High-Performance
Semiconductors
» Magnetic Sensor Integrated Circuits
» Power Integrated Circuits

Automotive ASIC Model Based Design

Jamie Haas
Director of Design Engineering
Advanced Sensor Technology Business Unit

Sensor Applications

- **Automotive**
  - **Steering**
  - **Engine**
  - **Transmission**
  - **Hybrid/Electric Motor**
  - **Seat Belt**
  - **More!**
- **Consumer Applications**
- **Industrial Controls**
- **Current Sensing**

An Overview of Allegro's MBD Success

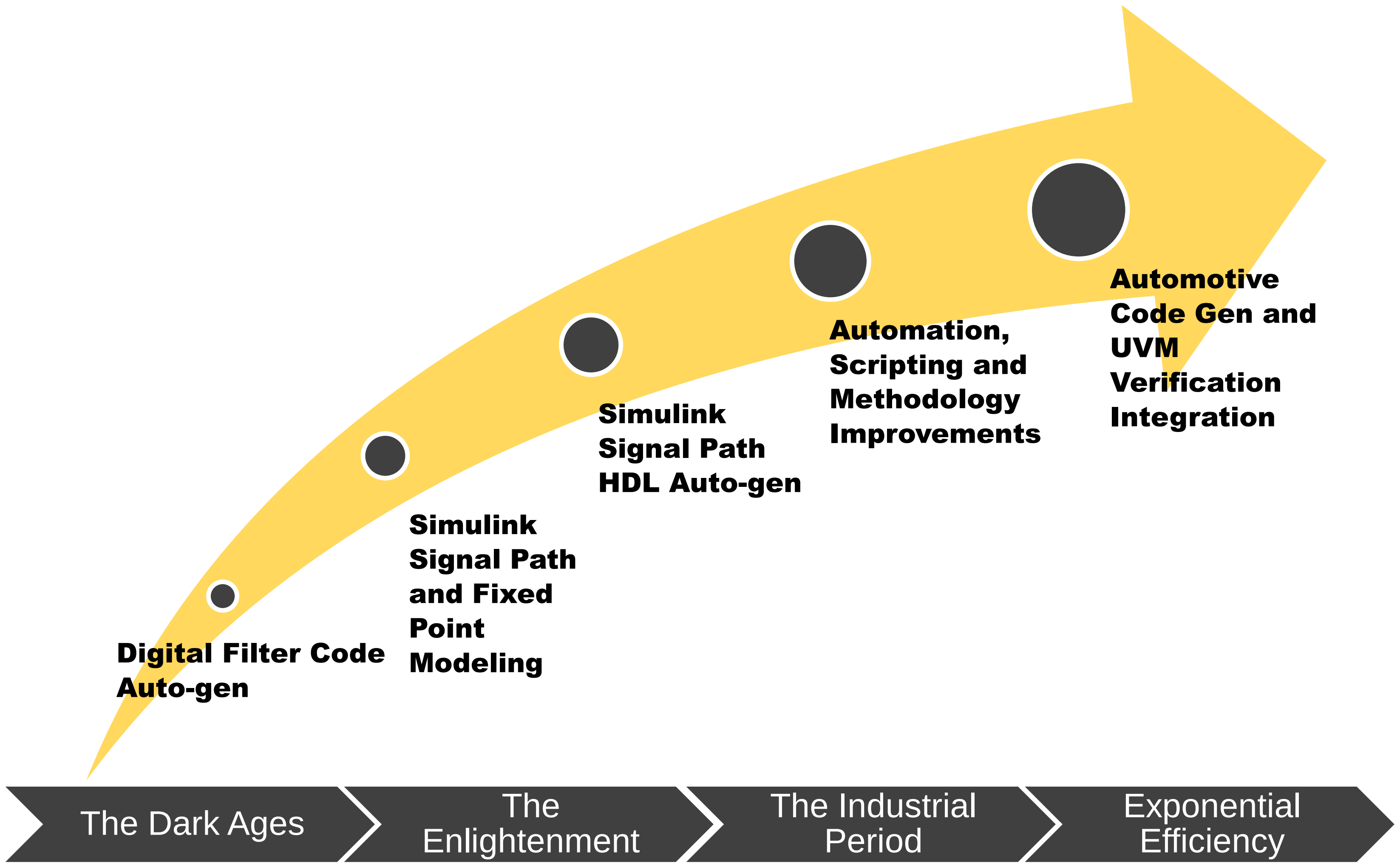
- ❑ **We Started taking a serious look at Simulink MBD for ASIC Development in 2014**

- ❑ **Team Adoption hit the "knee" of the curve in 2015**

- ❑ **A total of 8 device digital control systems and signal paths auto generated from Simulink**
  - ❑ Interpolation engines
  - ❑ Digital filters
  - ❑ Signal Processing Algorithms
  - ❑ Digital PLL's
  - ❑ Digital Sigma Delta DAC's

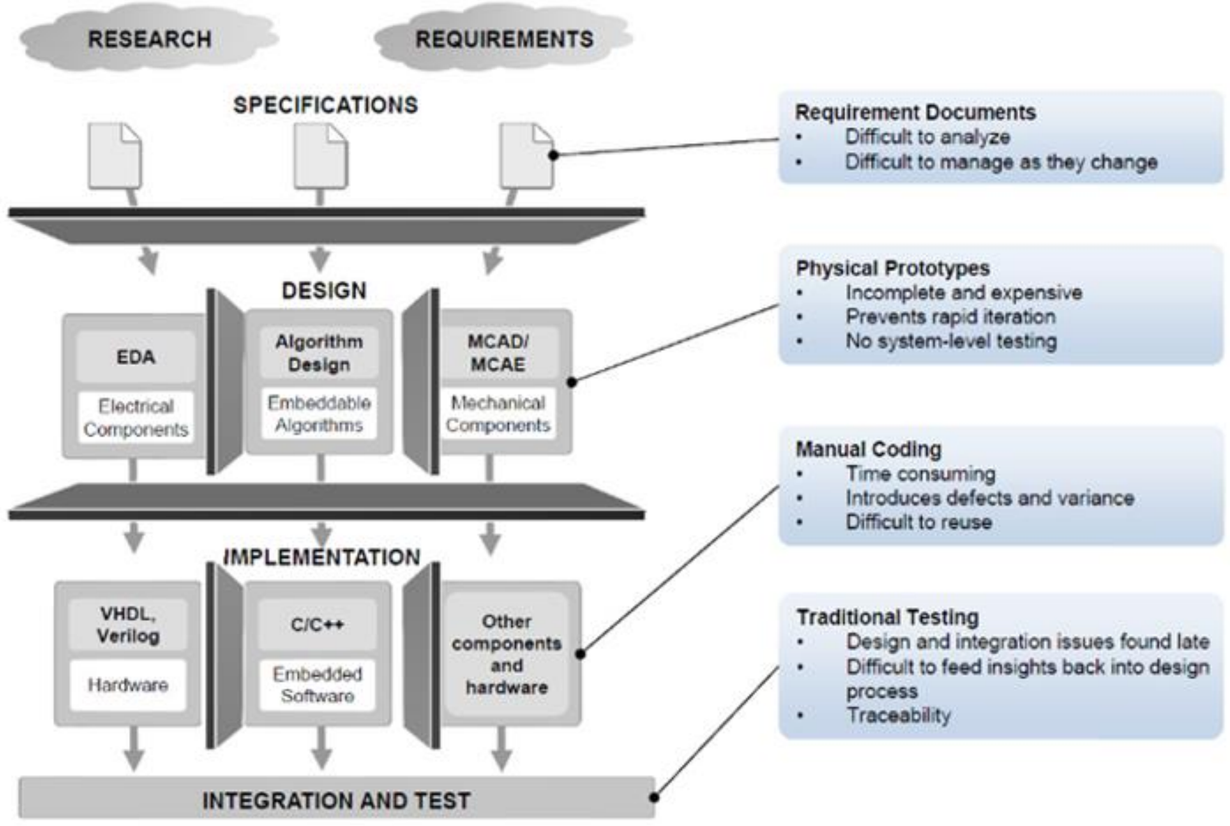
- ❑ **Silicon Evaluation has demonstrated ZERO bugs from auto generated code to date.**

The Evolution of Allegro's Model Based Design (MBD) Flow


Digital Filter Code Auto-gen
Simulink Signal Path and Fixed Point Modeling
Simulink Signal Path HDL Auto-gen
Automation, Scripting and Methodology Improvements
Automotive Code Gen and UVM Verification Integration
The Dark Ages
The Enlightenment
The Industrial Period
Exponential Efficiency

"The Dark Ages"
Traditional ASIC Design

I just found out my algorithm is wrong and we are releasing in 2 weeks!!



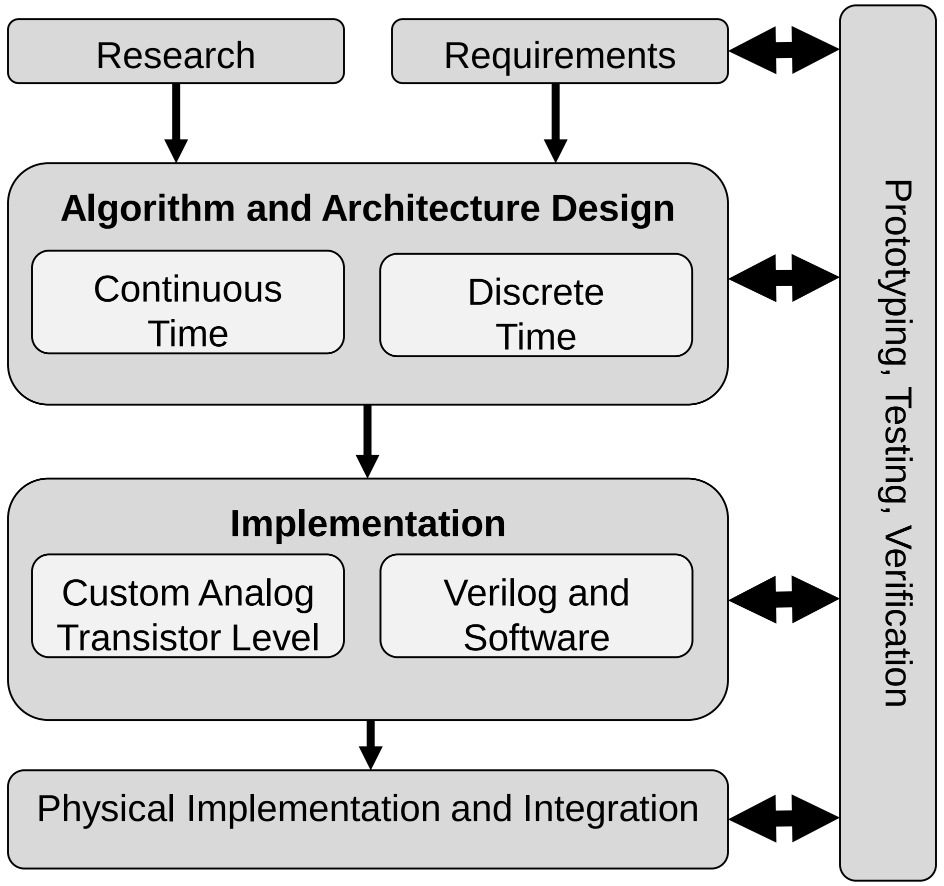
Requirement Documents
- Difficult to analyze
- Difficult to manage as they change

Physical Prototypes
- Incomplete and expensive
- Prevents rapid iteration
- No system-level testing

Manual Coding
- Time consuming
- Introduces defects and variance
- Difficult to reuse

Traditional Testing
- Design and integration issues found late
- Difficult to feed insights back into design process
- Traceability

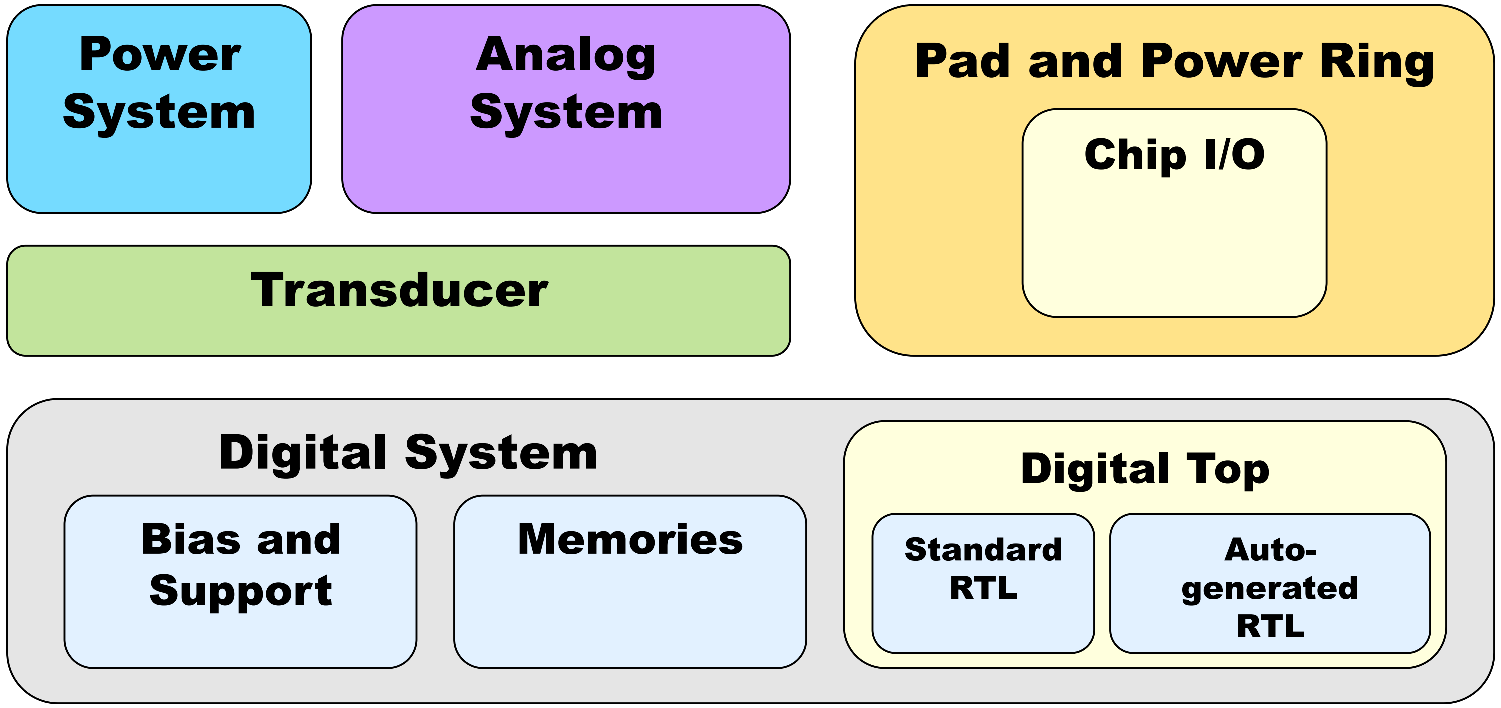
The Enlightenment: Model Based Design


Research
Requirements
Algorithm and Architecture Design
Continuous Time
Discrete Time
Implementation
Custom Analog Transistor Level
Verilog and Software
Physical Implementation and Integration
Prototyping, Testing, Verification

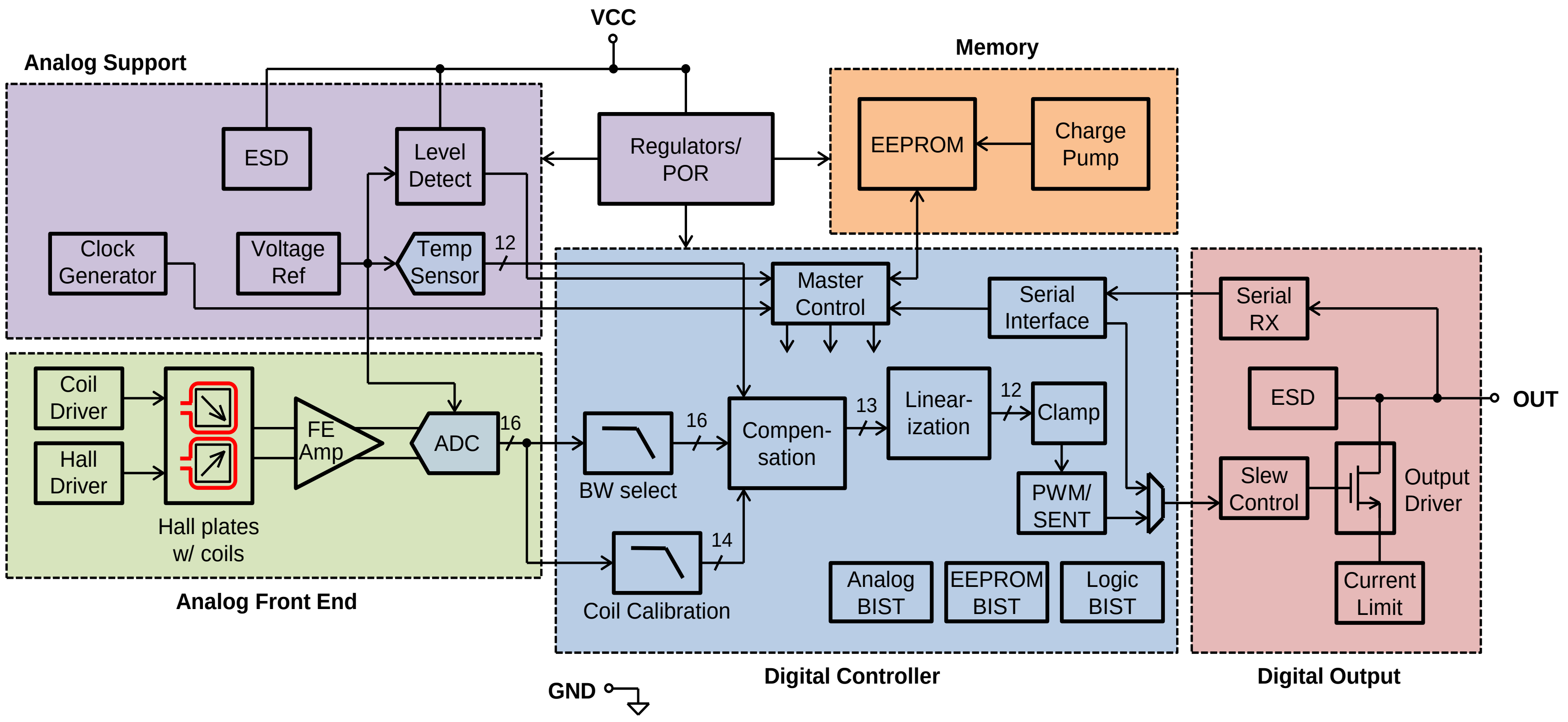
- **Architecture and Algorithm Design Evolve into Executable Specifications**
- **Front load testing and verification**
- **Development is “parallelized”**
- **Continuous Equivalency Testing is utilized**
- **.... And of course auto-generated production code**

ASIC Sensor Components

Mixed Signal ASIC Components



Precision Automotive Magnetic Sensor

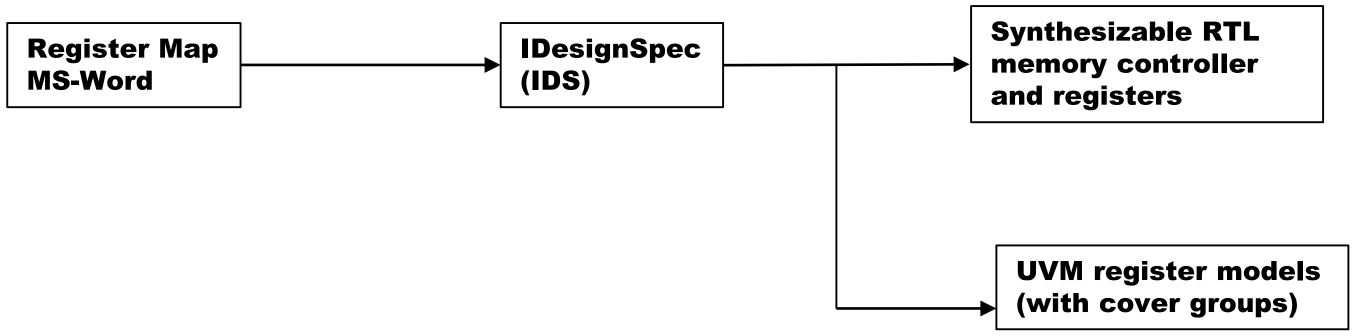


ASIC Sensor Simulink Modeling

- ❑ How do we handle the memory (EEprom configuration)

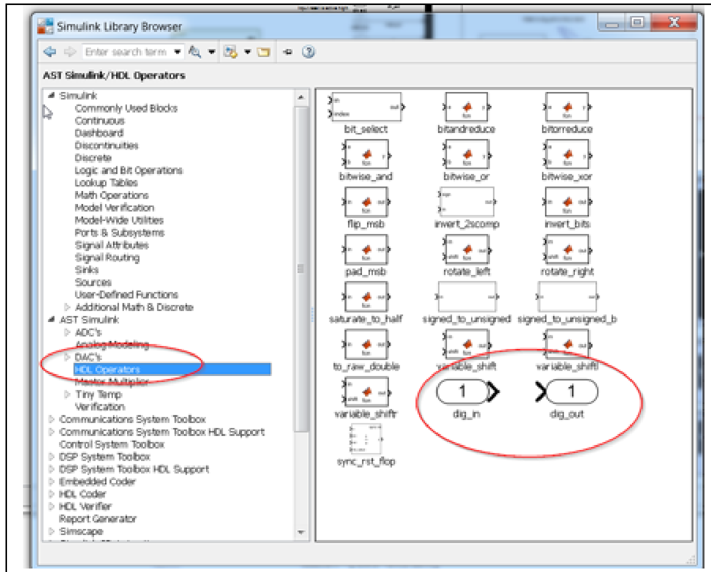
- ❑ Simulink Model Types
  - ❑ Spec Model
  - ❑ HDL Gen: Stateflow
  - ❑ HDL Gen: Matlab Function
  - ❑ Validation Model

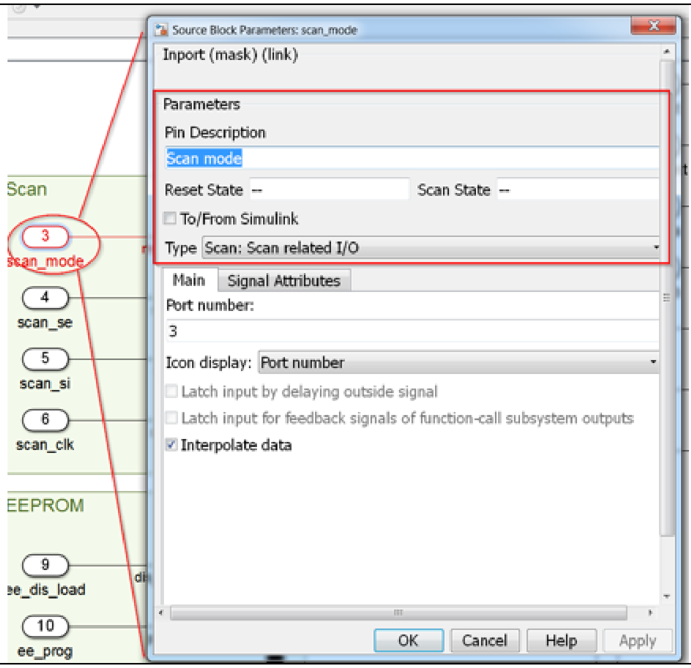
Memories: Automatic Register Generation


Register Map MS-Word
IDesignSpec (IDS)
Synthesizable RTL memory controller and registers
UVM register models (with cover groups)

- **The Word based register map is the single point for documenting all registers**
- **RTL is generated using the Agnisys IDesignSpec tool**
- **Register models required by DV (Design Verification) are generated using the Agnisys IDesignSpec tool**

Memories:
Link Simulink Model to Memory Map





Inport (mask) (link)

Parameters

Pin Description

Scan mode

Reset State -- Scan State --

☐ To/From Simulink

Type Scan: Scan related I/O

Main | Signal Attributes

Port number:

3

Icon display: Port number

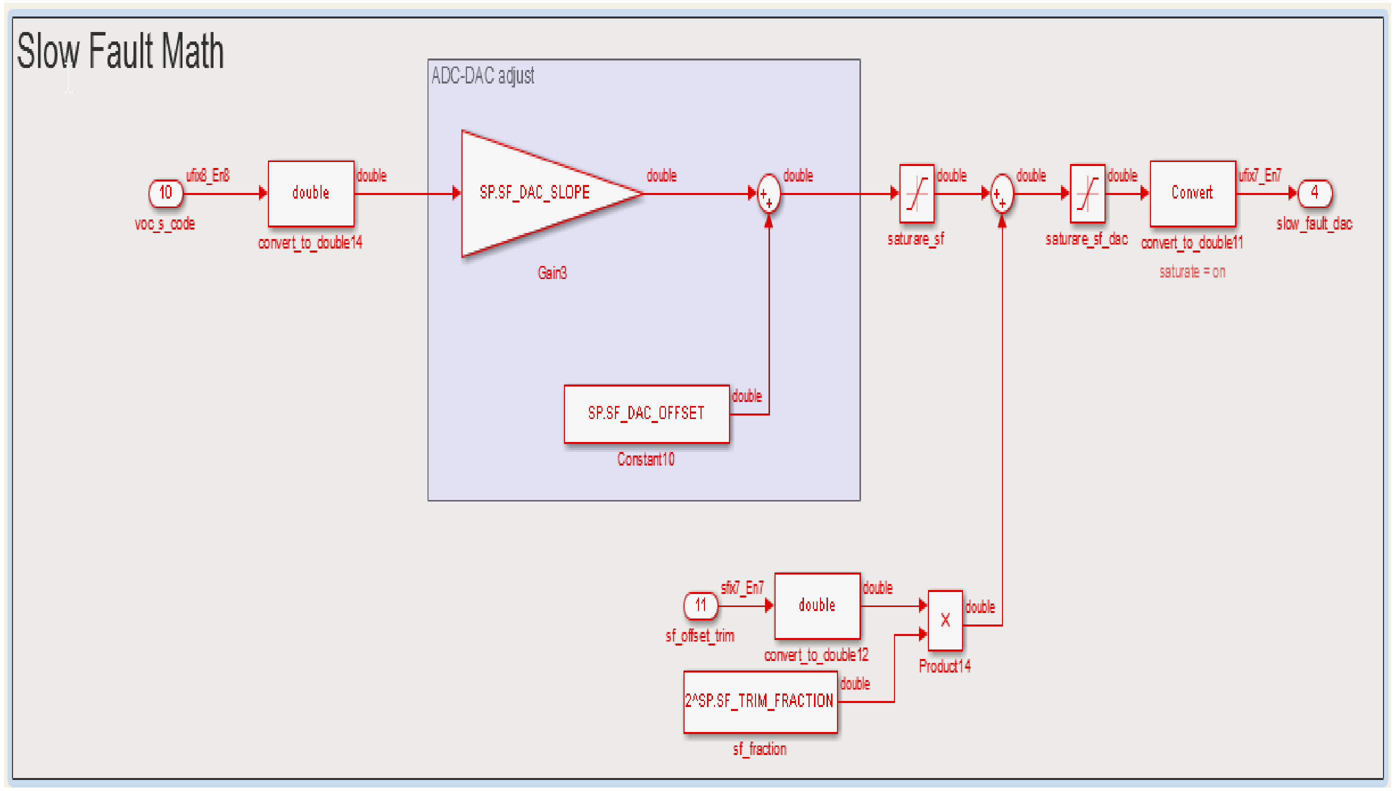
☐ Latch input by delaying outside signal

☐ Latch input for feedback signals of function-call subsystem outputs

☑ Interpolate data

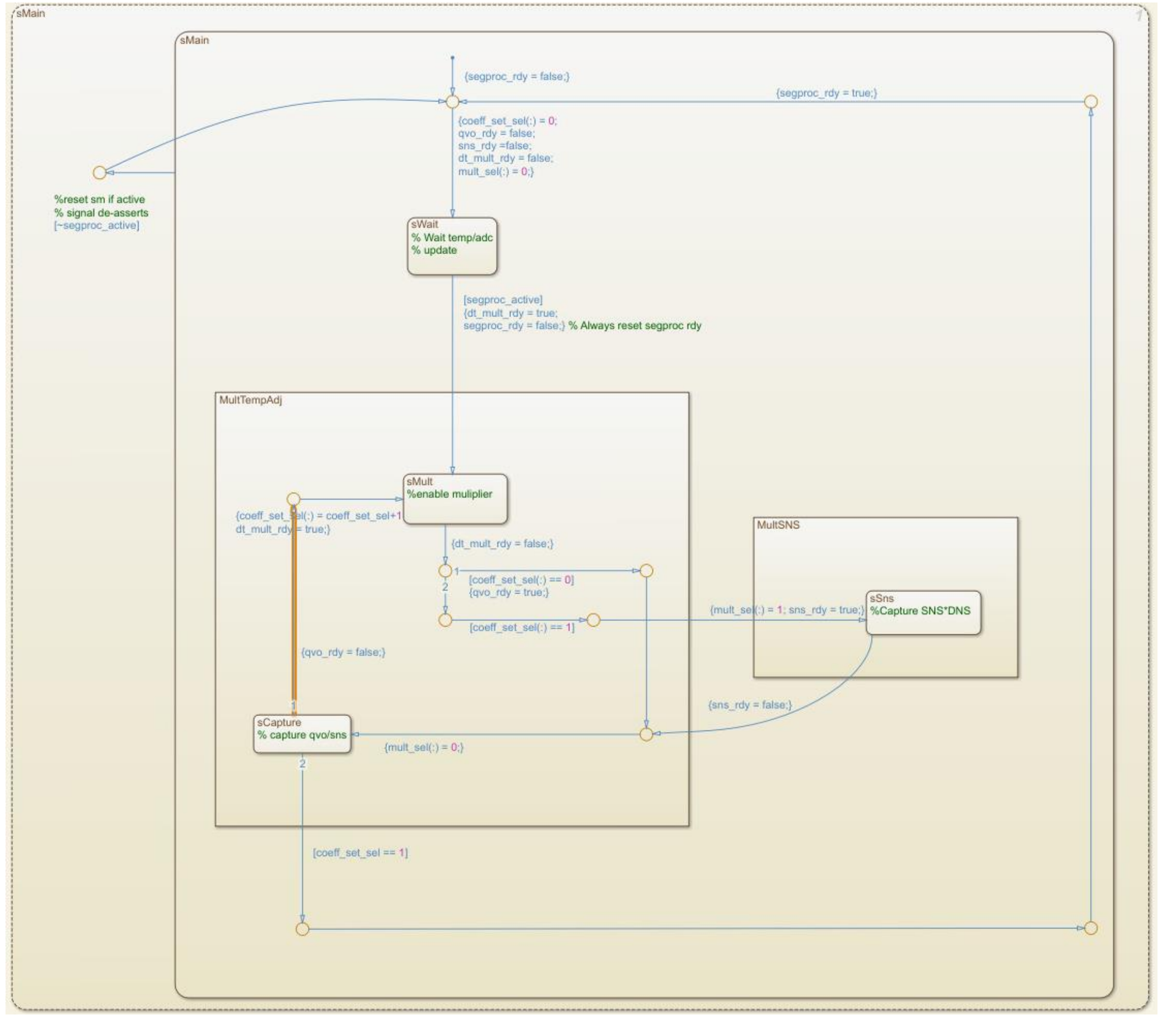
- ❑ **Custom Inports and Outports with attributes where created**
- ❑ **Additional attributes are used by Matlab Report Generator for automated document generation**

Simulink Spec Model



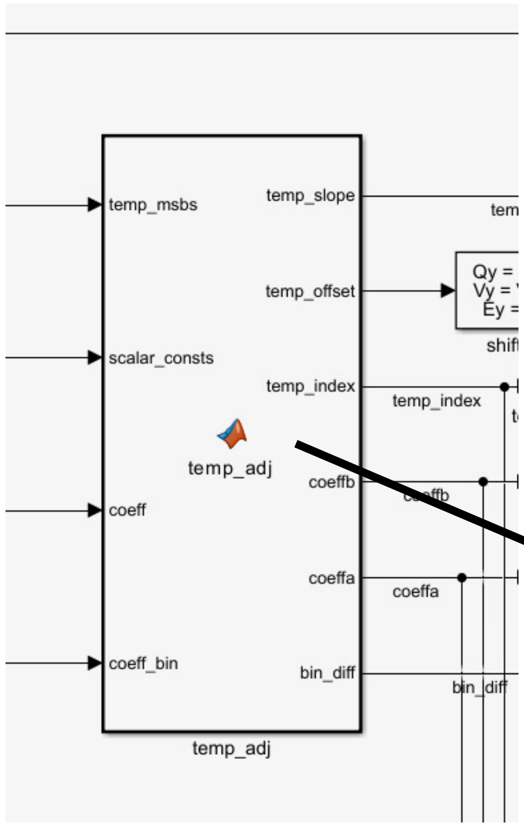
- ❑ **A Spec Model is the "golden model" of what you are trying to achieve.**
  - ❑ **Easy to read**
  - ❑ **Simple as possible**
- ❑ **Requirements are linked to the spec models**
- ❑ **Avoid**
  - ❑ **HDL optimizations**
- ❑ **Keep data types as doubles where possible**
- ❑ **Using DPI-C , System Verilog models are created from the spec models**

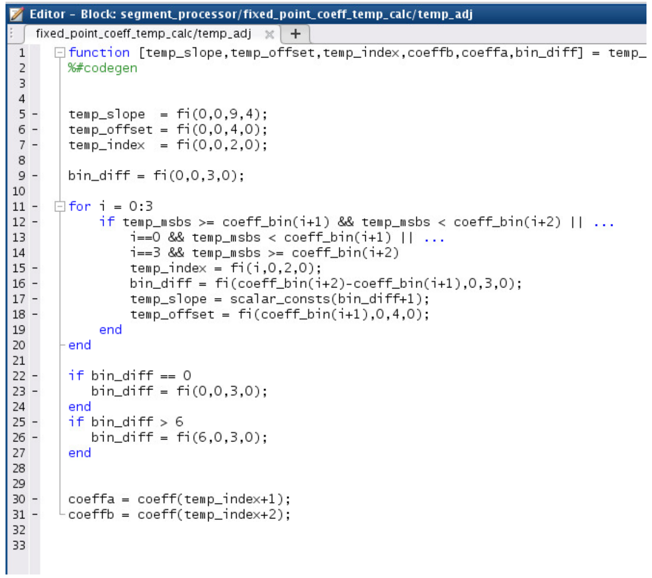
The Power of Stateflow



- ❑ **Stateflow is a universal tool**
  - ❑ **Stateflow allows for efficient control logic design that is self documenting and translates to efficient RTL(**)**
    - ❑ **SAR ADC Controller**
    - ❑ **Multiplier Sharing**
  - ❑ **Stateflow can be used as a verification driver (handles asynch events)**
  - ❑ **Stateflow can be used to model analog switched capacitor charge transfer!**
  - ❑ **Stateflow charts can be embedded within Stateflow charts**

Flexibility of Matlab Function Block





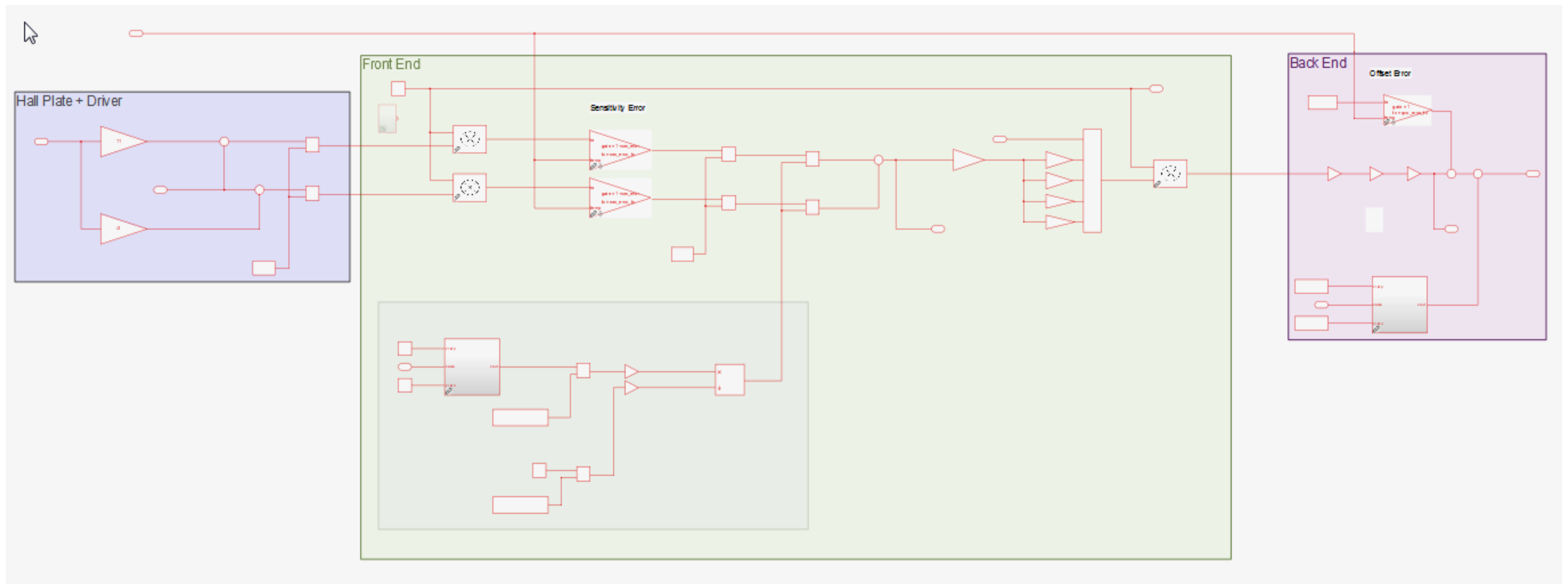
Editor - Block: segment_processor/fixed_point_coeff_temp_calc/temp_adj

fixed_point_coeff_temp_calc/temp_adj

```
function [temp_slope,temp_offset,temp_index,coeffb,coeffa,bin_diff] = temp_
%#codegen


temp_slope  = fi(0,0,9,4);
temp_offset = fi(0,0,4,0);
temp_index  = fi(0,0,2,0);

bin_diff = fi(0,0,3,0);

for i = 0:3
    if temp_msbs >= coeff_bin(i+1) && temp_msbs < coeff_bin(i+2) || ...
        i==0 && temp_msbs < coeff_bin(i+1) || ...
        i==3 && temp_msbs >= coeff_bin(i+2)
        temp_index = fi(i,0,2,0);
        bin_diff = fi(coeff_bin(i+2)-coeff_bin(i+1),0,3,0);
        temp_slope = scalar_consts(bin_diff+1);
        temp_offset = fi(coeff_bin(i+1),0,4,0);
    end
end

if bin_diff == 0
   bin_diff = fi(0,0,3,0);
end
if bin_diff > 6
   bin_diff = fi(6,0,3,0);
end


coeffa = coeff(temp_index+1);
coeffb = coeff(temp_index+2);
```

- ❑ **Code generation is not limited to Simulink fixed point and Stateflow**
- ❑ **Many functions and operations are more efficiently written as a Matlab function**
- ❑ **Floating point functions**
    - ❑ **coder.approximate**
- ❑ **Simulink is your canvas**

Simulink Validation Model

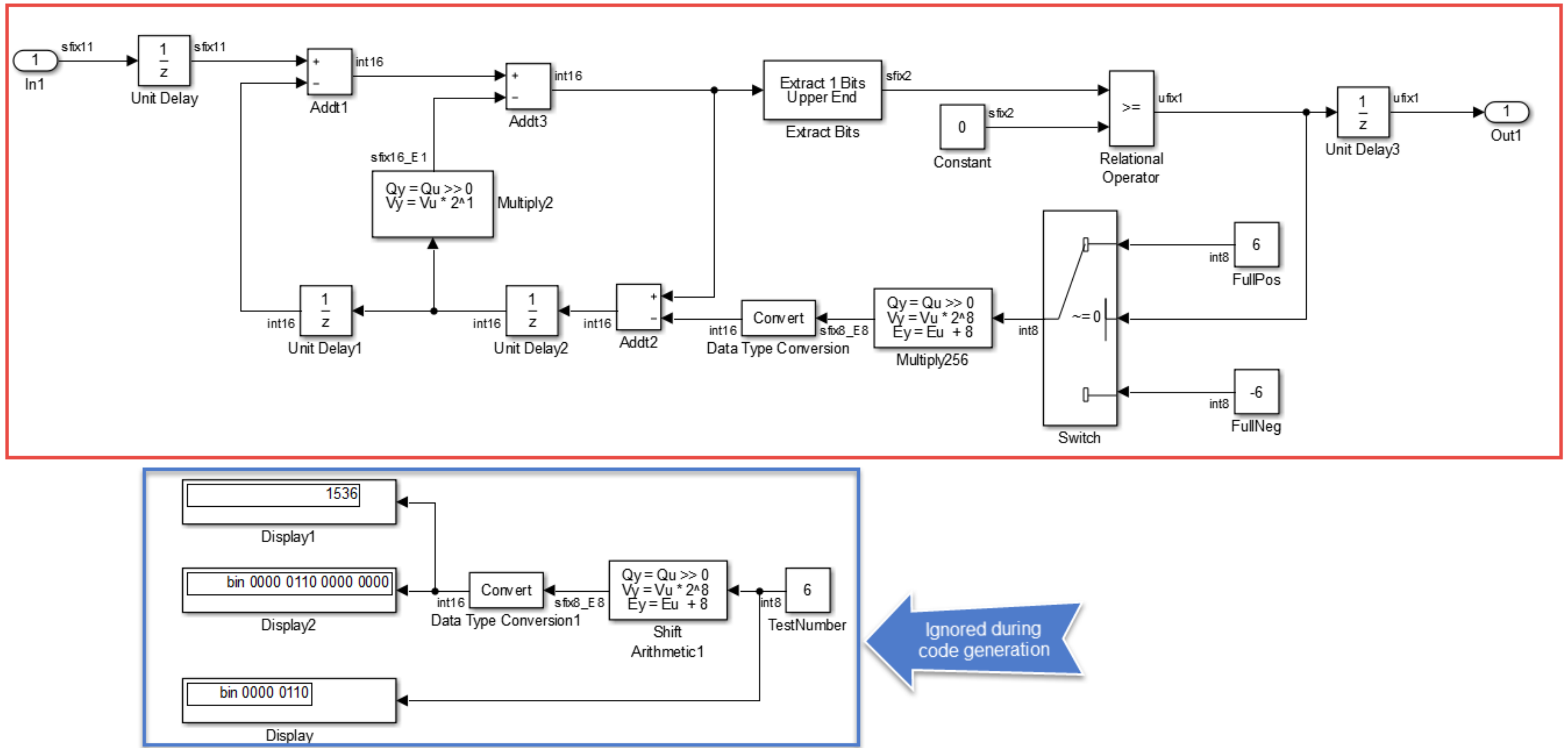


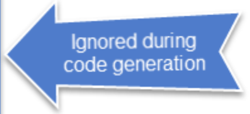
- A validation model generates “evidence” that specific requirements are met
  - Automated regressions require that a pass/fail criteria be used

- Asserts are your friend!
  - We hope to see the asserts pass through to the auto generated HDL very soon!

Simulink Code Generation for Digital (and Analog ?!)

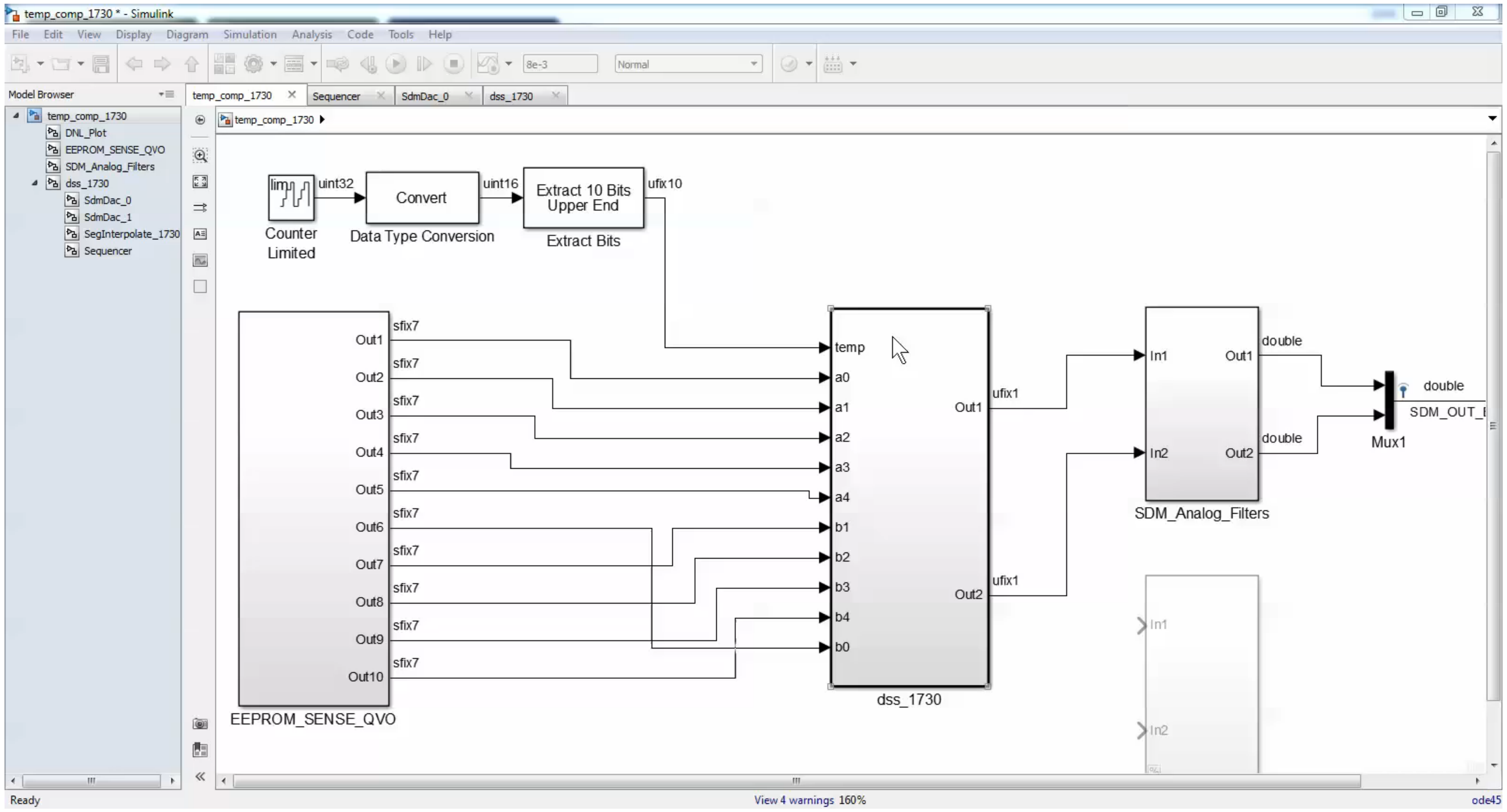
Digital Example: SDM DAC



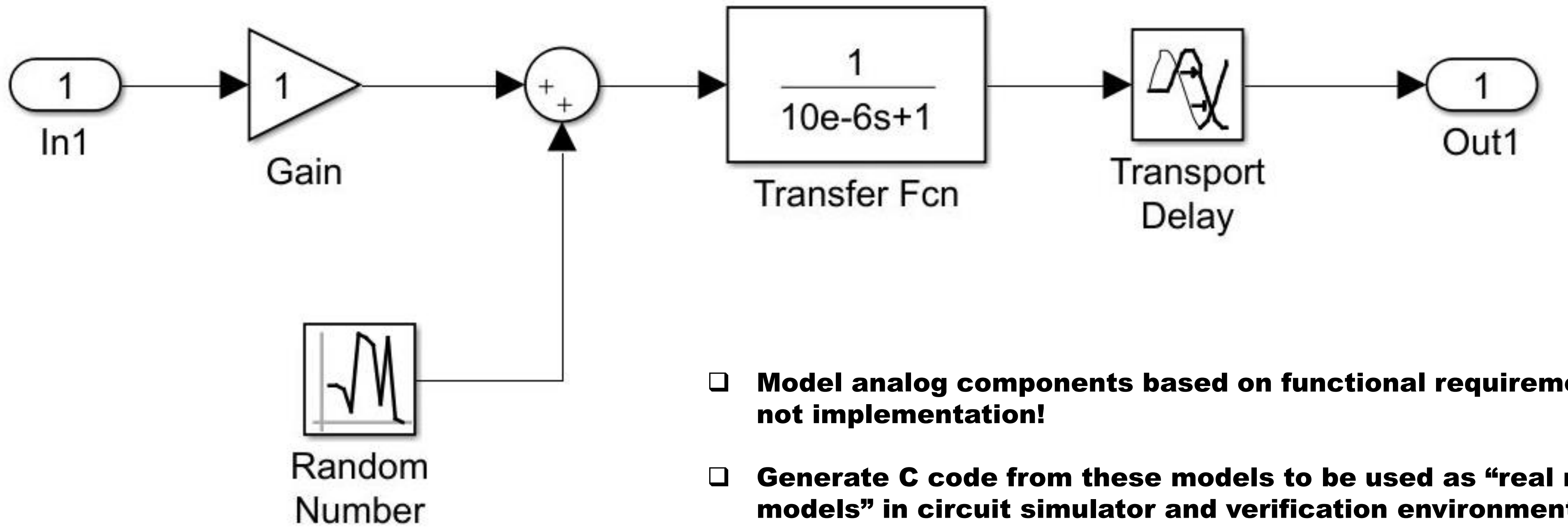


- Fixed Point Modeling is VERY powerful in Simulink
  - Fixed point optimization and area/accuracy tradeoffs are rapidly analyzed

Digital DAC: Code Generation Video

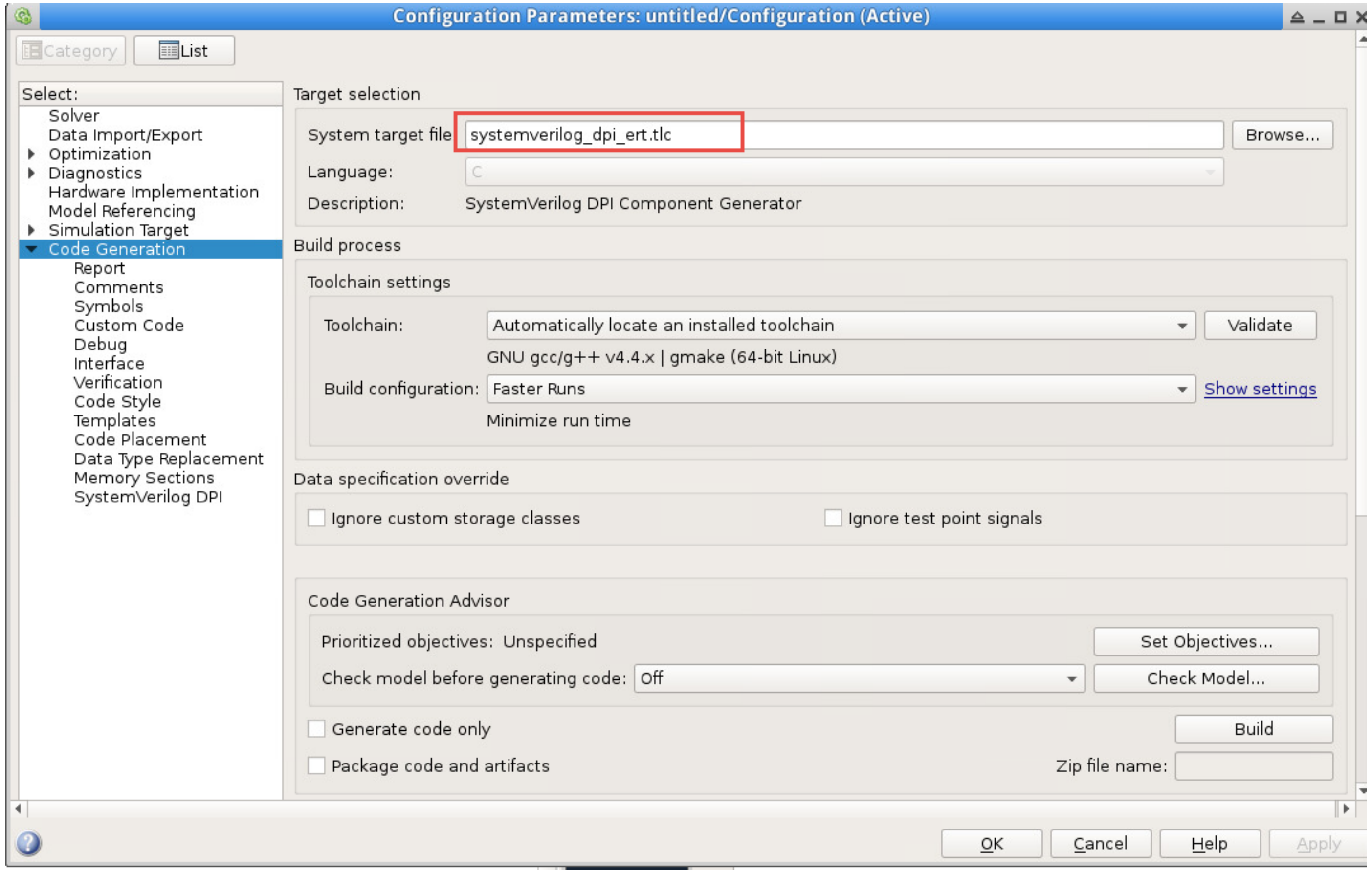


Simulink Analog Models using DPI-C: Passive Filter with Noise and Delay



- Model analog components based on functional requirements, not implementation!
- Generate C code from these models to be used as “real number models” in circuit simulator and verification environment
- This will allow for traceability and equivalence checking under the ISO26262 Automotive Specification.
- Analog designers will feel a strong attraction to SimElectronics. Leave the implementation for the circuit simulator (Cadence, etc.)

Generate the System Verilog / C-code Model

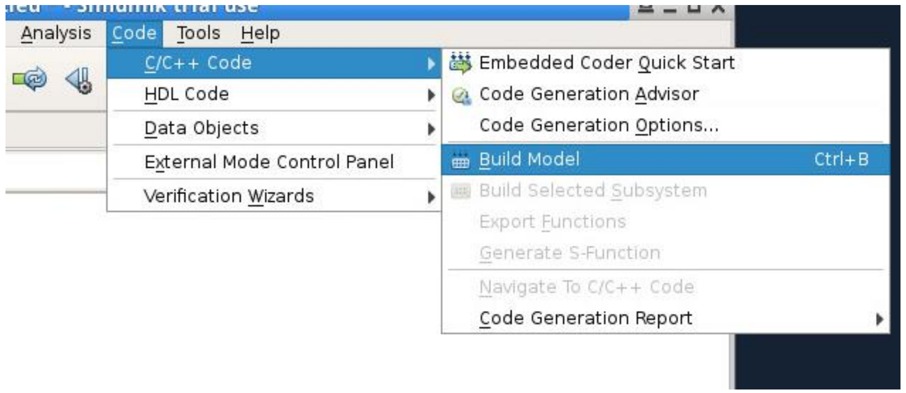

Configuration Parameters: untitled/Configuration (Active)

Target selection
System target file: systemverilog_dpi_ert.tlc
Language: C
Description: SystemVerilog DPI Component Generator

Build process
Toolchain settings
Toolchain: Automatically locate an installed toolchain
GNU gcc/g++ v4.4.x | gmake (64-bit Linux)
Build configuration: Faster Runs
Minimize run time

Data specification override
Ignore custom storage classes
Ignore test point signals

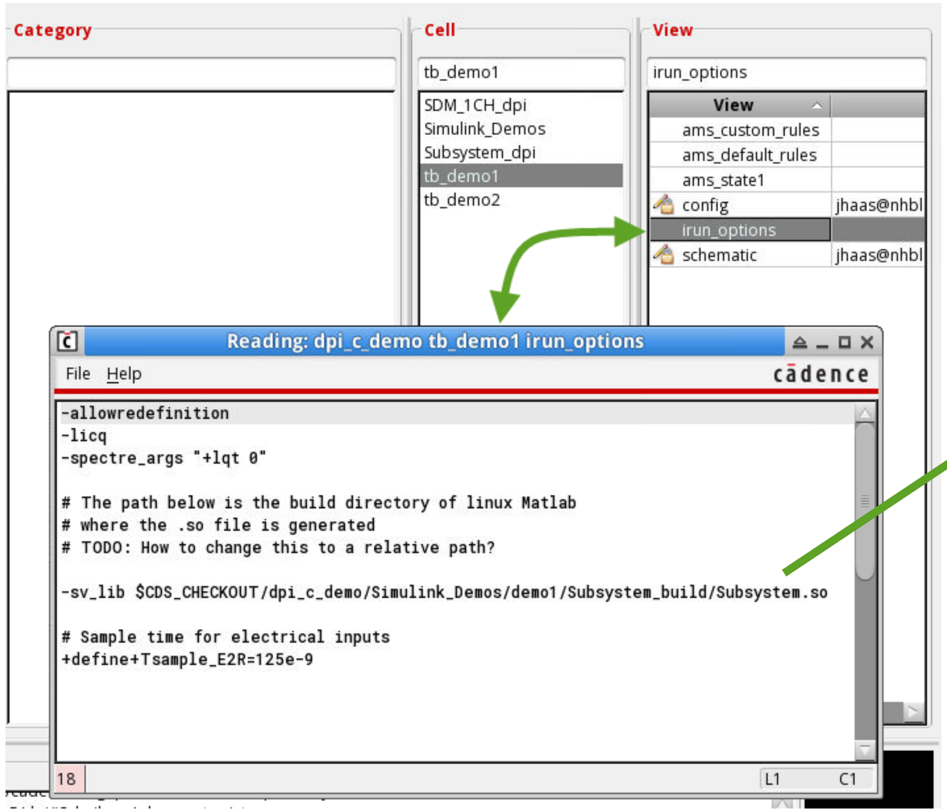
Code Generation Advisor
Prioritized objectives: Unspecified
Check model before generating code: Off
Generate code only
Package code and artifacts
Zip file name:



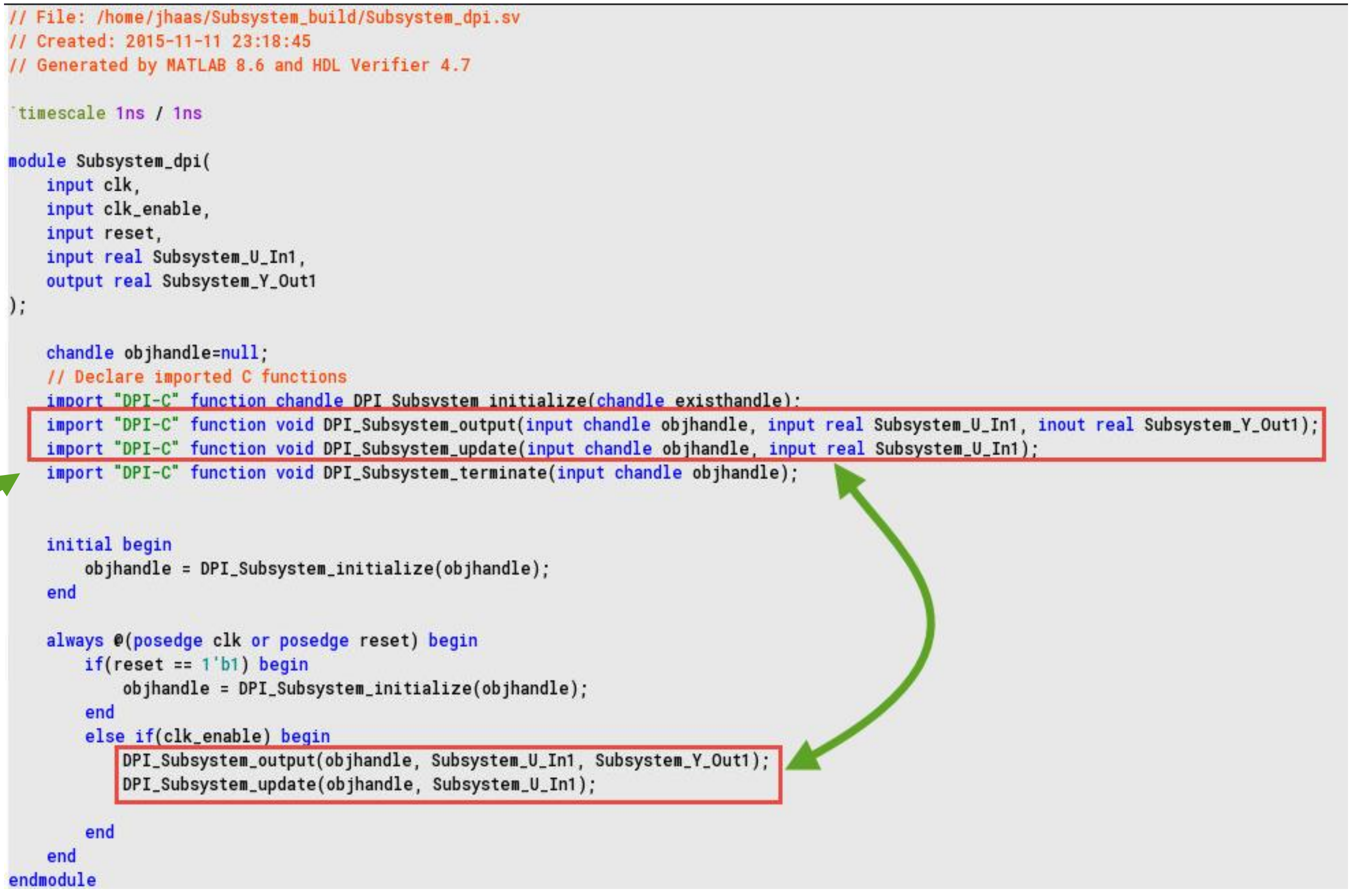
- ❑ **Select “systemverilog_dpi_ert.tlc”**
- ❑ **Files to be generated**
  - ❑ **Model_name.sv “wrapper”**
  - ❑ **Model_name.so**

The System Verilog Wrapper and the System Object Functions

- ❑ **Within AMS Options, add the path to the irun_options file**

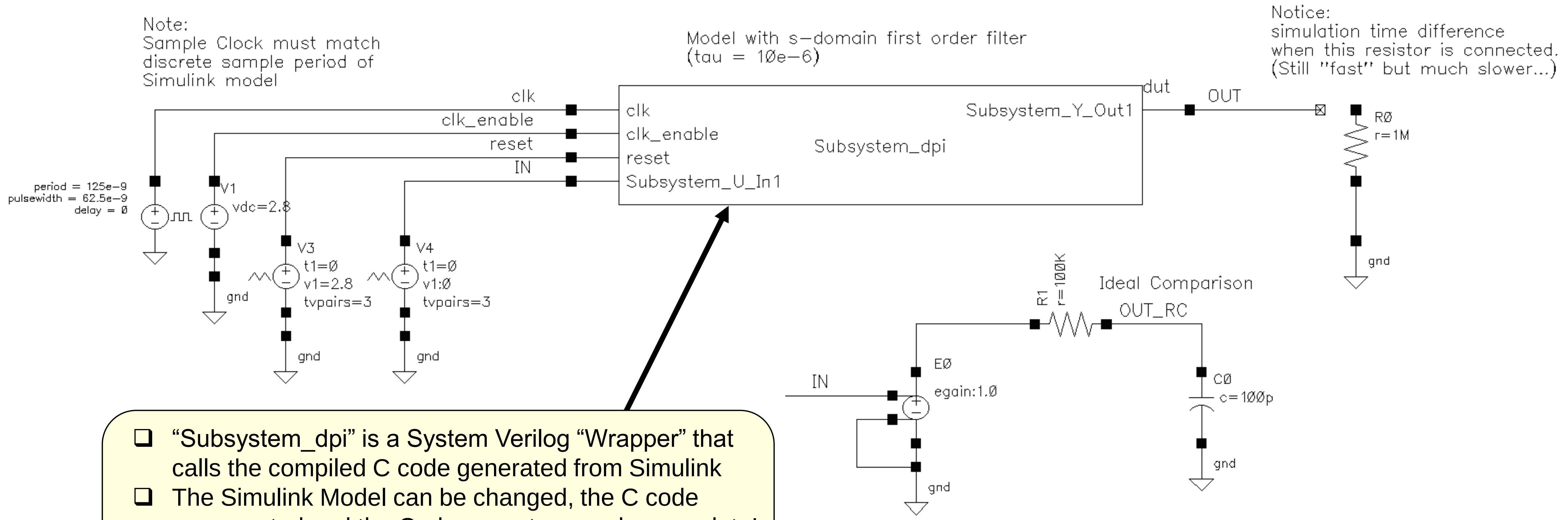


```
-allowredefinition
-licq
-spectre_args "+lqt 0"

# The path below is the build directory of linux Matlab
# where the .so file is generated
# TODO: How to change this to a relative path?

-sv_lib $CDS_CHECKOUT/dpi_c_demo/Simulink_Demos/demo1/Subsystem_build/Subsystem.so

# Sample time for electrical inputs
+define+Tsample_E2R=125e-9
```



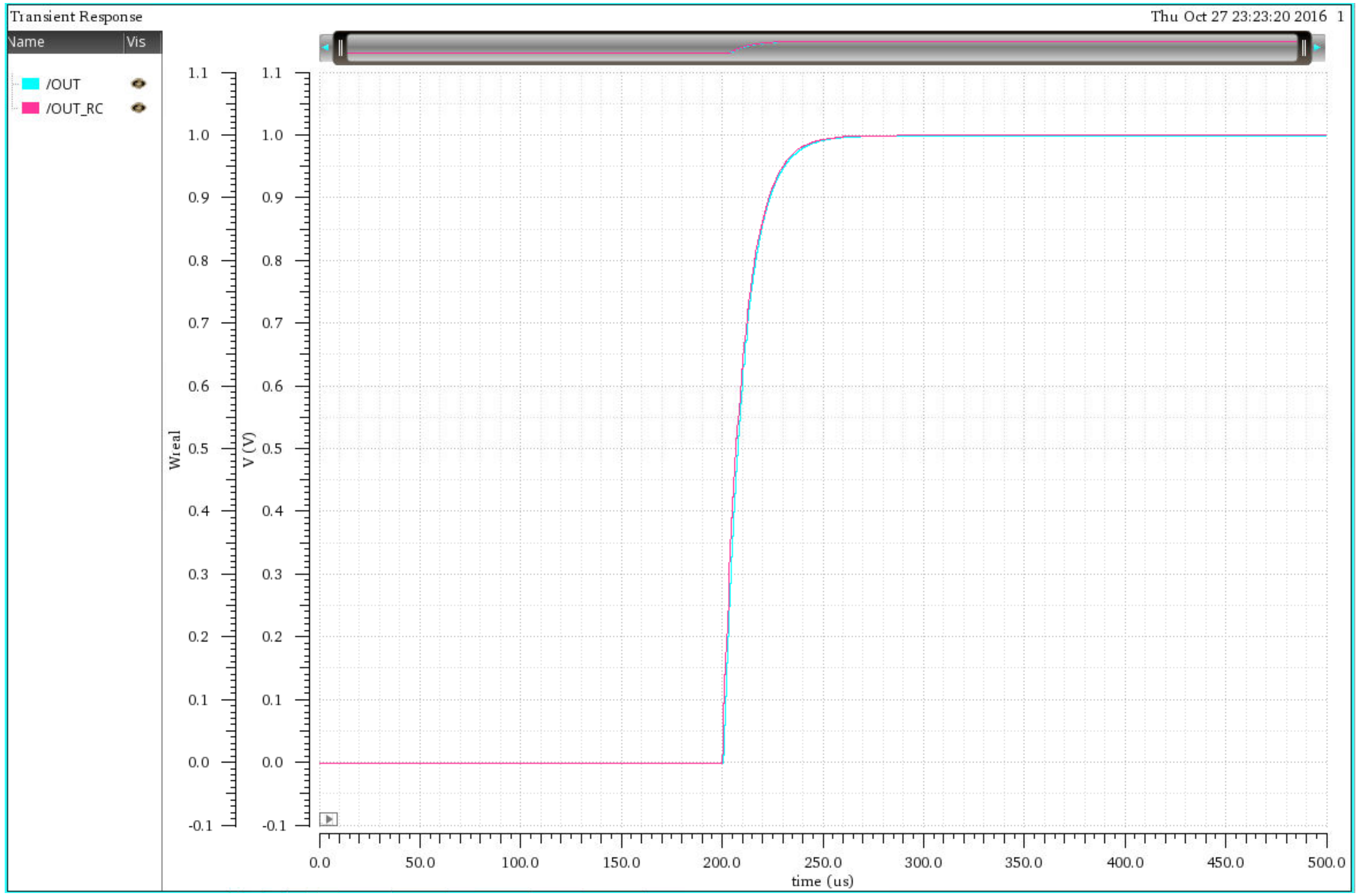
```
// File: /home/jhaas/Subsystem_build/Subsystem_dpi.sv
// Created: 2015-11-11 23:18:45
// Generated by MATLAB 8.6 and HDL Verifier 4.7

`timescale 1ns / 1ns

module Subsystem_dpi(
    input clk,
    input clk_enable,
    input reset,
    input real Subsystem_U_In1,
    output real Subsystem_Y_Out1
);

    chandle objhandle=null;
    // Declare imported C functions
    import "DPI-C" function chandle DPI_Subsystem_initialize(chandle existhandle);
    import "DPI-C" function void DPI_Subsystem_output(input chandle objhandle, input real Subsystem_U_In1, inout real Subsystem_Y_Out1);
    import "DPI-C" function void DPI_Subsystem_update(input chandle objhandle, input real Subsystem_U_In1);
    import "DPI-C" function void DPI_Subsystem_terminate(input chandle objhandle);


    initial begin
        objhandle = DPI_Subsystem_initialize(objhandle);
    end

    always @(posedge clk or posedge reset) begin
        if(reset == 1'b1) begin
            objhandle = DPI_Subsystem_initialize(objhandle);
        end
        else if(clk_enable) begin
            DPI_Subsystem_output(objhandle, Subsystem_U_In1, Subsystem_Y_Out1);
            DPI_Subsystem_update(objhandle, Subsystem_U_In1);

        end
    end
endmodule
```

Cadence AMS Test Bench

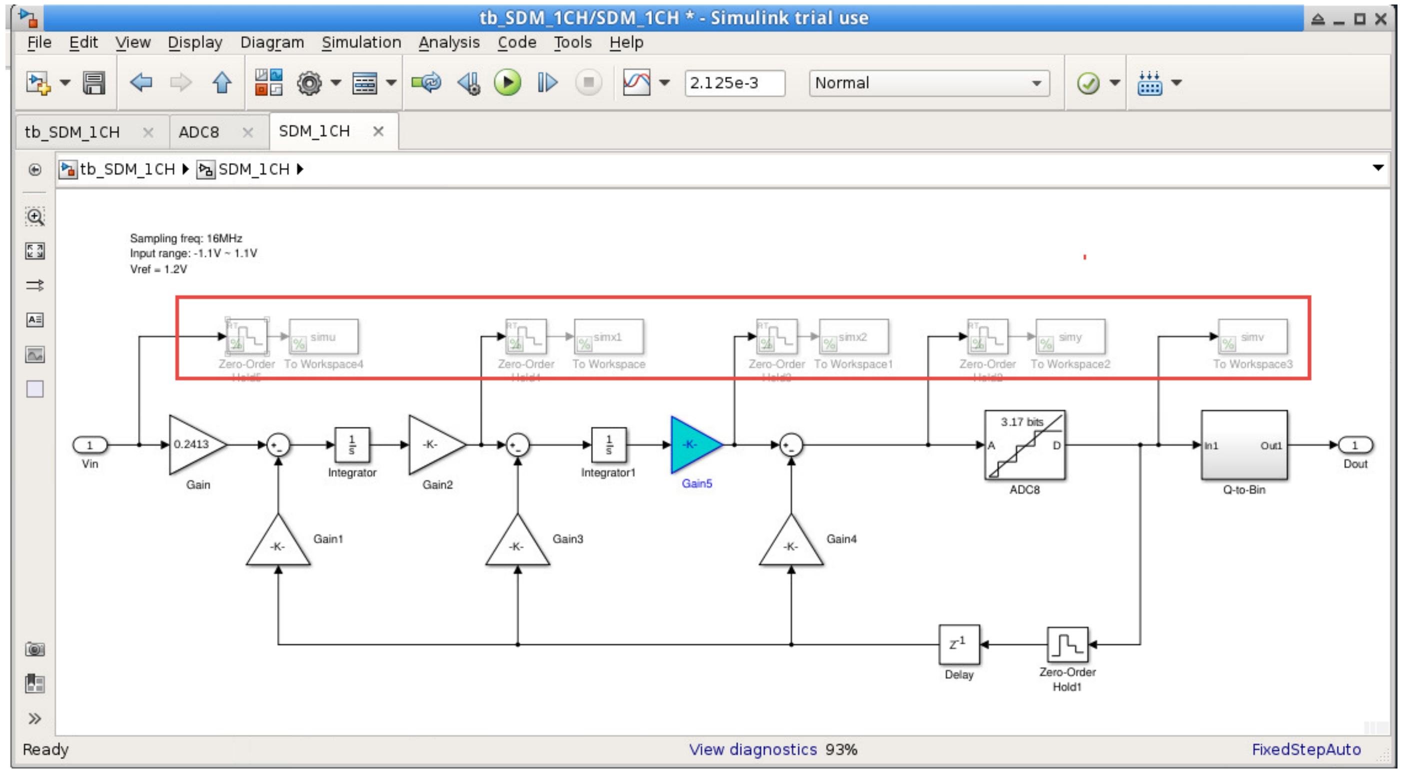


- "Subsystem_dpi" is a System Verilog "Wrapper" that calls the compiled C code generated from Simulink
- The Simulink Model can be changed, the C code regenerated and the Cadence setup needs no update!
    - Makefile is automatically called upon Simulink Code Generation

Cadence Simulation of Simulink “C” Model and Ideal RC



Simulink Analog Models using DPI-C: Continuous Time Sigma Delta ADC



- ❑ Make sure and "comment out" any elements in the subsystem that you do not want in the generated code.
  - ❑ Those elements are shown inside the red box.

ASIC MBD Summary
Present and Future

- Simulink and Matlab have been instrumental in the development of an agile Automotive Mixed Signal ASIC Sensor Flow
    - High level model exploration allows for accelerated insights and convergence on architecture and algorithms
    - Traditional duplication efforts (model – spec – another model) are minimized
    - Upfront models expedite the verification efforts and front load issue discovery
    - Powerful Real Time Simulation Platforms (Speed Goat) allow for testing algorithms in the lab before design team is heavily engaged!
    - Powerful modeling, automated code generation and robust traceability are paving the way for agile development in an ISO26262 world!

Thank You!

For more information:
http://www.allegromicro.com

Email:
jhaas@allegromicro.com