Rapid Development Platform for C-Programmable DSP using MATLAB and Simulink

Texas Instruments India,
Audio and Imaging Group

• Supriyo Palit
• Doug Roberson
• Mukund Navada
• Diljith Thodi
Outline

- Problem Statement
- Existing Workflow
- Proposed Solution
- Benefits
- Description
- Development Workflow
- Impact
- Conclusion
- Q&A
Problem Statement

- Reduce turn-around-time from discovery of a problem to solution.
- Validate our solutions with customers using SOC emulation on FPGA before investing in Silicon.
- Bit-accurate comparisons between model and target.
- Intuitive GUI for controlling both model and target.
- Automated tool flow to production
Typical Workflow: Exploration

- Analysis/Design
- Modeling
- Define/Refine
- Evaluate
Typical Workflow: Development

- Model Translation
- Build: (Firmware/ROM)
- Validation
Typical Workflow: Tooling, UI & Deployment

- Tuning Application Development (User Interface)
- Drivers and Tools
- Integration
- Demos/Training
- Bugs/Enhancement
Typical Workflow: Summary

Model solution in Simulink/MATLAB.

Manual translation of above verified to C.

Generate firmware and test on FPGA/Silicon.

Develop a standalone app to tune firmware.
Workflow Limitations

• Greater effort required to evaluate solution feasibility.

• Re-implementation of simulation blocks on target hardware
  – Wasted Effort
  – Bug-prone

• Error-prone translation and integration steps at various phases.

• Model, Implementation & GUI toolchains are independent and the designs have to be manually kept in sync.

• What is shipped is different from what is modelled.

• Inability to simulate system issues in model.
The New Model-Based Workflow
Model-Based Workflow

• Phase 1: Algorithm Development
  – Develop and maintain algorithm in MATLAB/Simulink (Source Code)
  – Access to rich library of DSP and computation toolboxes
  – Helps in modelling complex systems accurately
  – Convenience in debugging system level problems
  – Probe points for live debug in hardware
  – Linked graphical navigation between model components and C-code
Model-Based Workflow

- Phase 2: Automatic Code Generation and Build
  - Eliminating hand-written code, faster time-to-market
  - 100% bit-exact (Simulink/FPGA/production code)
  - “Push-button” to generate and deploy code, and go to listening experience
  - Automated tool flow to generate firmware binaries for target.
  - Zero turnaround between algorithm and production firmware
  - Optimized C code available from the tool
Model-Based Workflow

- **Phase 3: Integration & Packaging**
  - Develop a tuning GUI that can talk to both hardware and model.
  - Any system level tuning setting can now be run through the model.
    - More visibility
    - Easier debug
    - Quick correlation between EVM and high level model
    - Two-way exchange of configuration between hardware and Simulink
  - Code is generated from Simulink model, so firmware and model are compatible – easier for GUI to control
Workflow Details

• Simulink to Firmware
  – Embedded/Simulink Coder package
    • Converts fixed-point Simulink algorithm to optimized C code
    • Target level support through code replacement library and custom s-functions
  – DSP-Toolchain (C-compiler/Linker)
    • Converts C code to DSP assembly
  – In-house development
    • Framework to control DSP subsystem
    • Scripts to convert firmware object code to I2C format
    • System level scripts to communicate between Simulink/Toolchain/Tuning GUI
  – Supports both FPGA (with 30Mbit fast download) and SOC
Workflow Details

• Embedded Coder Configuration for efficiency and embedded control
  – DSP custom rule replacement library
    • Identifies processing macros and replaces with DSP intrinsics
  – Custom DSP library components
    • s-function models for specific DSP blocks (e.g. biquad filter) with optimized embedded implementation
  – Memory constraints
    • Flexibility to declare variables across memory banks from the coder using custom Signal and Parameter classes – required for single cycle MAC
  – ROM-ability
    • Support for single-structure re-entrant function for ROM
  – Probe points
    • Ability to add probe variables in embedded code for live debug on hardware
  – Supports both sample- and block-based processing
Workflow Details

- Deployable Application
  - Front-end based on MATLAB GUIDE framework
  - Speaker Measurement and Tuning
  - Controls both hardware and Simulink
  - Real-time update for live audio tuning
  - System level tuning parameters are accessible to Simulink for debugging
Case Study

• Coder Efficiency (on typical building blocks)

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Handwritten</th>
<th>Embedded Coder</th>
<th>% Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles</td>
<td>Instructions</td>
<td>Cycles</td>
</tr>
<tr>
<td>RMS Filter</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Polynomial Evaluation</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Attack-Decay Filter</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Noise Floor Estimator</td>
<td>44</td>
<td>49</td>
<td>41</td>
</tr>
</tbody>
</table>

• Coder Efficiency on Top-Level Block
  – Saved ~100 Cycles/Sample using Coder compared to handwritten code.
Benefits

- High-level Model-based programming tool
  - Conversion to optimized firmware taken care by the prototyping flow
  - No hand-written code for the algorithm – robust, faster time-to-market
  - Faster and easier debugging
  - Probe points for hardware debug
  - Bit-exact correlation
  - Platform independent – easily portable to newer architectures

- Real-time Application to control FPGA, Silicon and Simulink
  - Complete firmware/hardware stack from MATLAB call through USB, I2C into target H/W
  - Fast and smooth
    - Needed for the tedious sound tuning of new algos as well as algo verification and debugging
Impact

• Impact to Business
  ➢ Faster turnaround time
    • Recently, we were able to make a new Audio Algorithm available for SOC ROM within a week of release. In the past, using the typical workflow, firmware development itself would have taken about eight man-weeks; test and integration would have taken several more man weeks.

• Impact to Development
  ➢ Easier management of algorithm source
    • Managed at a design level
    • Target architecture agnostic
  ➢ Platform independent and therefore make cross-platform deployment faster.

• Impact to Testing
  ➢ Faster iteration between development and testing.
  ➢ Easier to bit-exact verification.
Challenges

• Optimality
  – Achieved using s-functions
    • Limited debug capability
    • May result in sub-optimal code in the vicinity of the block
  – Block-level replacement techniques are being explored to overcome these limitations.

• Block Processing
  – Code generation for block-based signals with recursive structures has limitations.
  – S-functions can be used to work around the limitations temporarily, but they reduce the overall efficiency.
Conclusions

• Rapid prototyping flow speeds up solution development and results in efficient and faster debug of end-product.

• The prototyping flow has already been deployed on SOC.

• Development work on having a larger set of optimized primitives in the DSP library is ongoing.

• Work on standardizing and simplifying the build and deployment process is in progress.
Q&A