MATLAB Simulink Code generation for industrial inverters: myths & techniques

Dr Robert Turner, R&D, ABB Napier. 10 May 2016
AC 800PEC Product Presentation

Introduction

- Setting the scene
- Control model development the “old way”
  - Its problems
  - Scaling up
- Code generation for control models
  - Simulate all parts of system
  - Modelling philosophy
- Code generation
  - What to expect
  - Myths
- What we’ve achieved

Goal: Productivity gains through automation
Setting the scene
What we make and why we need code generation

We build large grid-connected inverter products by connecting many smaller inverters in parallel

- Each inverter is smart
- System controller performs application control & interface
- Each inverter is $250\text{kVA} @ 480V_{\text{L-L, RMS}}$
- Systems can be further paralleled!
May 16, 2016

© ABB Group
Setting the scene
Many parts, many control algorithms

- Smart PEBBs with best-in-industry control algorithms
  - High control bandwidth
  - Ensure all working together & accurately share current
  - Individual diagnostics & protections
  - Different operating modes
- System controller
  - Coordinate PEBBs coming & going
  - Application functionality
  - Inter-system application coordination
The “old way”

Problems

- Time consuming & redundant

Simulate
- Develop & test control algorithms

Translate
- Discretise
- Floating to fixed point
- Identify standard blocks

Code
- Write blocks
- Copy-paste for each phase

Debug
- Identify differences between implementation & sim
- Update sim

Prone to human error

May 16, 2016 | Slide 7

© ABB Group
Simulation Productivity
Simulation matching the hardware

- For a simulation with closer match to hardware need to model all functional parts:
  - Controllers & communications – where code generation comes in
  - Interfaces (ADCs, PWM etc)
  - Plant (electrical, mechanical etc)

- For simulation speed, component models can often be simplified for frequencies-of-interest. Potentially simplify:
  - Component parasitics. Are they necessary?
  - “Really fast” blocks eg FPGA IP like PWM modulators. For closed-loop control algorithms a linear modulator which accurately captures effective delays is sufficient and achieves a huge simulation speed-up
  - ADCs. We probably don’t really need to model the internals of a SAR ADC!
Modelling Philosophy

Model cohesion

Cohesion: the degree to which the elements of a module belong together

None
- No abstraction; everything at the top level

Temporal
- Grouped based on sample rates

Domain
- Grouped based on domain, e.g., control and electrical

Functional
- Parts of module are grouped because they contribute to a well-defined task
Modelling philosophy underpins long-term maintainability and productivity

- Going beyond “little” Simulink models
- Concepts from software engineering. Many can write c / c++, but not many can write good c / c++
- Strive for functional cohesion

- With effort, Simulink can achieve functional cohesion
  - Simulink Project nesting assists in organisation & reuse
  - Heavy use of libraries and model references
  - Careful consideration of physical domain modelling
Making code generation work for you

Myths

- The big one: Code generation must be less efficient than hand written
  - By definition, substitution must be as good, so benchmark is that code gen is at least as good
  - Code gen can make optimisations that allow model verbosity but produce optimum code

- What about Fixed Point?
Making code generation work for you
What to expect

- Productivity gains, and not just from where you expect them
  - Rapid change turn around
  - Good bye human errors!!!!!

- There are still risks that simulation can’t capture!
What we’ve achieved

- Truly functional components
  - Library of functional components with same interfaces as hardware counterparts

- Simulation harnesses targeting different aspects
  - Single & multiple inverter performance
  - Thorough inter-inverter communication tests
  - Automated regression tests
  - Single & multiple inverters in a product with different grid & load conditions

- Typical product model has:
  - ~10 Simulink Projects
  - ~10 libraries (each with automated or manual test harnesses)
  - Many grid scenario harnesses

- Developer productivity is easily order-of-magnitude higher
Code gen for controllers
System simulation harnesses

- Simulation and code generation of system & products.
- Simulate integration with customer grid, load and other systems
Simulatable functional blocks include all functional parts. For example a PEBB, a PEBB cabinet, or complete product.

Blocks have functionally equivalent ports, e.g., communications & electrical ports (and monitoring).

Including:
- Code-generation model (Model Reference)
- Interface (including FPGA sim)
- Electrical circuit (Plecs / SimScape)
Modelling layout
System functional cohesion

- Nest functional blocks. For example PEBBs inside cabinet; PEBB cabinets & master & electrical in product; multiple products in a system harness.
Communications harness
What we’ve achieved

Some unexpected awesomenesses

- Simulink is pretty good for developing communications protocols
  - Communications is fundamentally data flow
  - Area of Simulink getting development attention – some workflows needed massaging to ensure efficiency
  - Invaluable being able to simulate bugs found on the hardware
  - Simulation of non-controller parts (eg FPGA) good for proving interface specs

- For large & long-term developments, refactoring efficiency is key
  - Programmers have different abilities
  - Concepts evolve
  - Eases transition from rapid development to structured
Simulink Code Generation

Conclusions

- With careful modelling layout consideration, Simulink can achieve functional cohesion, which is key for large projects
- Code generation offers huge productivity improvements
  - *Safer* than hand coding
  - *More* efficient than hand coding
- Having 1-to-1 model of hardware is invaluable for rapid controller development
Discussion
Power and productivity for a better world™