Direct MATLAB® to AMD Vitis™ HLS Workflow

Transforming Algorithm to Hardware Implementation

MathWorks®

AMD
together we advance_

Agenda

1. Introduction to MATLAB to AMD Vitis HLS Design Flow
2. Technical Details of MATLAB to Vitis HLS Design Flow
3. Coding Guidelines and Considerations
4. Examples and Benchmarks
5. Next Steps

Introduction to MATLAB to AMD Vitis HLS Design Flow

A Partnership Between MathWorks and AMD

MATLAB for Algorithm Development

MATLAB offers toolboxes ideal for architectural exploration and algorithm development

Challenges in Hardware Implementation

High cost and time associated with manual RTL coding

Models can be out of sync between RTL and MATLAB – Iterative process

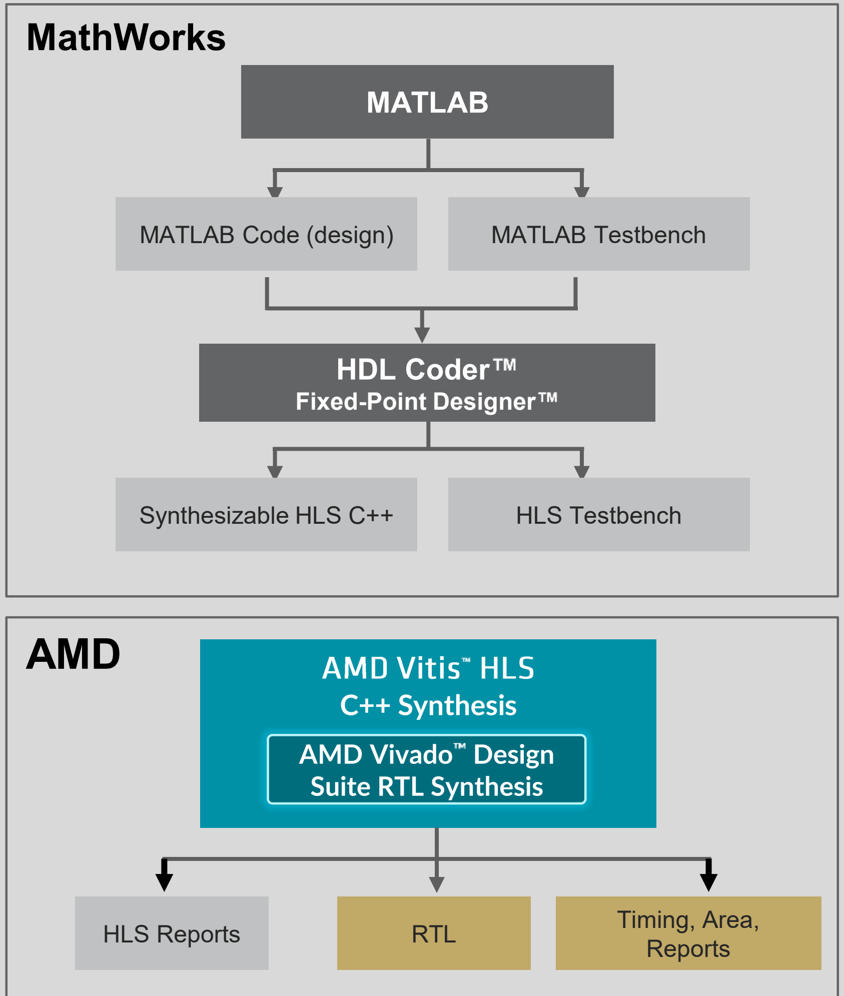
Solution: AMD & MathWorks Partnership

AMD and MathWorks partnership now provides a flow that can take MATLAB algorithmic description to AMD Adaptive SoC-optimized RTL

MATLAB to AMD Vitis HLS Workflow

Direct path from MATLAB to AMD Vitis HLS…

- MATLAB **code** ⇒ Vitis HLS friendly **C++**
- Leverages **Fixed-Point Designer** to optimize floating and fixed-point algorithms
- Translates MATLAB testbench to C++ testbench for Vitis HLS
- Launches Vitis HLS csim and cosim
- Creates a Vitis HLS project

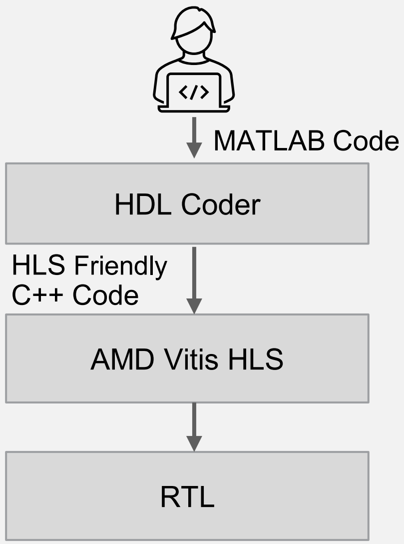


AMD Vitis™ HLS project created by MATLAB

MATLAB to AMD Vitis HLS Workflow – Usage Scenarios

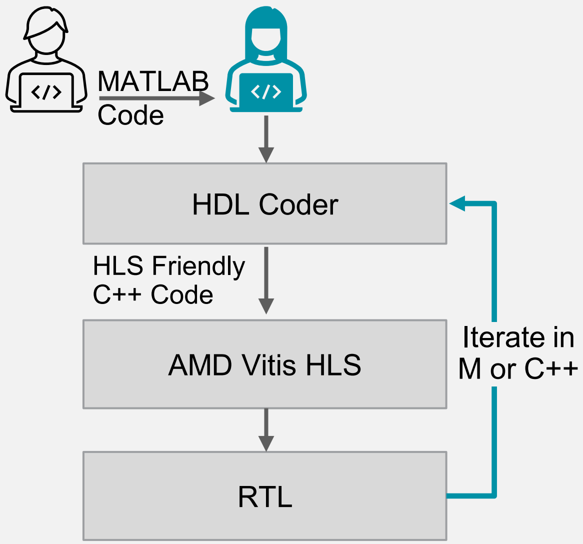
Scenario 1

Algorithm developer alone works until RTL generation



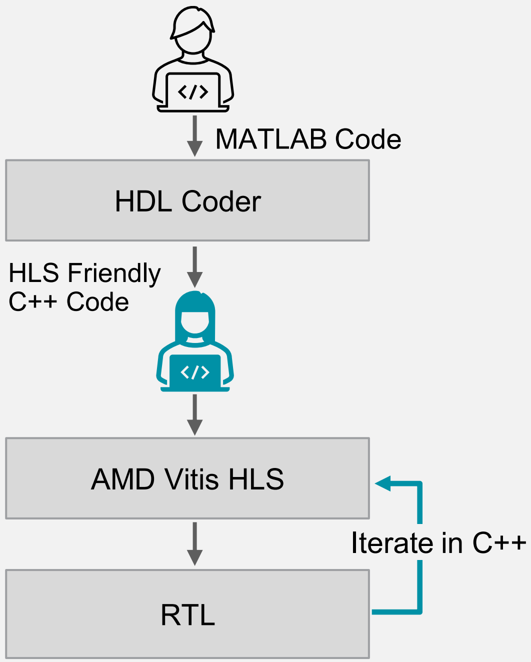
Scenario 2

Algorithm developer gives code to hardware developer



Scenario 3

Algorithm developer gives C++ code to hardware developer

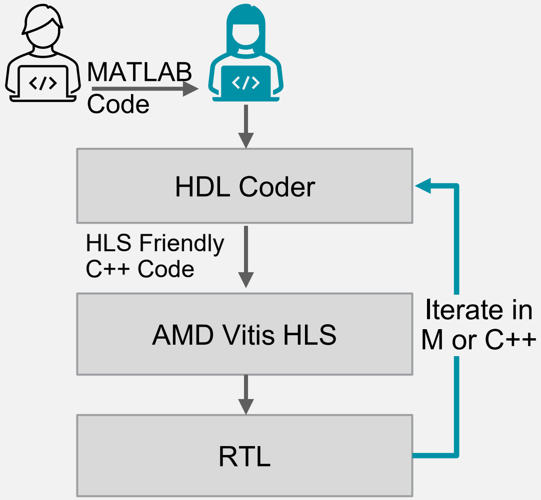


MATLAB to AMD Vitis HLS Workflow – Use Models

Involving hardware developers is recommended…

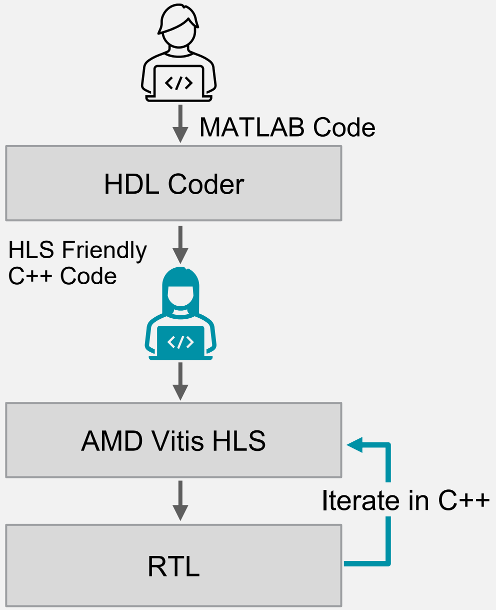
Scenario 2

Algorithm developer gives code to hardware developer



Scenario 3

Algorithm developer gives C++ code to hardware developer



Benefits of MATLAB to AMD Vitis HLS Design Flow

Algorithm Developer

- Stay within familiar MATLAB programming environment
- Convert verified MATLAB algorithms to RTL without HDL programming
- Get an early estimate of FPGA resource and power requirements

Hardware Developer

- Accommodate last minute changes in hardware implementation by leveraging automated flow - Eliminate errors
- Faster simulation using MATLAB / C++ code
- Easy to retarget to different architectures – Source code is technology independent

Technical Details of MATLAB to AMD Vitis HLS Workflow

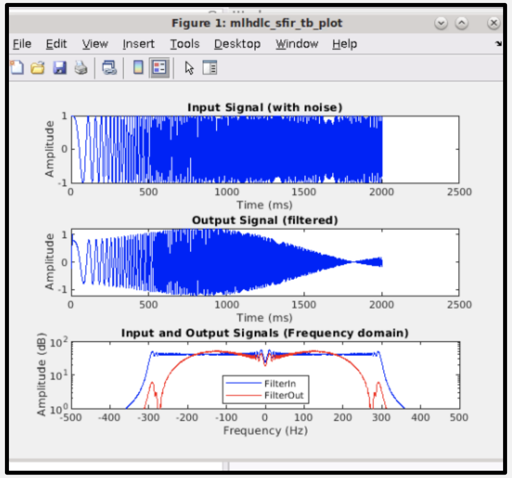
Technical Details of MATLAB to AMD Vitis HLS Workflow

MATLAB	**Algorithm Design**	Design in MATLAB using MATLAB syntax and functions, best practices
	Fixed-Point Conversion	Convert floating-point code to fixed-point code, and optimize fixed-point data types
	Code Generation	Generate AMD Vitis HLS C++ code and C++ testbench. Supports insertion of Vitis HLS pragmas in MATLAB code
AMD Vitis	**User-Driven Optimization**	Improvements through resource sharing, streaming, pipelining, array partitioning, and RAM mapping in Vitis HLS
	Verification	Supports simulation and verification of generated Vitis HLS C++ and RTL in Vitis HLS
	Deployment	Supports generation of synthesis scripts and deploy Vitis HLS generated HDL code to AMD adaptive SoCs and FPGAs

Standard Algorithm Development Flow Using MATLAB

```
%#codegen
function [y_out, delayed_xout] = mlhdlc_sfir(x_in,h_in1,h_in2,h_in3,h_in4)
% Symmetric FIR Filter

% declare and initialize the delay registers
persistent ud1 ud2 ud3 ud4 ud5 ud6 ud7 ud8;
if isempty(ud1)
    ud1 = 0; ud2 = 0; ud3 = 0; ud4 = 0; ud5 = 0; ud6 = 0; ud7 = 0; ud8 = 0;
end

% access the previous value of states/registers
a1 = ud1 + ud8; a2 = ud2 + ud7;
a3 = ud3 + ud6; a4 = ud4 + ud5;

% multiplier chain
m1 = h_in1 * a1; m2 = h_in2 * a2;
m3 = h_in3 * a3; m4 = h_in4 * a4;

% adder chain
a5 = m1 + m2; a6 = m3 + m4;

% filtered output
y_out = a5 + a6;

% delayout input signal
delayed_xout = ud8;

% update the delay line
ud8 = ud7;
ud7 = ud6;
ud6 = ud5;
ud5 = ud4;
ud4 = ud3;
ud3 = ud2;
ud2 = ud1;
ud1 = x_in;
end
```

```
dt = 0.001;
N = T/dt+1;
sample_time = 0:dt:T;

df = 1/dt;
sample_freq = linspace(-1/2,1/2,N).*df;

% input signal with noise
x_in = cos(2.*pi.*(sample_time).*(1+(sample_time).*75)).';

% filter coefficients
h1 = -0.1339; h2 = -0.0838; h3 = 0.2026; h4 = 0.4064;

len = length(x_in);
y_out = zeros(1,len);
x_out = zeros(1,len);

for ii=1:len
    data = x_in(ii);
    % call to the design 'mlhdlc_sfir' that is targeted for hardware
    [y_out(ii), x_out(ii)] = mlhdlc_sfir(data, h1, h2, h3, h4);
end
```

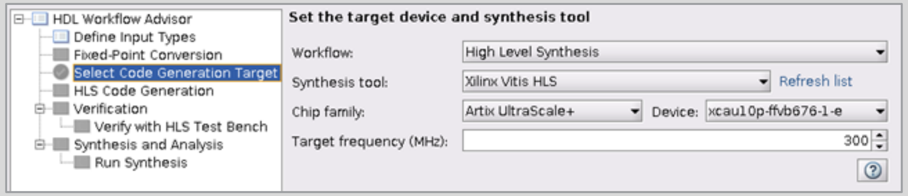


MATLAB Design → → **Simulate Design**

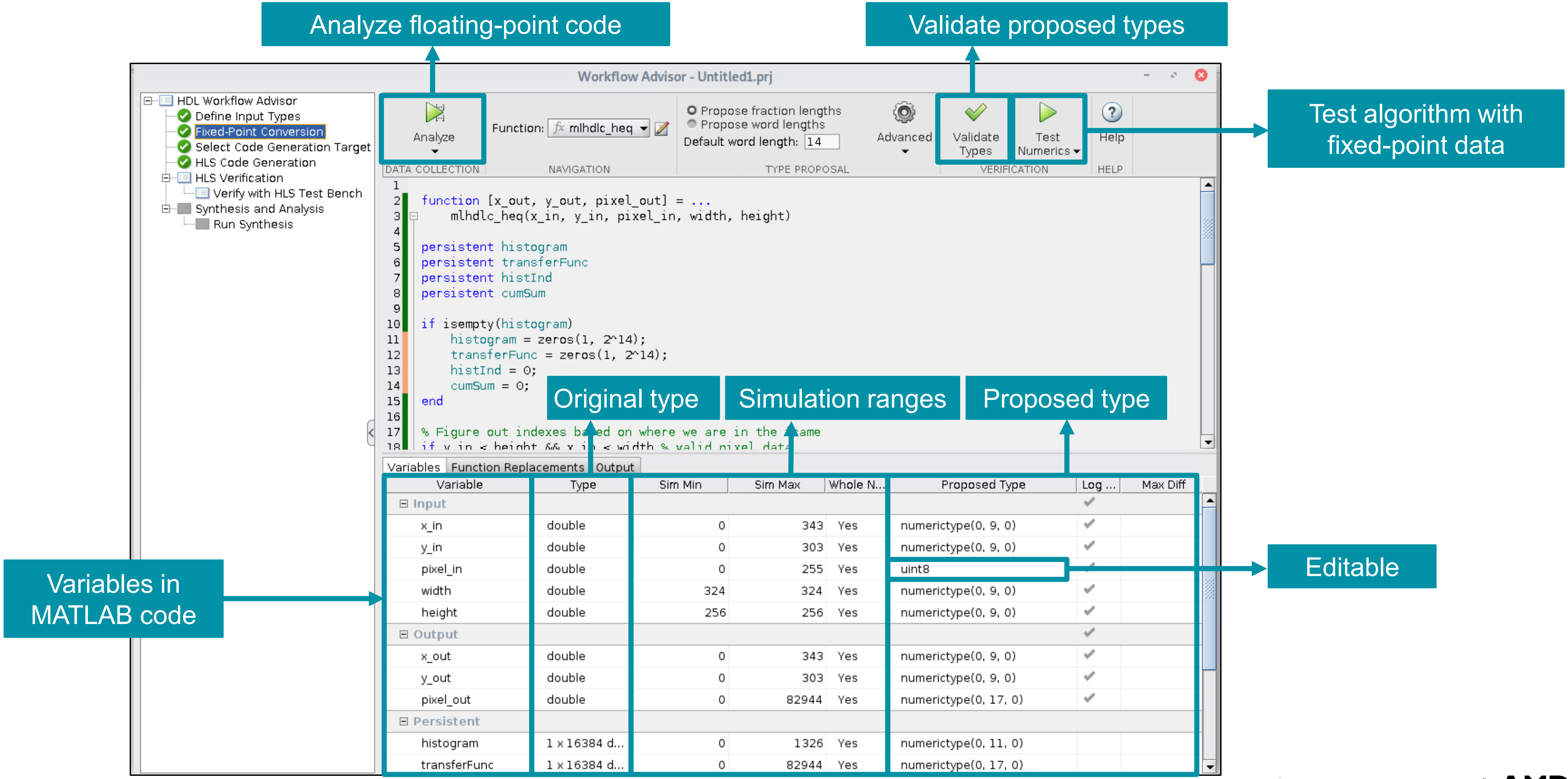
AMD
together we advance_

Code Generation Tool Flow Setup

- Create an HDL project and add the MATLAB design and testbench files
- Configure the code generation flow for MATLAB to HLS using the HDL Workflow Advisor
  - Float to fixed-point conversion
  - Target device
  - Clock frequency
- Flow automatically identifies fixed-point ranges and MATLAB datatypes through simulation-based analysis

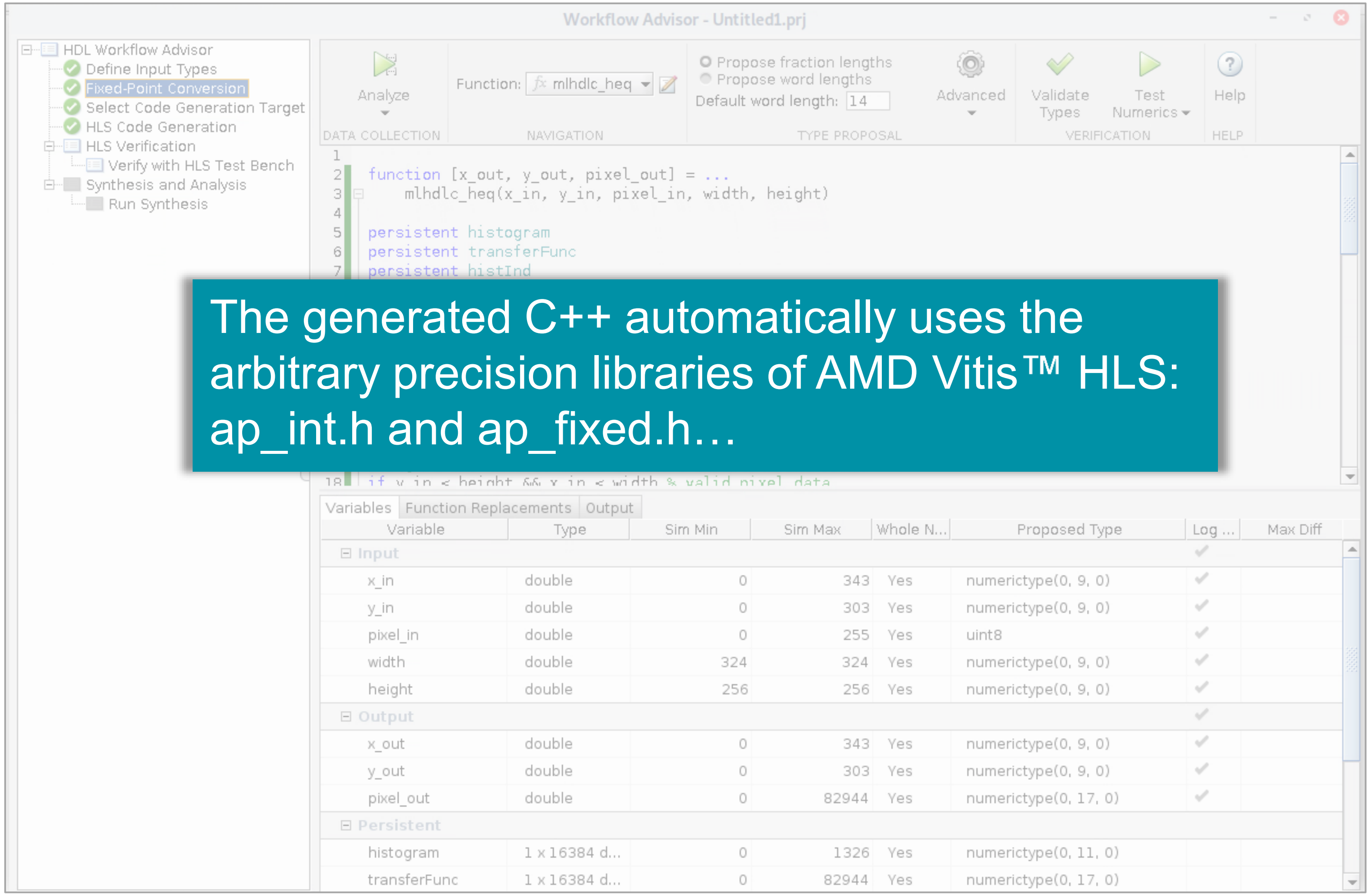


Fixed-Point Designer for AMD Vitis HLS Fixed-Point



MathWorks® | AMD together we advance_

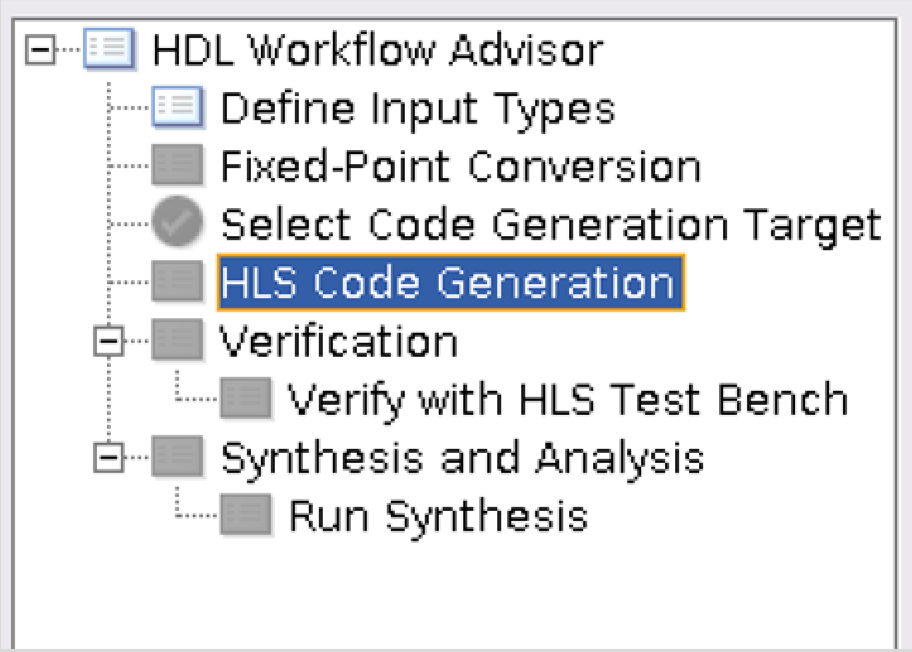
Fixed-Point Designer for AMD Vitis HLS Fixed-Point



The generated C++ automatically uses the arbitrary precision libraries of AMD Vitis™ HLS: ap_int.h and ap_fixed.h…

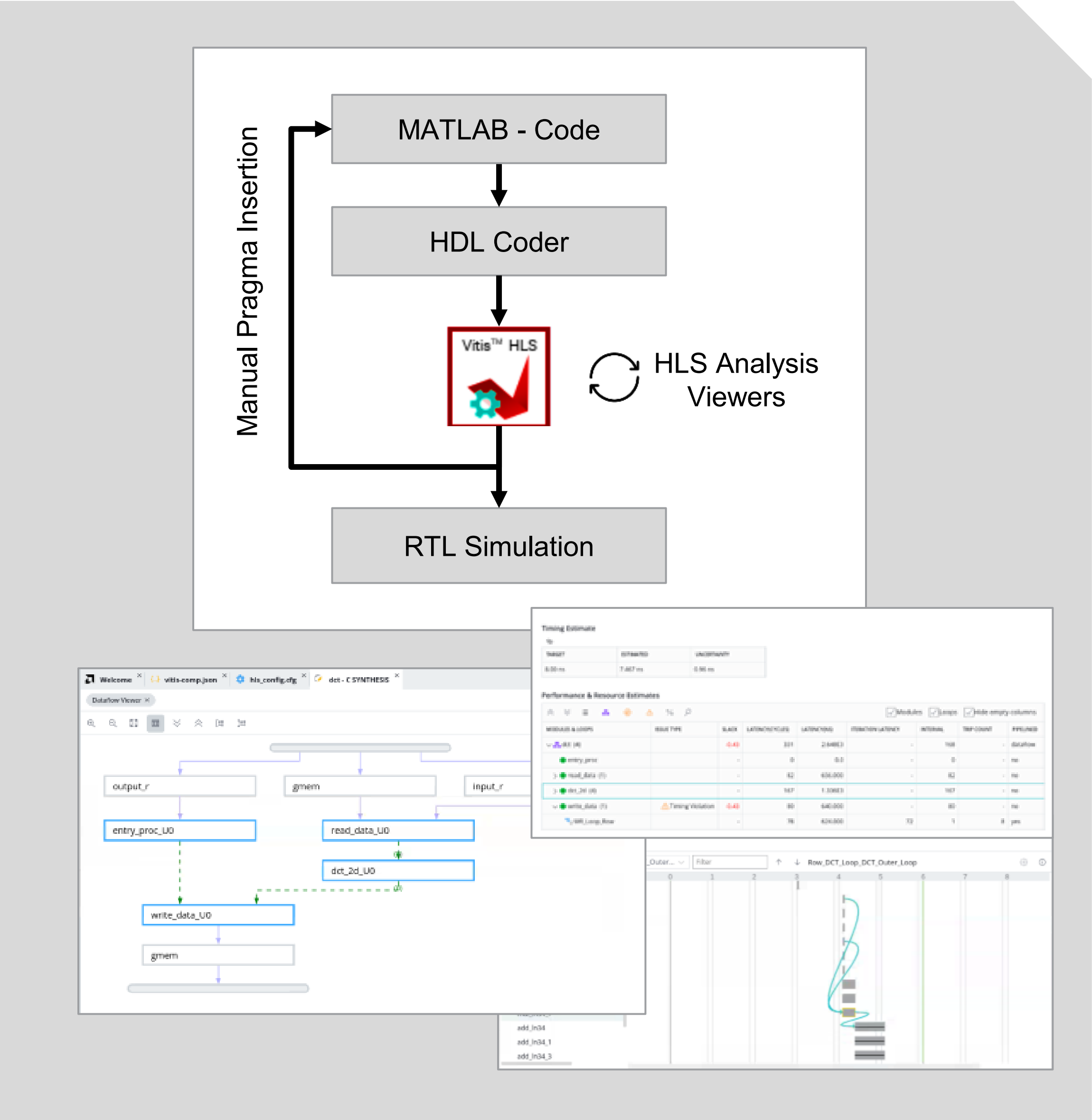
Automatic HDL Code Generation and Verification

- Code Generation: HDL Coder generates synthesizable C++ code optimized for AMD Vitis™ HLS
- Verification: Invokes Vitis HLS C simulation and verifies the generated C++ code
- Synthesis: Invokes Vitis HLS to run C++ synthesis and RTL simulation



User-Driven Optimization

- Identify performance gaps using AMD Vitis HLS Summary Report and Analysis Viewers
- Manually add pragmas such as pipeline, inline, etc., to optimize and improve performance
- Iteratively refine and optimize results



Coding Guidelines and Considerations

Supported MATLAB Syntax/Constructs for HLS Code Generation

MATLAB

- Recursion
- Unbounded sizes
- Datetime/duration
- Strings
- Half precision
- Try/catch
- Sparse matrices
- Toolbox functions
- Tables
- System objects
- Cell arrays
- Variable-sized arrays
- Visualization
- Matrices > 3D
- Java®/Python™

HLS Code Generation

- Scalars
- break/continue
- Rounding Modes
- 1D/2D/3D arrays
- return
- Overflow Modes
- int/float/fixed-point
- Fixed-size data
- Structs, enums
- If-else, switch, loops
- Sub-functions
- Complex

Supported MATLAB Features/Capabilities for HLS Code Generation

- **Features**
  - Streaming of scalars, matrices, and structures
  - Persistent variables and arrays
  - Loading constants from MATLAB file using coder.load
  - HLS-specific options in the source code
    - coder.hdl.loopspec
    - coder.hdl.constrainlatency
  - Instantiation of user-provided C functions
  - Sample-based and frame-based coding styles

- **Limitations**
  - Sizes and types of all variables need to be constant and known at compile time
  - Persistent variables must be read before written
  - System objects not supported

```
%#codegen
function [filtered_signal, y, fc] = mlhdlc_lms_fcn(input, ...
                                        desired, step_size, reset_weights)
% 'input'  : The signal from Exterior Mic which records the ambient noise.
% 'desired': The signal from Pilot's Mic which includes
%            original music signal and the noise signal
% 'err_sig': The difference between the 'desired' and the filtered 'input'
%           It represents the estimated music signal (output of this block)
%
% The LMS filter is trying to retrieve the original music signal('err_sig')
% from Pilot's Mic by filtering the Exterior Mic's signal and using it to
% cancel the noise in Pilot's Mic. The coefficients/weights of the filter
% are updated(adapted) in real-time based on 'input' and 'err_sig'.

% register filter coefficients
persistent filter_coeff;
if isempty(filter_coeff)
    filter_coeff = zeros(1, 40);
end

% Variable Filter: Call 'tapped_delay_fcn' function on path to create
% 40-step tapped delay
delayed_signal = mtapped_delay_fcn(input);

% Apply filter coefficients
weight_applied = delayed_signal .* filter_coeff;

% Call treesum function on matlab path to sum up the results
filtered_signal = mtreesum_fcn(weight_applied);

% Output estimated Original Signal
td = desired;
tf = filtered_signal;
esig = td - tf;
y = esig;

% Update Weights: Call 'update_weight_fcn' function on MATLAB path to
% calculate the new weights
updated_weight = update_weight_fcn(step_size, esig, delayed_signal, ...
                                   filter_coeff, reset_weights);

% update filter coefficients register
filter_coeff = updated_weight;
fc = filter_coeff;
```

Examples and Benchmarks

Demos and Examples

- Filters
  - Symmetric FIR – “ **sfir** ”
  - Kalman Filter - “ **Kalman_c** ”
  - Sobel Filter - “ **sobel** ”
  - Least Mean squares filter - “ **Lms_fcn** ”
  - IIR Filter – “**iir_filter**”
- Vision
  - Histogram Equalizataion - “ **heq** ”
  - Image Scaling - “ **Image_scale** ”
  - RGB Conversion - “ **Rgb2yuv** ”
  - 2-D FIR Filter - “ **2DFIR** ”
  - Median Filter - “ **Median_filter** ”
- Crypto
  - Advanced Encryption System - “ **aes** ”
- Comms
  - Communication Example - “ **Comms_data_packet** ”
  - Viterbi Algorithm - “ **viterbi** ”
  - Discrete Fourier Transform - “ **df2t_filter** ”
- Controls
  - Discrete Time Integrator - “ **dti** ”

Demos are included in the MATLAB R2025a installation

A demo can be setup by the following command:

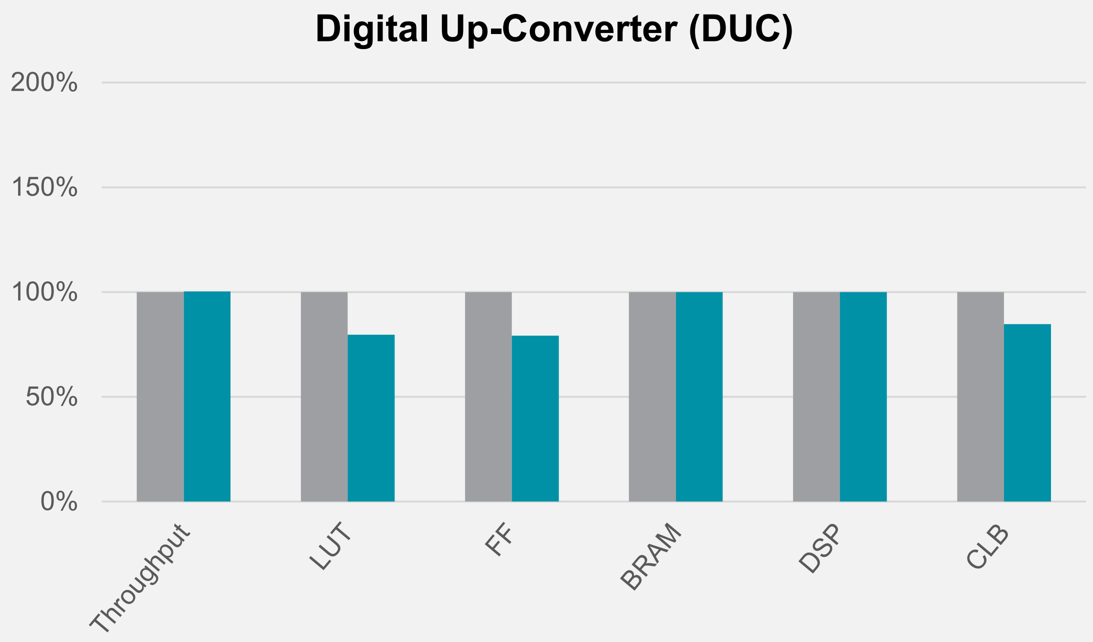
```
>> mlhdlc_demo_setup(“<demo_name>");
```

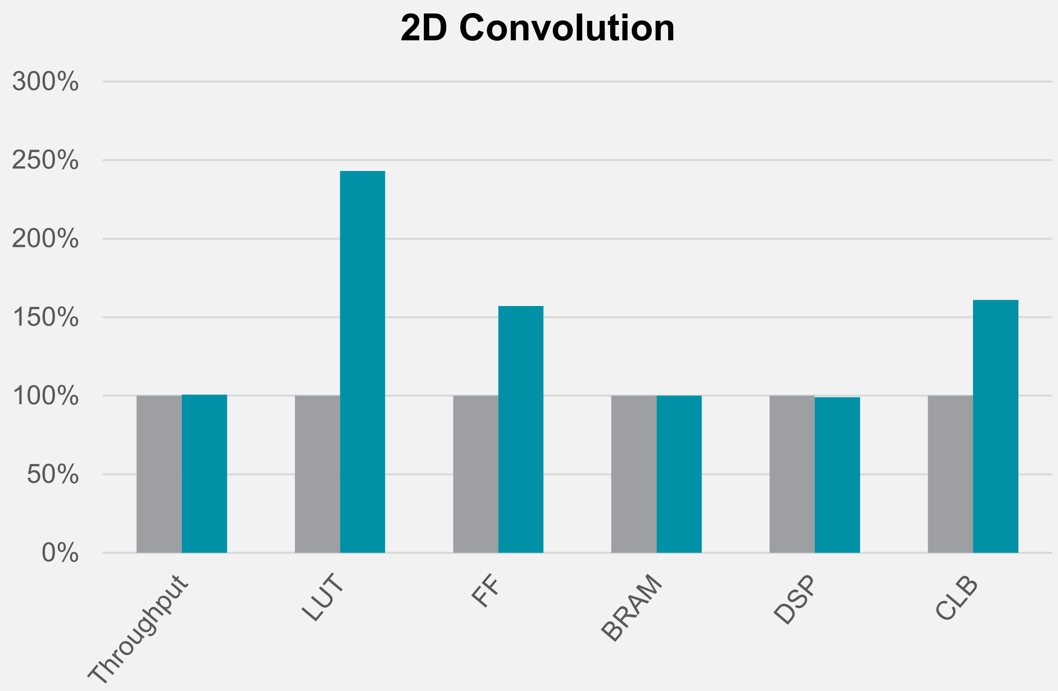
For example, the Histogram Equalization Demo can be setup using:

```
>> mlhdlc_demo_setup("heq")
```


AMD
together we advance_

Relative Performance and Utilization

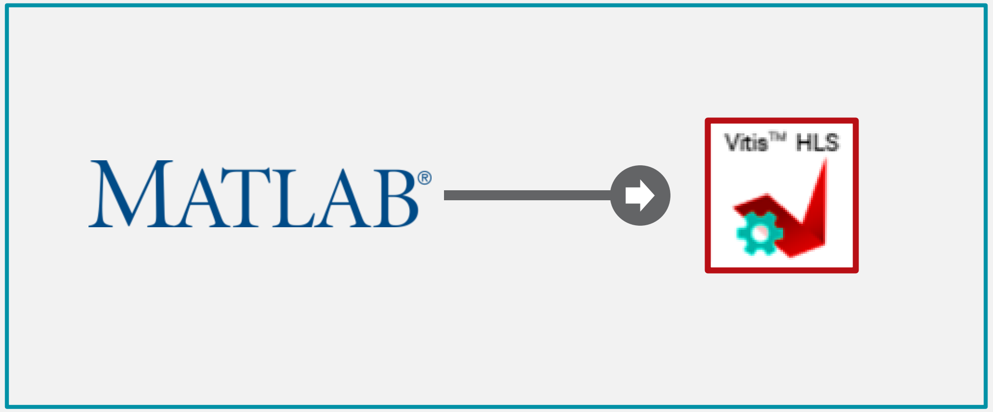




Native HLS C/C++ MathWorks MATLAB to HLS

AMD together we advance_

Next Steps & Call to Action



Download the MATLAB R2025a release!

General Disclaimer and Attribution Statement 2025

AMD