Accelerating FASGW(H) / ANL Image Processing with Model-Based Design

Issue 2

Andy Sherriff
Technical Expert Image Processing
MBDA UK Ltd
7th October 2015
Introduction

• Why Model-Based Design is of interest to MBDA?
  • Quicker to market
    - Fewer iterations
    - Savings on back-back testing and reviewing
  • Development efficiencies
    - WYSIWYG, automatically generated code directly into Target Processor
  • Robustness
    - Fewer human mistakes as there's less textual interfacing of requirements

• Application context for this presentation is FASGW(H) / ANL
  • FASGW(H) / ANL is adopting Rapid Development principles
    - Aggressive schedule
    - Requires innovative engineering approaches
What is FASGW?

- Future Anti-Surface Guided Weapon (FASGW)
- Family of guided weapons comprising of light and heavy missile systems
- Combination of light & heavy capable of defeating wider target threat
- “FASGW will improve the firepower and effectiveness of the Royal Navy ‘Helicopter Maritime Attack‘ (HMA), currently consisting of the Sea Lynx carrying the Sea-Skua missile”.

What is FASGW(H)/ANL?

- Cooperative development signed by UK & French governments
- MBDA chosen as lead
- French requirements for ANL (Anti-Navire Legere) are similar to British MoD FASGW(H) requirements

My Role on FASGW(H)/ANL

- Responsible for the FASGW(H)/ANL Image Processing algorithms using Model-Based Design and methodology
- Role matured since successful delivery on Fire Shadow
FASGW(H) / ANL Summary

• Versatile
  • FIACs up to Corvettes in complex littoral environments
  • Capability against coastal land targets

• High launch platform survivability
  • Launched at standoff ranges

• High precision
  • IR guidance with Man-in-the-Loop capability allowing target selection / impact point selection / abort

• Start of Development early 2013
• Requires high IP maturity early in D&M
FASGW(H) / ANL – Automatic code generation from Simulink

- Simulink automatic code generation for mainstream project
**IP Design Intention**

- **Common model and embedded target processor code**
  - Desirable to reduce time & cost

- **Use Embedded Coder to produce the embedded target processor code from the model**
  - Fire Shadow was the first MBDA full scale development project to adopt this approach
  - Now adopted on new MBDA full scale development projects

- **Achieved Embedded Coder code quality**
  - Use
    - Embedded MATLAB
    - Embedded MATLAB within Stateflow Charts
    - C-code within Model Source blocks and Stateflow Charts
      - Sparingly – Pointers on large structures, Telemetry and Timing Markers
    - Standard Simulink building blocks & toolboxes
  - Don’t Use
    - S-functions, m-scripts
Use Reference Models

- Allows IP team to work on the algorithms simultaneously
  - Easily maintained in a configuration system e.g. Dimensions
  - Requires sensible algorithmic split – not just for the sake of it
- Enforces strict interface definition
- Allows better configuration for sub-model testing & re-use
- Allows combination of variable and fixed step solvers
• As with any tool, within MATLAB/Simulink there are a number of ways to do the same task
  • Some work well
  • Some work not so well

• Code of Practice defines the standards and guidelines used within MBDA
  • Formal models and algorithm definition
  • Particularly when definitions are to have code automatically generated to product software

• Code of Practice has been developed in conjunction with MathWorks
  • Over a number of years; a number of projects; to determine what works and what doesn’t,
  • Regularly updated
  • This can be supported by the use the Model Advisor tool
Model Advisor Tool

- Provides a report based analysis of either MATLAB code or Simulink diagrams for compliance with a CoP

- Cannot guarantee that MATLAB or Simulink models are correct

- Can be used to check for general layout and use/absence of certain constructs within a model
  - ensure that development standards and CoPs are being complied with
  - ensure consistency within a project and across projects
  - increase the potential for reuse

- If all Simulink models comply with the CoP, then they are consistent in terms of structure, style, and appearance

- The CoP is good practice, that greatly enhances productivity and portability

- This approach will save time and money in conducting reviews since only the configured Model Advisor needs to be reviewed
The Model Advisor Tool is configured by establishing rules that must be complied with during an algorithm development.

Facilitated from Model Advisor Settings Application Tool:

- Enter Simulink model to test.
- Codes of Practice Excel spreadsheet that contains the ‘Model Advisor Settings’ sheet.
- Checks consistency of the selections and writes the test settings to a file ‘MAchecksConfig.txt’.
Model Advisor Example for CoP v2.8

35 MBDA specific tests selected from 46

26 MathWorks standard tests selected from 132

61 tests in total
Simple Code Generation Example

• In the CoP, it is recommended that the Math Function block, when set to POW and an integer power (u2, u3, etc), is not used for code efficiency reasons.
Target Timings Process

- Target Timing markers are automatically introduced around each Reference Model during code generation
  - Optionally controlled via `EnableTiming` flag

- The automatically generated code is transferred onto a standalone Compaq netbook, with an Intel ATOM N270 1.6GHz CPU

- Compiled using GCC under a Linux operating system

- Run against a consistent set of FASGW Open Loop scenarios

- Times between the timing markers are counted and output to an external file

- Test Vectors are output which are compared between the desktop and the Target for algorithmic consistency

- The limitation of this approach is that all target timings only apply to the IP algorithmic functions and data transfers; they do not account for any software infrastructure
• **Using a commercial P2020 Reference Design Board (RDB)**
  • 1.2GHz CPU Cores, Platform 600MHz, and DDR3 memory 800MHz
  • Contains two e500v2 cores, each equivalent (on paper) to a single Intel Atom Z530 core
  • The board has been configured to FASGW P2020 requirements

• **P2020 RDB includes a pre-built Linux 2.6.35 Kernel held in NOR flash, which allows the board to boot and communicate with the Compaq netbook**
  • Allowing download and execution of the FASGW IP components
  • As a single or dual threaded application
MATLAB Multicore Processor Target Support

• Automatically generate code for algorithms from Simulink models
  • Typically single threaded algorithms have developed within MBDA
    - Limited to the processing capacity of a single processor core
    - Not scalable for future multicore targets
  • Require methods for producing multi-threaded code
    - Without extensive hand coding/modifications
    - Supported by embedded targets (currently PowerPC, ARM in the future)
    - Utilising the processing capacity of additional cores

• MATLAB 2011b onwards provides support for multicore targets
  • Models can be configured for “Concurrent Execution”
    - Model blocks mapped to periodic tasks or asynchronous interrupts
    - For execution concurrently on available processor cores
    - Simulink Coder produces multi-threaded code compatible with the host
      • Windows threads if MATLAB is running on Windows
      • POSIX threads if MATLAB is running on Linux
• Configuring a model from concurrent execution on a multicore target
  • Starting with a simple Simulink model
    - A counter as an input
    - Applying two gains as example operations
    - Displaying the input, intermediate and output signals on a scope

• Concurrent execution requires the model to be split into Reference Models
  • Reference Model blocks are mapped onto tasks
  • The Top Level Model must only be Reference Models
**Image Processing Algorithm Model Configuration**

- **Simulink model of FASGW IPMM used for investigation**
  - Test Harness: Input and Display blocks
  - Image Conditioning algorithms (~65% processing time)
  - Image Compression operation (~35% processing time)
  - Compression Quality Control function provides feedback to the Image Compression operation to keep image data size within required range

- **Image compression operation allocated to second task**
  - Simplest split of processing load into two tasks, optimised for dual core target
  - Avoids algebraic loop between Image Compression and Quality Control function blocks
  - Minimises overhead introduced by data transfers between cores

- **Produced a 45% increase in frame rate on the P2020 target after configuration of the model for concurrent execution using two tasks**
  - No modifications required to automatically generated code after generation
  - Only requires a Configuration file change to switch between single and dual core operation
  - Fixed Periodic time is a limitation, needs to be the maximum, often the Test Harness block
  - Large amount of time required to re-configure complex hierarchy of models for concurrent execution (days)
    - Would be reduced if the model was initially designed for concurrent execution
Summary

- All Image Processing Model Code is in
  - Embedded MATLAB
  - Embedded MATLAB within Stateflow Charts
  - Some C-code within Model Source blocks and Stateflow Charts
    - Pointers on large structures, Telemetry and Timing Markers
  - Standard Simulink building blocks & toolboxes

- Code of Practice
  - Ensures Simulink models are consistent in terms of structure, style, and appearance
  - Good practice, that greatly enhances productivity and portability (re-use)
  - Saves time and money conducting reviews

- Simulink allows
  - Visual and rigorous architecture
  - Multi-threaded automatically generated C-code compatible with the host
  - Allows rapid prototyping, albeit efficient automatically generated C-code requires experience/training
  - Clear and rigorous Interface, clearly interpretable by software
  - Diagnostics/Displays can be included and switched out during code generation
  - Easy debugging, albeit some information isn’t in the workspace viewer, and indexing
  - Rapid turnaround from complete algorithm change, through automatic code generation to test vector/timings on the target processor ~1hr

- Can only become easier with experience & later releases of MATLAB
End