Automatic Code Generation

EECS 461
Winter 2008
jeffcook@eecs.umich.edu
Topics

- Code Generation from Models
- Model-based Software Engineering
- Lab 8

Model-based SW Engineering: Process

- Control law validated by simulation and rapid prototyping
- Executable software specification (algorithm model)
- Software Development (or automatic generation)
- HIL verification of embedded implementation
- Models permit V&V at every step of the process from requirements to implemented code.
Examples: Automatic Code Generation

- Deep Space 1
- Deep Impact
- Pluto-Kuiper Belt
- MER
- MRO
- MESSENGER
- X-37
- DAWN

Advantages of Automatic Code Generation (dSPACE website with editorial comments in red)

- Less time-consuming and error-prone than hand coding
- Shorter development times, often reduced by more than 40%
- Model and C code always consistent (really?)
- Uniform standard for coding (really?)
- No implementation errors (assuming the model is correct!)
- Code documentation always up-to-date (really?)
Disadvantages of Automatic Code Generation

- Code size (not as big a problem as it once was)
- Integration with legacy code
- Consistency between model and code (temptation to tweak the code rather than revise the model and re-generate)
Model-based Software Engineering: Statistics

Model-based Software Development vs. Historical for Major Powertrain Control Project

More time on modeling, validation, verification

Model-based software works "first time"

-43%

Cumulative Errors

1st Release

Time

J.A. Cook 07March08
Automatic Code Generation Tools

- **dSPACE Targetlink**
  - Code generation from Simulink/Stateflow
  - Extended Targetlink block set for fixed-point code generation and implementation specific information

- **ETAS ASCET**
  - Code generation from ETAS graphical modeling environment
  - New product supports translation from Simulink/Stateflow

- **National Instruments LabVIEW**
  - FPGA code generation from LabVIEW “Virtual Instrument” modeling environment

- **The MathWorks**
  - Code generation from Simulink/Stateflow
  - Real-time Workshop (RTW) and RTW with Embedded Coder
Lab #8: Automatic Code Generation from Simulink Models

- Adding program from LAB #1
  - Implement as Simulink model and code generate
- Spring-mass-damper virtual world
- Double spring-mass-damper
  - Fast and slow systems
  - Multitasking
- Please read through the full lab document since the format has changed for this lab
Lab #8: Hardware Specific Functions and Low Level Operations

- Lab #1 adder requires low level (“bit pushing”) operations in Simulink – how do we do this?
  - Simulink block set
  - S-functions
- Hardware I/O and processor initialization?
  - Special Simulink blocks from Freescale
- Real-time Workshop and Embedded Coder from TMW for code generation
Lab #8 Part 1: Bit Manipulation - 32 bit unsigned integer into four 8 bit unsigned integers
Bit Manipulation and Low-level Operations

- Signal Attributes/Data Type Conversion
- Logic and Bit Operations/Bitwise Operator
- Logic and Bit Operations/Shift Arithmetic
Lab #8 Part 2: Virtual Mass-Spring System

- Simulation model:
  - Select $k$, $J\omega$, $b$, and $T$
- C code to implement on the $\mu$P has hardware specific tasks:
  - Get wheel position from QD function of eTPU
  - Convert wheel position from eTPU in encoder counts to degrees
  - Convert calculated torque in N-mm to duty cycle
  - Update duty cycle and send to PWM function of eMIOS
  - Do data type conversions
  - Initialize eTPU and eMIOS
- How do we do all these things in a model?
Freescale RAAppID

- Special Freescale block set in Simulink Library Browser
- Move from simulation environment to implementation without writing low-level C code
- Microprocessor initialization and peripheral device set-up blocks
Freescale RAAppID Toolbox for μP and Peripheral Initialization
Top Level Spring-Mass-Damper Model: Execution Timing

Triggered subsystem executes at the periodic rate specified by the function call.
Inside the triggered subsystem, we need functions to read wheel position and convert counts to degrees, and output torque as PWM signal.
Freescale
RApIID

J.A. Cook 07March08 18
Device Driver Blocks: Quadrature Decode

- Special Simulink blocks from Freescale
- Configure eTPU for QD
- Encoder counts to wheel angle, degrees
Device Driver Blocks: eMIOS PWM

\[ DC = \frac{T \times 18}{773.4 \times 128} + 0.5 \]
Some Other Details Before We can Code Generate

- Update rate
- Parameter initialization

J.A. Cook 07March08
Function Generator

Source Block Parameters: Fast Task Trigger

- Function-Call Generator (mask) [link]
  
  This block implements an iterator operation. On each time-step as defined by the sample time field, this block will execute the function-call subsystem(s) that it drives for the specified number of iterations.

  Demux the block’s output to execute multiple function-call subsystems in a prescribed order. The system connected to first demux port is executed first, the system connected to second demux port is executed second, and so on.

- Parameters
  
  Sample time: [T]

  Number of iterations: [1]
Model Parameter Initialization

- From the Simulink model window
- File/Model Properties/Callbacks/InitFcn
- Assign values in an M-file specified in the initialization window without the .m suffix
Triggered Subsystem

Simulink/signal routing/environment controller
Lab#8 Part 3: Tasks, Priorities and Shared Data

- Spring-Mass-Damper
  - Single task rate
  - No shared data
- Real system
  - Multiple tasks
  - Rate monotonic priority scheme
  - RTOS
  - Shared data

Double Spring-Mass-Damper System: 2 tasks with different sample rates
Multi-rate System: 2 S-M-D

- Fast S-M-D is 10 times faster than slow system
- Separate tasks at different rates
- Fast and slow systems have different integration time steps
Rate Transition Blocks

- Deterministic transfer of data with data integrity between blocks operating at different speeds at the cost of maximum latency of data transfer
  - ZOH for fast-to-slow transitions
  - Unit delay for slow-to-fast transitions
Code Generation

- Processor initialization at highest level
- Device driver blocks inside the fast system
- RTW code generation
  - Tools-&gt;Real-Time Workshop -&gt;Build Model
Real Embedded Software

- Large, complex, developed by many people, integrated at the end, and expected to work.
- Typical automotive control “feature”
  - Much more than just the control law
  - Multiple versions address program-to-program variability
  - Average feature has
    - 1-2 execution contexts
    - 20 inputs
    - 14 outputs
- ~60-100 features per vehicle with more than 2000 connections among features
Model-based Software Engineering

- Modeling environment requires
  - Flexible, interchangeable and reusable model components
  - Seamless process for component “plug-and-play”
  - Data and complexity management
  - Systems and software analysis tools
Style Matters

- Models must be clear, readable, modular, documented and precise
- Automatic code generation does not eliminate human error – just moves it higher in the process
- Order of execution, execution context, data types – must be specified in the model!
- Naming conventions, data scoping, annotations and comments, ...

Reference: “Style Matters - Applying the lessons from the software industry to Autocoding with Simulink” by Peter Gilhead, Ricardo Tarragon
Hatley-Pirbhai Model Methodology

- Simulink diagrams model data flow; Stateflow diagrams model control flow
- Process specifications (P-specs) modeled using Simulink blocks and/or Stateflow diagrams, depending on the nature of the algorithm
- Control specifications (C-specs) are modeled using Stateflow
- One Simulink subsystem per execution context (10ms, 100ms, etc.)

Style Matters

- Attempt to form diagrams that have no crossing signal lines
- Use consistent style for readability and documentation