Automatic Testbench Build to Reduce Cycle Time and Foster Reuse

Joachim Geishauser

Alexander Schilling

SECURE CONNECTIONS
FOR A SMARTER WORLD

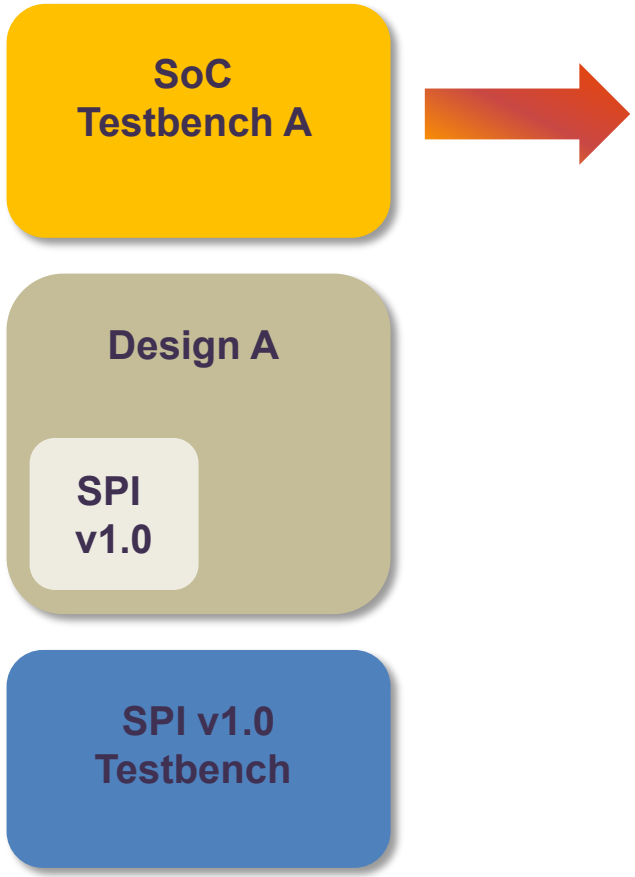
accellera
SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

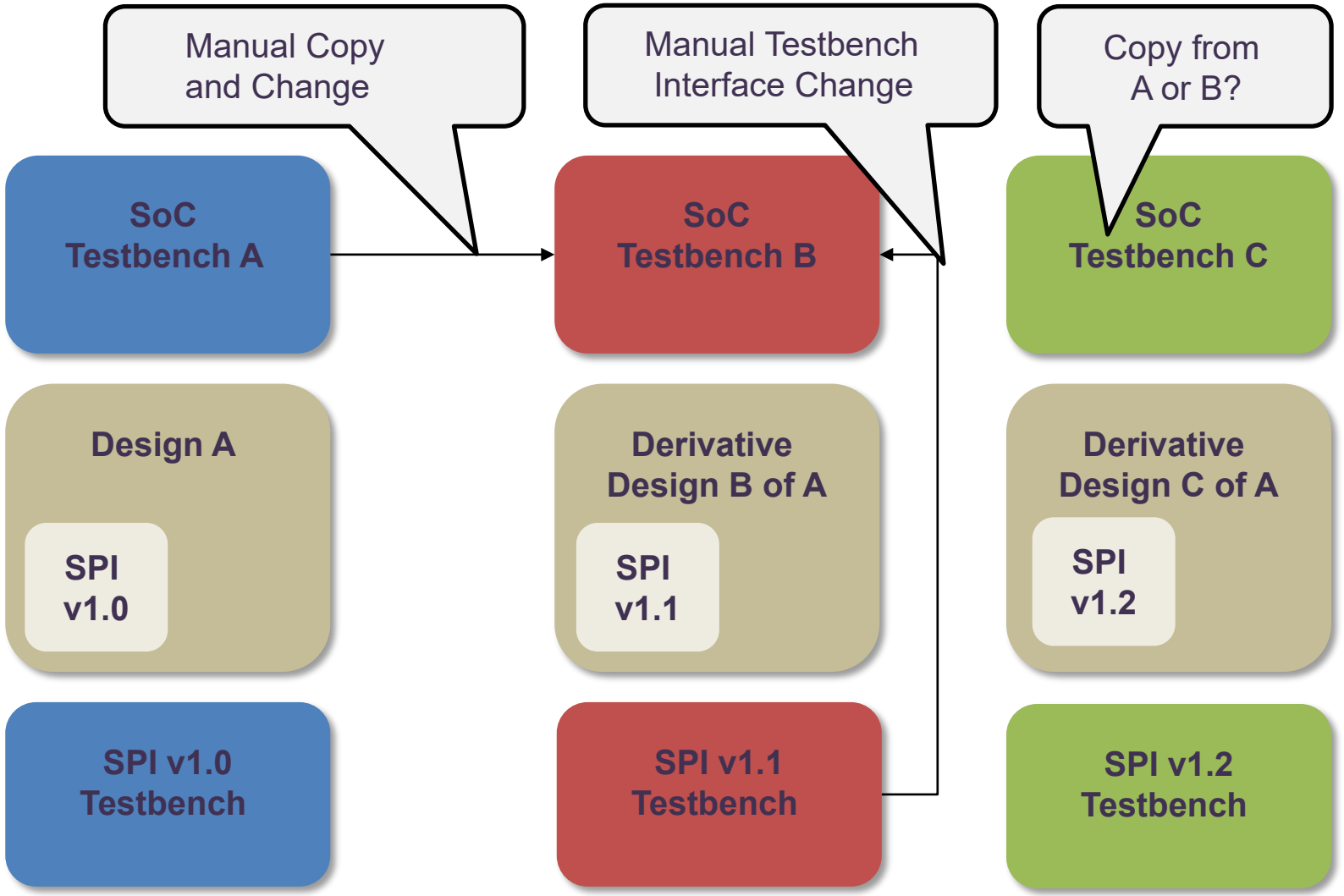
- Introduction
- Role Models
- Function Details
- MagniV Implementation

Testbench Bill of Material

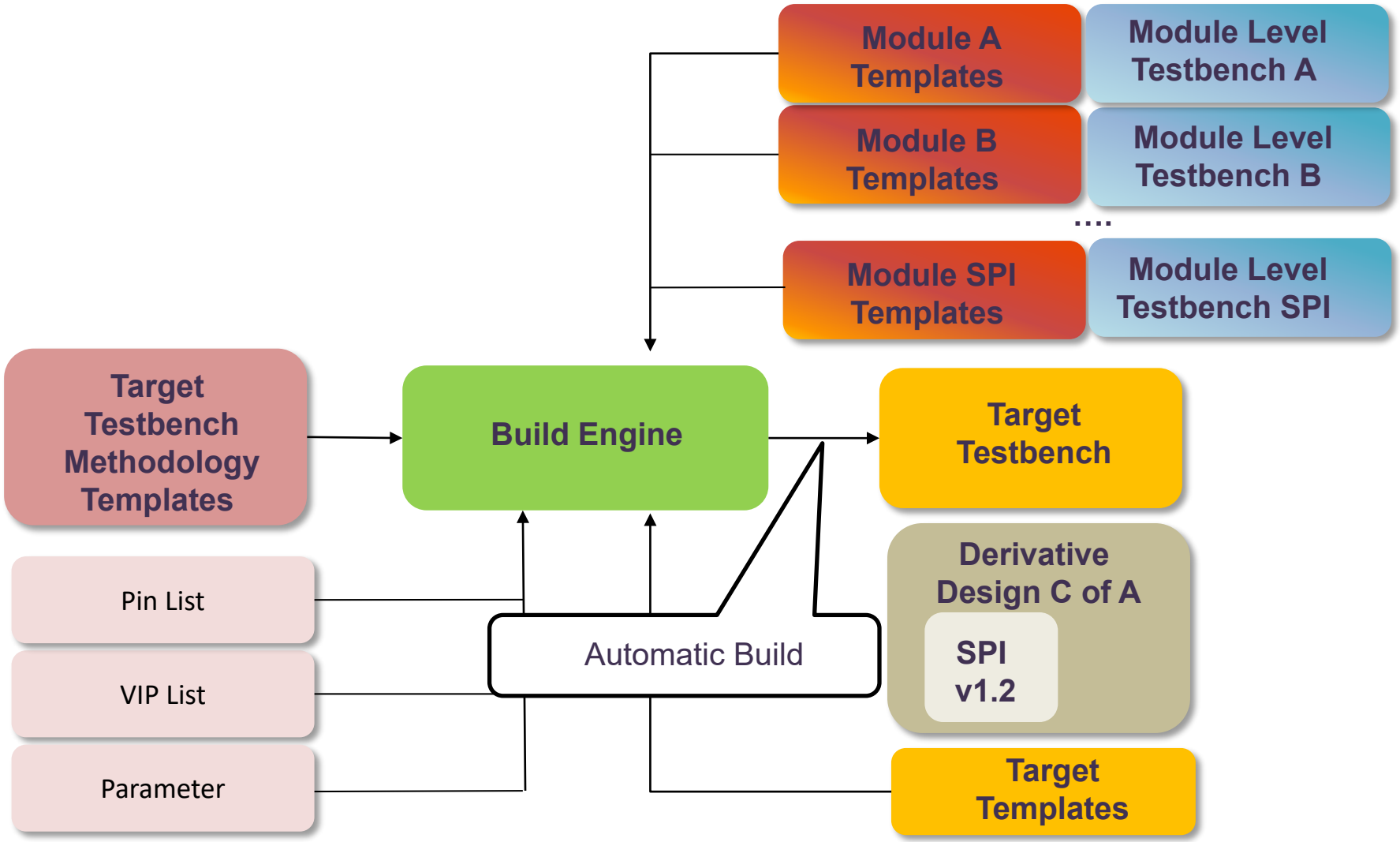


- IPDSS Directory Structure
- Stingray Meta Data
- System Verilog Testbench
- C Stimulus
- SoC Header Creation
- Regression
- Verification Guide
- Tool Configurations
  - Simulation (VCS/IUS)
  - Ifv
  - Certitude
- Xfilter Definitions
- Creation of ClearCase Elements

Traditional Testbench Setup



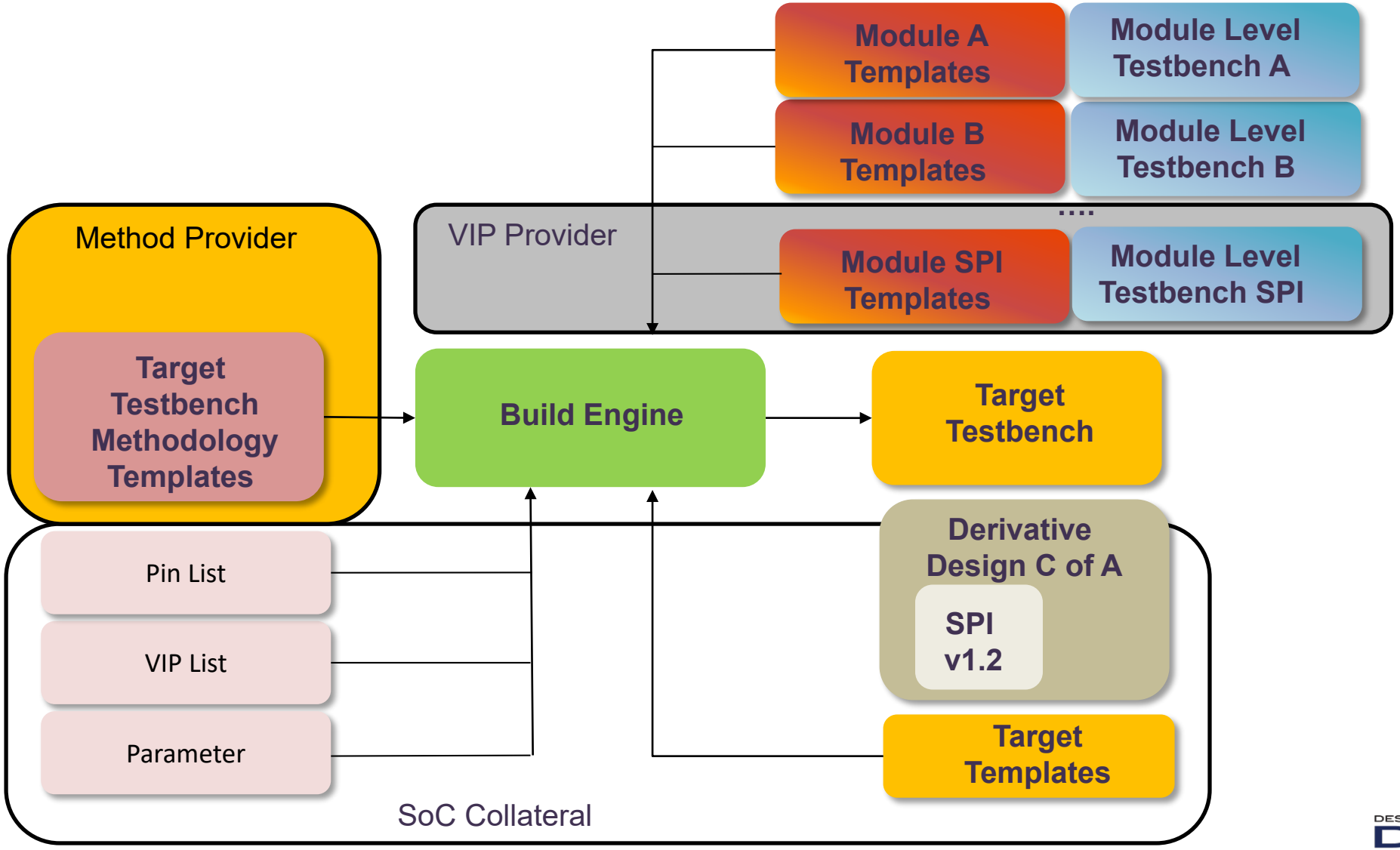
Automation Components Overview



Agenda

- Introduction
- Role Models
- Function Details
- MagniV Implementation

Role Models

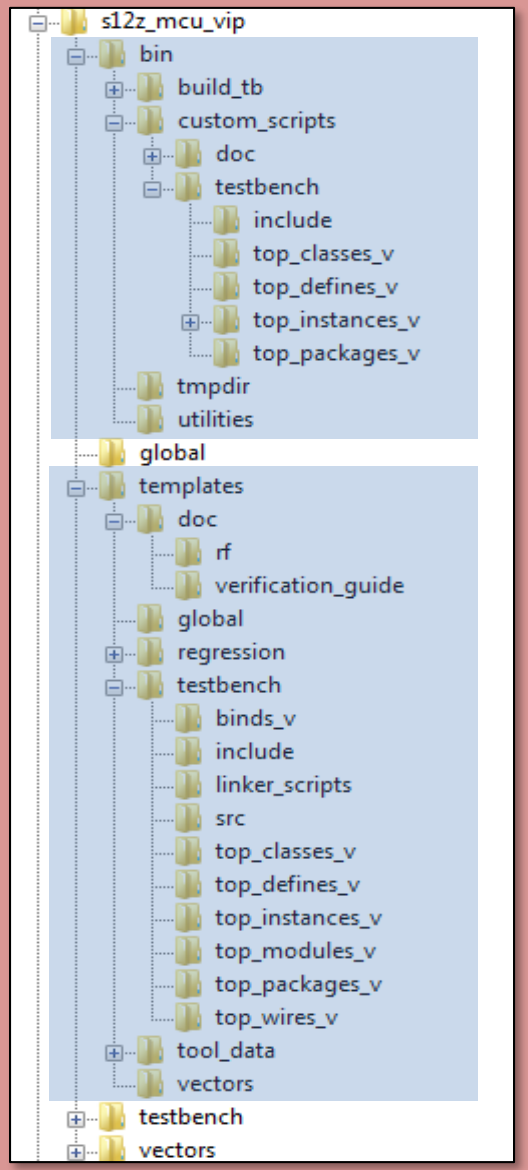

Module A Templates
Module Level Testbench A
Module B Templates
Module Level Testbench B
....
Module SPI Templates
Module Level Testbench SPI
VIP Provider
Method Provider
Target Testbench Methodology Templates
Build Engine
Target Testbench
Derivative Design C of A
SPI v1.2
Target Templates
Pin List
VIP List
Parameter
SoC Collateral

Method Provider

- Depends on Testbench Architecture
- Defines Testbench Build Mechanisms
- Provides Base Templates
- Provides Base Scripts

- Implements extended IPDSS directory structure

Method View



VIP Provider

VIP View

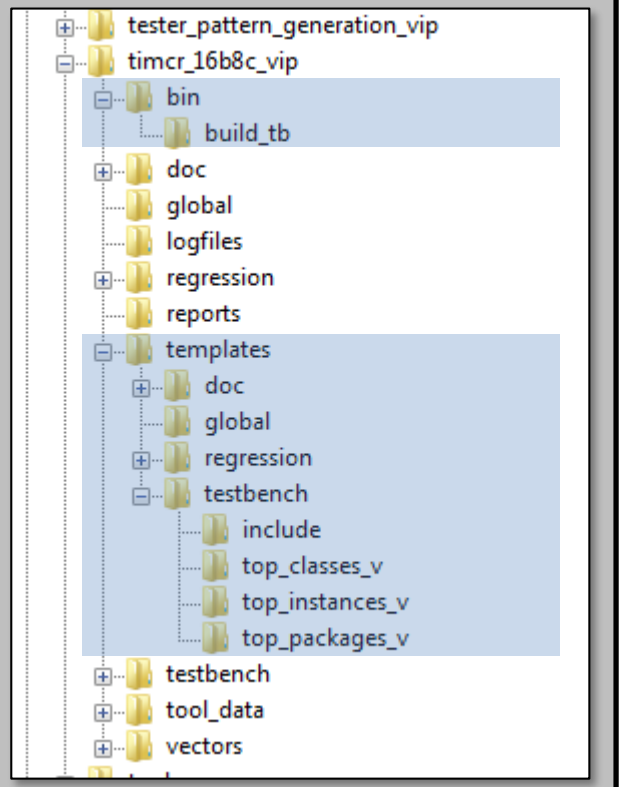
Perl Functions

Verification Guide

Regression Files

Integration Stimulus

Verification Drivers and Monitors



- Provides Integration Stimulus
- Provides required drivers and monitors
- Provides Verification Guide (VG) sections for SoC VG
- Provides assertions
- Provides regression files
- Provides register definitions for SoC
- Implements extended IPDSS directory structure

SoC Collateral

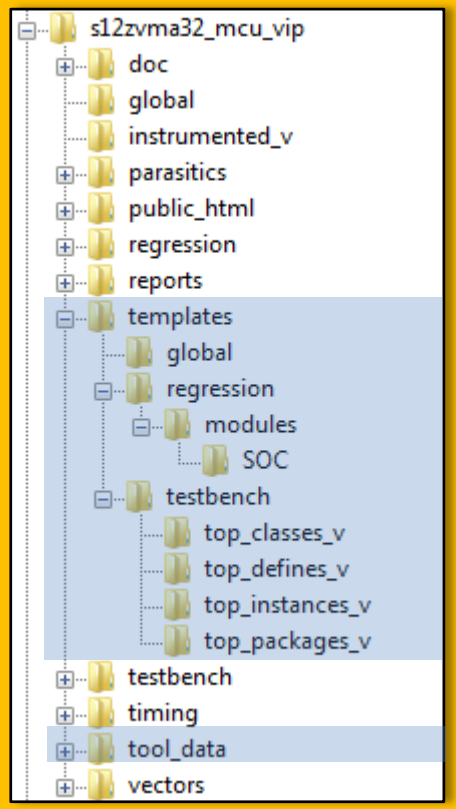
SoC View

SoC template files

Pin List

VIP List

Parameter



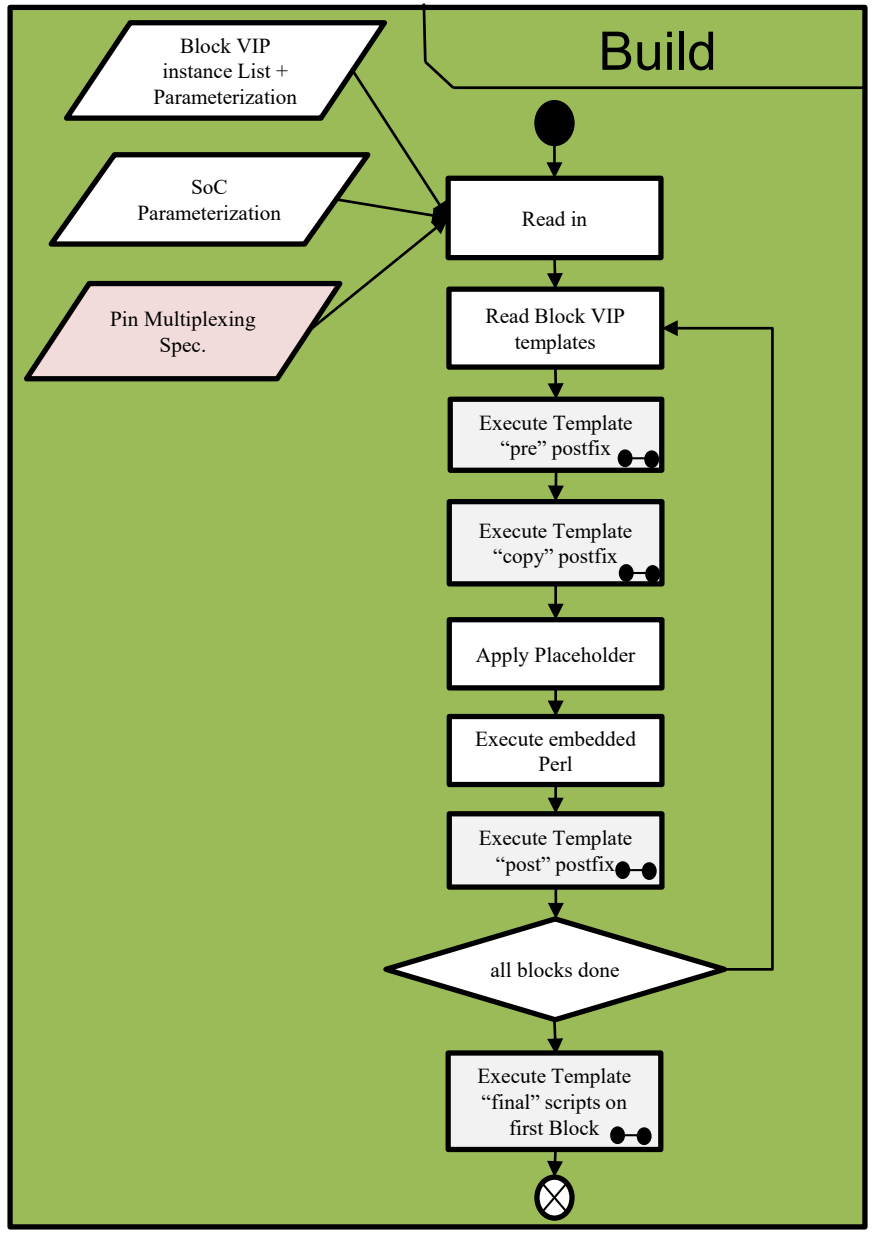
- Defines SoC parameters
- Provides pin definitions
- Provides VIP list
- Provides SoC testcases
- Provides SoC regression files
- Implements extended IPDSS directory structure

Agenda

- Introduction
- Role Models
- Function Details
- MagniV Implementation

Build Engine

- Implemented in Born Shell
- Small Engine Script (~400 lines)
- Common Functions (~200 lines)
- Uses extended IPDSS – based on `templates` and `bin` directory add-on
- Automates copy and replace work
  - simple replace
  - Perl template replace



Build Engine (cont.)

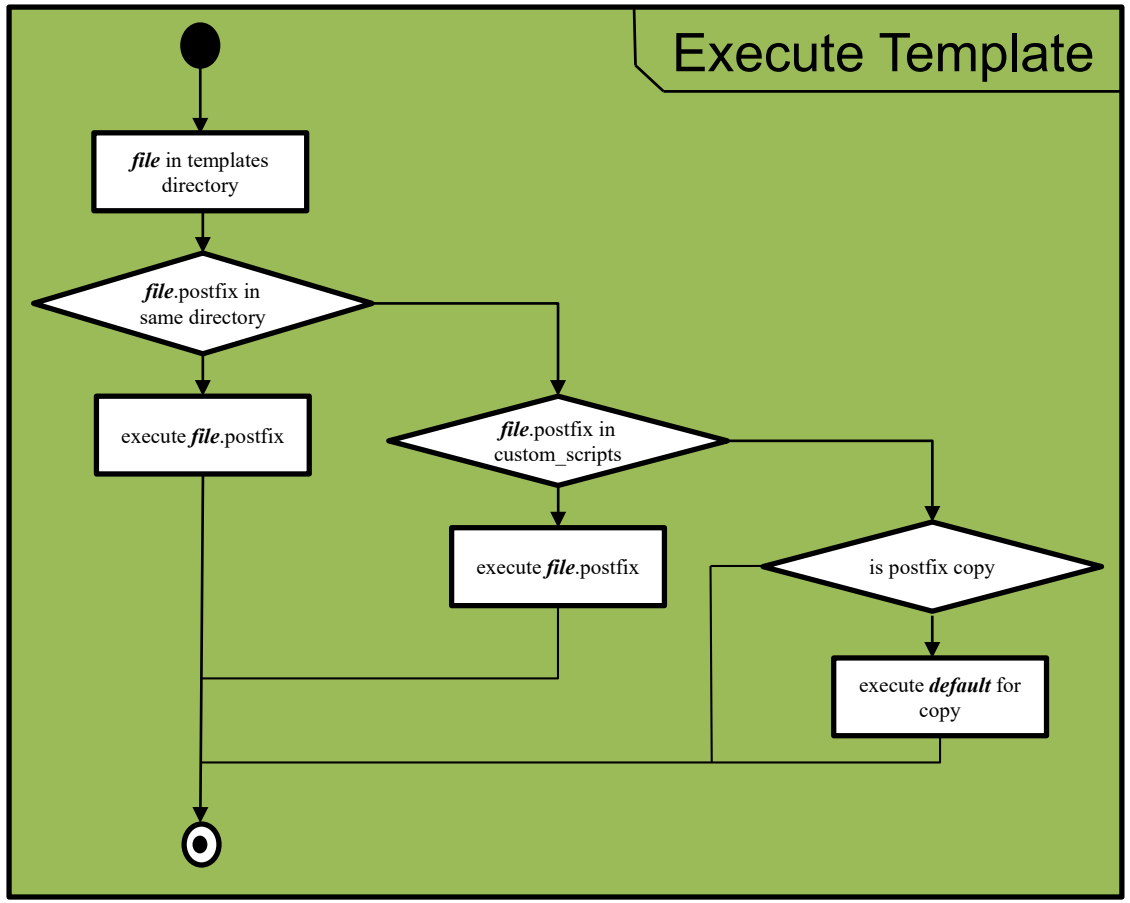
For a file *filename* in the template tree the following is looked for

Called first (if supplied)
filename.pre.sh

Called when supplied,
otherwise a copy/mkdir is called
filename.copy.sh

Called third (if supplied)
filename.post.sh

Called only for first block files
filename.final.sh



Manipulation files reside either local or in the `custom_scripts` directory

```
spi_vip/templates/testbench/top_packages_v/pin_multiplex_names_pkg.sv
spi_vip/templates/testbench/top_packages_v/pin_multiplex_names_pkg.sv.copy.sh
```

SoC View – Detailed Example

SoC View

Pin List

```
BCTL,|,,,,,,|,LQFP48,LQFP64,|,BCTL
PT0,|,SPI0_MOSI,SCI0_RXD,XIRQ,|,LQFP48,LQFP64,|,PT[0]
PT1,|,TIM0_IOC1,SPI0_MISO,SCI0_TXD,|,LQFP48,LQFP64,|,PT[1]
PP1,|,KWP1,PMF0_PWM5,,,|,LQFP48,LQFP64,|,PP[1]
VDD,|,,|,LQFP64,|,VDD
VSS1,|,,,,,|,LQFP48,LQFP64,|,VSS1
```

First Block

VIP List

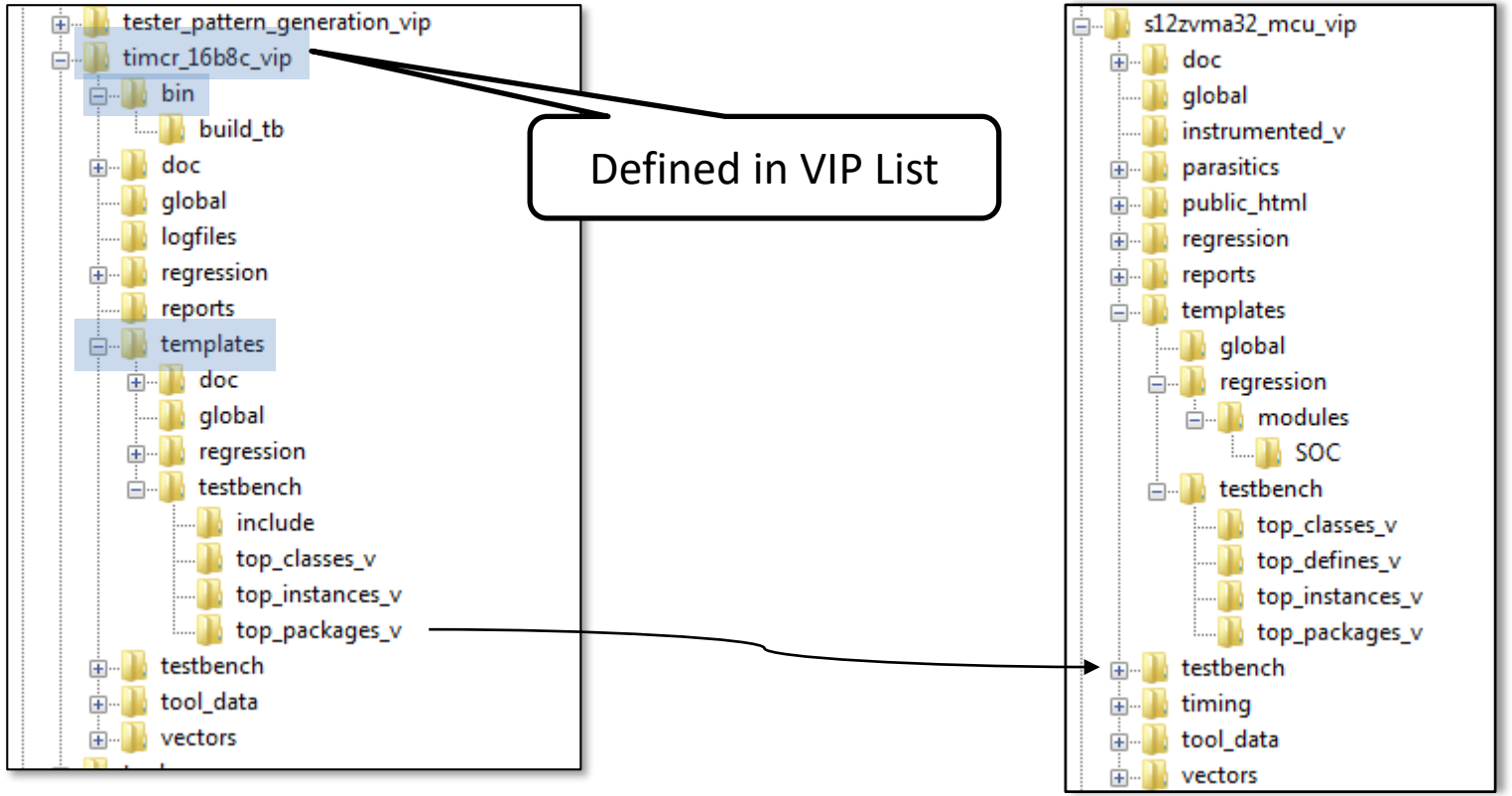
```
s12z_mcu_vip       ,s12zvma32_mcu_vip
s12zvma32_pim_vip  ,pim
s12zvma32_bist_vip ,bist
s12_cpmu_uhv_vip   ,cpmu
timcr16c8_vip      ,tim0,__BASE_ADDRESS__=32'h780
```

Parameter File

```
### NO WHITESPACES ALLOWED __..._=...
### General
__PART__=s12zvma32
__SOC_NAME__=S12ZVMA32
```

Copy - File Structure Definition

Content of
`timcr_16b8c_vip/templates/testbench/top_packages_v/pin_multiplex_names_pkg.sv`
Gets applied to
`s12zvma32_mcu_vip/testbench/top_packages_v/pin_multiplex_names_pkg.sv`



Replace - Simple Parameter

SoC View

Pin List

```
BCTL,|,,,,,,|,LQFP48,LQFP64,|,BCT
PT0,|,SPI0_MOSI,SCI0_RXD,XIRQ,|,I
PT1,|,TIM0_IOC1,SPI0_MISO,SCI0_TX
PP1,|,KWP1,PMF0_PWM5,,,|,LQFP48,I
VDD,|,,|,LQFP64,|,VDD
VSS1,|,,,,,|,LQFP48,LQFP64,|,VSS1
```

VIP List

```
s12z_mcu_vip        ,s12zvma32_mcu_vip
s12zvma32_pim_vip   ,pim
timcr16c8_vip       ,tim0,__BASE_ADDRESS__=32'h780
```

Parameter

```
### NO WHITESPACES ALLOWED __..._=.
### General
__PART__=s12zvma32
__SOC_NAME__=S12ZVMA32
```

VIP View

TopStimManager.sub_stim_manager_new.sv

```
begin : __INSTANCE__
    __instance__StimManager = new (
        .parentStimulusManager (this),
        .name          ("__INSTANCE__"),
…
        .base_addr     (__BASE_ADDRESS__));
end : __INSTANCE__
```

```
begin : TIM0
    tim0StimManager = new (
        .parentStimulusManager (this),
        .name          ("tim0"),
…
        .base_addr     (32'h780));
end : TIM0
```

Replace - Perl Template Support

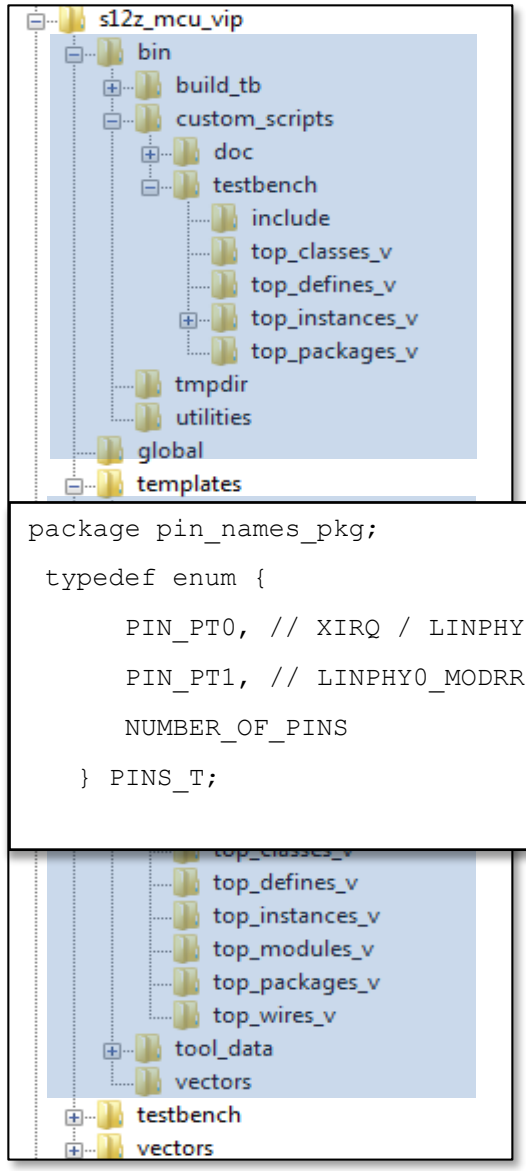
Method View

top_packages_v/pin_names_pkg.sv

```
package pin_names_pkg;

   typedef enum {

      __[ create_pin_list (@pins); ]__

      NUMBER_OF_PINS

   } PINS_T;

            ;
```

build_tb/build_tb.pl

```
sub create_pin_list {
   my @pins = @_;
   my $code = "\n";
   $code .= "// START: create_pin_list \n";
   foreach my $pin (@pins) {
      $code .= "      PIN_$pin->{pin}, // ";
      foreach my $func (@{$pin->{"func"}}) {
         $code .= "$func / ";
      }
      $code .= "\n";
   }
   $code .= "// END: create_pin_list \n";
   return $code;
}
```




```
package pin_names_pkg;
 typedef enum {
      PIN_PT0, // XIRQ / LINPHY0_MODRR0_06_LP0RXD
      PIN_PT1, // LINPHY0_MODRR0_06_LP0TXD
      NUMBER_OF_PINS
   } PINS_T;
```

Perl API

- Parameter passed
- Placeholder name
- Placeholder value

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Method View Definition Example

Method View

TopStimManager.sub_stim_manager_new.sv.copy.sh

```
#! /bin/sh
argv=$*
PROGRAM=`basename $0` # used in all included modules
TEMPLATE_DIR=$1
  . $TEMPLATE_DIR/mk_common.sh

cat_file ${relative_source}
apply_placeholder ${soc_blocks}/${block_name}/${relative_source}
```

TopStimManager.sv.final.sh

```
#! /bin/sh
# Assemble the TopStimulusManager fragments into one
argv=$*
PROGRAM=`basename $0` # used in all included modules
TEMPLATE_DIR=$1
  . $TEMPLATE_DIR/mk_common.sh
…
  merge_file TopStimManager.parameter.sv
  merge_file TopStimManager.sub_stim_manager.sv
  merge_file TopStimManager.new.sv
  merge_file TopStimManager.sub_stim_manager_new.sv
…
```

VIP View

TopStimManager.sub_stim_manager_new.sv

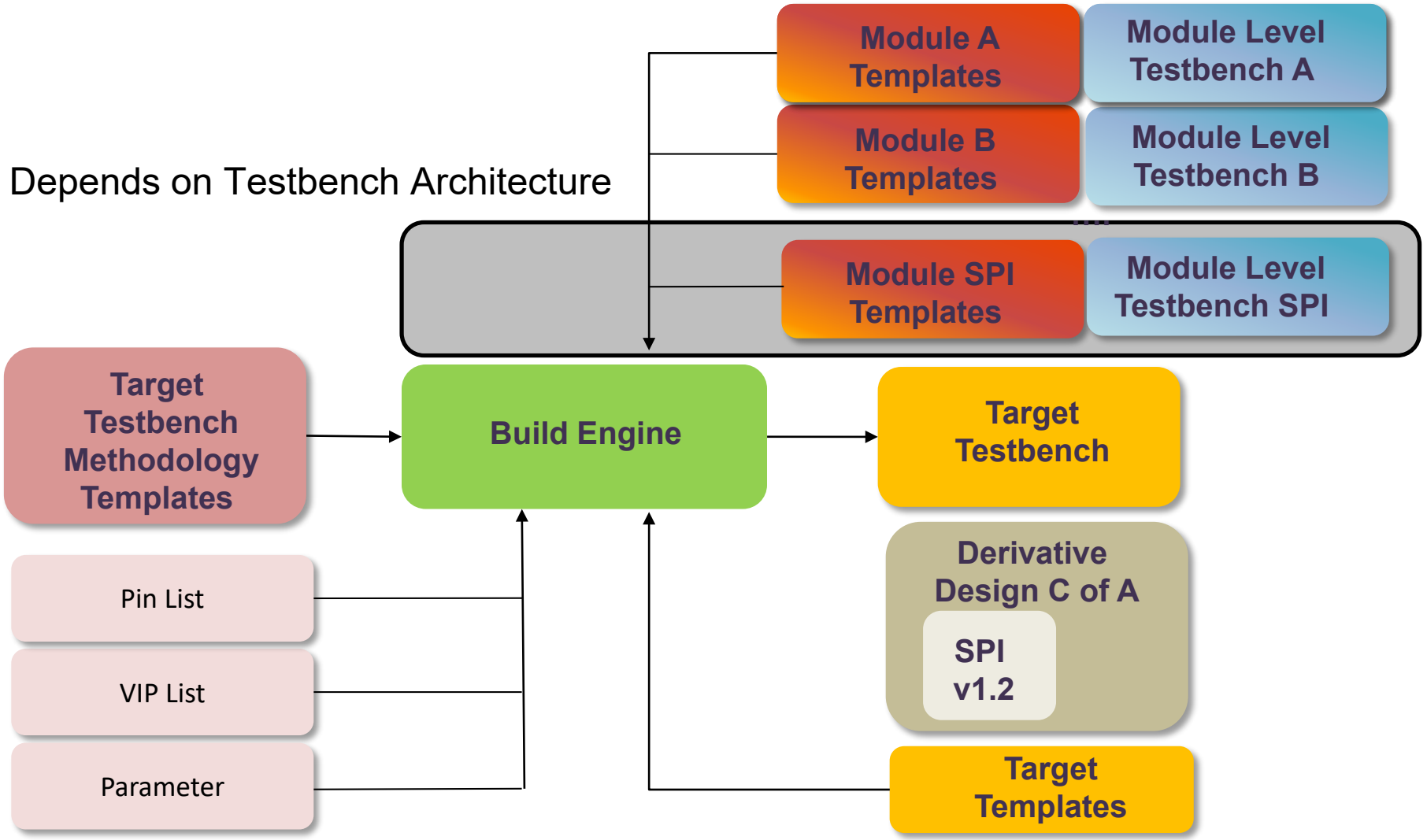
```
begin : __INSTANCE__
   __instance__StimManager = new (
      .parentStimulusManager (this),
      .name          ("__INSTANCE__"),
…
```

```
class TopStimManager #(/*! START: s12z_mcu_vip */
…
 function new (StimManagerBase parentStimulusManager,
…
 begin : SPI0
   spi0StimManager = new (
      .parentStimulusManager (this),
      .name("SPI0"),
…
```

Agenda

- Introduction
- Role Models
- Function Details
- MagniV Implementation

IP View S12Z MagniV Example



IP View S12Z MagniV Example

- Integration stimulus is written at module level
- Tester pattern can be written at module level

- Requirements
  - API definition for all functions to allow reuse on SoC

- Benefits for module level development
  - Independent of SoC design and testbench
  - Fast compile and simulation time
  - Small environment eases debugging
  - Share module and integration testbench infrastructure

IP View S12Z MagniV Example - APIs

Module to SoC reuse concepts

- Interrupt API
- Stimulus Manager
- Register Flash Preloading
- Running System Verilog on the Embedded Core - Core Slave Mode – SNUG 2004
- System Verilog and CAPI Register Access API – DATE 2002, SNUG 2005
- Reusable DMA Stimulus API – SNUG 2006
- Reusable Clock API – SNUG 2010
- Reusable Low Power Stimulus API - SNUG 2011
- Control Loop API – SNUG 2014

Most of the concepts got implemented in different languages and base classes over time

Summary

Status

- 3 SoC testbenches were build 100% automatic
- 28 block VIPs are available with templates
- SoC, IPBI and PIM target methodology templates are available

Pro's

- Testbench setup can be 100% reproduced
- Module design parameters can be applied to all created files
- Testbench setup for an experienced testbench user is reduced from 5 to 1 day
- Verification input is shifted to Design Specification Level

Con's

- Additional effort on the module level to provide the templates

Conclusion

- The testbench automation reduces verification effort and automates handover from module to SoC level.

Questions