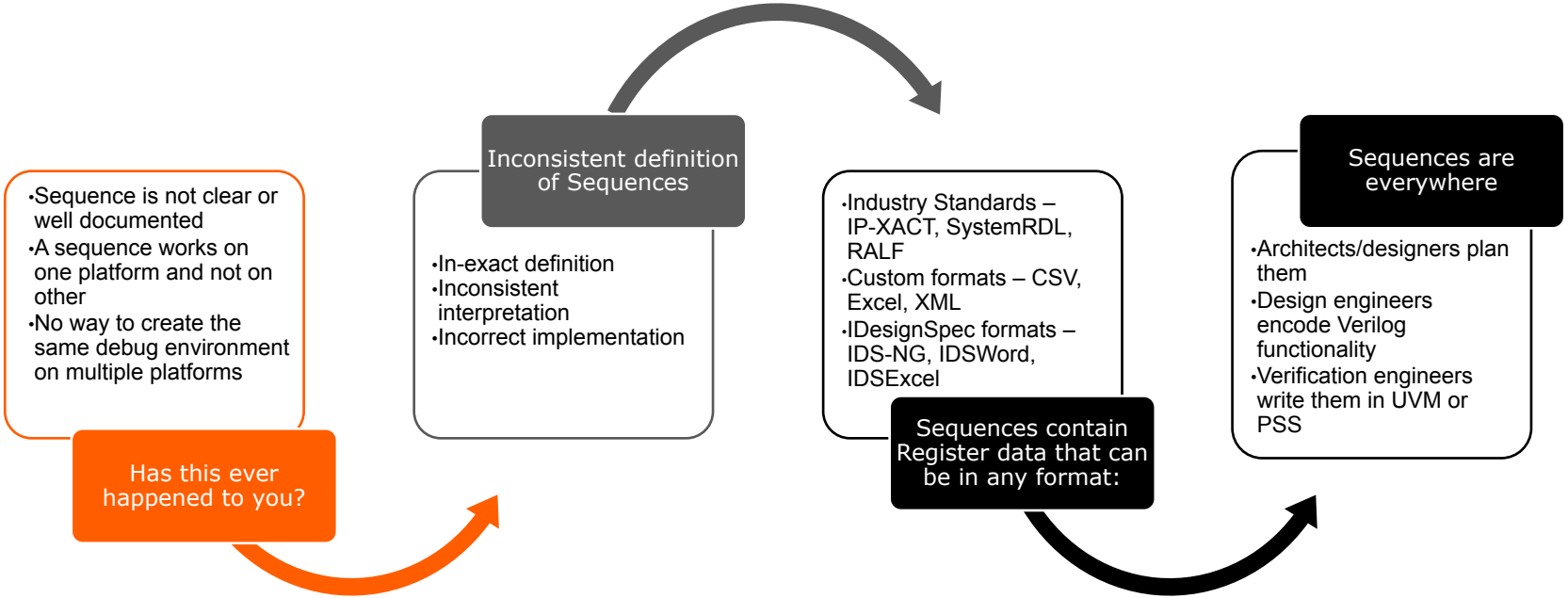
HW/SW Interface of a Typical SoC



Typical SoC Development

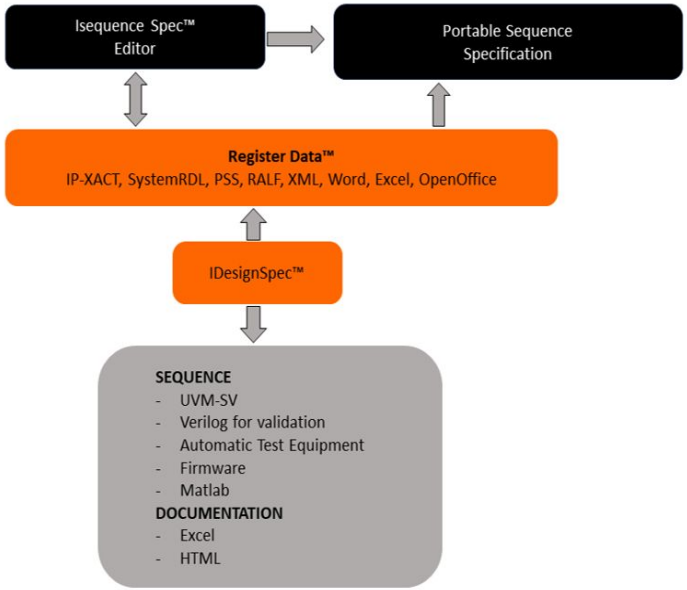


Challenges Development Teams Face with Sequences



An Ideal Solution

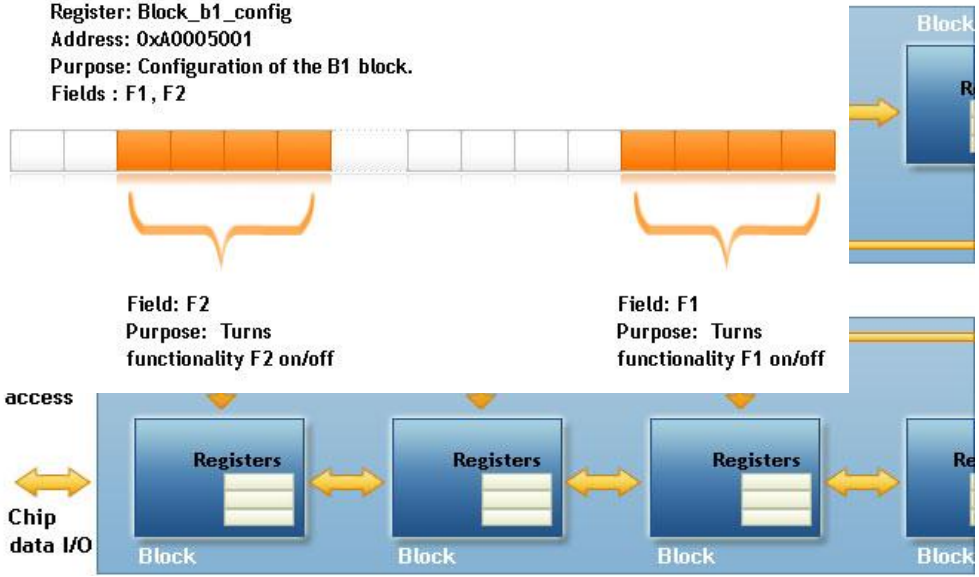
- Describe the programming and test sequences of a device and automatically generate sequences ready to use from an early design and verification stage to post silicon validation
- Centralize creation of sequences from a single specification and generate various output formats for multiple SoC teams
  - 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



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 :

◉ AMBA-AHB	○ AVALON	○ OCP	○ PROPRIETARY	○ AMBA-AXI		
○ AMBA-AXI4FULL	○ AMBA-APB	○ AMBA3-AHB-lite	○ WISHBONE	○ SPI -beta	○ I2C -beta	



SystemRDL (System Register Description Language)

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

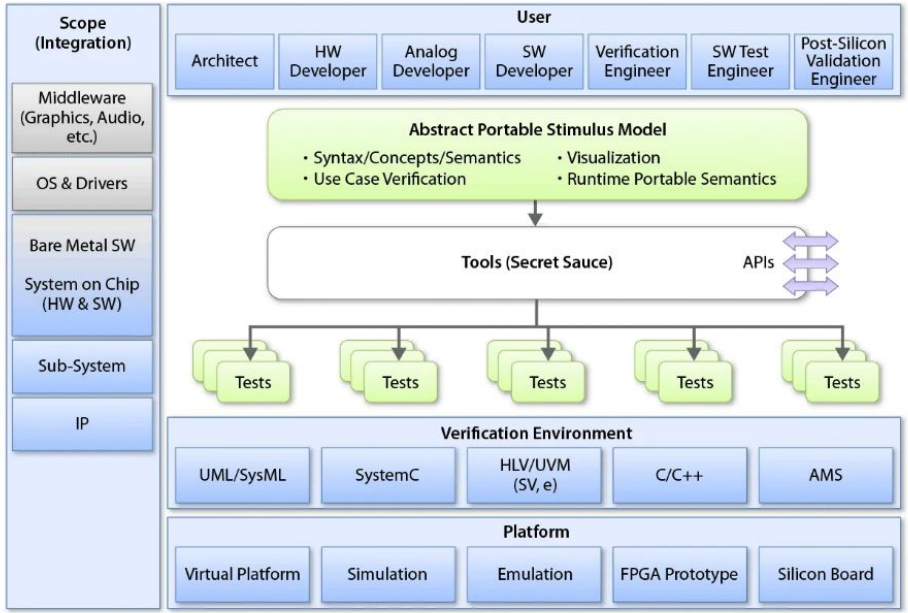
- "Excerpt from "Introduction"
  The SystemRDL language was designed specifically for describing and implementing registers and memory.
  SystemRDL allows developers to automatically generate and synchronize register specifications in hardware design, software development, verification, and documentation.

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;
} ;
```

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

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 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

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

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

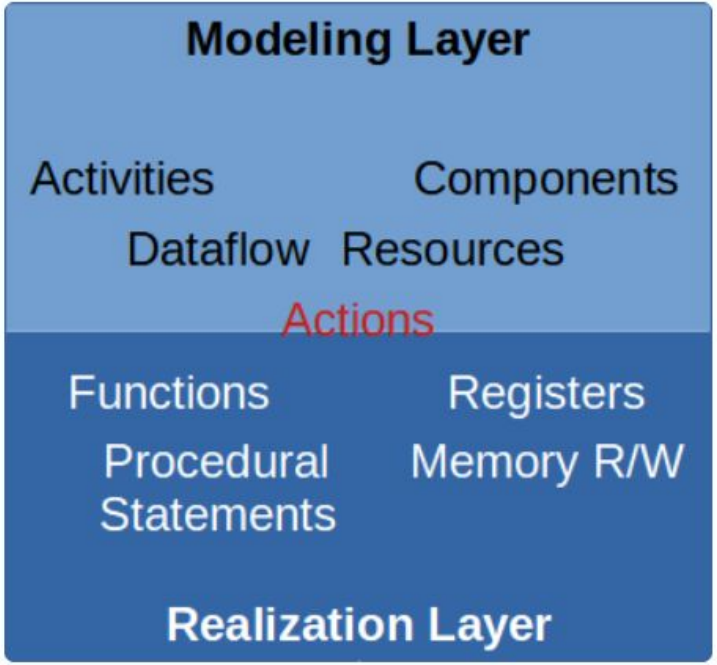
- Widest / Most comprehensive Register/Memory definition
- Pioneer in Sequence/Functions for IP/SoC

Agnisys offers

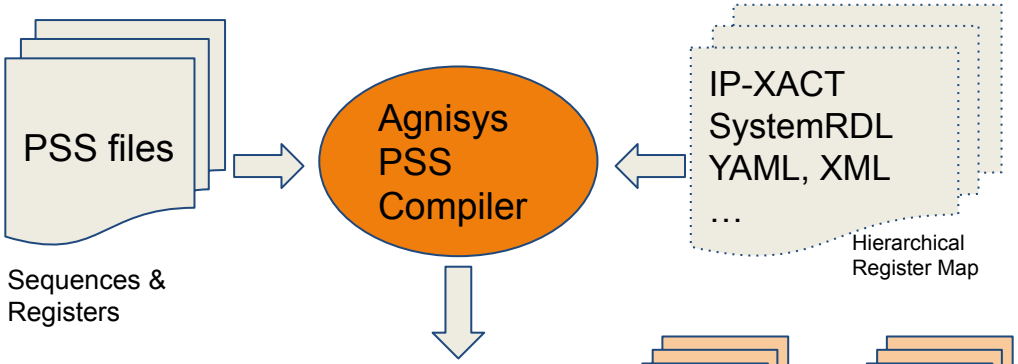
- Use PSS (or Excel, Python, GUI (NG)) to create Golden Spec for Sequences
- Generate C functions and UVM Sequences

Key Benefits

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



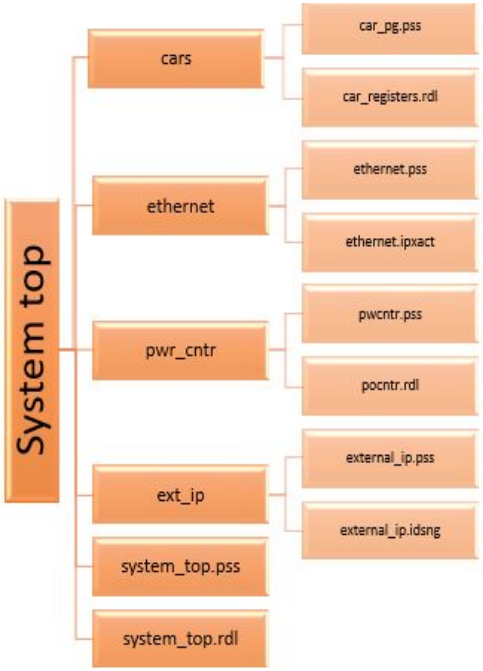
Possible Outputs From PSS Files: Tests



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 Compiler

Directory Structure



SystemRDL Input

```
property chip {type=string; component=addrmap;};
addrmap system_top {
  chip= true;
  addrmap top_chip {
     chip =true;
     refpath="top_chip.idsng:top_chip";
     };
  addrmap extrenal_ip{
      refpath="elevator controller system.idsng:elevator";
     };
  addrmap pwctrl {
      refpath = "power_controller.idsng:Machine_power_controller";
      };
    top_chip top_chip;
    external_ip external_ip;
    pwctrl  pwctrl;
};
```

PSS Input

```
extend component car_c {
    import car_hsi_target_interface::*;

    action car_NVENC_Enable_clk_act{
        exec body {
            target_car_NVENC_Enable_clk_function();
        };
    };

    action car_NVENC_Reset_act{
        exec body {
            target_car_NVENC_Reset_function();
        };
```

C Output

```
#include "top_chip.h"
#include "alldefine.h"
#include "car_reg_iss.h"
#include <stdio.h>


int car_nvenc_enable_clk( ) {

  FIELD_WRITE(top_chip_car_registers_CLK_OUT_ENB_NVENC_SET_ADDRESS, 0x80000000, TOP_CHIP_CAR_REGISTERS_CLK_OUT_ENB_NVENC_SET_SET_CLK_ENB_NVENC_MASK,
  TOP_CHIP_CAR_REGISTERS_CLK_OUT_ENB_NVENC_SET_SET_CLK_ENB_NVENC_OFFSET);

  return 0;

}

int car_nvenc_reset( ) {

  FIELD_WRITE(top_chip_car_registers_RST_DEV_NVENC_SET_ADDRESS, 0x80000000, TOP_CHIP_CAR_REGISTERS_RST_DEV_NVENC_SET_SET_SWR_NVENC_RST_MASK,
  TOP_CHIP_CAR_REGISTERS_RST_DEV_NVENC_SET_SET_SWR_NVENC_RST_OFFSET);

  return 0;

}
```

UVM Output

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

    uvm_status_e status;

    car_registers_block rm ;

    function new(string name = "uvm_car_nvenc_disable_clk_seq") ;
        super.new(name);

    endfunction

    task body;

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

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

        rm.CLK_OUT_ENB_NVENC_CLR.CLR_CLK_ENB_NVENC.write(status, 'h1, .parent(this));
```

Additional Automated Outputs Generated

- RAL Model, UVM, UVM sequences with verification environment
- RTL
- PDF/HTML
- C tests & C sequences with validation environment
- Headers

accellera SYSTEMS INITIATIVE™

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

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)
- Documentation outputs such as HTML and flowchart