2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

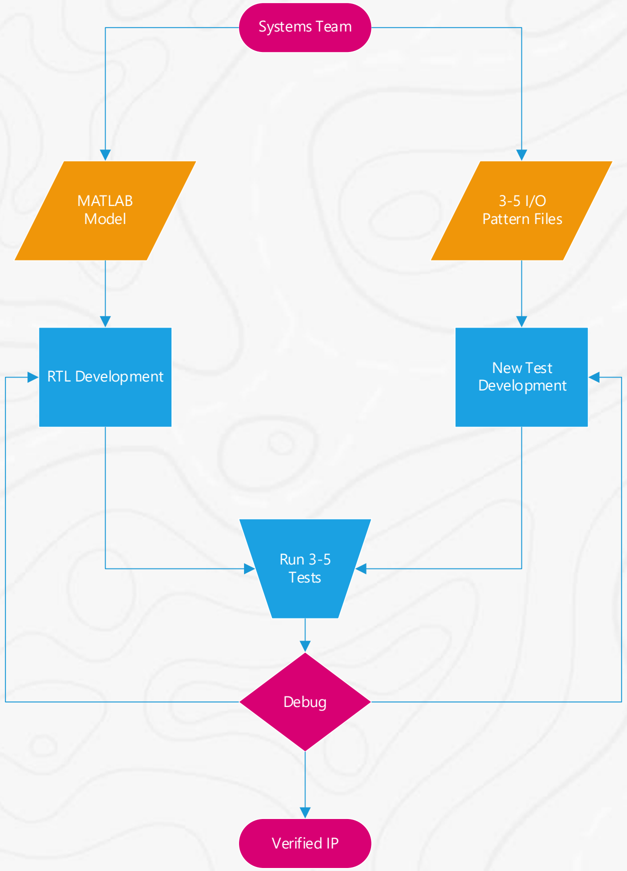
Flattening the UVM Learning Curve: Automated Solutions for DSP Filter Verification

Avinash Lakshminarayana, Eric Jackowski – Silicon Labs

Eric Cigan, Mark Lin - MathWorks

MathWorks® SILICON LABS accellera SYSTEMS INITIATIVE

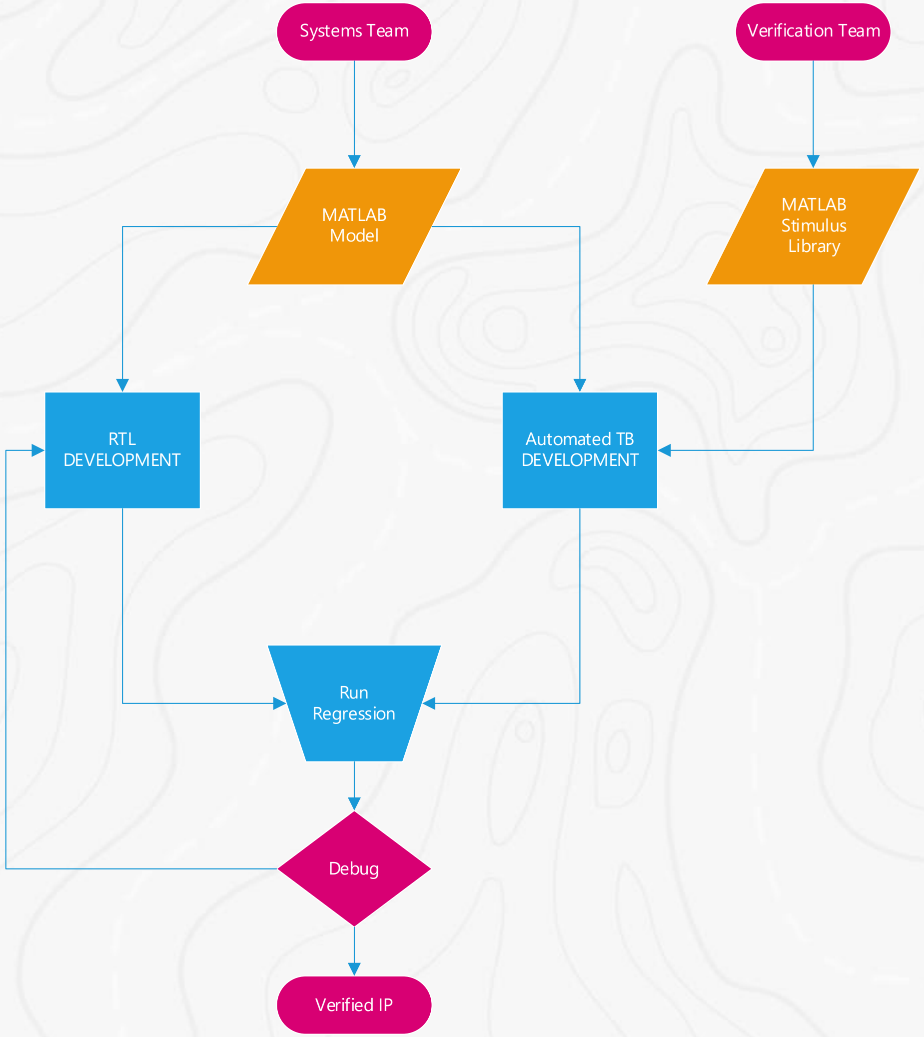
Introduction



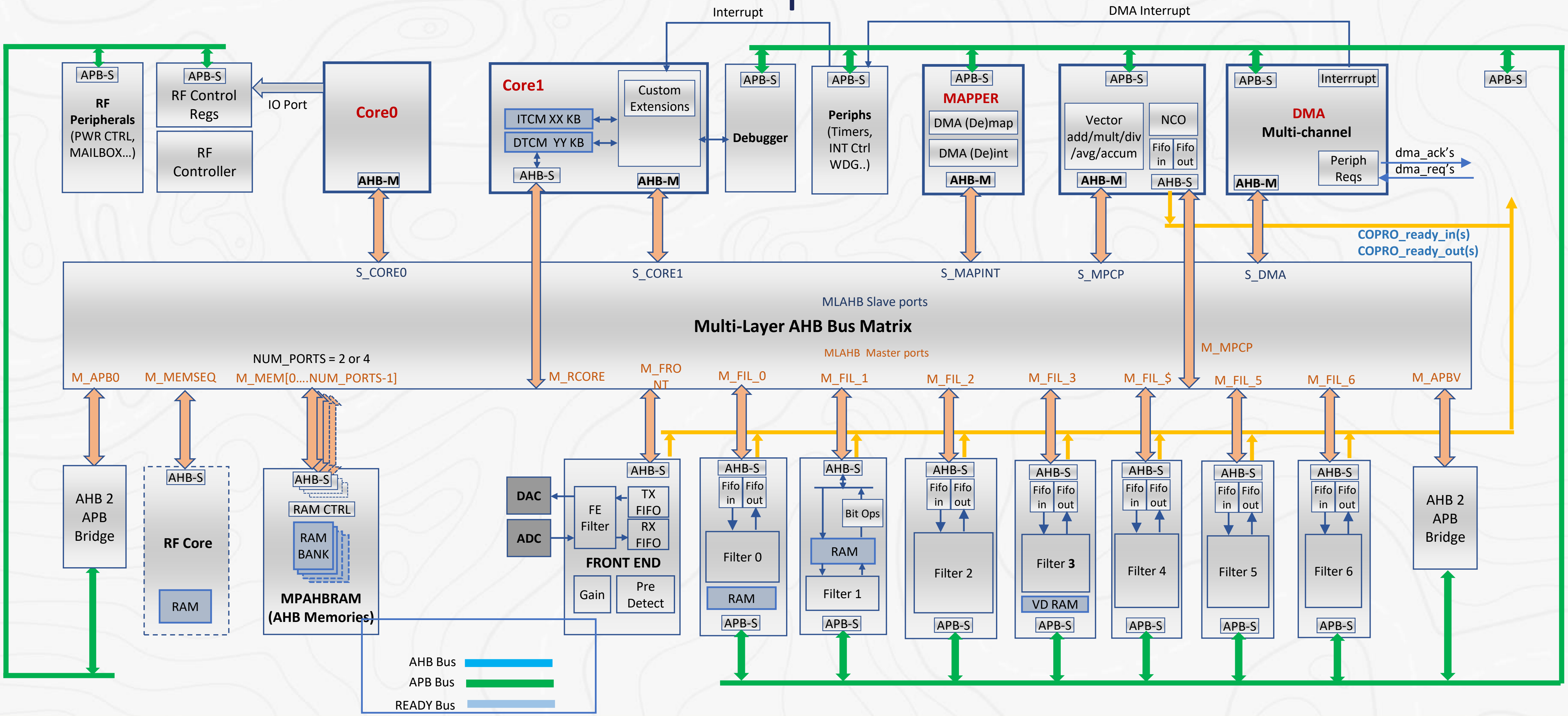
Existing Flow

- Need for Constrained-Random Verification
- Expansion of control logic

Proposed Flow



DSP Blocks in an IoT Chip



Challenges & Opportunities

- Small footprint
- Large number of filters
- Math-intensive scoreboards
- Cross-functional teams
- Shared I/O characteristics
- Simple control path
- Stimulus reuse

Previous UVM Support with High-Level Tools

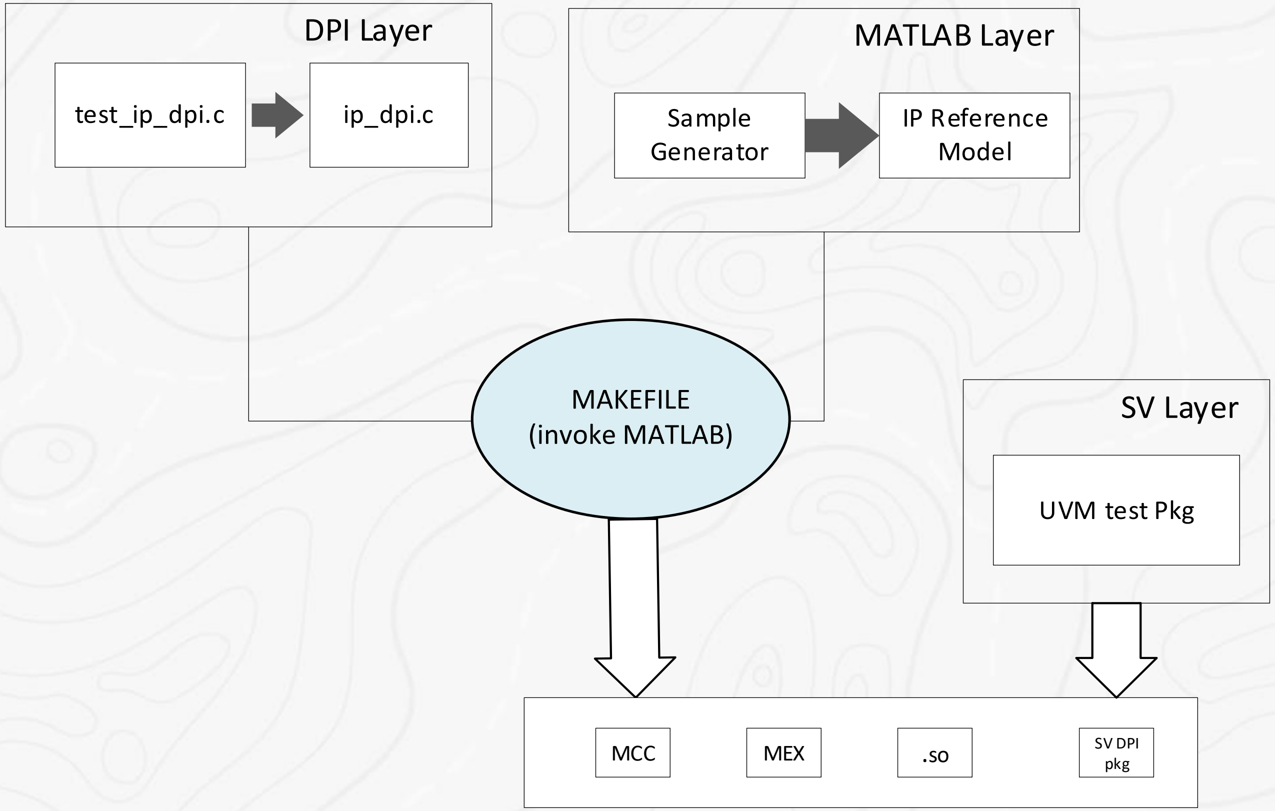
- UVM Framework (Mentor Graphics® / Siemens® EDA)
  - Integrated with MATLAB® through MATLAB Engine API
  - Supported other high-level tools: e.g., Catapult® HLS

- SystemVerilog DPI generation from MATLAB / Simulink®(MathWorks®)
  - Produces shared library from either MATLAB functions or Simulink subsystems for specified arguments and data types.
  - Generates a directory structure with DPI-C wrappers, header files, makefiles, SystemVerilog testbenches, and simulation scripts
  - Supports variety of commercial HDL simulators

Three Case Studies

1. Semi-automated solution using MATLAB Compiler™
2. Coprocessor testbench generation
3. Automated UVM environment generation from MATLAB and Simulink with `dpigen()` and `uvmbuild()`

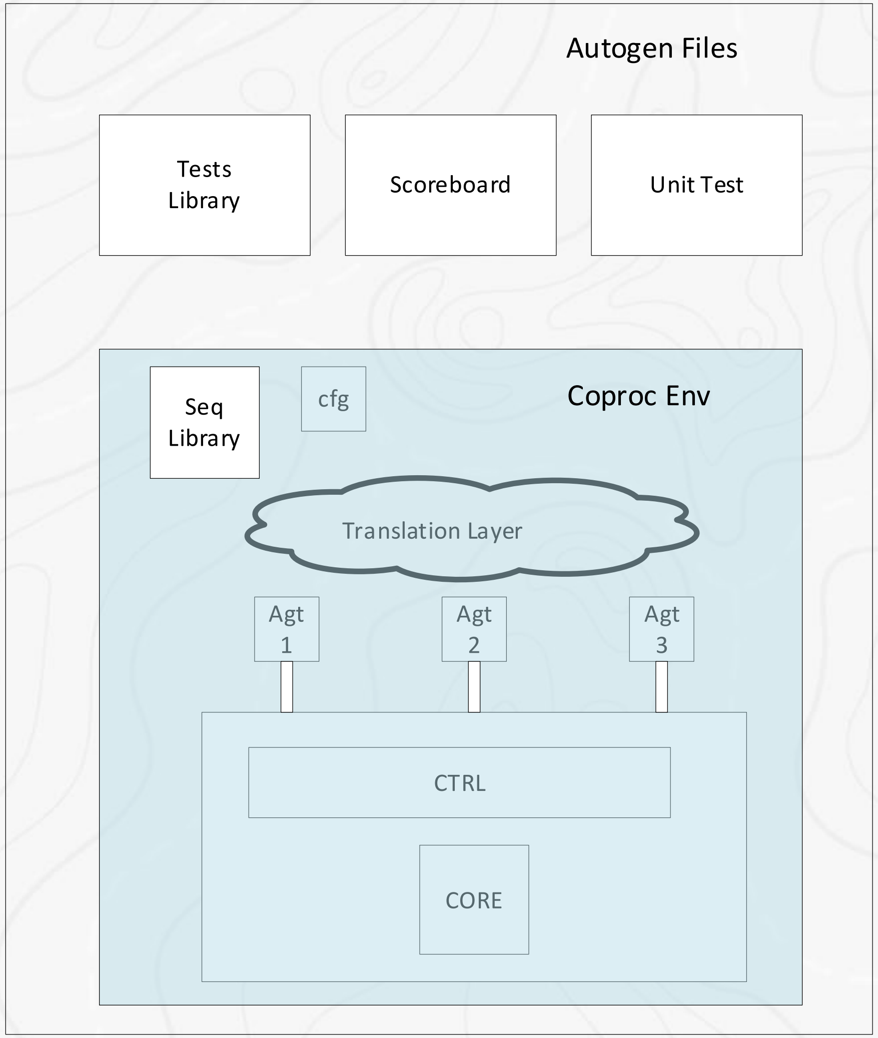
Semi-automated Solution



DPI Function Calls

```
//
// DPI C Functions
//

import "DPI-C" function int  firFilterDpiInitializeMatlab(string options[MAX_MATLAB_OPTIONS], int count);
import "DPI-C" function int  firFilterDpiInitializeLibrary();
import "DPI-C" function int  firFilterDpiRunSampleGenerator(input int unsigned  sample_type,
                                                            input int unsigned  tone_in_hz,
                                                            input int unsigned  latency_ctrl,
                                                            input real          coeff[NUM_OF_SYMETRIC_COEFFS],
                                                            input int unsigned  size_of_coeff,
                                                            input int unsigned  random_seed,
                                                            input int unsigned  max_rtl_samples,
                                                            output int unsigned size_of_in,
                                                            output int unsigned in_i [RADIO_MODEL_MAX_SAMPLES],
                                                            output int unsigned in_q [RADIO_MODEL_MAX_SAMPLES],
                                                            output int unsigned size_of_out,
                                                            output int          out_i [RADIO_MODEL_MAX_SAMPLES],
                                                            output int          out_q [RADIO_MODEL_MAX_SAMPLES]);
import "DPI-C" function int  firFilterDpiTerminateLibrary();
import "DPI-C" function int  firFilterDpiTerminateMatlab();

//
// SV Function calls wrapping DPI C funcitons
//
matlab_options[matlab_options_count++] = "-nosplash";
matlab_options[matlab_options_count++] = "-nodisplay";
matlab_options[matlab_options_count++] = "-nojvm";
fir_filter_initialize_matlab(matlab_options,matlab_options_count);

fir_filter_initialize_radio_model();

fir_filter_run_radio_model(sample_type,
                           tone_in_hz,
                           latency_ctrl,
                           coeff_for_matlab,
                           size_of_in,
                           in_i,
                           in_q,
                           size_of_out,
                           out_i,
                           out_q);

fir_filter_terminate_radio_model();

fir_filter_terminate_matlab();
```

Coprocessor Testbench Generation



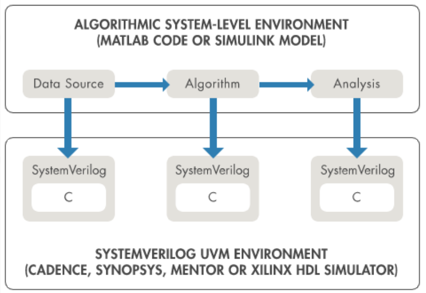
Coprocessor Verification Environment

- Parametrization of variables and interfaces
- Generic Coprocessor transaction
- Translation layer to add protocol specific details
- Config objects – protocol specific and generic
- Sequence Library, Assertions and Cover groups

Reusing MATLAB models with SystemVerilog DPI

Generate SystemVerilog DPI-C models from MATLAB stimulus sequence and reference models with the `dpigen()` feature of HDL Verifier, then reuse in RTL testbenches.

```
function [out, d] = tb_dec8(in)
    .
    .
    .
    out = dec8 (in);
    out = int32(out);
end
```



HDL Verifier

`dpigen()`

```
import tb_dec8_dpi_pkg::*;

module tb_dec8_dpi(input bit clk,
                   input real in[0:19],
                   …
                   output real out_im);
initial
  begin
    DPI_tb_dec8_initialize();
  end

always @(posedge clk begin
  if(reset== 1'b1) begin
      DPI_tb_dec8_reset(…);
    end
  else begin
      DPI_tb_dec8(…);
     end
  end
end module
```

Generating C for DPI from MATLAB

- MATLAB function `dec8.m` models a decimate-by-8 CIC filter, with a subfunction call to `calc_cic_filter.m`
- We created a wrapper function (`tb_dec8.m`) to meet code generation requirements and avoid modification of original MATLAB function `dec8.m`

```
function [matlab_out, dummy] = tb_dec8(matlab_in)
    matlab_out = dec8 (matlab_in);
    matlab_out = int32(matlab_out);
end
```

- We used `dpigen()` to generate C for DPI from tb_dec8.m. Data types of input and output argument must be provided to `dpigen()`

```
dpigen -args {double(ones(20,1))} tb_dec8
```

MATLAB Model Constraints:

1. Ensure function argument type and size don't change during execution
   - Use wrapper functions to provide type conversion, limit maximum size, and avoid modification to original code
2. Use `persistent` variable to maintain state

Generated DPI-C Functions

```
function chandle DPI_tb_dec8_initialize(input chandle existhandle);
function chandle DPI_tb_dec8_reset(input chandle objhandle,
                                   input real matlab_in [20],
                                   output real matlab_out_re,
                                   output real matlab_out_im,
                                   output real dummy);
function void DPI_tb_dec8(input chandle objhandle,
                          input real matlab_in [20],
                          output real matlab_out_re,
                          output real matlab_out_im,
                          output real dummy);
function void DPI_tb_dec8_terminate(input chandle existhandle);
```

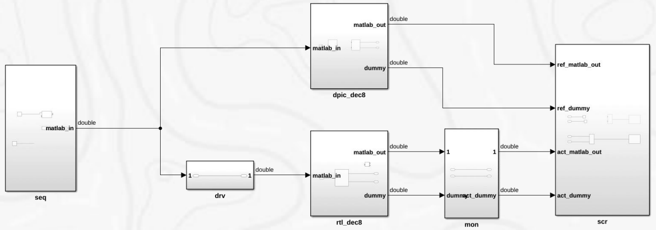
Integrating Generated DPI-C Functions

- Verilog module tb_dec8_dpi.sv was provided automatically along with generated DPI-C
- DPI-C functions generated using `dpigen()` can be integrated manually into UVM test benches

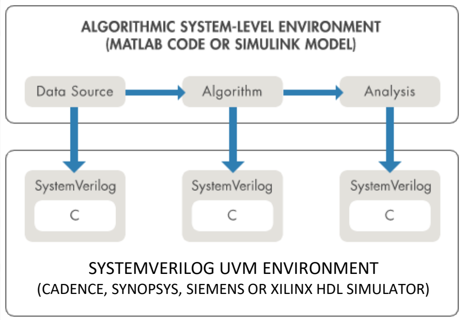
```
import tb_dec8_dpi_pkg::*;
module tb_dec8_dpi(input bit clk,
                   input real matlab_in [0:19],
                   …
                      output real matlab_out_im);
…
    initial begin
        objhandle=DPI_tb_dec8_initialize(objhandle);
    end

    final begin
        DPI_tb_dec8_terminate(objhandle);
    end

    always @(posedge clk or posedge reset) begin
        if(reset== 1'b1) begin
         objhandle=DPI_tb_dec8_reset(…);
            …
        end
        else if(clk_enable) begin
         DPI_tb_dec8(…, matlab_out_im_temp);
         matlab_out_im <= matlab_out_im_temp;
         …
        end
    end
end module
```

Reusing Simulink as UVM Testcase

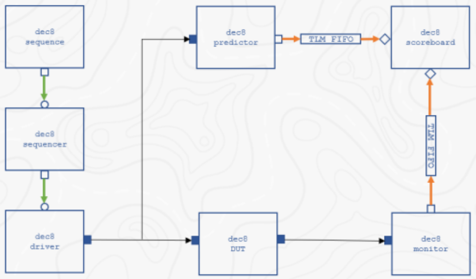
We automatically generated a complete UVM test case from the Simulink testbench with the `uvmbuild()` feature of HDL Verifier



Simulink testbench



HDL Verifier



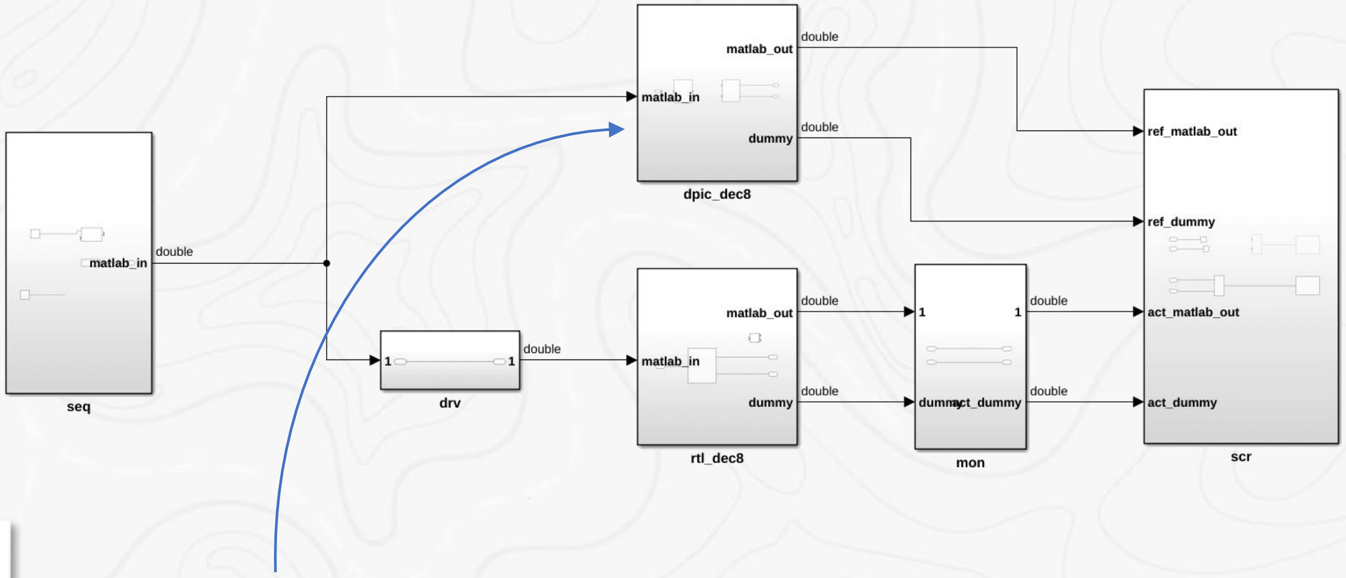
UVM

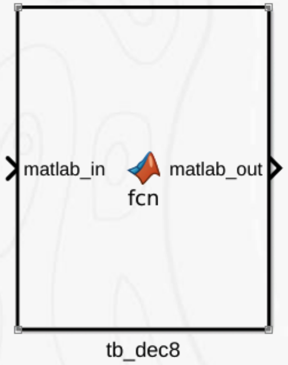
`uvmbuild()`

Generating UVM from Simulink

- UVM generation requires that the architecture of the Simulink model mimics UVM
- We incorporated the MATLAB model `tb_dec8.m` into Simulink model using the MATLAB Function Block
- We then generated the UVM testbench with `uvmbuild()`, using the script `makeuvm.m` to expedite the process

```
dut = 'sldec8/ref_dec1';
ref = 'sldec8/ref_dec8';
seq = 'sldec8/seq';
scr = 'sldec8/scr';
drv = 'sldec8/drv';
mon = 'sldec8/mon';

uvmbuild(dut,seq,scr,'Driver', drv,
                     'Monitor', mon,
                     'Predictor', ref);
```

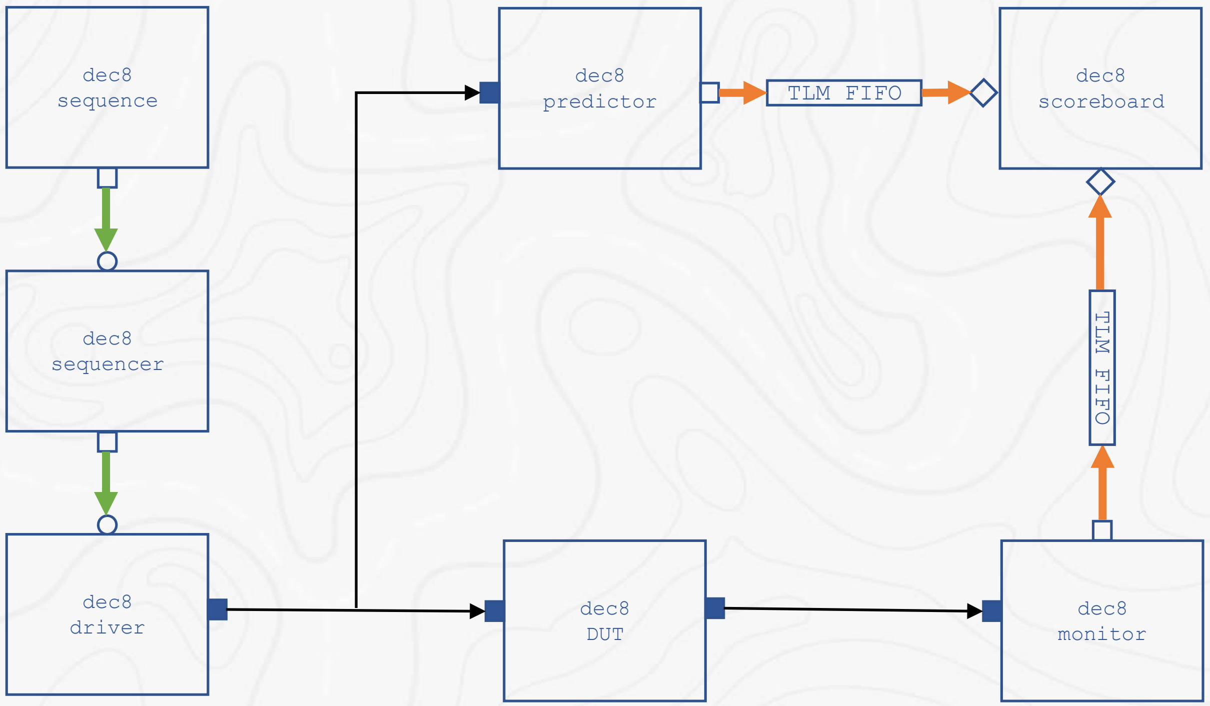





```
function matlab_out  = fcn(matlab_in)

  [matlab_out] = tb_dec8(matlab_in);
```

Generated UVM Component and Objects

- Simulink block results as an `uvm_component`
- Each Simulink connection results in a TLM with its corresponding `uvm_object`
- Simulink model connections define the TLM connections between components
- Supporting components like `env` and `agent` are generated automatically



sil_dec8_test.sv
sil_dec8_env.sv
sil_dec8_prd.sv
sil_dec8_scr.sv

sil_dec8_agt.sv
sil_dec8_seq.sv
sil_dec8_drv.sv
sil_dec8_mon.sv

sil_dec8_seq_trans.sv
sil_dec8_scr_trans.sv

Using Generated UVM Testcase

- Generated testcase was executed as is using QuestaSim via generated .do file

```
$ vsim -do run_tb_mq.do
```

- Each component – e.g., agent, driver, or scoreboard – could be independently integrated into existing UVM testbench.

Future Work

- Deploy assertions & coverage as part of the automated flow
- Model & Incorporate complex control path in SystemC to the Simulink model
- Investigate use of cosimulation of MATLAB and Questa to enable feedback from the RTL into the MATLAB model
- Work with System designers to explore generation of RTL from MATLAB and Simulink for prototyping

Conclusions

- Adopted hybrid approach
  - For legacy blocks – generated DPI-C using MATLAB Compiler
    - Avoids MATLAB code changes
  - For new blocks – generate DPI-C / UVM using HDL Verifier™
    - Provided guidance to System team so new MATLAB models will be code generate-able
- Some Measurable Results
  - Number of tests increased from 3 to 100
  - New testbench development time reduced from 2-3 weeks to under 2 days
  - Non-DV engineers can change testbenches using GUI

Additional Resources

- Company websites
  - Silicon Labs: www.silabs.com
  - MathWorks: www.mathworks.com
- **`dpigen()`** reference: mathworks.com/help/hdlverifier/ref/dpigen.html
- **`uvmbuild()`** reference: mathworks.com/help/hdlverifier/ref/uvmbuild.html
- Examples: mathworks.com/help/hdlverifier/examples.html?category=systemverilog-dpi-generation
- Products used:
  - HDL Verifier: mathworks.com/verify
  - MATLAB Coder: mathworks.com/products/matlab-coder
  - MATLAB Compiler: mathworks.com/products/compiler