Implementation of Hardware and Software Systems using Model-Based Design

Part of DATE’08 Tutorial:
Automatically Realizing Embedded Systems from High-Level Functional Models

Monday, 10 Mar 2008
Don Orofino, PhD
Director, Signal Processing Development
The MathWorks, Inc.

Tutorial Segment

- Discuss use of Model-Based Design (MBD) to implement hardware and software systems
- Incrementally elaborate an example model from abstract toward concrete implementation
- Discuss practical requirements imposed on the tool chain
- Engineers must make choices
  - Implementation choices should be minimally invasive and maximally deferred
  - Key questions underlying those choices...
Traditional Implementation Decisions

Abstract notion of an algorithm

Example: Digital filter, prescribed specifications, floating point

1. Re-code/simulate
2. Re-code/simulate
3. Re-code/execute
4. Re-code/execute

Multi-stage cascade, floating point
Executing on DSP?
1: TI DM642: fixed point
2: 16-bit banked coefficients, pipelined code
3: Assembly language optimizations …
4: Re-code/simulate
5: Re-code/simulate
6: Re-code/execute
7: Re-code/execute

Executing on FPGA?
1: Xilinx Virtex-4: fixed point
2: 18-bit multipliers on a DSP48 slice
3: Core-generator implementation …

Steps are highly invasive (significant recoding, cannot easily “undo”) Steps cannot be deferred (and must be done manually)
Traditional Implementation Decisions

- Additional Areas for Improvement
  - Performance scales with the engineer's experience (highly variable outcomes)
  - Time consuming manual iterations
  - Initial design requires expertise at all decision points
  - Code maintenance and enhancement require significant experience
  - Algorithm changes not easily accommodated
  - Design space exploration is manual, slow
  - Verification not built into process, must be separately considered
  - Design artifacts and simulation results not uniformly reproducible at each step
- More...

- Traditional Implementation Steps are highly invasive (significant recoding, cannot easily "undo")
- Steps cannot be deferred (and must be done manually)

MBD Implementation Decisions

- Focus on the complete algorithm-design lifecycle
- MBD demands tight integration of tools
- Maintains model abstraction (defers implementation)
- Simplifies:
  - Understanding
  - Verification
  - Maintenance
  - Enhancement
  - Re-targetability
MBD Implementation Decisions

Abstract notion of an algorithm

*Example: Digital filter, prescribed specifications, floating point*

I1: Multi-stage cascade, floating point
Executing on DSP
   I2: TI DM642, fixed point
   I3: 16-bit banked coefficients, pipelined code
   I4: Assembly language optimizations...

Executing on FPGA
   I2: Xilinx Virtex-4, fixed point
   I3: 18-bit multipliers on a DSP48 slice
   I4: Core-generator implementation...

I2: Elaborate, verify
I3: Elaborate, verify
I4: Integrate, verify

Model-Based Design

Automotive Design Example:
Lane Departure Warning

Results from Simulink Model of Lane Departure Warning System
MBD: Executable Specification

**Executable Specifications**
- Lane Departure Warning
- Floating-point specification
- Verification test-bench

**System Model:** Implement edge detection algorithm as part of lane departure warning system
**Design Entry: Simulink**

Start by developing a golden specification of the edge detection algorithm.

**Floating-point System Specification**

- **High-level blocks:** Video Edge Detector
  - Video and Image Processing Blockset
- **Lower-level blocks:** Sum, Abs, 2-D Filter
  - Simulink, Signal Processing Blockset

**Design Entry: Embedded MATLAB**

- Embedded subset of MATLAB Language
- Brings MATLAB algorithms into Simulink and Stateflow
Insight on Implementation Choices

1. **Design Languages**

   - **MATLAB**: Largely behavioral, un-timed, array-based semantics
   - **Simulink**: Structural, time-based simulation engine, hybrid solvers
   - **Stateflow**: Reactive systems design, finite state machines
   - **SimEvents**: Discrete-event, data driven modeling

---

MBD: Design with Simulation

- **Design with Simulation**
  - Fixed-point model migration
  - Design Advisors, Test-Benching
MBD: Design with Simulation

- Migrate the design using verified iterations
- Make decisions appropriate to deployment
  - Serial processing (2D to vector data)
  - Fixed-point operation (float-to-fixed conversion)

Migration to Fixed-point

- Fixed-point data types – full manual control

```
fixed-point data types – full manual control
```

```
“sfix11_En2”
signed fixed-point number
11 bit word, 2 bit fraction
```
Migration to Fixed-point

<table>
<thead>
<tr>
<th>Fixed-point data types</th>
<th>Signed</th>
<th>Word length</th>
<th>Fraction length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>Same as input</td>
<td>Same as input</td>
<td>Same as input</td>
</tr>
<tr>
<td>Product output</td>
<td>Same as input</td>
<td>Same as previous</td>
<td>Same as previous</td>
</tr>
<tr>
<td>Accumulator</td>
<td>Same as product output</td>
<td>Same as previous</td>
<td>Same as previous</td>
</tr>
<tr>
<td>Output</td>
<td>Binary point scaling</td>
<td>Same as previous</td>
<td>2</td>
</tr>
</tbody>
</table>

Locked scaling against changes by the autoscaling tool

```
sfix11_En2
```
signed fixed-point number
11 bit word, 2 bit fraction

Migration to Fixed-point

Autoscaling Tool: Advisor to help set fixed-point scale factors
Design Advisors

- GUI-driven Advice and Assistance
- Fixed-Point Tools
  - Record design requirements
  - Dynamic range assessment
  - Design auto-scaling
- Simulink Model Advisor
  - Choose advisor goals
    - Modeling Guidelines
    - Best Practices
    - Code Safety
    - Performance Optimization
  - Step-by-step guidance

Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3. 
4. 
5. 

- Use block-based implementations (Edge Detector)
- Adjust fixed-point settings in each block dialog
- Use target-specific optimized blocks (TI, Xilinx, etc)

- Fixed Point Tool (Simulink Fixed Point)
  - Automatically suggests fixed point settings for blocks
- Filter Wizard (Signal Processing Toolbox)
  - Automatically generates Simulink models using basic blocks
- Simulink HDL Coder
  - Generates new models based on implementation choices
MBD: Automatic Code Generation

- Coders
  - Code generation from models
  - Language options
  - Code interfacing, optimization

- Links
  - Verification tool integration
  - Project generation, build, download
  - Co-simulation, SIL/PIL/HIL

- Targets
  - Processor & memory specific optimization
  - Device drivers, board support
  - Schedulers, RTOS integration

Model-Based Design supports both Software and Hardware systems

C Code Generation, Links, Targets
HDL Code Generation, Links, Targets
C / ASM
VHDL / Verilog
MCU
DSP
FPGA
ASIC
Code Gen: Embedded Software

- Code Generation
  - Real Time Workshop – ANSI/ISO C code for rapid prototyping, acceleration
  - Real Time Workshop Embedded Coder – Embedded deployment

- Links
  - Altium TASKING
  - Analog Devices VisualDSP++
  - Green Hills MULTI
  - TI Code Composer Studio

- Targets
  - TI C6000 DSP
  - TI C2000 DSP
  - Infineon C166 Microcontrollers
  - Freescale MPC5xx Microcontrollers

Let's take Lane Departure to an FPGA …

Code Gen: FPGA Hardware

- Code Generation
  - Simulink HDL Coder – FPGA and ASIC deployment using VHDL and Verilog
  - Filter Design HDL Coder – Filter implementation from MATLAB

- Links
  - Mentor ModelSim/Questa
  - Cadence VCS/Incisive
  - Synopsys Discovery

- Targets
  - Altera DSPBuilder
  - Xilinx SystemGenerator
  - More…

Let's take Lane Departure to an FPGA …
Code Gen: FPGA Hardware

Simulink HDL Coder
Correct-by-construction VHDL and Verilog code

Generated Verilog code

Design Space Exploration for HDL

NEED:
- Speed
  How fast can this design run?
- Area
  Can I use a smaller chip?
- Power
  Can I target a mobile device?

SOLUTION:
Code Generation Control Files
Controls implementation used to generate code for blocks / components
- Locally: for an individual block
- Globally: for all blocks of a given type
- Hierarchically: scope within a model

- Sum, Product
  - Linear
  - Cascade
  - Tree
- Gain
  - Multiplier
  - CSD
  - ICSD
- Min/Max
  - Tree
  - Cascade
- Lookup Table
  - Inline
  - Hierarchical
Example Control File

Textual interface

Initialize control file constraint object

CSD multiplier "GainCSD"

Factored-CSD multiplier "GainFCSD"

Synthesis: Edge Detection

Device utilization summary:

Selected Device: 4vex25f668-12

- Number of Slices: 1105 out of 10240 10%
- Number of Slices Flip Flops: 1900 out of 20480 9%
- Number of 4-input LUTs: 156 out of 18400 0%
- Number of IOs: 1
- Number of Bonded IOBs: 1 out of 310 0%

Design statistics:
- Minimum period: 4.106ns
- Maximum frequency: 241.54MHz
- Minimum input required time before clock: 1.3ns
- Maximum output delay after clock: 7.323ns
Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3. Design Constraints
4. 
5. 

**Block Dialogs**
Example block-level controls:
- Time- vs. Frequency domain algorithm options
- Table-lookup vs. on-line internal computations

**Code Gen Options**
- Memory Layout
- Interfaces & Naming
- Clocking & Reset
- Pipelining Control
- Assertion Levels

**HDL Control Files**
Specify concrete implementations
- For individual algorithms
- Across all similar algorithms
Re-simulate with constraints
- Bit-true, cycle-accurate
- Automatic model creation

MBD: Continuous Test and Verification

Test and Verification
- Simulation, Test-benching
- SIL/PIL/HIL, Co-simulation
Design Advisors for HDL

GUI-driven Advice and Assistance
- Compatibility Checker
  - Identify unsupported semantics
  - Optimize for design goals
- Test-bench Generation
  - VHDL, Verilog, script file generation

Co-simulation of Generated Code

HDL code executing on ModelSim simulator
Making Full Use Of The System Model

- Promotes parallelism in design and verification tasks
- Accelerates system level verification

Verification of Synthesized Hardware

- Verify implementation using Hardware-In-the-Loop (Third-Party Integration)
Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3. Design Constraints
4. Verification Links
5. Advisory Tools

Conclusions?
- No single “silver bullet” to support modern workflow
  - Need a well-devised arsenal
- Well-defined and executable semantics are required
- Need integrated simulation, code generation, verification
- Multi-language support seems essential
  - Domain-specific, and combined textual & graphical
Trends in Model-Based Design

- Unifying multiple engineering groups
  - One design environment
  - Multi-domain simulation (analog, digital, physical, …)
- Multi-platform designs
  - Embedding MATLAB within Simulink and Stateflow
- Connecting stages of development
  - Design – Implementation – Verification
  - Coder products: C and HDL generation
  - Link products: Verification and Validation
- Harnessing high-performance computing
  - Distributed computing tools for MATLAB and Simulink
  - Acceleration from multi-core simulation

Questions?

Accelerating the pace of discovery, innovation, development, and learning in engineering and science.