Introduction to C and HDL Code Generation from MATLAB

이웅재 차장
Senior Application Engineer
Algorithm Development Process

Requirements

Research & Design
- Explore and discover
- Gain insight into problem
- Evaluate options, trade-offs

Implementation

Desktop
- .dll
- .exe
- .c, .cpp

Embedded
- C
- VHDL / Verilog
- Structured Text

Test & Verification
The Algorithm Design Challenge

- How can we:
  - Implement designs faster?
  - Reuse designs on a variety of hardware?
Solution: C and HDL Code Generation

- Design, execute, and verify algorithms in MATLAB
- Automatically generate C or HDL code
- Deploy generated code on hardware
Code Generation Products for C/C++

- **Embedded Coder™**
  - Automatically generate C and C++ optimized for embedded systems

- **Simulink® Coder™**
  - Automatically generate C and C++ from *Simulink* models and *Stateflow* charts

- **MATLAB® Coder™**
  - Automatically generate C and C++ from *MATLAB* code
Code Generation Products for VHDL/Verilog

HDL Coder™
Automatically generate VHDL or Verilog from MATLAB code and Simulink Model

MATLAB® Coder™
Automatically generate C and C++ from MATLAB code
MATLAB Coder Capabilities
Why Engineers translate MATLAB to C today?

Integrate MATLAB algorithms w/ existing C environment using source code and static/dynamic libraries

Prototype MATLAB algorithms on desktops as standalone executables

Accelerate user-written MATLAB algorithms

Implement C code on processors or hand-off to software engineers
Challenges with Manual Translation from MATLAB to C

- Separate functional and implementation specification
  - Leads to *multiple implementations* that are inconsistent
  - Hard to modify requirements during development
  - Difficult to keep reference MATLAB code and C code in-sync

- Manual coding errors

- Time consuming and expensive
With MATLAB Coder, design engineers can

- Maintain **one design** in MATLAB
- Design faster and **get to C quickly**
- Test more systematically and frequently
- Spend more time improving **algorithms** in MATLAB
MATLAB Coder Key Features

- Code generation for MATLAB
  - Portable ANSI/ISO C code
  - Floating-point or Fixed-point
    (requires Fixed-Point Toolbox)
  - Processors specific optimizations
    (requires Embedded Coder)

- Verification
  - Generates MEX test-benches

- Design and Build
  - Automated compile and build for
desktop execution
  - Ability to cross-compile and build for
execution on other platforms
Using MATLAB Coder: 3-Step Workflow

Prepare your MATLAB algorithm for code generation
  • Make implementation choices
  • Use supported language features

Test if your MATLAB code is compliant
  • Validate that MATLAB program generates code
  • Accelerate execution of user-written algorithm

Generate source code or MEX for final use
  • Iterate your MATLAB code to optimize
  • Implement as source, executable or library
Example
\[ c = a \times b \]

- Coder UI
- Testbench
- Code Generation options
- Generate code
- Browse through report

>> Demo
Demo: Newton/Rhapson Algorithm

workflow example

Preparing your MATLAB code

- Pre-allocate
- Identify more efficient constructs
- Building options

Test & Generate

\[
x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}
\]

```matlab
function [x,h] = newtonSearchAlgorithm(b,n,tol)
% Given, "a", this function finds the nth root of a
% number by finding where: x^n-a=0.
coder.inline('never');

notDone = 1;
aNew = 0; %Refined Guess Initialization
a = 1; %Initial Guess
cnt = 0;
h=zeros(50,1);
h(1)=a;
while notDone
    cnt = cnt+1;
    [curVal,slope] = f_and_df(a,b,n); %square
    yint = curVal-slope*a;
    aNew = -yint/slope; %The new guess
    h(cnt)=aNew;
    if (abs(aNew-a) < tol) %Break if it's converge
        notDone = 0;
    elseif cnt>49 %after 50 iterations, stop
        notDone = 0;
        aNew = 0;
end
```

>> Demo
MATLAB Language Support for Code Generation

- Java
- sparse
- nested functions
- graphics

- visualization
- variable-sized data
- struct
- malloc
- functions
- numeric
- System objects
- complex
- fixed-point
- arrays
- global
- classes
- persistent

- nested functions
Embedded Coder for Optimized Code

Embedded Coder extends MATLAB Coder with:

- Processor-specific code generation
  - Built-in support for select processors
  - Open APIs for use with any processor
- Speed, memory, and code appearance advanced features
HDL Coder Capabilities
HDL Coder

Generate VHDL and Verilog Code for FPGA and ASIC designs

MATLAB
Simulink
HDL Coder
Verilog and VHDL

New: MATLAB to HDL

- Automatic floating-point to fixed-point conversion
- HDL resource optimizations and reports
- Algorithm-to-HDL traceability
- Integration with simulation & synthesis tools
Which benefits Engineers get from MATLAB to HDL?

- **Automate for implementing HDL** without direct RTL hand-coding.
- **Less LOC** by using MATLAB TestBench and **Reusing** it through Entire Design Process.
- **Easy conversion** from Floating-Point to Fixed-Point.
- **Fast to simulate** compare with using HDL DUT/TB in the HDL simulator environment.
HDL Coder Key Features

- Code Generation for MATLAB and Simulink
  - Target-independent RTL level HDL Code
  - IEEE 1376 compliant VHDL
  - IEEE 1364-2001 compliant Verilog
  - Automate Fixed-Point conversion

- Verification
  - Generate HDL test-bench
  - Generate Scripts for Synthesis tool & HDL simulator

- Design automation
  - Synthesize and P&R using integrated Xilinx and Altera synthesis tool interface
  - Optimize for area or speed
Using HDL Coder: 5-Step Workflow

Prepare
- Prepare your MATLAB algorithm for code generation
  - Use supported language features
  - Make implementation choices

Fixed-Point
- Fixed-Point MATLAB code generation from your floating-point design using your MATLAB TestBench
  - Accelerate TestBench for fast simulation
  - Automatically propose Fixed-Point type
  - Iterate data-type customization to optimize
  - Verify Fixed-Point code against original Floating-Point code.

Generate
- Generate synthesizable RTL & TestBench code from Fixed-Point MATLAB code for final use
  - Iterate your MATLAB code to optimize
  - Implement as source, executable or library

Simulate
- Simulation the generated HDL code with test vectors from the test bench using the specified simulation tool

Synthesis, P&R
- Synthesis, Place and Route the generated RTL code by creating project with ISE/Quartus II
  - Check timing analysis report to optimize
Workflow Advisor

- Automatically convert floating point to fixed-point
- Automatic HDL Verification
- Integration with FPGA implementation workflows
- Automatic HDL code generation with built-in optimizations

```
### Begin VHDL Code Generation
### Working on sobel_edge_detection_FixPt/u_d_ram/dualPortRAM_128x8b as codegen\sobel_edge_detection\hdlsrc\dualPortRAM_128x8b.vhd
### Working on sobel_edge_detection_FixPt/u_d_ram as codegen\sobel_edge_detection\hdlsrc\u_d_ram.vhd
### Working on sobel_edge_detection_FixPt/u_d_ram/dualPortRAM_128x8b_block.vhd
### Working on sobel_edge_detection_FixPt/u_d_ram as codegen\sobel_edge_detection\hdlsrc\u_d_ram.vhd
```
Example: Symmetric FIR Filter

```matlab
function [y_out, delayed_xout] = mlhdlc_sfir(x_in,
    % 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;

    always @(posedge clk or posedge reset)
        begin : ud2_reg_process
            if (reset == 1'b1) begin
                ud2_1 <= 0;
            end
            else begin
                if (enb) begin
                    ud2_1 <= ud2;
                end
            end
        end
    assign tmp_4 = ud2_1;
```
Demos

Generate Modular Code for Functions

Fixed-Point Type Conversion and Refinement

Distributed Pipelining: Clock Speed Optimization

Resource Sharing of Multiplier: Area Optimization
MATLAB Language Support for

HDL Code Generation

- struct
- visualization
- Java
- complex
- sparse
- matrix
- nested functions
- graphics
- classes

Arithmetic Operators
Rational Operators
vector
persistent
Logical Operators
Control Flow Statement
System objects (*limited*)
fixed-point

MathWorks
Customer Successes
Ono Sokki Reduces Development Time for Speed Measurement Device Using MATLAB Generated Code

Challenge
Develop a high-precision speedometer for prototype vehicles within a tight schedule

Solution
Use MathWorks tools for Model-Based Design to develop algorithms, generate production code for an embedded processor, and perform processor-in-the-loop verification

Results
- Development time cut significantly
- Code base reduced dramatically
- System model reused as a test bench

“With MathWorks tools we have a seamless environment for development, simulation, code generation, and processor-in-the-loop verification. The advantages of Model-Based Design over hand-coding in C can’t be overestimated.”

Kazuhiro Ichikawa
Ono Sokki
FLIR Accelerates Development of Thermal Imaging FPGA

**Challenge**
Accelerate the implementation of advanced thermal imaging filters and algorithms on FPGA hardware

**Solution**
Use MATLAB to develop, simulate, and evaluate algorithms, and use HDL Coder to implement the best algorithms on FPGAs

**Results**
- Time from concept to field-testable prototype reduced by 60%
- Enhancements completed in hours, not weeks
- Code reuse increased from zero to 30%

“With MATLAB and HDL Coder we are much more responsive to marketplace needs. We now embrace change, because we can take a new idea to a real-time-capable hardware prototype in just a few weeks. There is more joy in engineering, so we’ve increased job satisfaction as well as customer satisfaction.”

Nicholas Hogasten
FLIR Systems

Link to user story
Summary

- Design using MATLAB
- Generate C and HDL
- Deploy code to Processors and Hardware
- Accelerate simulation speed
- Easy conversion from floating-point to fixed-point
- Iterate your designs faster
Additional Information

- MATLAB Coder
  - Choosing the Right Deployment Solution
  - Working with Simulink

- HDL Coder
  - Working with Simulink
  - Simulink Design for HDL Code Generation
Choosing the Right Deployment Solution

MATLAB Coder and MATLAB Compiler
Choosing the Right Deployment Solution

**MATLAB Coder and MATLAB Compiler**

<table>
<thead>
<tr>
<th></th>
<th>MATLAB Coder</th>
<th>MATLAB Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td>Portable and readable C source code</td>
<td>Executable or software component/library</td>
</tr>
<tr>
<td><strong>MATLAB support</strong></td>
<td>Subset of language Some toolboxes</td>
<td>Full language Most toolboxes Graphics</td>
</tr>
<tr>
<td><strong>Runtime requirement</strong></td>
<td>None</td>
<td>MATLAB Compiler Runtime (MCR)</td>
</tr>
<tr>
<td><strong>License model</strong></td>
<td>Royalty-free</td>
<td>Royalty-free</td>
</tr>
</tbody>
</table>
Working With Simulink

for C code generation

- MATLAB Function block in Simulink
Working With Simulink
for HDL code generation

- MATLAB Function block in Simulink

```matlab
function e = fcn(u, thresh, C)
% Pipeline Sobel Edge Detection algorithm

[xfo, yfo] = filter(u, C);
xfd = filterdelay12(xfo);
yfd = filterdelay13(yfo);
a = abs(xfd);
y = abs(yfd);
t = (a + y >= thresh);
e = filterdelay14(t);
end

% Compute convolution of serialized image
function [xfo, yfo] = filter(u, C)

lbl1 = line_buffer1(u, Col);
lbl2 = line_buffer2(lbl1, Col);
u_d1 = filterdelay1(u);
u_d2 = filterdelay2(u_d1);
```

```matlab
u_d_next_14 <= resize(resize(u_d_10, 12) + resize(u_d_11, 12), 13) +
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
```
Simulink Library Support for HDL

HDL Supported Blocks

- Almost 170 blocks supported

- Core Simulink Blocks
  - Basic and Array Arithmetic, Look-Up Tables, Signal Routing (Mux/Demux, Delays, Selectors), Logic & Bit Operations, Dual and single port RAMs, FIFOs, CORDICs

- Signal Processing Blocks
  - NCOs, FFTs, Digital Filters (FIR, IIR, Multi-rate, Adaptive), Rate Changes (Up & Down Sample), Statistics (Min/Max)

- Communications Blocks
  - Psuedo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoders
MATLAB & Stateflow for HDL

HDL Supported Blocks

- **MATLAB**
  - Relevant subset of the MATLAB language for modeling and generating HDL implementations
  - *eml_hdl_design_patterns*: Useful MATLAB Function Block Design Patterns for HDL

- **Stateflow**
  - Graphical tool for modeling Mealy and Moore Finite State Machines
Model-Based Design flow using Simulink
from Algorithm to FPGA Implementation