Simulink to Embedded Hardware

Paul Peeling
MathWorks
Model-Based Design for Hardware

- Stakeholder Needs
- Environment Models
- Physical Components
- Algorithms

- Subsystem Design
- System Specification
- High-Level Designs
- Low-Level Design
- Code

- Manage Requirements Traceability
- Develop Acceptance Tests
- Develop Test System (e.g., HIL)
- Define Golden Reference

- System Integration and Test
- Complete Integration and Test

- Production Code Generation

- IMPLEMENTATION

- Requirements
Creating a legacy model

- Model begins life as proof of concept
- Elaborated to address project-specific requirements
- Code generation to run on target hardware
- Model and/or code optimised to meet constraints
- Model can no longer be simulated
Hardware Requirements

- My design must run at XXX Hz
- Available memory is limited
- The algorithm must be implemented in single precision
- I need to read from/write to peripherals
- I need to call libraries written for the target
- The hardware is not yet available
- Run on multiple hardware platforms
Topics for this session

- Model architecture for hardware designs
- Simulation, debugging and logging on hardware
- Analysis tools and reports
- Importance of hardware abstraction
Exercise: Parametric Audio Equalizer

- Run a signal processing algorithm on a STM32F4 Discovery Board
- A working desktop simulation exists: dspparameq
- Interactive:
  - Switch between recorded (intended) and white noise (good for testing)
  - Prototype user interface to manipulate filter response
Partitioning a model and generating code

1. Generating C code from an *atomic* subsystem.
2. Tracing between model and code through the code generation report.
3. Analyse generated code through static code metrics.
4. Using model reference to define a component boundary.
Model References

Central file repository

Implementation

Test Harness

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Virtual subsystem</th>
<th>Atomic subsystem</th>
<th>Model reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readability</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Traceability</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Reusability</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Concurrent development</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Unit testing</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
Connecting to Low Cost Hardware

Engineer’s computer
MATLAB algorithm or Simulink model

Data I/O
Ethernet / USB / Bluetooth
MATLAB Hardware Support Packages

Target

Low Cost Hardware
Simulink Hardware Support Packages

Get Support Package Now
STM32F4-Discovery support package
Running a Processor-in-the-Loop Simulation

1. Download and install a target support package
2. Configure model for hardware and profiling
3. Run model block in PIL mode
4. Verify outputs and logged signals
5. Analyse profiling data
Processor-in-the-Loop (PIL)
Verify compiled object code matches simulation

- Verify numerical equivalence
- Assess execution time
- Collect code coverage
- Create certification artifacts

Non-real-time execution: synchronized with simulation

- Software-In-the-Loop (SIL) for host-based execution
Function Execution Times for Code Running on Embedded Hardware

Comprehensive measurement of function call execution times

- Call-site instrumentation to measure execution time of functions in the generated code
- Includes initialization, shared utility and math library functions
- Configurable units for reporting of measured execution times

```matlab
/* Outputs for Atomic SubSystem: '<Root>/Band 1 DF1' */
profileStart_P_acf95d8ceb4917b3(1U);
Parametric_Equalizer_Band1DF1();
profileEnd_P_d63e7c572a73650c(1U);

/* Outputs for Atomic SubSystem: '<Root>/Band 2 DF2' */
profileStart_P_acf95d8ceb4917b3(3U);
Parametric_Equalizer_Band2DF2();
profileEnd_P_d63e7c572a73650c(3U);

executionProfile.report( 'ScaleFactor', '1', 'NumericFormat', '%0.6f' )
```
Optimising models and verifying behaviour

1. Creating an alias for data types used throughout a model
2. Verifying behaviour through simulation
3. Running models through multiple test harnesses using model reference
4. Using configuration references to manage model parameter sets
Design Data and Variants

1. Parameterising sample times and data lengths
2. Creating tunable parameters
3. Moving design data from base workspace to data dictionary
4. Linking model hierarchies to dictionaries
Data Dictionary

- Model workspace
  - B 3
  - C 3

OR

Data dictionary .sldd

Configurations

Global Design Data
- Signals
- Parameters
- Data objects
Key Tools

- Data Dictionary
- Configuration Reference
- Processor-in-the-Loop
- Model Reference
- Variants
- Alias Types
- Tunable Parameters
Key Takeaways

1. Generated code can be executed and verified from within Simulink
2. Start designing with hardware in mind from the onset
3. Maintain abstraction from hardware
Thank You!
Questions?
On-Target LDRA Testbed® Code Coverage

Collect code coverage metrics for production code executing in PIL simulation

- Execute tests on simulator or target hardware to support high integrity workflows such as DO-178B and ISO 26262
- Works with any existing PIL target
- Support new targets via the PIL Target Connectivity API
  - Arduino and TI C2000 reference implementations available