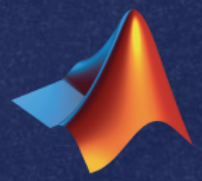
2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

How to leverage the power of MATLAB from Functional Verification Test Benches

Tom Richter - MathWorks



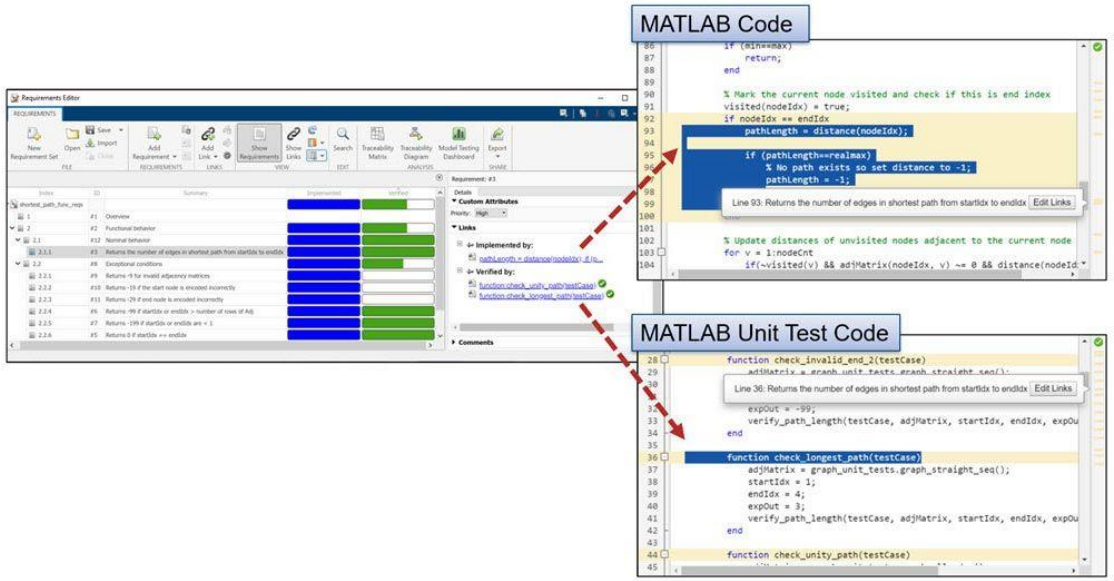
MathWorks®

accellera SYSTEMS INITIATIVE

Agenda

- Verification Challenges
- Why MATLAB?
- Verification options
  - Co-simulation with MATLAB® on top
  - Co-simulation with HDL Simulator on top
  - SystemVerilog DPI component generation
  - UVM testbenches and components
- Questions

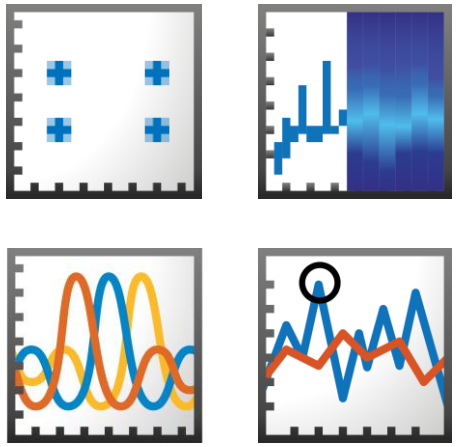
Verification Challenges



- The complex nature of the design
- Time and resource limitation
- Stimulus generation
- Golden reference model creation
- Verify that the deployed algorithm works the same as the reference
- Are all requirements implemented and tested?

Why MATLAB?

- Used in many domains such as signal processing, image processing and communications
- There, it is de facto standard language to explore, evaluate, and design architectures and algorithms
- MATLAB environment and its add-on products:
  - Intuitively learnable high-level programming language
  - Huge library of Functions, Classes, and System Objects
  - Diverse visualization options
  - Many apps, functions and objects for signal generation



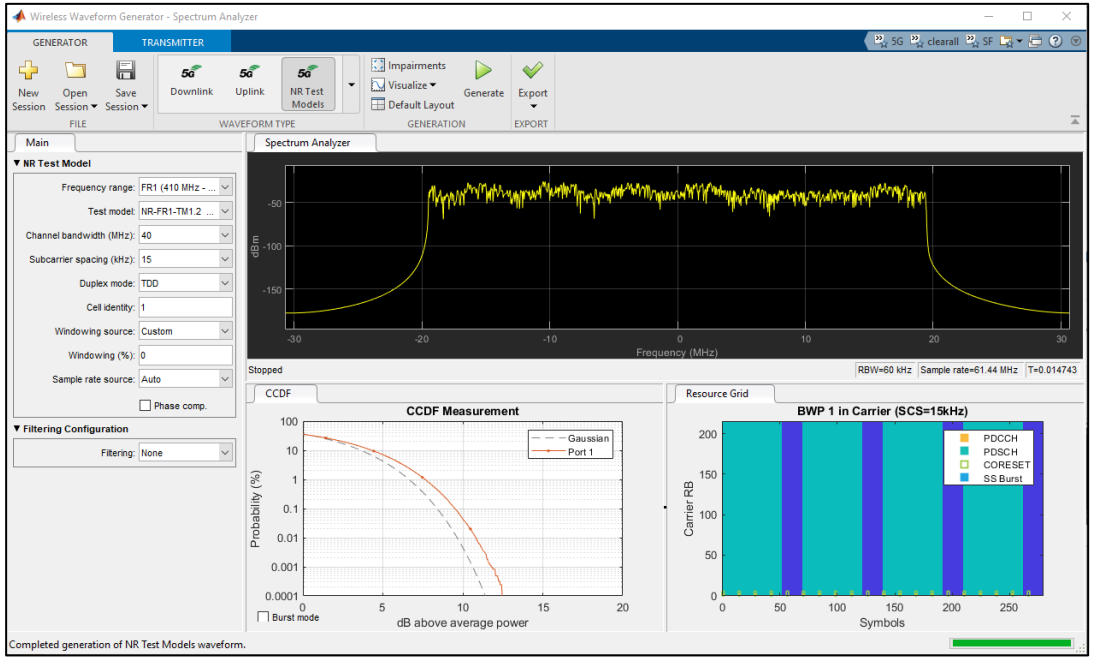
Requirement Implementation and Testing

- MathWorks offers also tools to
  - associate requirements with MATLAB code and
  - plain-text external code, such as C code,
- This is achieved by
  - creating selection-based links with the Requirements Editor or
  - creating links programmatically at the MATLAB command line.
- Verify requirements with MATLAB code by creating links to MATLAB unit tests and running the tests
- View and edit links to code

Stimulus - Wireless Waveform Generator App

Certain MATLAB add-on come with rich features for waveform generation, full system simulation, and visualization.

- Waveform Generator App from 5G Toolbox offers:
  - Off-the-shelf waveforms : NR-TMs/FRCs
  - Custom downlink & uplink waveforms
  - CCDF measurement
  - Export waveform or generate code



Verification with MATLAB and Simulink

- Early verification and prototyping capabilities
  - FPGA-in-the-loop on FPGA and SoC boards
  - Automated insertion of probes into FPGA netlists for viewing and analysis
  - Hardware-based testing through read/write access to AXI registers and DDR
- Capabilities for functional verification
  - Co-simulation with HDL simulators from Siemens® EDA, Cadence®, and AMD®
  - SystemVerilog DPI components and testbenches from MATLAB and Simulink
  - Universal Verification Methodology (UVM) testbenches and components

Three ways to do Co-Simulation in MATLAB

There are three options to achieve co-simulation with MATLAB
by using a MATLAB:

- System Object **`hdlverifier.HDLCosimulation`** or **`hdlcosim`**
- Callback function in combination with the instance **`matlabtb`**
- Callback function in combination with the instance **`matlabcp`**

MATLAB on Top - Workflow

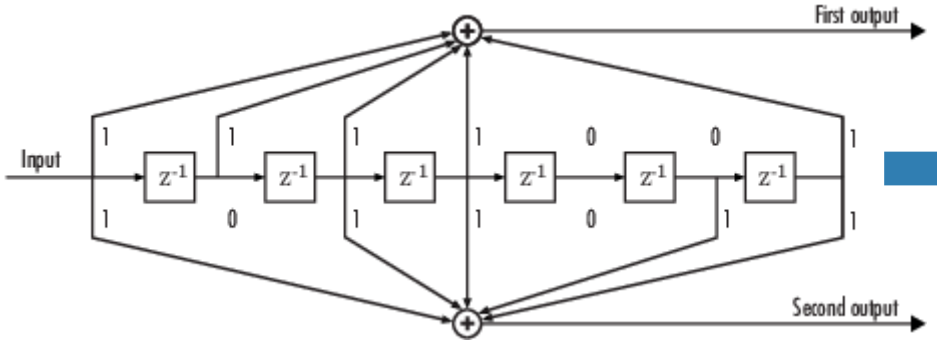
Using a Co-Simulation System Object is considered to be a "MATLAB on top – workflow":

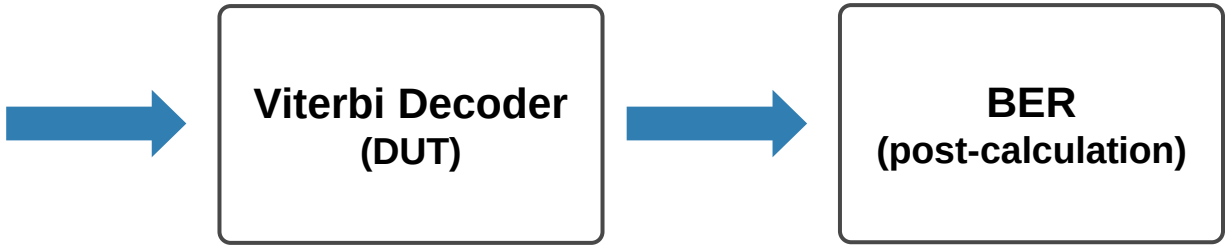
- MATLAB will be the manager
  - Creates stimuli for the device under test (e.g.: manually written RTL)
  - Controls the compilation of the HDL files
  - Launches the HDL simulator tool (GUI or batch mode)
  - Sends stimuli to the HDL simulator continually (sample or frame based)
  - Receives simulated data from the HDL simulator
  - Compares, visualizes, or postprocesses the received data

Example – Verify a Viterbi Decoder (VVD)

The Viterbi decoding algorithm is still used in space communications, voice recognition, …, and DNA sequencing. Assume:

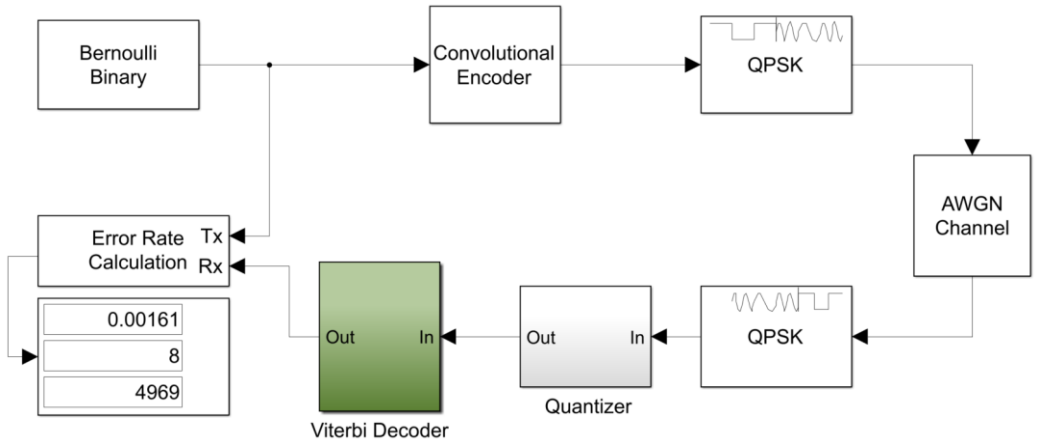
- The algorithm was developed and tested in MATLAB
- VHDL code was manually written following the same specs
- You think about reusing the MATLAB code for verification





Example VVD – Reference Algorithm

- System design
- Write code (unstructured, floating point)
- Test algorithms




```
EsNo = 3;          % Energy per symbol to noise power spectrum density ratio in dB
FrameSize = 1024;  % Number of bits in each frame
```

Convolution Encoder

```
hConEnc = comm.ConvolutionalEncoder; % TrellisStructure = poly2trellis(7, [171 133]) - default
```

QPSK Modulator

```
hMod    = comm.QPSKModulator('BitInput',true);
```

AWGN channel

```
hChan   = comm.AWGNChannel('NoiseMethod', ...
                           'Signal to noise ratio (Es/No)', ...
                           'SamplesPerSymbol',1, ...
                           'EsNo',EsNo);
```

QPSK Demodulator

```
hDemod  = comm.QPSKDemodulator('DecisionMethod','Log-likelihood ratio', ...
                               'Variance',0.5*10^(-EsNo/10),'BitOutput',true);
```

Viterbi Decoder (MATLAB simulation)

```
hDecSim = comm.ViterbiDecoder('TracebackDepth',32,...   % TrellisStructure = poly2trellis(7, [171 133]) - default
    'InputFormat','Soft','SoftInputWordLength',3);      % Soft-processing with 3 bits (0 .. 7)
```

Error Rate Calculator

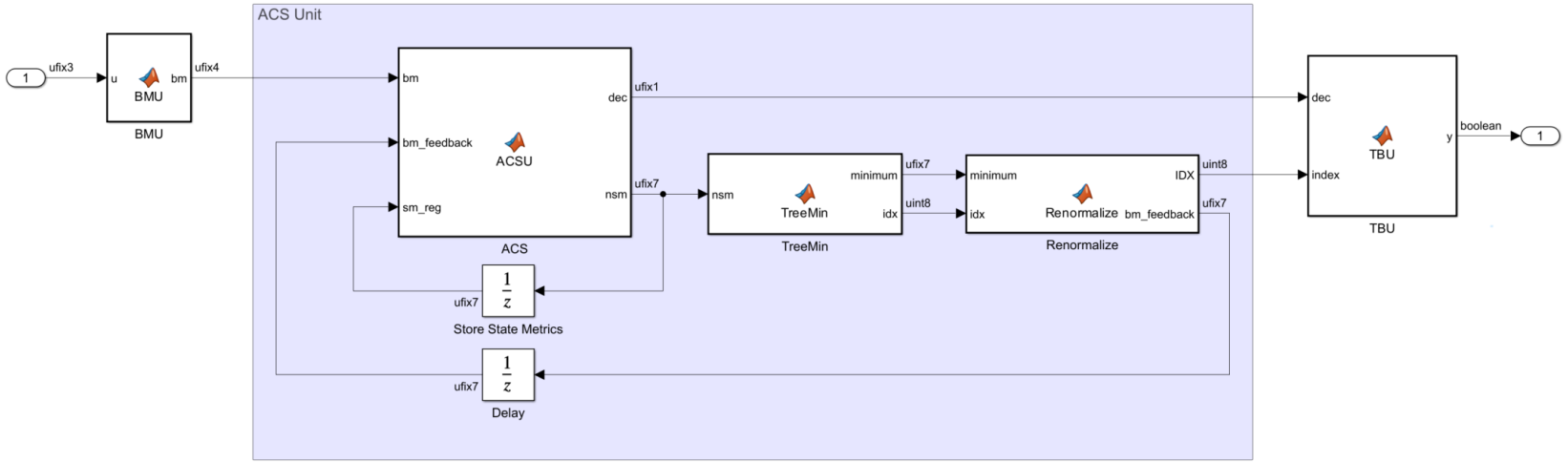
```
receiveDelay = 32; % TracebackDepth
hError  = comm.ErrorRate('ComputationDelay',100,'ReceiveDelay',receiveDelay);
```

Visualization

```
hScope = timescope("NumInputPorts",2,"LayoutDimensions",[2,1],"ChannelNames", {'Input','Received'},...
    "SampleRate",100e6,"TimeUnits","seconds","TimeSpan",10e-7,"PlotType","stairs");
hScope.ActiveDisplay = 1;
hScope.YLimits = [0 1];
hScope.ActiveDisplay = 2;
hScope.YLimits = [0 1];
hScope.Position = [27 225 800 500];

hConstDiag = comm.ConstellationDiagram("NumInputPorts",2,"ChannelNames", {'Modulated','Received'},...
    "XLimits",[-3 3],"YLimits",[-3 3],"ShowReferenceConstellation",false,'ShowLegend',true,'ColorFading',true);
hConstDiag.Position = [833 175 600 600];
```

Example VVD – Refine Reference Algorithm

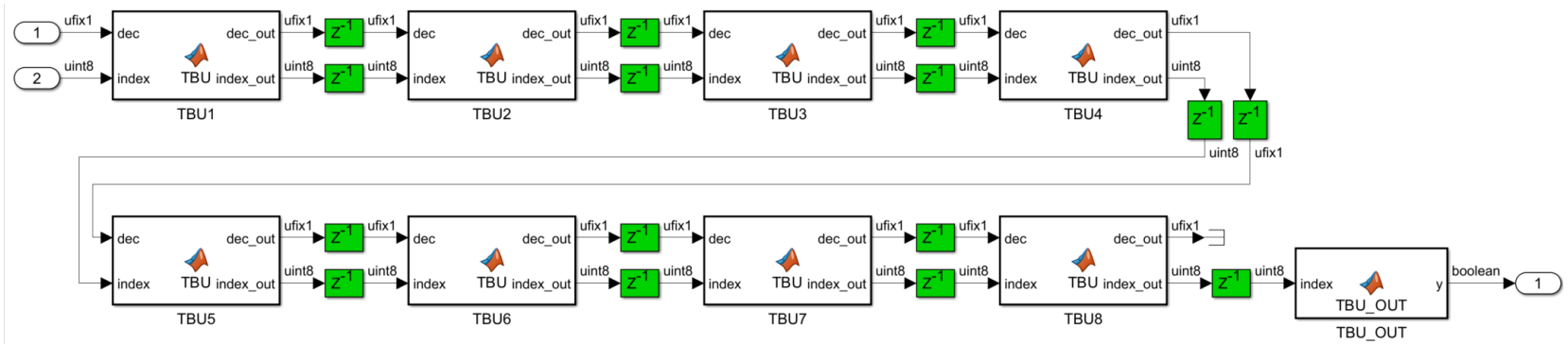
- Research on algorithm implementation
- Write code (unstructured, fixed-point)
- Test algorithm against reference




```
function y = TBU(dec, index)

% traceback storage register
persistent tb_reg;
if isempty(tb_reg)
    tb_reg = getfi(zeros(1,64), 0, 32, 0);
end

%-------------------------------------------------------------
% TBU (Trace Back Unit)
%-------------------------------------------------------------

% stores path metric decision inforamtion from PMU unit
tb_reg = bitconcat(bitsliceget(tb_reg, 31, 1), dec);

% trace back unit
for i=1:32
  index = tbunit(index, tb_reg, i);
end

% the 6th bit of the 32th trace back step is the decoder output
tband = bitget(index, 6);
y =  logical(tband);

end

function new_idx = tbunit (current_idx, tb_reg, stage)

idx = int32(current_idx) + int32(1);
tbread = bitget(tb_reg(idx), stage);

shift_idx = bitconcat(bitsliceget(current_idx, 5, 1), tbread);
new_idx = bitconcat(getfi(0, 0, 2, 0), shift_idx);

end
```

Example VVD – Streaming Algorithm

- Research on IP requirements
- Refine code (streaming, fixed-point)
- Test algorithm against unstructured fixed-point reference

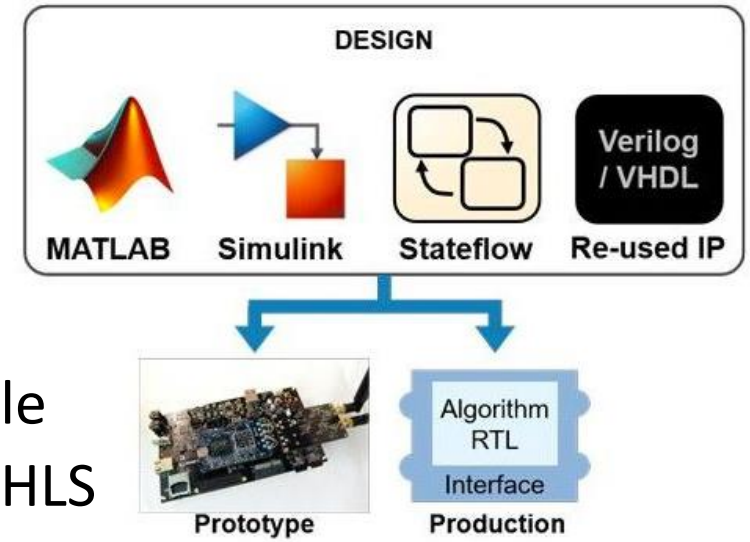



```
function [dec_out, index_out] = TBU(dec, index)
% TBU (Trace Back Unit)

% initialize 4-stage trace back storage shift register
persistent tb_reg1 tb_reg2 tb_reg3 tb_reg4
if isempty(tb_reg1)
    tb_reg1 = getfi(zeros(1, 64), 0, 1, 0);
end
if isempty(tb_reg2)
    tb_reg2 = getfi(zeros(1, 64), 0, 1, 0);
end
if isempty(tb_reg3)
    tb_reg3 = getfi(zeros(1, 64), 0, 1, 0);
end
if isempty(tb_reg4)
    tb_reg4 = getfi(zeros(1, 64), 0, 1, 0);
end

% 4-stage trace back storage shift register
dec_out = tb_reg4;
tb_reg4 = tb_reg3;
tb_reg3 = tb_reg2;
tb_reg2 = tb_reg1;
tb_reg1 = dec;

% stage one
idx = int32(index) + int32(1);
tbread = tb_reg1(idx);
shift_idx = bitconcat(bitsliceget(index, 5, 1), tbread);
index = bitconcat(getfi(0, 0, 2, 0), shift_idx);

% stage two
idx = int32(index) + int32(1);
tbread = tb_reg2(idx);
shift_idx = bitconcat(bitsliceget(index, 5, 1), tbread);
index = bitconcat(getfi(0, 0, 2, 0), shift_idx);
```

Example VVD – Integration Decision

Depending on your final target (FPGA/ASIC/SoC) and preferences you may now decide how to resume in the integration process.

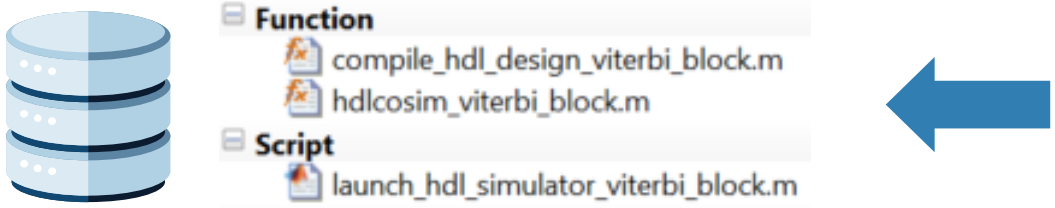
- Manually writing RTL code
- High-level synthesis
- Automatic code generation, e.g.: HDL Coder™ :
  - Synthesizable VHDL, Verilog, and SystemVerilog code
  - Fully generic, target independent, readable, and traceable
  - Synthesizable SystemC code and testbench for Stratus™ HLS
  - Support for vertical products for AI, DSP, Comms, and Vision



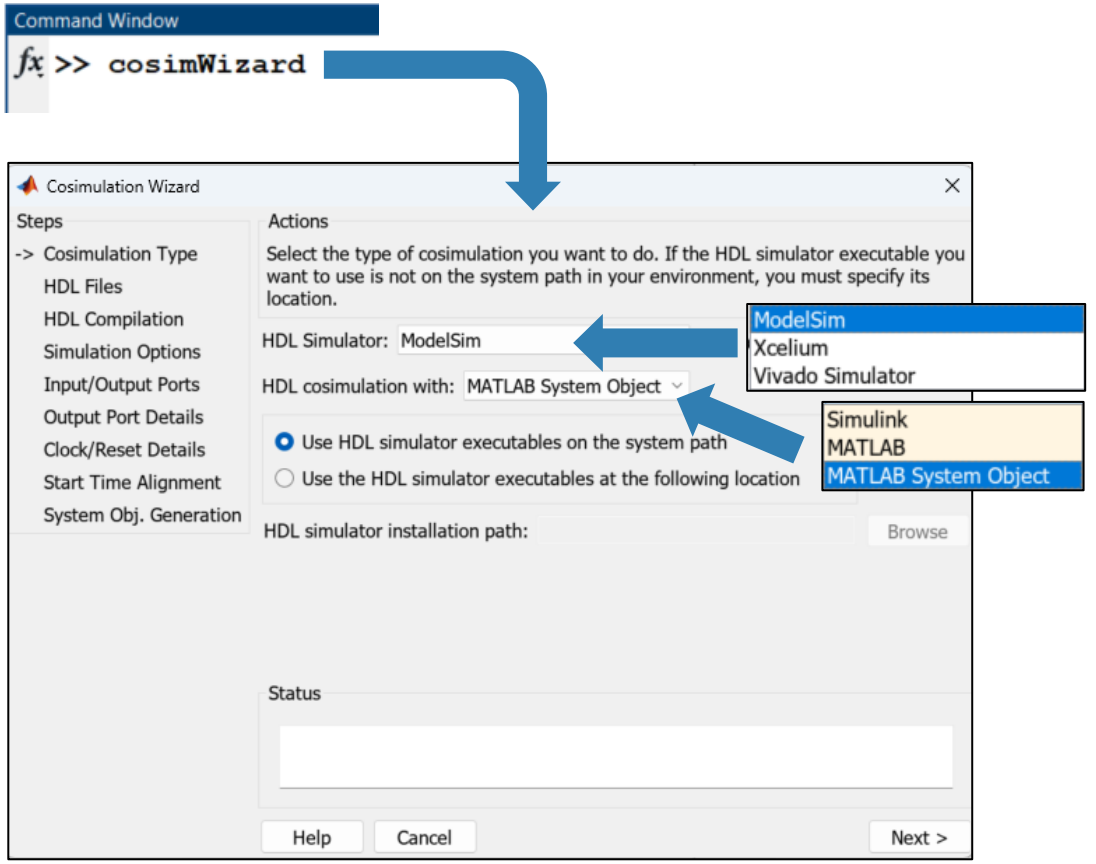
Example VVD – Prepare Co-Simulation

The Cosimulation Wizard helps you with:

- Creating a MATLAB System Object or callback function from a template
- Preparing the tcl-commands for the HDL simulator



Adapt the generated files for your needs



Example VVD – Launch HDL Simulator

- Modify the System Object code if necessary

```
hDec = hdlcosim(  'InputSignals', {'/viterbi_block/In1','/viterbi_block/In2'}, ...
                  'OutputSignals', {'/viterbi_block/Out1'}, ...     % inputs/outputs matching top-level HDL
                  ...
                  'OutputSigned', false, ...                        % how output should be casted for use in MATLAB
                  'OutputFractionLengths', 0, ...                   %   scalar -> all outputs treated the same
                  ...                                               %   vector -> outputs treated separately
                  ...
                  'TCLPostSimulationCommand', 'echo "done";', ...   % Post-simulation tcl-commands for HDL simulator
                  'PreRunTime', {10,'ns'}, ...                      % Pre run time for signal settling (reset/enable/clock)
                  'Connection', {'SharedMemory'}, ...               % Connection to the HDL simulator ('SharedMemory' or 'Socket')
                  'SampleTime', {10,'ns'});                         % Clock or sample time
switch Simulator
  case 'ModelSim'
      hDec.TCLPreSimulationCommand = ...                            % Pre-simulation tcl-commands for HDL simulator
          'force /viterbi_block/clk_enable 1 0; force /viterbi_block/clk 0 0 ns, 1 5 ns -repeat 10 ns; force /viterbi_block/reset 1 0 ns, 0 8 ns; ';
  case 'Xcelium'
      hDec.TCLPreSimulationCommand = ...
          'force :clk B"0" -after 0ns B"1" -after 5ns -repeat 10ns; force reset B"1" -after 0ns B"0" -after 8ns; force :clk_enable B"1" -after 0ns';
end
```

- Launch the HDL Simulator (that compiles the code and sets up the tool)

```
switch Simulator
  case 'ModelSim'
      vsim('tclstart',viterbi_cosimulation_tclcmds('vsimmatlabsysobj'),"runmode","Batch");
  case 'Xcelium'
      nclaunch('tclstart',viterbi_cosimulation_tclcmds('hdlsimmatlabsysobj'));
end
```

Example VVD – Run Co-Simulation

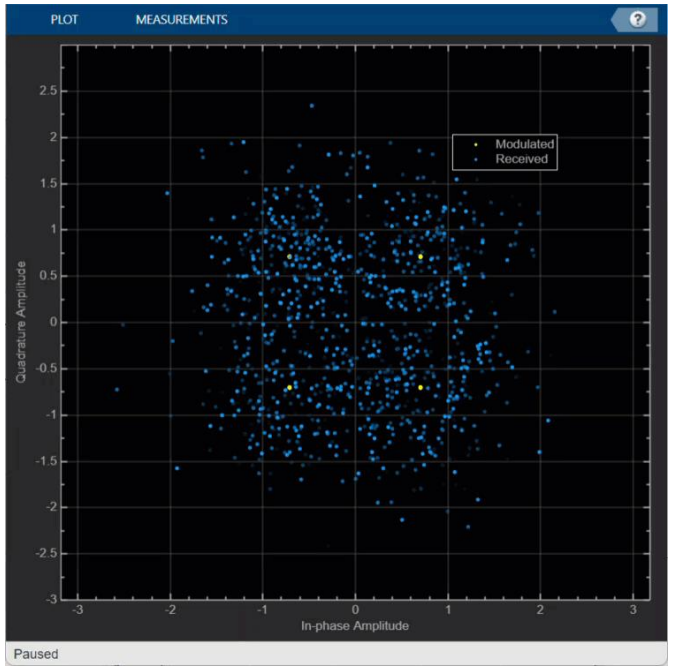
- Create a loop to run the whole transceiver chain

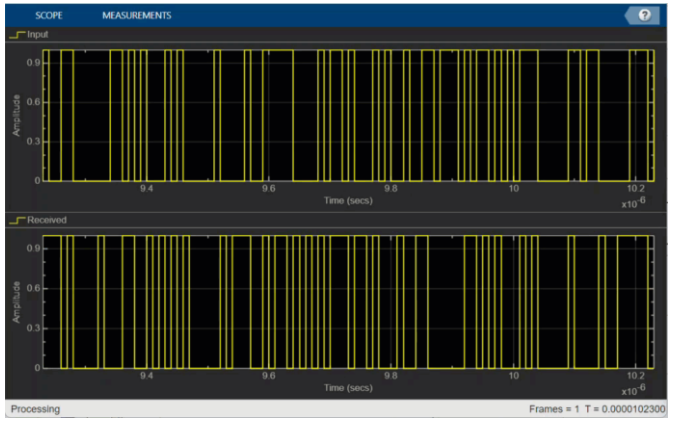
```
receiveDelay = 58;                                  % TracebackDepth + Pipeline Delay
hError  = comm.ErrorRate('ComputationDelay',100,'ReceiveDelay',receiveDelay);
hScope.reset;
rng(1)
for counter = 1:20480/FrameSize                     % Run 20480 samples with frame-processing 1024 samples/frame
    data            = randi([0 1],FrameSize,1);     % Generate a pseudo-random bit sequence initialized (seed 1)
    encodedData     = hConEnc(data);                % Convolution Encoder
    modSignal       = hMod(encodedData);            % QPSK Modulator
    receivedSignal  = hChan(modSignal);             % AWGN Channel
    demodSignalSD   = hDemod(receivedSignal);       % QPSK Demodulator
    quantizedValue  = fi(4-demodSignalSD,0,3,0);    % Quantization
    input1          = quantizedValue(1:2:2*FrameSize);  % Input Preparation
    input2          = quantizedValue(2:2:2*FrameSize);
    receivedBits    = hDec(input1, input2);         % Run Viterbi Decoder on HDL Simulator
    errors          = hError(data, double(receivedBits)); % Calculate Bit Error Rate
    hScope(data,receivedBits)                       % Show transmitted and received random sequence in a Scope
    hConstDiag(modSignal,receivedSignal)            % Show constellation of modulated transmitted and received signal
end
```

- Compare the results with the simulation reference

```
fprintf('Bit Error Rate is %d\n',errors(1))
```

```
Bit Error Rate is 3.493751e-03
```



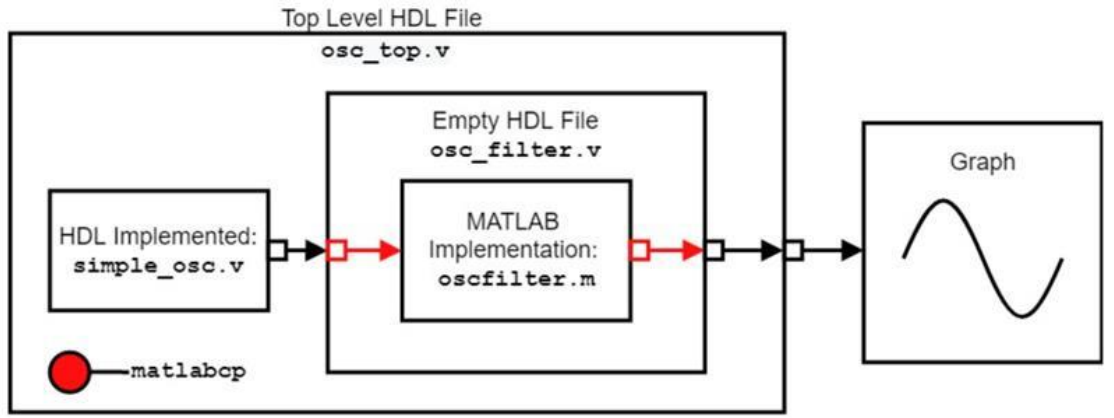
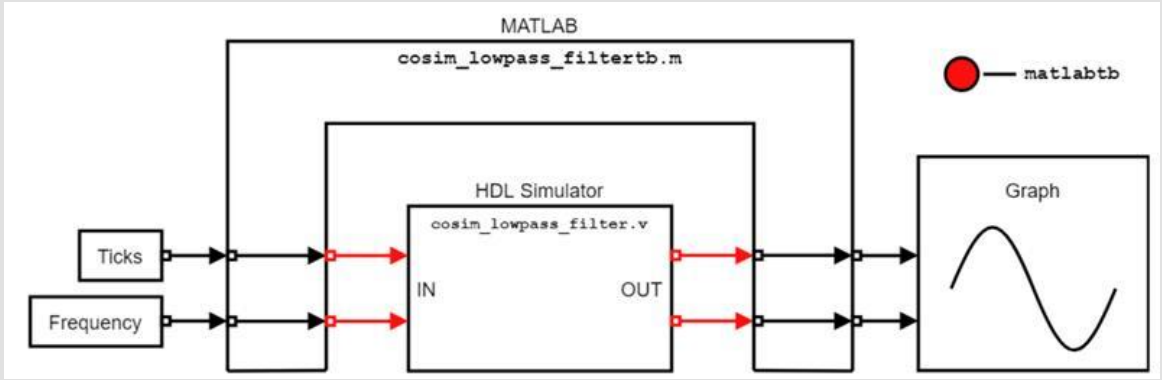


HDL Simulator on Top - Workflow

Using callback functions in MATLAB and instances like **`matlabtb`** or **`matlabcp`** made known to the HDL Simulator we call the workflow “HDL Simulator on top ”:

- MATLAB just works in the background
  - It needs to run the HDL Link MATLAB server using shared memory inter-process communication
- HDL Simulator controls the whole simulation
  - Triggers the callback function in MATLAB
  - Sends/requests data to/from MATLAB

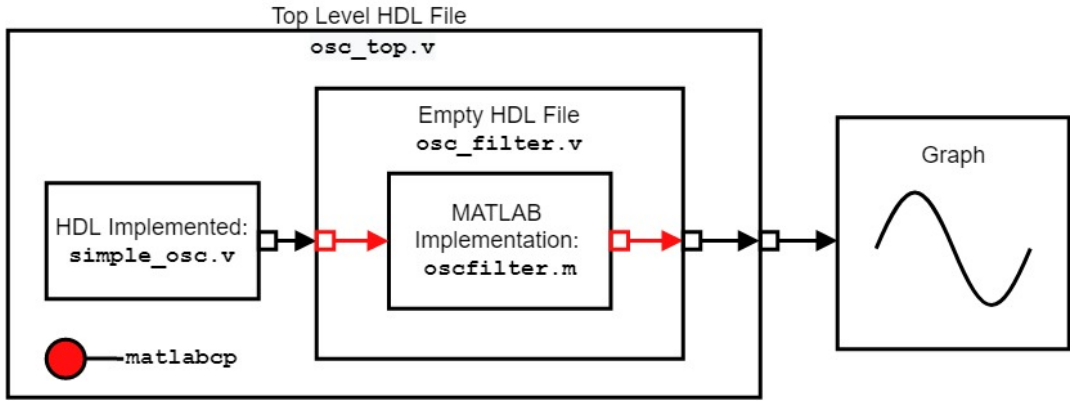
Component vs. Testbench

matlabcp	matlabtb
Test your HDL code in an HDL Simulator but a certain component which does not yet exists in HDL will be simulated through MATLAB. Testbench exists in EDA tool.	Drive stimuli from MATLAB to an HDL component and send the output back to MATLAB. Testbench exists in MATLAB but is controlled (called) from the HDL Simulator.
It sends output from HDL Simulator to MATLAB and receives input to HDL from Simulator MATLAB	Acquire input to HDL Simulator from MATLAB and send back output from HDL Simulator to MATLAB
	
Implement Filter Component of Oscillator in MATLAB	Cosimulation for Testing Filter Component Using MATLAB Test Bench

Example – MATLAB Filter Component (MFC)

The following steps are required to create and test a MATLAB component that can be used from an HDL Simulator:

- Generate a Callback Template
- Integrate your MATLAB function into the template files
- Modify the generated tcl-file
- Start the HDL Link MATLAB server
- Run existing testbench in HDL Simulator



Example (MFC) – Cosimulation Wizard

The Cosimulation Wizard helps again with:

- Creating a callback function from a template
- Preparing the tcl-commands

Example (MFC) – Callback Template

Two options for MATLAB component function writing:

- Using the HDL Instance Object (used for the template)
  - MATLAB function prototype: **`function matlabFuncName(obj)`**
  - Object field examples:
    - **`obj.portvalues, obj.tnow, obj.userdata, obj.simstatus, obj.argument, obj.portinfo`**
  - TCL command: `matlabcp hdlInstanceName -mfunc matlabFuncName -use_instance_obj`

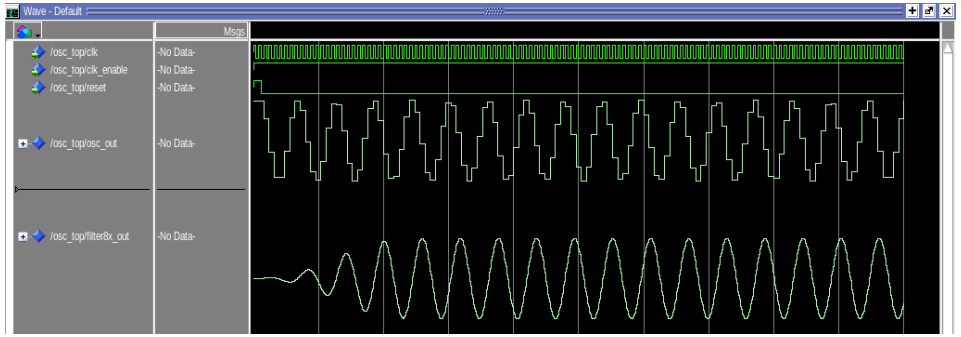
```
function oscfilter(obj)
%
% MATLAB Callback function template
% OSCFILTER HDL simulator example "Osc"
```

- Using Port Information
  - MATLAB function prototype: **`function [oport, tnext] = matlabFuncName(iport, tnow, portinfo)`**
  - TCL command: `matlabcp hdlInstanceName -mfunc matlabFuncName`

```
function [oport,tnext] = oscfilter(iport, tnow, portinfo)
%
% MATLAB Callback function template
% OSCFILTER HDL simulator example "Osc"
```

Example (MFC) – Run Co-Simulation

Run the HDL Link MATLAB server for communication between the HDL simulator and MATLAB:

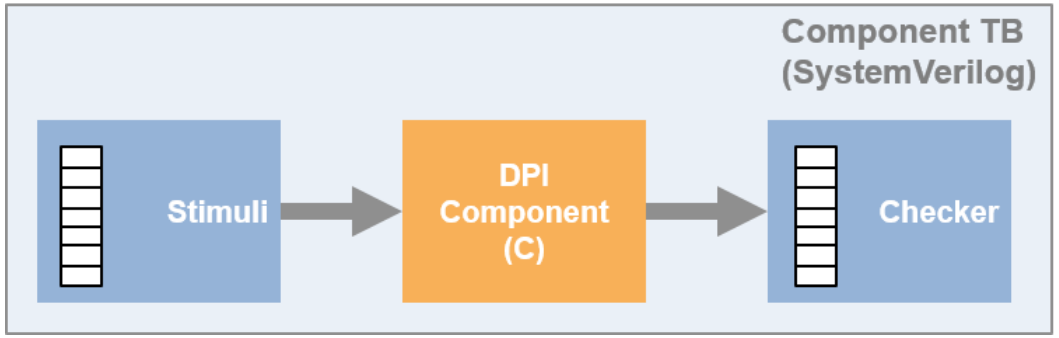


- In an open MATLAB session run:
  `>> hdldaemon`
- You can also open a MATLAB session in batch mode:
  - Windows cmd: **`matlab -nodesktop -r "hdldaemon"`**
  - Linux shell: **`xterm -e "matlab -nodesktop -r "hdldaemon"" &`**
- Start the cosimulation
  - Windows cmd: **`vsim -do qcommands_osc_w.tcl`**
  - Linux shell: **`vsim -do qcommands_osc_l.tcl`**

(here ModelSim was used)

SystemVerilog – DPI Component Generation

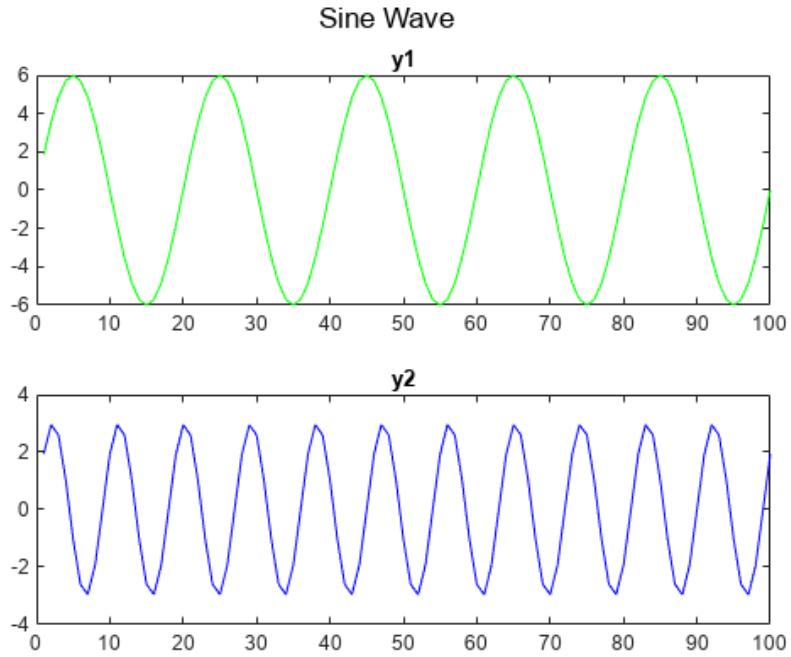
- Export a MATLAB function as a component with a direct programming interface (DPI) for use in a SystemVerilog simulation
  - MATLAB Coder is used for generating C code with a DPI wrapper
  - The DPI wrapper communicates with a SystemVerilog interface function
  - The SystemVerilog component can be used within a SystemVerilog testbench
- Get DPI component shared library for:
  - Linux (.so) or Windows (.dll)
- Templates to influence component
  - Sequential – sequential design, with registers
  - Combinational – with no registers
- Different choices for port data types



Example – Sine Wave Generator (SWG)

The following steps are required to generate a SystemVerilog DPI component:

- Prepare a MATLAB function for code generation
- Create a MATLAB testbench file (optional)
- Define and set a configuration object (optional)
- Generate the component using **`dpigen`**
- Verify your component with a generated testbench (optional)



Example (SWG) – Prepare Function

Before generating the component from MATLAB code ensure to:

- Initialize variables and define them on all execution paths
- Define static variables as persistent (states, registers)
- Avoid dynamic memory allocation for efficiency
  - Rather use fixed-size arrays and variable-size arrays (size < threshold)
- To identify issues:
  - Use **`%#codegen`** pragma to instruct the MATLAB Code Analyzer to help with finding and fixing violations
  - Run the Code Screener

```
1  function y = sineWaveGen(amp,freq)
2  %#codegen
3
4      for i = 1:100
5          y(i) =  amp*sin(i*2*pi/(freq));
6      end
7  end
```

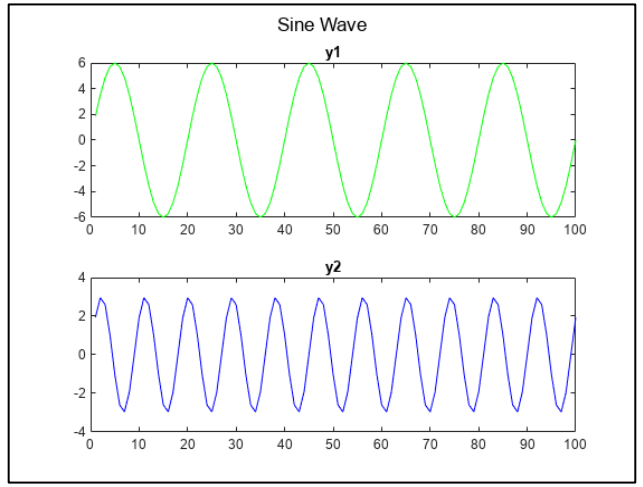
```
1  function y = sineWaveGen(amp,freq)
2  %#codegen
3      y = zeros(1,100);
4      for i = 1:100
5          y(i) =  amp*sin(i*2*pi/(freq));
6      end
7  end
```

Example (SWG) – MATLAB Testbench

A MATLAB testbench is an optional file. However, it helps with:

- Defining port
  - data types,
  - sizes, and
  - complexity
- Testing and debugging the MATLAB function
- Verifying the generated component

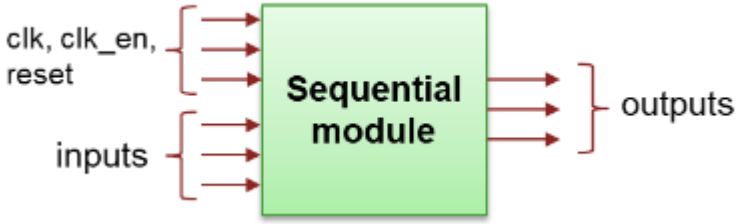
```
function sineWaveGen_tb
    in1_amp = 6;
    in1_freq = 20;
    in2_amp = 3;
    in2_freq = 9;
    y1 = sineWaveGen(in1_amp,in1_freq);
    y2 = sineWaveGen(in2_amp,in2_freq);
    figure();
    subplot(2,1,1)
    plot(y1,'g');
    title('y1');
    subplot(2,1,2);
    plot(y2,'b');
    title('y2');
    sgtitle('Sine Wave');
end
```




```
function y = sineWaveGen(amp,freq)
%#codegen
    y = zeros(1,100);
    for i = 1:100
        y(i) =  amp*sin(i*2*pi/(freq));
    end
end
```

Example (SWG) – Configuration

- Create a **`svdpiConfiguration`** object
  - The default configuration points to the templates for a sequential module (component and testbench template)
  - Change component and testbench name, and SystemVerilog port types



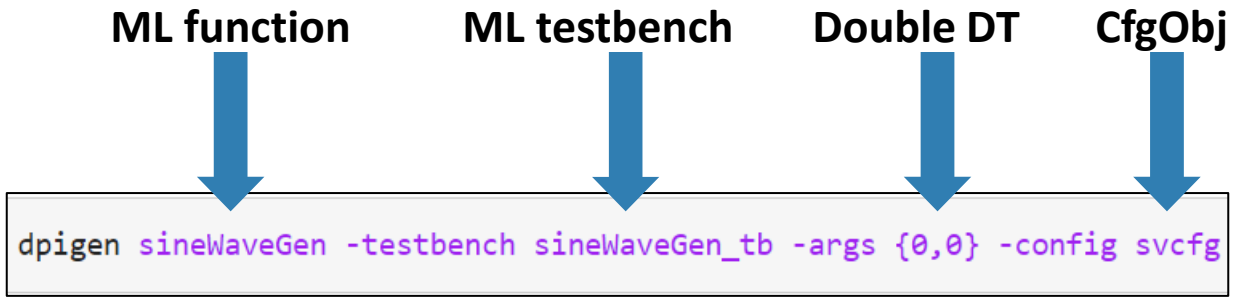

```
svcfg=svdpiConfiguration
```

```
svcfg =
  svdpiConfiguration with properties:

              ComponentKind: 'sequential-module'
          CoderConfiguration: [1x1 coder.EmbeddedCodeConfig]
           ComponentTypeName: ''
           TestBenchTypeName: ''
          TemplateDictionary: []
                  PortGroups: []
    ComponentTemplateFiles: {'/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/hdlverifier/dpigenerator/rtw/SequentialModuleML.svt'}
    TestBenchTemplateFiles: {'/mathworks/devel/bat/Bdoc23a/build/matlab/toolbox/hdlverifier/dpigenerator/rtw/SequentialTestBenchML.svt'}
```

Example (SWG) – Component Generation

Use the **`dpigen`** function to generate the component artefacts. You can:

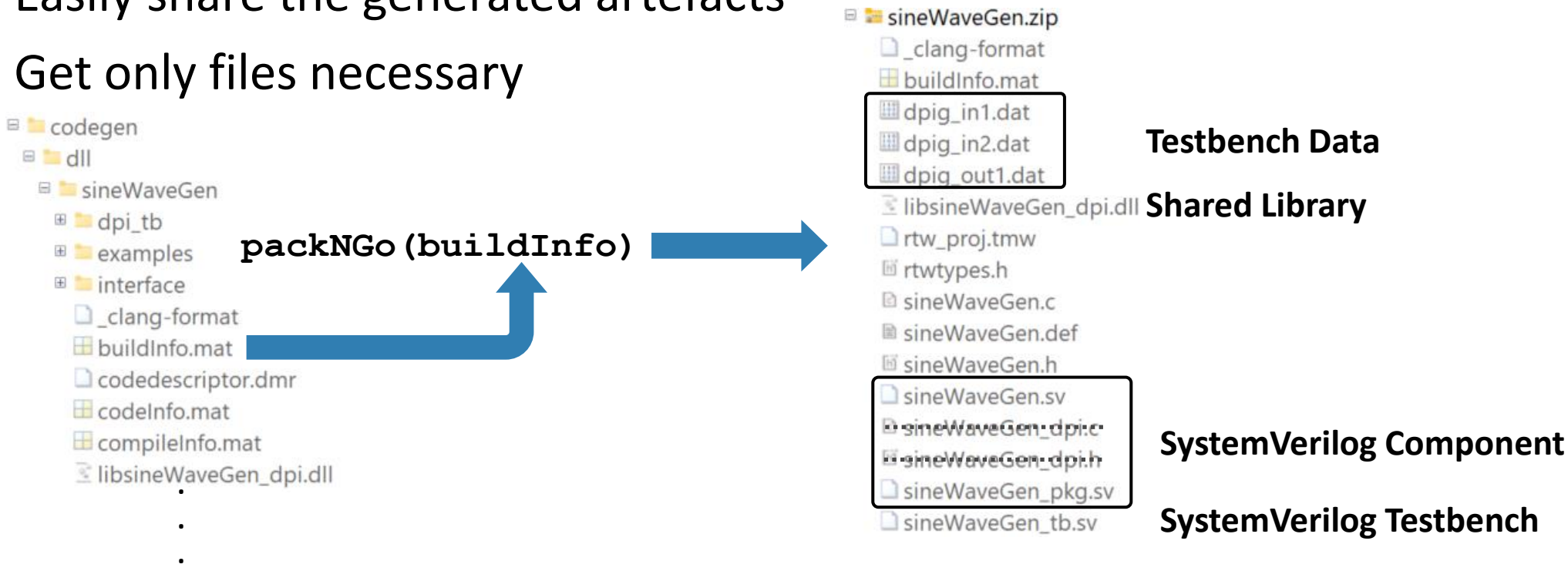
- Define the MATLAB function and optionally the MATLAB testbench
- Provide sample data for input arguments
- Set a configuration object or separately:
  - Custom include files
  - Compiler options
  - SystemVerilog Port data types
  - Component templates
- Generate and launch a code generation report



Example (SWG) – Package Required Files

With the **`packNGo`** function you can zip all the required Files.

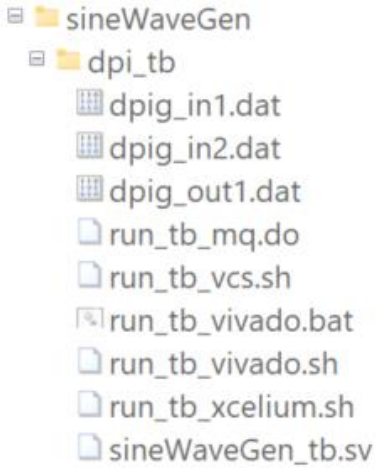
- Easily share the generated artefacts
- Get only files necessary



Example (SWG) – Verify Component

When generating the component including testbench you can directly verify it with an HDL simulator

- A *_tb.sv file is generated together with data files for stimuli and expected output
- tcl-scripts provided for compiling and running the testbench in:
  - ModelSim™/QuestaSim™
  - Cadence® Xcelium™
  - Synopsys® VCS™
  - AMD® Vivado™




```
C:\WINDOWS\SYSTEM32\cmd
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

C:\codegen\dll\sineWaveGen\dpi_tb>vsim < run_tb_mq.do
```

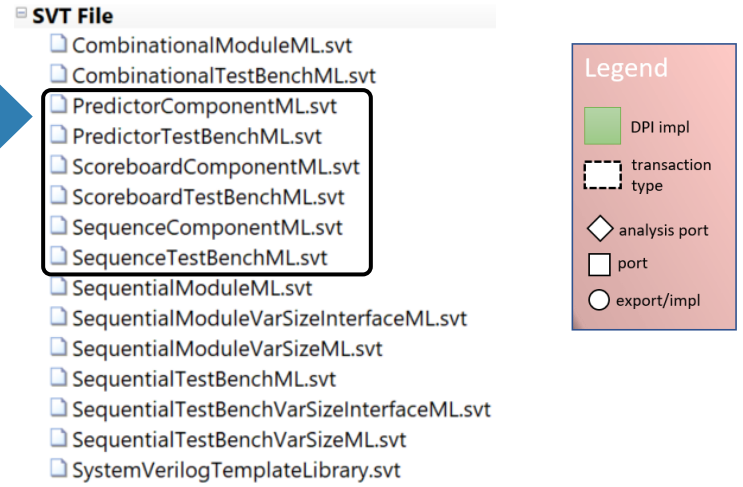
.
.
.

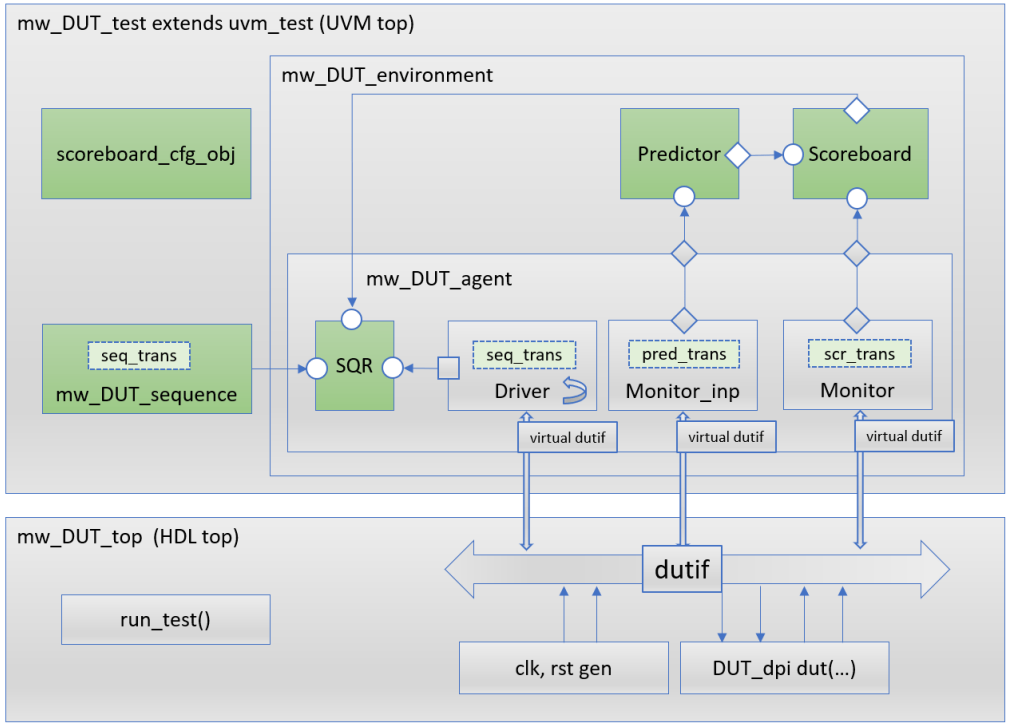
```
# ***************TEST COMPLETED (PASSED)**************
# ** Note: $finish    : ./sineWaveGen_tb.sv(91)
#    Time: 52 ns  Iteration: 0  Instance: /sineWaveGen_tb
# End time: 10:48:21 on Sep 01,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

UVM Component Generation from MATLAB

In addition of generating a general SystemVerilog DPI component, further template files enable also the generation of UVM components.

- Can integrate into a full UVM testbench
- Templates for component and testbench:
  - Predictor
  - Sequence
  - Scoreboard

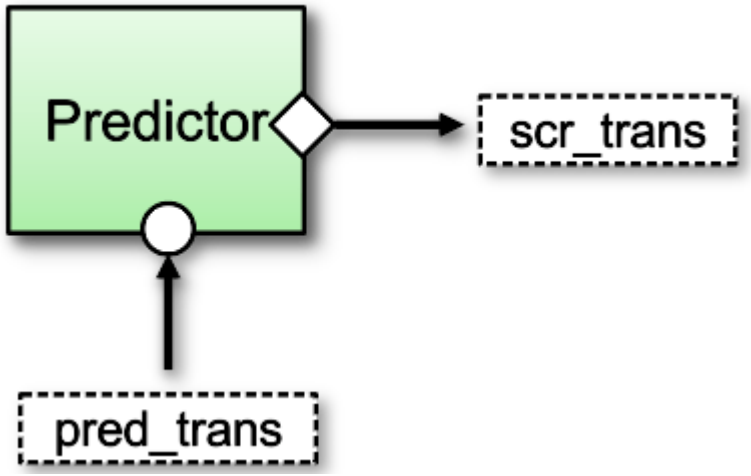




UVM – Predictor component

The predictor template generates a UVM predictor module that has:

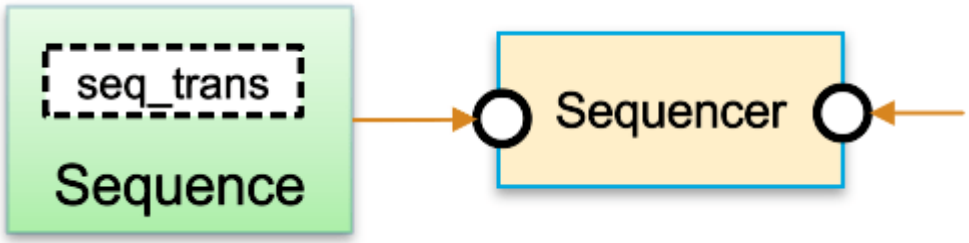
- an export that inputs a predictor transaction, and
- an analysis port that outputs a scoreboard transaction.
- The predictor template includes these variables:
  - `ComponentTypeName`
  - `TestBenchTypeName`
  - `ComponentPackageTypeName`
  - `InputTransTypeName`
  - `OutputTransTypeName`



UVM – Sequence component

The sequence template generates a UVM sequence module.

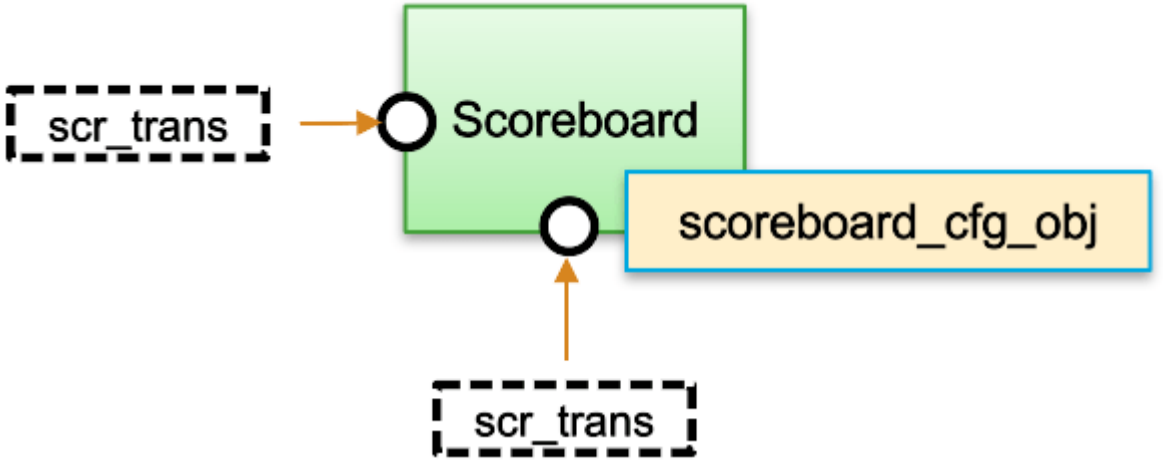
- It includes these variables:
  - `ComponentTypeName`
  - `TestBenchTypeName`
  - `ComponentPackageTypeName`
  - `SequenceTransTypeName`
  - `SequencerTypeName`
  - `SequenceCount`
  - `SequenceFlushCount`



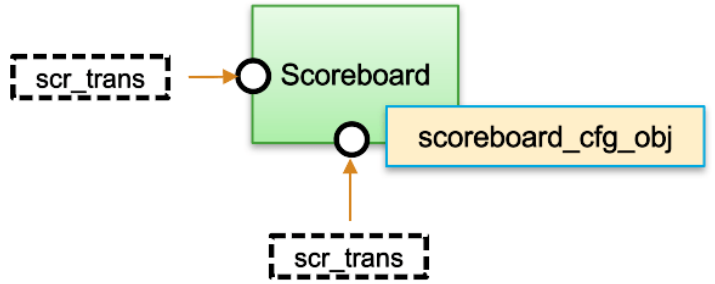
UVM – Scoreboard component

The scoreboard template generates a UVM scoreboard module that has two exports that input a scoreboard transaction.

- The scoreboard template includes these variables:
  - `ComponentTypeName`
  - `TestBenchTypeName`
  - `ComponentPackageTypeName`
  - `InputTransTypeName`
  - `OutputTransTypeName`
  - `ConfigObjTypeName`
- Map ports to port groups by using the addPortGroup object function.

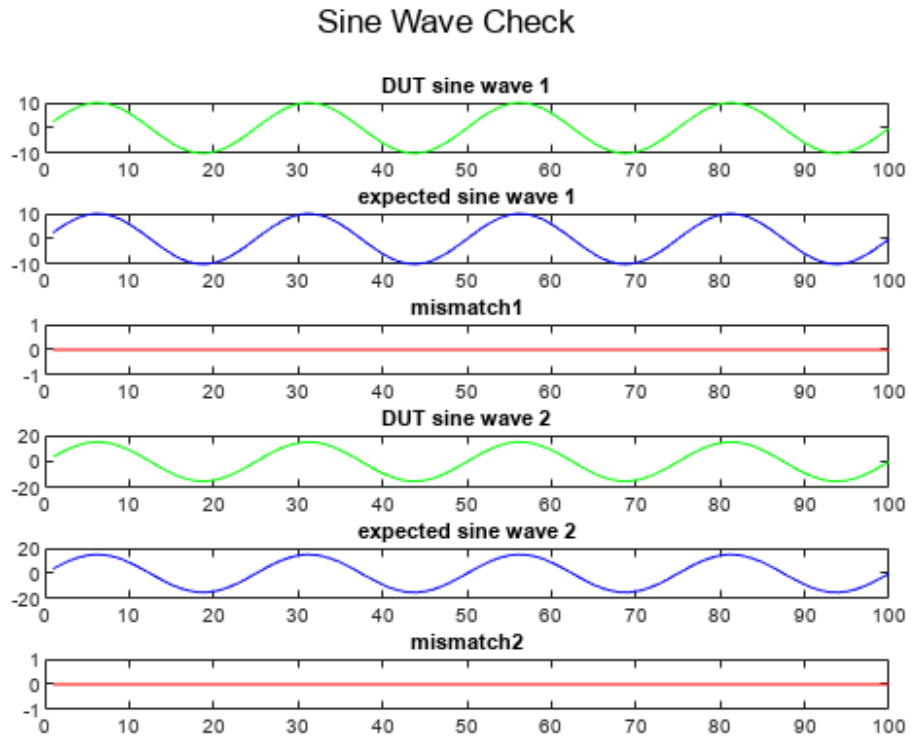


Example – Sine Wave Check (SWC)



The following steps are required to generate a UVM component:

- Prepare a MATLAB scoreboard function
- Create a MATLAB testbench file (optional)
- Define a configuration object (required)
- Provide UVM settings to the cfgobj
- Specify which port belongs to which port group
- Generate the component using **`dpigen`**
- Verify your component with a generated testbench (optional)



Example (SWC) – Scoreboard Function

Before generating the component from MATLAB code do the preparation steps necessary for SystemVerilog DPI components.

- Write a scoreboard function that compares
  - output of the DUT with
  - output of a golden reference
- Inputs are from Monitor and Predictor. Configurations are inputs, too.

```
1  function [mismatch1,mismatch2] = sineWaveCheck(in1FromMon,in2FromMon,inFromPred,amp1,amp2)
2  %inFromMon: sine wave from DUT
3  %inFromPred: normalized sine wave from golden reference (predictor)
4  %amp: amplitude of the sine wave from DUT
5  mismatch1 = (in1FromMon ~= inFromPred*amp1);
6  mismatch2=  (in2FromMon ~= inFromPred*amp2);
7  end
```

Example (SWC) – Configuration for UVM

- Create a **`svdpiConfiguration`** object and then change:
  - Change the kind of the component, and
  - Optionally the component name

```
clear svcfg;
svcfg=svdpiConfiguration;
svcfg.ComponentKind = 'uvm-scoreboard'
```

- Check and optionally change through the Template Dictionary
  - Name of component package file
  - Name of input/output transition
  - Name of configuration object

```
1  %</* This template requires the following vars and MUST be defined in the svdpiConfiguration.TemplateDictionary
2       PortGroups: PREDICTOR_INPUTS, MONITOR_INPUTS, CONFIG_OBJECT_INPUTS
3  */>
4  %</* This template requires the following vars. They can be overriden in the svdpiConfiguration.TemplateDictionary */>
5  %<BEGIN_LOCAL_DICTIONARY>
6  ComponentPackageTypeName=%<ComponentTypeName>_pkg
7  InputTransTypeName=scoreboard_input_trans
8  OutputTransTypeName=scoreboard_output_trans
9  ConfigObjectTypeName=scoreboard_cfgobj
10 %<END_LOCAL_DICTIONARY>
```

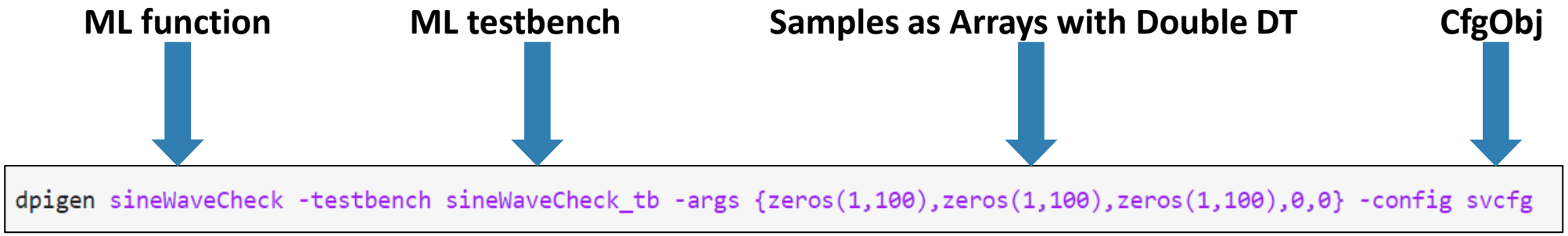
```
svcfg.TemplateDictionary = {'InputTransTypeName','sineWaveTrans'};
```

- Specify which port belongs to which port group

```
addPortGroup(svcfg,'MONITOR_INPUTS',{'in1FromMon','in2FromMon'});
addPortGroup(svcfg,'PREDICTOR_INPUTS',{'inFromPred'});
addPortGroup(svcfg,'CONFIG_OBJECT_INPUTS',{'amp1','amp2'});
```

Example (SWC) – Generate and Verify

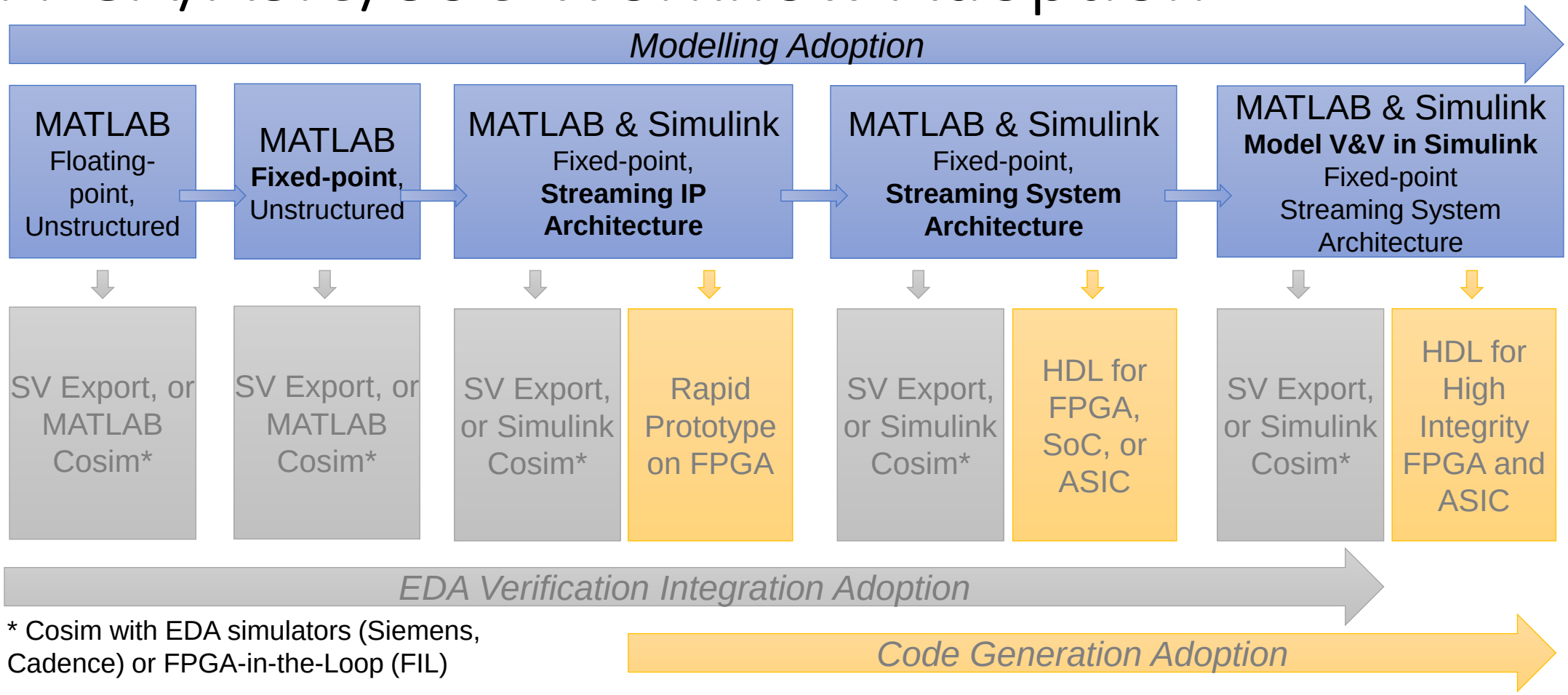
Use the **`dpigen`** function to generate the component artefacts.



Test the UVM component using the generated SystemVerilog testbench and tcl-file.

```
# **************TEST COMPLETED (PASSED)**************
# ** Note: $finish    : ./sineWaveGen_tb.sv(91)
#    Time: 52 ns  Iteration: 0  Instance: /sineWaveGen_tb
# End time: 10:48:21 on Sep 01,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

FPGA/ASIC/SoC Workflow Adoption



* Cosim with EDA simulators (Siemens, Cadence) or FPGA-in-the-Loop (FIL)

Questions