Reuse of Simulink Components within Chip-Level Design and Verification Environments

Simone Saracino
simone.saracino@st.com

Diego Alagna
diego.alagna@st.com

Agenda

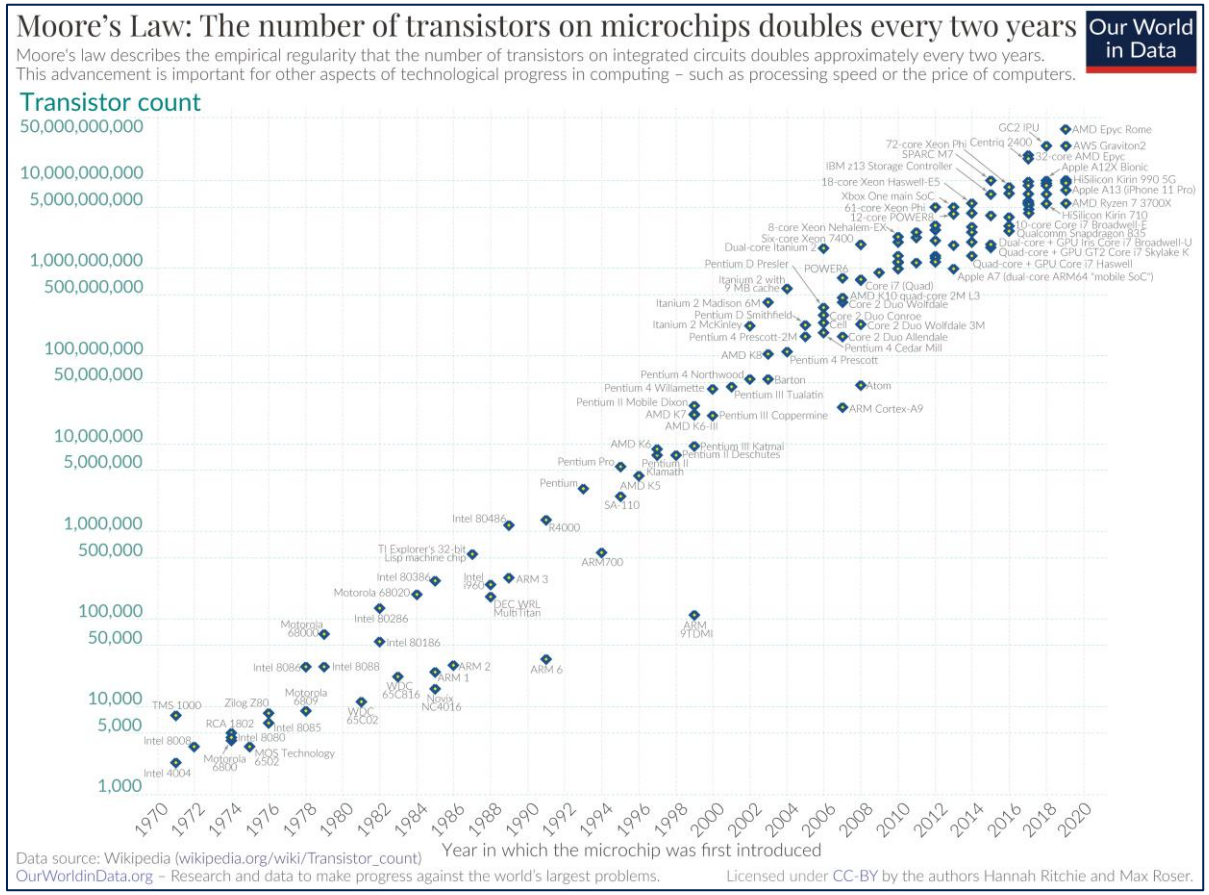
1. Introduction
2. System-level design model
3. System-level verification
4. Reuse of system-level verification IPs within the UVM testbench
5. Reuse of system-level design model
6. Results and future enhancements
7. Conclusions

life.augmented

Introduction

Introduction

- The number of transistors doubles every two years
- Moore’s law slowing down over the last decade
- Market requirements more and more aggressive
  - Performance, cost, time-to-market
- Architectural optimizations are key to drive IC enhancements



life.augmented

ST Automotive and Discrete Group - Smart Power R&D

Serving all automotive applications

Traditional Automotive
ICE powertrain, Chassis & Safety, Body, Infotainment

New Trends
Hybrid & battery electric vehicles, ADAS and new E/E architectures

New market perspective of Smart Power

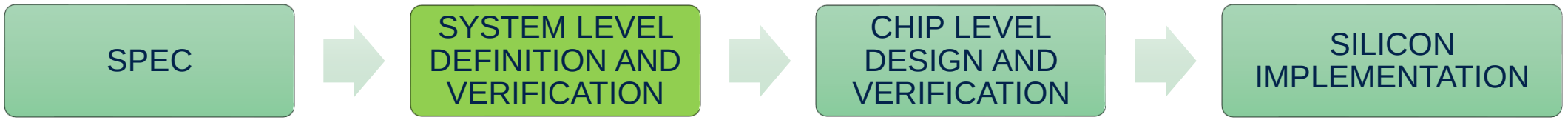
From traditional to electrification	Targeting new E/E architecture and vehicle digitalization
Isolated gate driver	Power management IC
Battery management system	DC-DC converter
Door-Zone & Multi-output HS/LS driver	System Basis Chip
Engine management system	Voltage regulator
Alternator regulator	Motor control

life.augmented

Architectural Optimizations

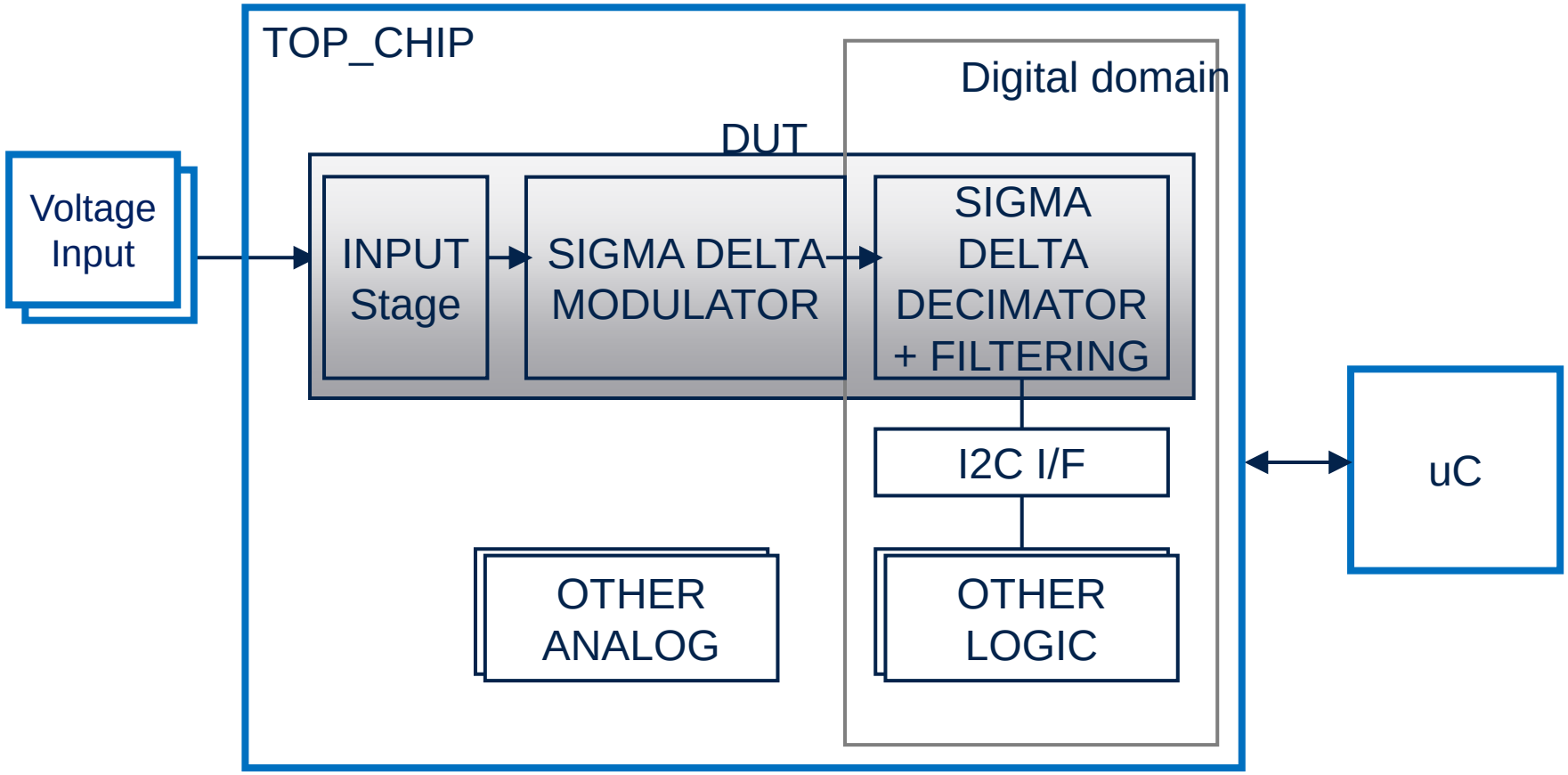
- Enhance system-level architectural exploration
  - Simulation
  - Analysis
  - Verification (also reduces bug lead time!)

- Reuse system-level collaterals within the rest of the design flow
  - Avoids overall additional effort



System-level design model

System-level design model



System-level design model

- Device Under Test (DUT, Main Measurement Unit) is modeled and verified using Simulink platform

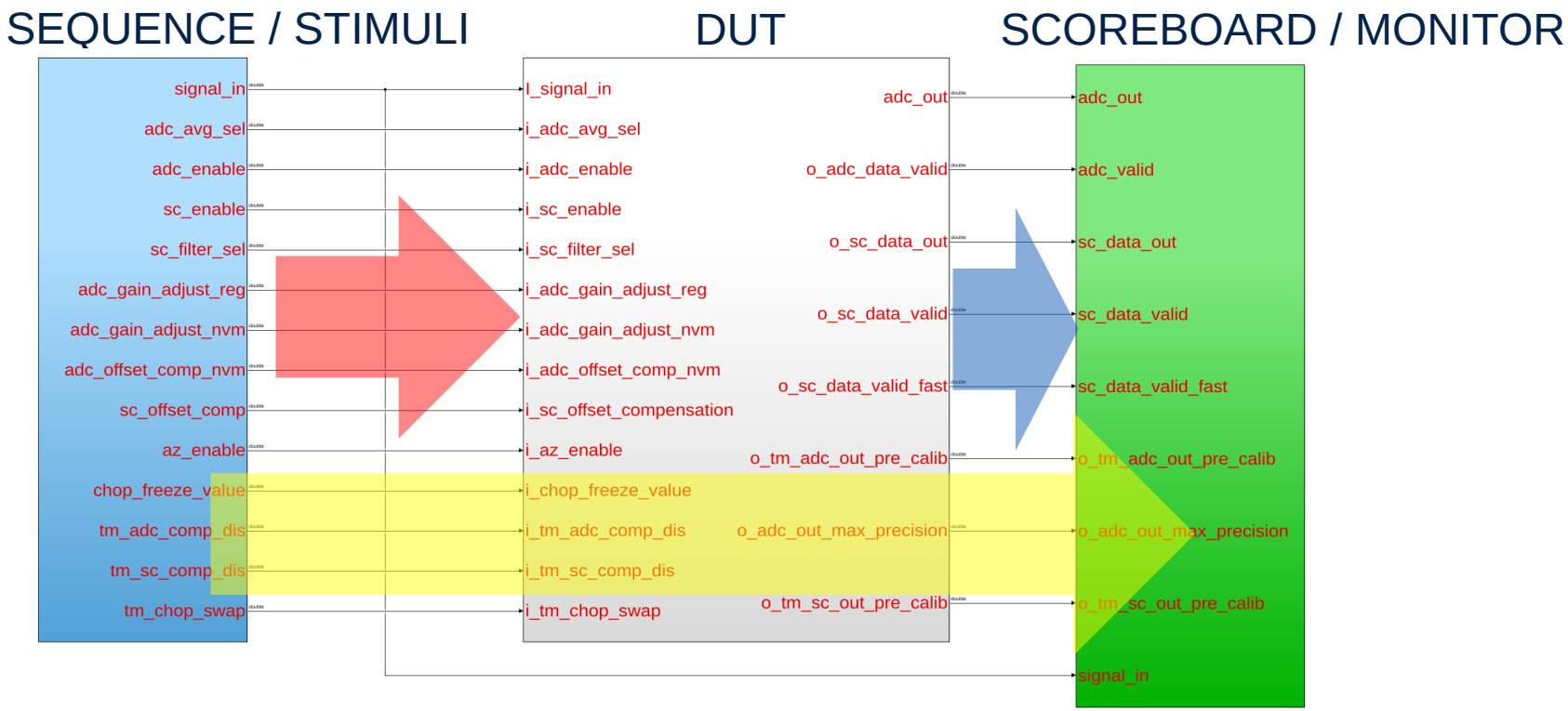
- Model includes both analog and digital components
  - Voltage input is modeled as ideal voltage generator
  - The analog blocks are modeled using real numbers in discrete-time domain with high sample rate compared to digital clock
  - Digital blocks are modeled in a fixed-point, bit-accurate and cycle-accurate way

- The rest of the IC components are not included in the system-level Simulink model
  - I2C interface is implemented with ST standard circuits

System-level verification

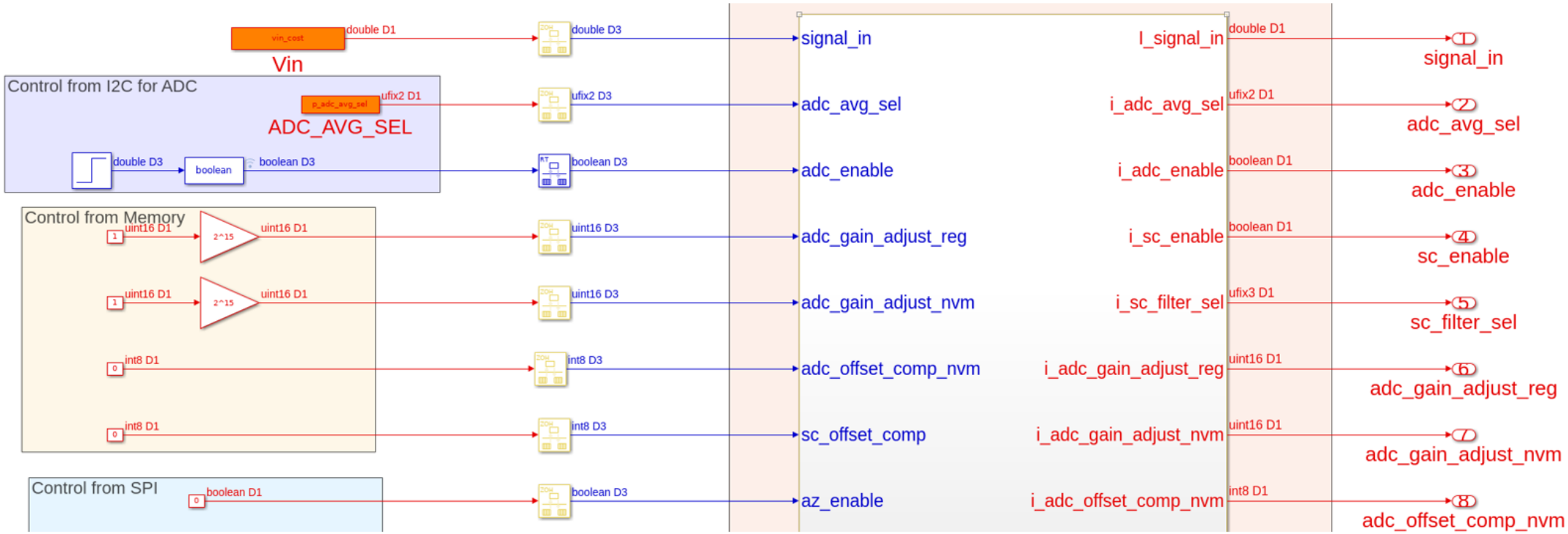
System-level testbench

- Allows DUT verification before its corresponding RTL code and SPICE netlist are available



life.augmented

Stimuli block



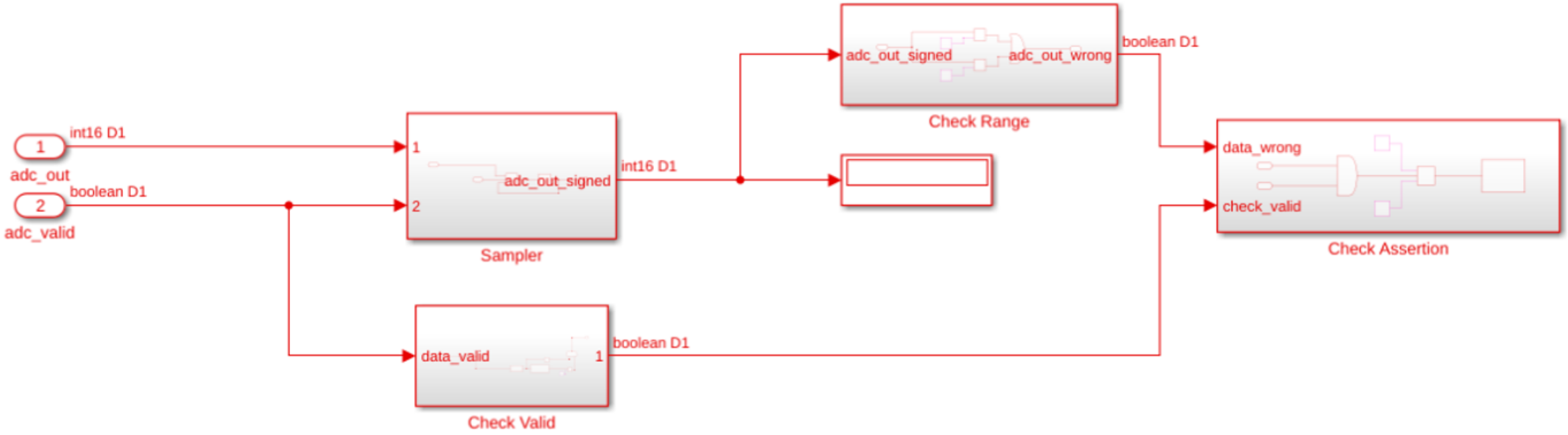
life.augmented

System-level stimuli

- Stimuli block drives the following DUT inputs
  - Analog voltage input (real number)
  - Configuration parameters (**«control from I2C for ADC»** box in the fig)
  - Digital correction for ADC errors (**«control from memory»** box in the fig)
  - Debug configuration (**«control from SPI»** box in the fig)
- DUT inputs are constants except for the Vin and the ADC's enable signal
- Parametrized stimuli are not randomized (future enh.)

Scoreboard

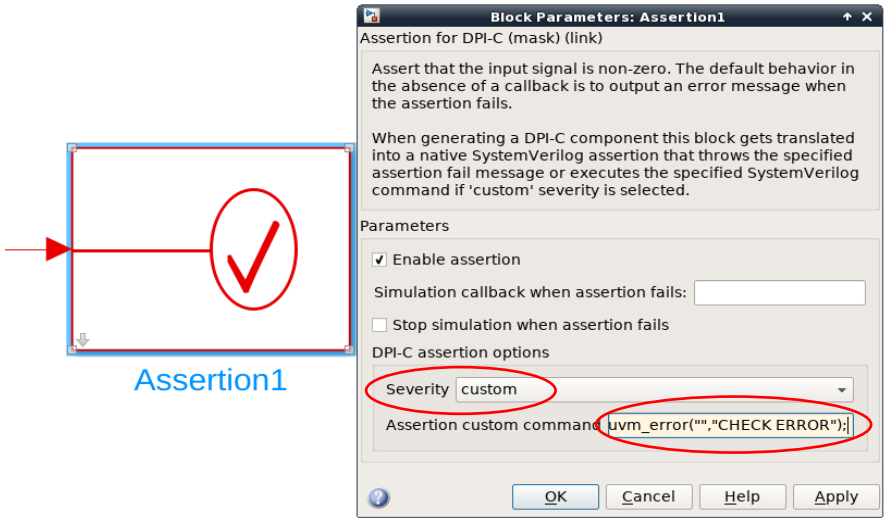
- Scoreboard is a verification component that contains checkers and verifies the functionality of a design
- Functional checks in the system-level verification



life.augmented

Assertion

- UVM assertion is a check embedded in verification, that validates the behavior of design during simulation
- Use **Assertion** Simulink component to:
  - Assert a simulation error during system-level verification
  - Insert an uvm_error in chip-level environment

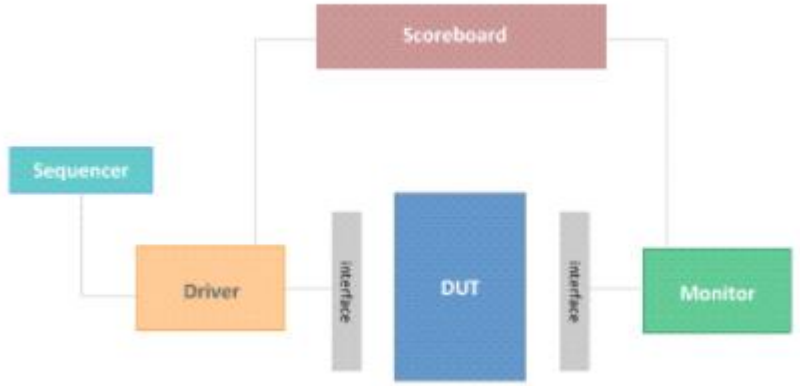


life.augmented

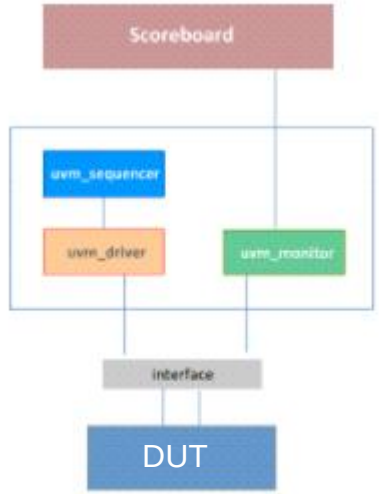
Reuse of system-level verification IPs within the UVM testbench

What is UVM?

- UVM (Universal Verification Methodology) is a standard methodology for IC verification
- Advantages:
  - Modular verification IP
  - Reusable verification environments
  - Scalable testbench structures
- Reference: https://www.chipverify.com/uvm/uvm-tutorial



Testbench based on classic SV approach

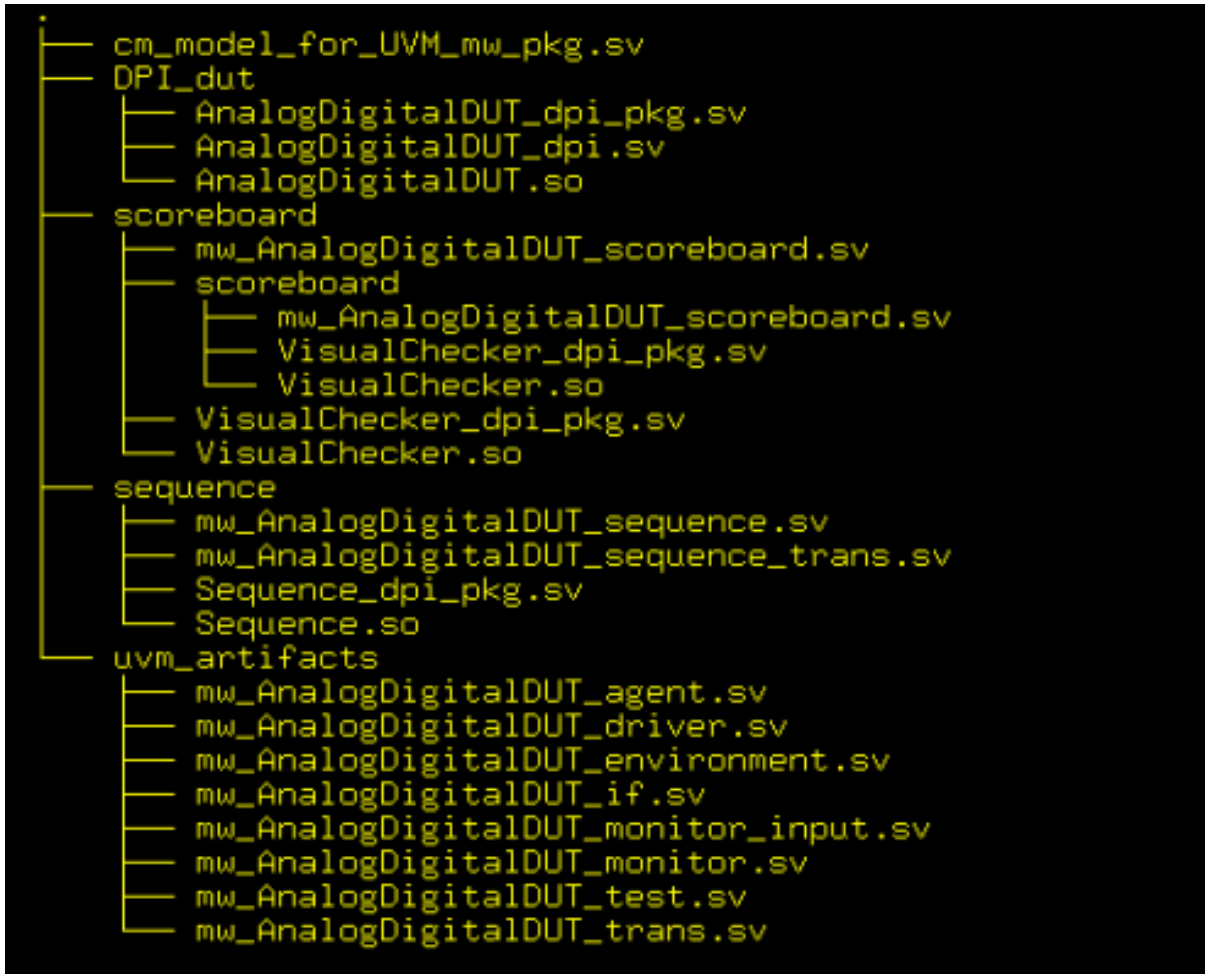


Testbench based on UVM approach

Reuse of system-level verification IPs

- The system-level verification test cases need to be re-run at chip level
- System-level testbench reused using Simulink UVM generation
  - HDL Verifier (MathWorks tool)
  - Avoid duplication of effort
- Simulink DUT doesn’t correspond to full IC: adaptations needed

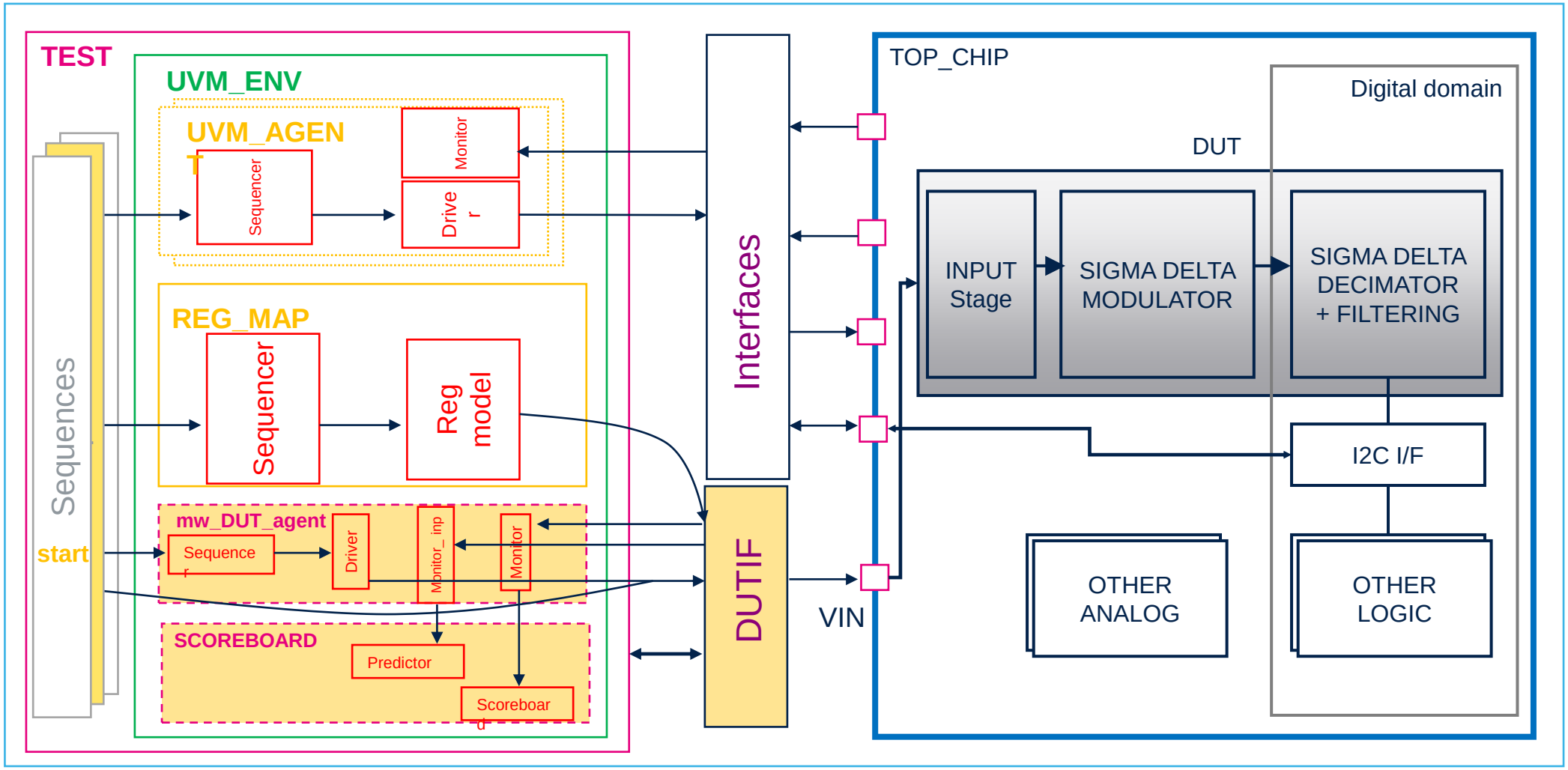
Generated UVM files



UVM testbench generation

- Generated UVM testbench is a stand-alone verification environment
  - In case DUT corresponds to full IC, it is possible to use it without modifications
- Simulink generates also UVM behavioral model for DUT
  - Ignored in our chip-level verification environment
  - Used RTL generation for digital portions, Verilog models for analog ones
- Each testcase corresponds to a single sequence
  - To generate a single sequence, it is necessary to generate the full UVM testbench environment
  - To have a good verification coverage, it needs a lots of sequences

Digital chip-level testbench environment



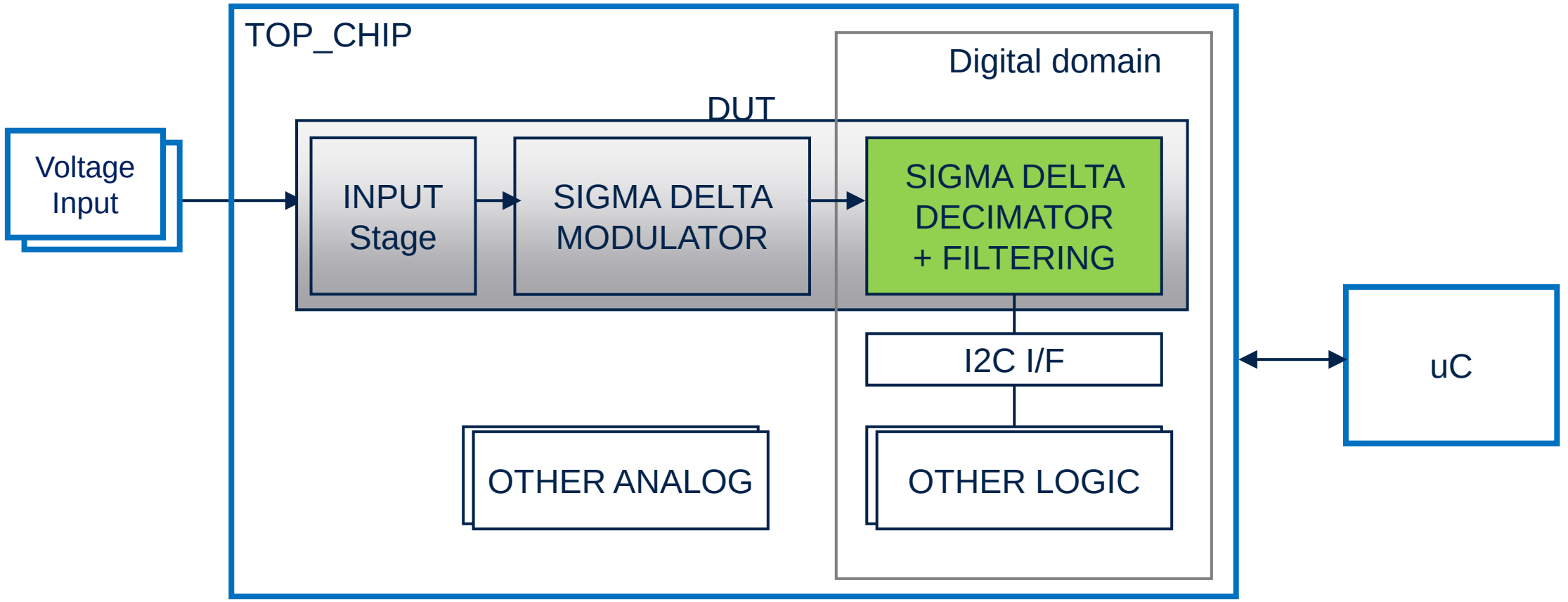
life.augmented

Integration of generated UVM components & objects

- Some manual modifications are needed to integrate the UVM components generated from Simulink within the chip-level UVM testbench:
  - Adapt files generated to UVM environment
  - Insert an interface decoder to sniff the output of DUTIF to generate existing high level transactions
  - Trigger insertion to synchronize the start of generated sequence with the starting time at system level

Reuse of system-level design model

System-level design model



RTL code generation

- RTL (Register-Transfer Level) is a design abstraction which models a synchronous digital circuit.
  - A synthesis tool is able to generate the real schematic starting from RTL
- Reuse also for digital part system-level design model (green block in the figure)
- Simulink add-on “HDL Coder” generates RTL code
- Both Simulink testbench components and design models are reused at chip-level

```
ENTITY adcv_corr IS
  PORT( i_clk                             :   IN    std_logic;
        i_reset_n                         :   IN    std_logic;
        i_adjust                          :   IN    std_logic_vector(15 DOWNTO 0);  -- uint16
        i_adc                             :   IN    std_logic_vector(19 DOWNTO 0);  -- sfix20
        i_comp_dis                        :   IN    std_logic;
        o_adc_cal                         :   OUT   std_logic_vector(19 DOWNTO 0)  -- sfix20
        );
END adcv_corr;
```

```
ARCHITECTURE rtl OF adcv_corr IS

  -- Signals
  SIGNAL i_adc_signed                     : signed(19 DOWNTO 0);  -- sfix20
  SIGNAL i_adjust_unsigned                : unsigned(15 DOWNTO 0);  -- uint16
  SIGNAL Product1_cast                    : signed(16 DOWNTO 0);  -- sfix17
  SIGNAL Product1_mul_temp                : signed(36 DOWNTO 0);  -- sfix37
  SIGNAL Product1_out1                    : signed(35 DOWNTO 0);  -- sfix36
  SIGNAL Extract_Bits1_out1               : signed(34 DOWNTO 0);  -- sfix35
  SIGNAL Gain1_cast                       : signed(35 DOWNTO 0);  -- sfix36_En15
  SIGNAL Gain1_out1                       : signed(19 DOWNTO 0);  -- sfix20
  SIGNAL Switch8_out1                     : signed(19 DOWNTO 0);  -- sfix20
  SIGNAL Unit_Delay6_out1                 : signed(19 DOWNTO 0);  -- sfix20

BEGIN
  -- Rescale down to the original fixed point
  -- and truncate

  i_adc_signed <= signed(i_adc);

  i_adjust_unsigned <= unsigned(i_adjust);

  Product1_cast <= signed(resize(i_adjust_unsigned, 17));
  Product1_mul_temp <= i_adc_signed * Product1_cast;
  Product1_out1 <= Product1_mul_temp(35 DOWNTO 0);

  Extract_Bits1_out1 <= signed(Product1_out1(34 DOWNTO 0));

  Gain1_cast <= resize(Extract_Bits1_out1, 36);
  Gain1_out1 <= Gain1_cast(34 DOWNTO 15);


  Switch8_out1 <= Gain1_out1 WHEN i_comp_dis = '0' ELSE
      i_adc_signed;

  Unit_Delay6_process : PROCESS (i_clk, i_reset_n)
  BEGIN
    IF i_reset_n = '0' THEN
      Unit_Delay6_out1 <= to_signed(16#00000#, 20);
```

RTL code generation - Advantages

- Avoid human errors in converting Simulink models to RTL
- Ensure alignment between Simulink and RTL simulation
- Bug fix at system-level model
- Improvement workflow efficiency

Results and future enhancements

Results

- RTL verification time cut in half
- Avoided duplication of effort between system-level and RTL-level verification
- Saved RTL coding time, reduced chance of human errors within RTL
- Single source of test cases and design models

Future Enhancements

- Simplify integration of generated UVM components within chip-level environment
- Test case management, options
  - Multiple Simulink testbenches
  - Simulink parametrization, different scenarios from the same testbench
- Randomization, options
  - At Simulink level
  - Introduced after UVM generation

Conclusions

Conclusions

- Shift-left untimed and loosely-timed portions of verification
  - Moving them to system-level
- Avoid duplication of effort
  - Reuse system-level verification IPs within RTL environment
  - Reuse system-level design models, converting them to RTL
- Early detect bugs, such as
  - Functional and performance issues in algorithms, state machines, etc.
  - Poor interactions between domains

Our technology starts with You

Find out more at www.st.com

life.augmented