The MathWorks™

MATLAB&SIMULINK

Large-Scale Modeling for Embedded Applications

Kerry Grand
Senior Technical Consultant
kgrand@mathworks.com

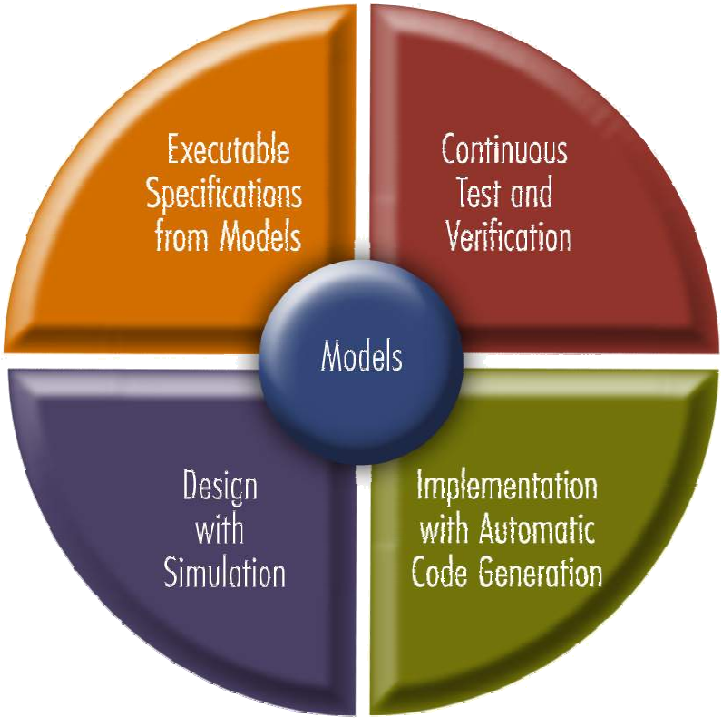
© 2009 The MathWorks, Inc.

Outline

- Introduction
- Large Scale Modeling for Embedded Systems Challenges
- Model Architecture Recommendations
- Design and Implementation Recommendations
- Next steps

Introduction

- Increasing demand for high-integrity and mission-critical embedded software
- The breadth and scope of projects Model-Based Design increasing
- Increasing team size
- Model-Based Design for large embedded systems creates a unique set of opportunities and challenges



Introduction (cont.)

- Definition of a **Large Scale Model**
  - too big for one person to know all its details
  - exceeds 100,000 blocks
  - contains over 100 inputs and outputs

- Project teams working with large models targeting an embedded application often experience a common set of challenges
  - Model architecture
  - Design and implementation

Large Scale Modeling for Embedded Systems Challenges

- How do you partition the model into manageable components to aid in parallel development and reuse?
- How do you define and manage a consistent set of interfaces?
- How do you control execution of the model?
- How does model architecture affect testing?
- How do you avoid rework?
- How do you ensure a consistent modeling environment?
- How do you select a modeling language?
- What code generation approach is appropriate?

Model Architecture Recommendations

1. Partition the top level model components using model reference architecture in Accelerator mode
2. Use atomic subsystem library models for components with fewer than 500 blocks
3. Design for portability and reusability by using Simulink data objects
4. Make buses virtual except for model reference component boundaries
5. Make your model interfaces explicit by minimizing or eliminating global data stores, global events, and global Goto/From blocks
6. Use function-call scheduling if the goal is to match the model to existing software architecture. Otherwise, let Simulink determine the best order based on data dependency analysis
7. Avoid the use of block priority to control the Simulink execution order
8. Architect the design such that engineers can independently test their areas of responsibility in the model domain
9. Implement a consistent method for signal injection and logging with the model architecture

Model Architecture Recommendations

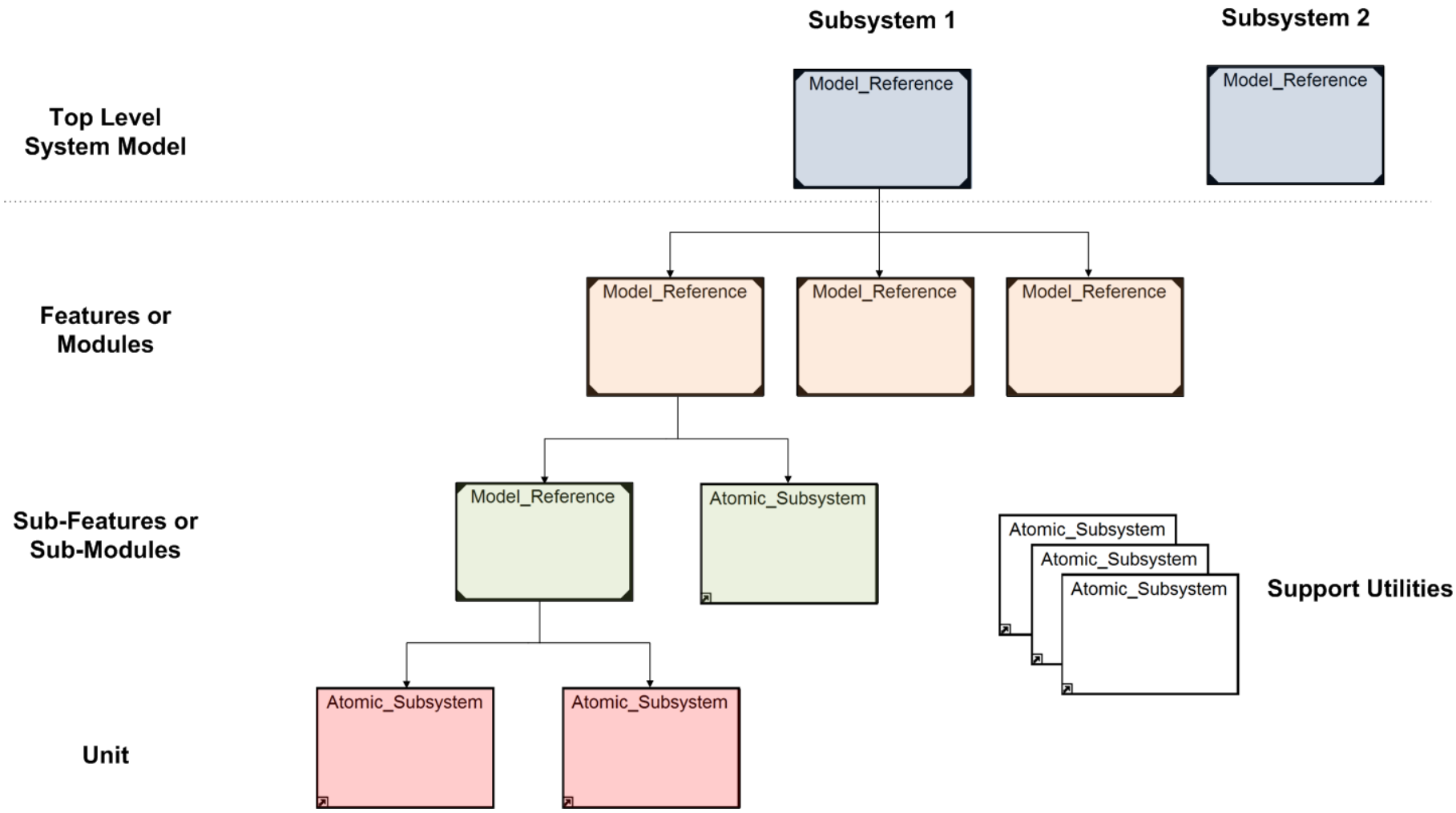
1. Partition the top level model components using model reference architecture in Accelerator mode
2. Use atomic subsystem library models for components with fewer than 500 blocks
3. Design for portability and reusability by using Simulink data objects
4. Make buses virtual except for model reference component boundaries
5. Make your model interfaces explicit by minimizing or eliminating global data stores, global events, and global Goto/From blocks
6. Use function-call scheduling if the goal is to match the model to existing software architecture. Otherwise, let Simulink determine the best order based on data dependency analysis
7. Avoid the use of block priority to control the Simulink execution order
8. Architect the design such that engineers can independently test their areas of responsibility in the model domain
9. Implement a consistent method for signal injection and logging with the model architecture

Model Architecture Recommendation – Componentization

- If a model is not properly componentized the following issues commonly arise:
  - Work requires other components or higher levels of the model
  - Exhaust CPU resources
  - Increased simulation update times

- **Recommendation 1**
  - Partition the top level model components using model reference architecture in Accelerator mode
- **Recommendation 2**
  - Use atomic subsystem library models for components with fewer than 500 blocks

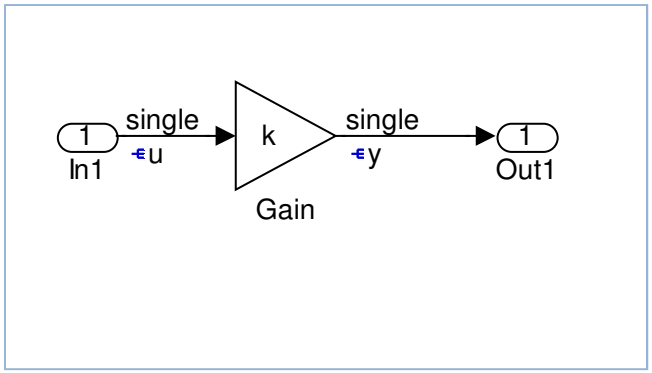
Componentization Example



Model Architecture Recommendation – Interface Management

- Uncontrolled interfaces to the model components produce a unique set of issues:
  - Unexpected model behavior due to data type mismatch
  - Model component reuse is limited
  - Integration and rework issues due to inconsistent interfaces

- **Recommendation 3 -** Design for portability and reusability by using Simulink data objects



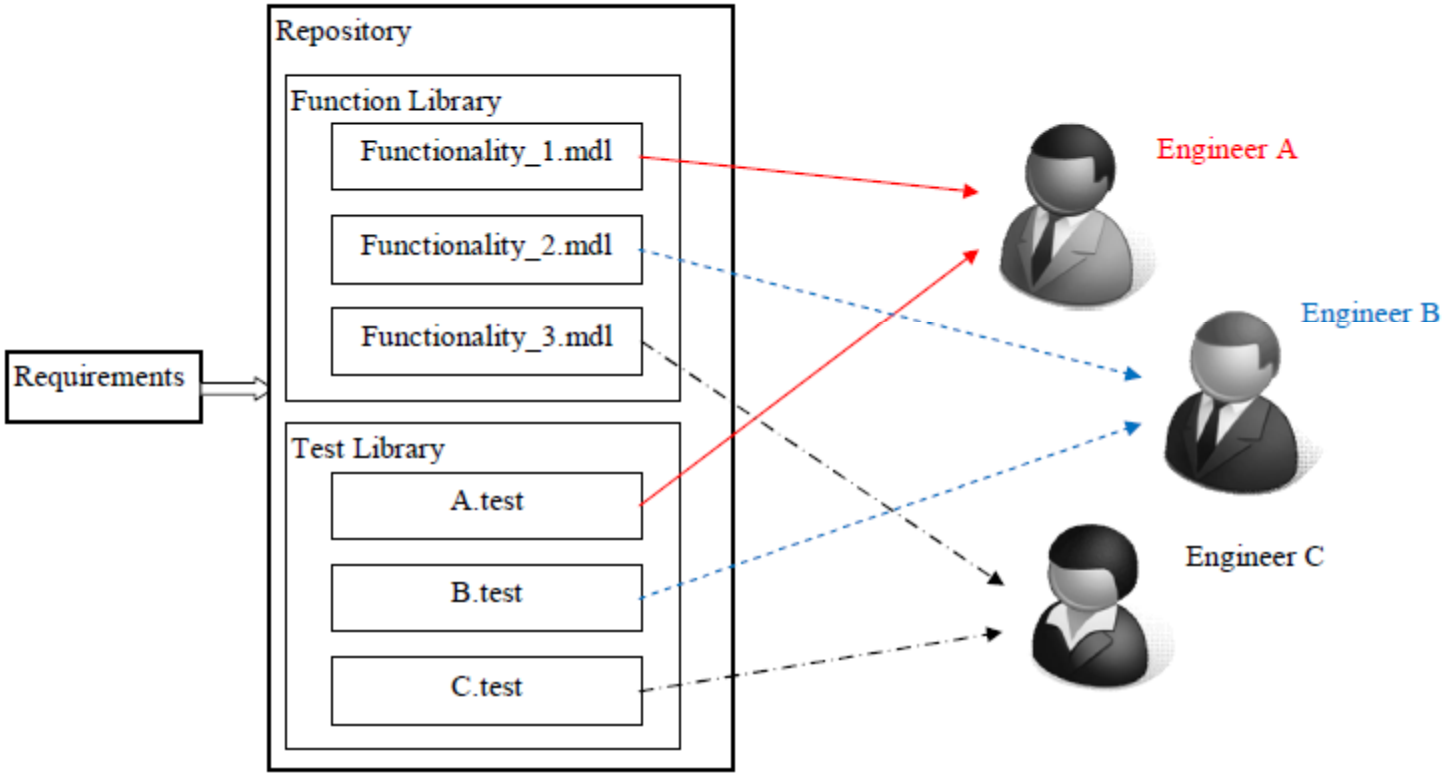

```
real32_T k = 10.0F;
real32_T u;
real32_T y;

/* Model step function */
void data_object_model_step(void)
{
  /* Gain: '<Root>/Gain' incorporates:
   *  Inport: '<Root>/In1'
   */
  y = k * u;
}
```

Model Architecture Recommendation - Architecture Impact On Testing

- Model architecture that couples components and artifacts required for unit and subsystem testing produce a set of issues:
  - Version control conflicts due to multiple engineers changing the same artifact
  - Loss productivity when one of the artifacts required for test has some sort of defect
  - Loss of productivity when one of the artifacts required for test has not be fully developed

- **Recommendation 8 -** Architect the design such that engineers can independently test their areas of responsibility in the model domain

Independent Areas of Model Testing Example

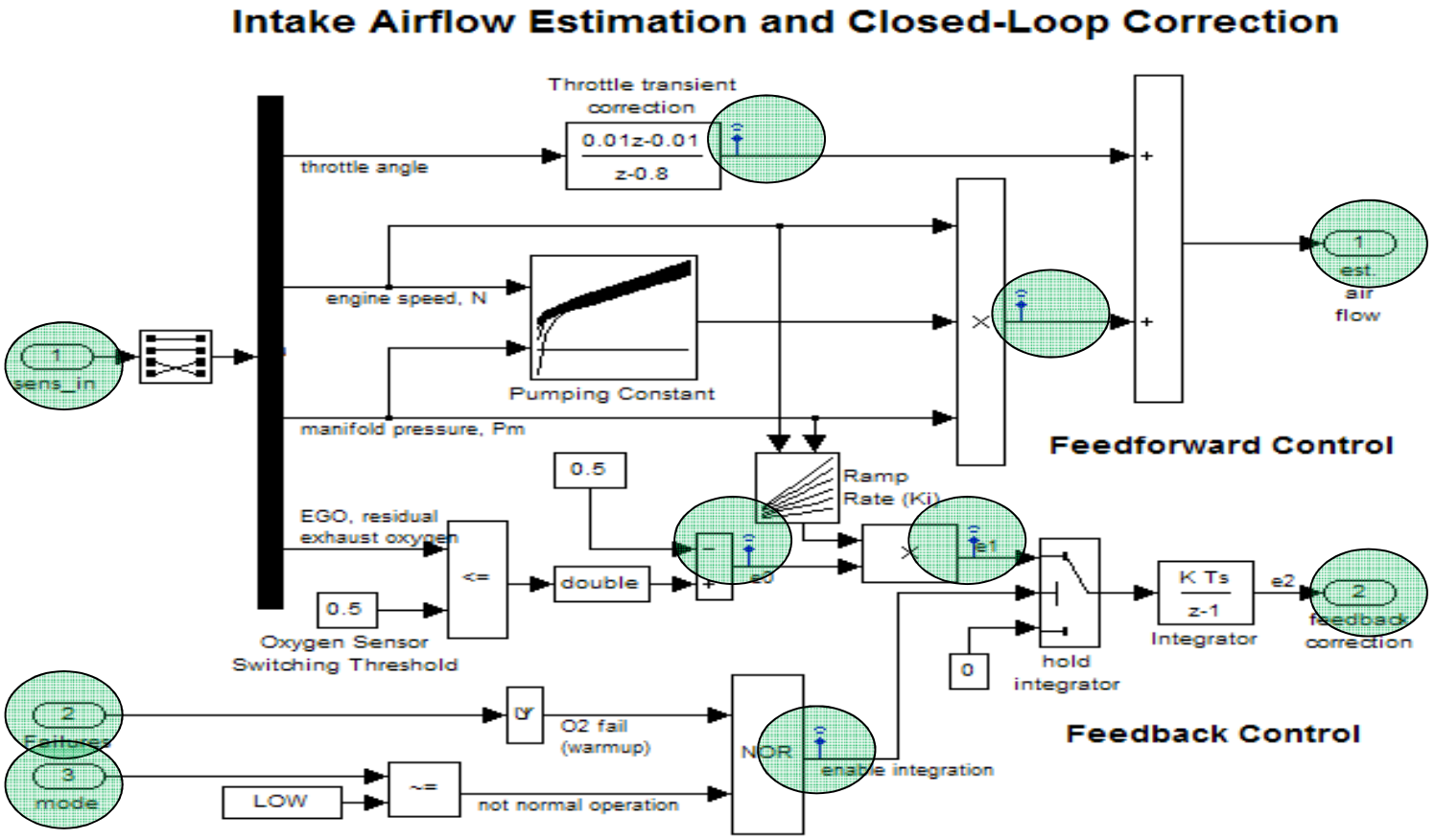


Model Architecture Recommendation - Architecture Impact On Testing

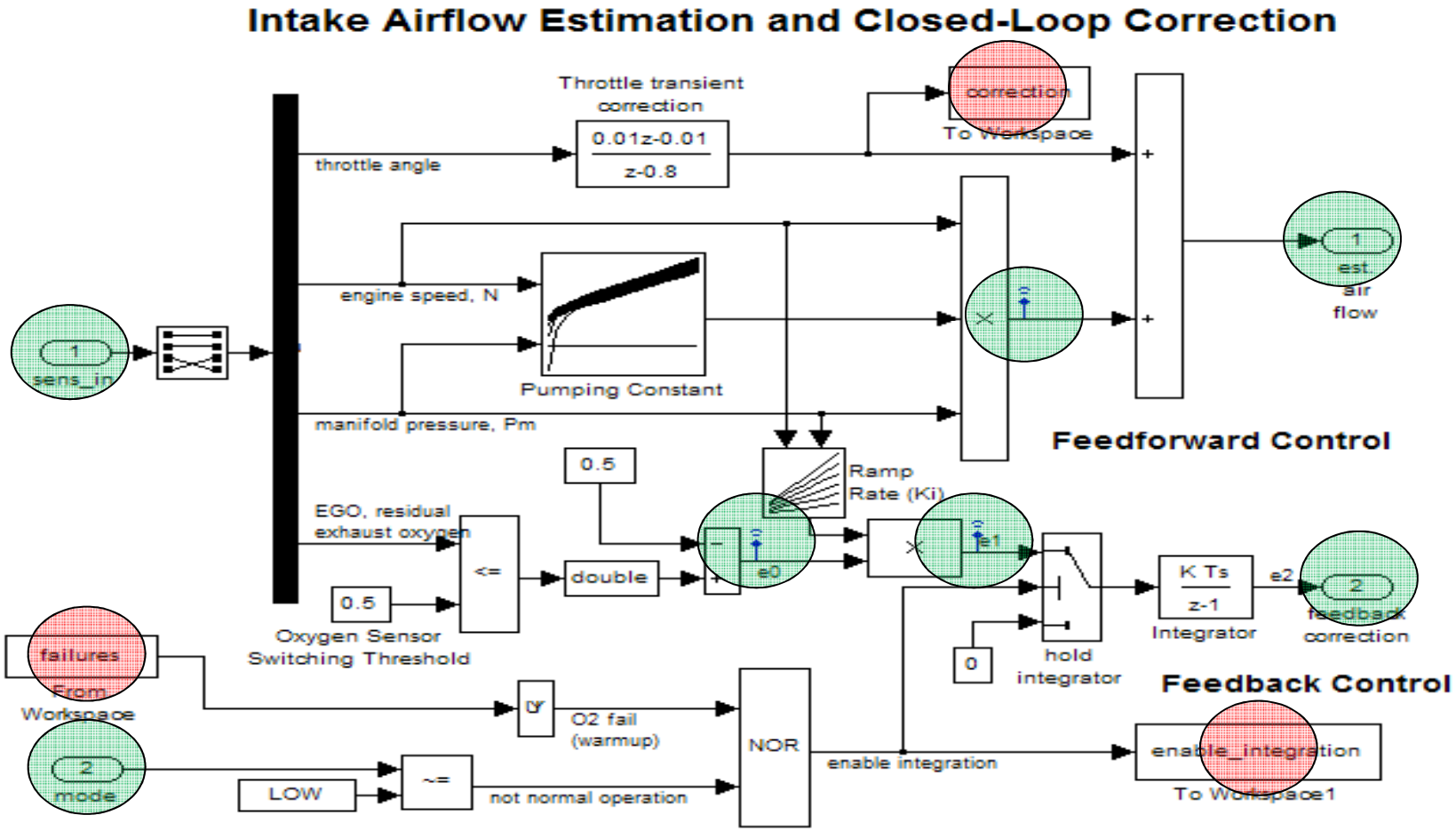
- Model architecture that does not account for test interfaces leads to the following issues:
  - Signal values and states that cannot be measured or injected due to model architecture
  - Inconsistent injections and logging mechanisms that make testing difficult to maintain
  - Model modifications required for unit testing or integration

- **Recommendation 9 -** Implement a consistent method for signal injection and logging with the model architecture

Consistent Signal Injection and Logging Example



Inconsistent Signal Injection and Logging Example



Design and Implementation Recommendations

10. Create a consistent modeling environment by using startup.m, cached MAT-file loading, referenced configuration sets, sl_customization.m, and system target file
11. Architect the model and scripting to support parallel model reference accelerator builds and code generation target builds using Parallel Computing Toolbox
12. Consider using:
    - Simulink for signal flow and feedback control algorithms
    - Stateflow for combinatorial logic, schedulers, and finite-state machines
    - Embedded MATLAB for matrix and single line equations.
13. Determine the PCG deployment strategy along with architecture design

Design and Implementation Recommendations

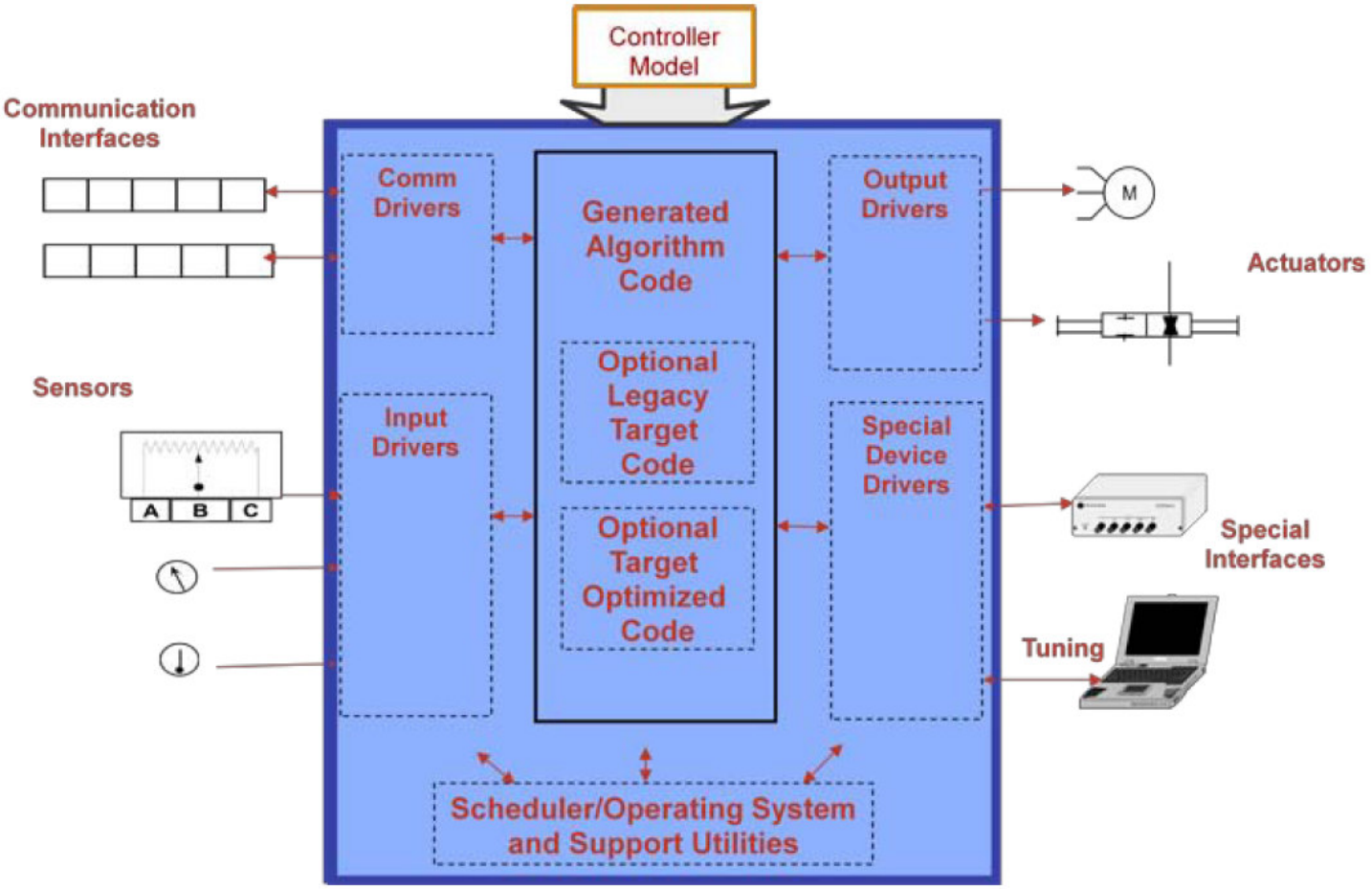
10. Create a consistent modeling environment by using startup.m, cached MAT-file loading, referenced configuration sets, sl_customization.m, and system target file
11. Architect the model and scripting to support parallel model reference accelerator builds and code generation target builds using Parallel Computing Toolbox
12. Consider using:
    - Simulink for signal flow and feedback control algorithms
    - Stateflow for combinatorial logic, schedulers, and finite-state machines
    - Embedded MATLAB for matrix and single line equations.
13. Determine the PCG deployment strategy along with architecture design

Design and Implementation Recommendations - Algorithm Export and Full Real-Time Target

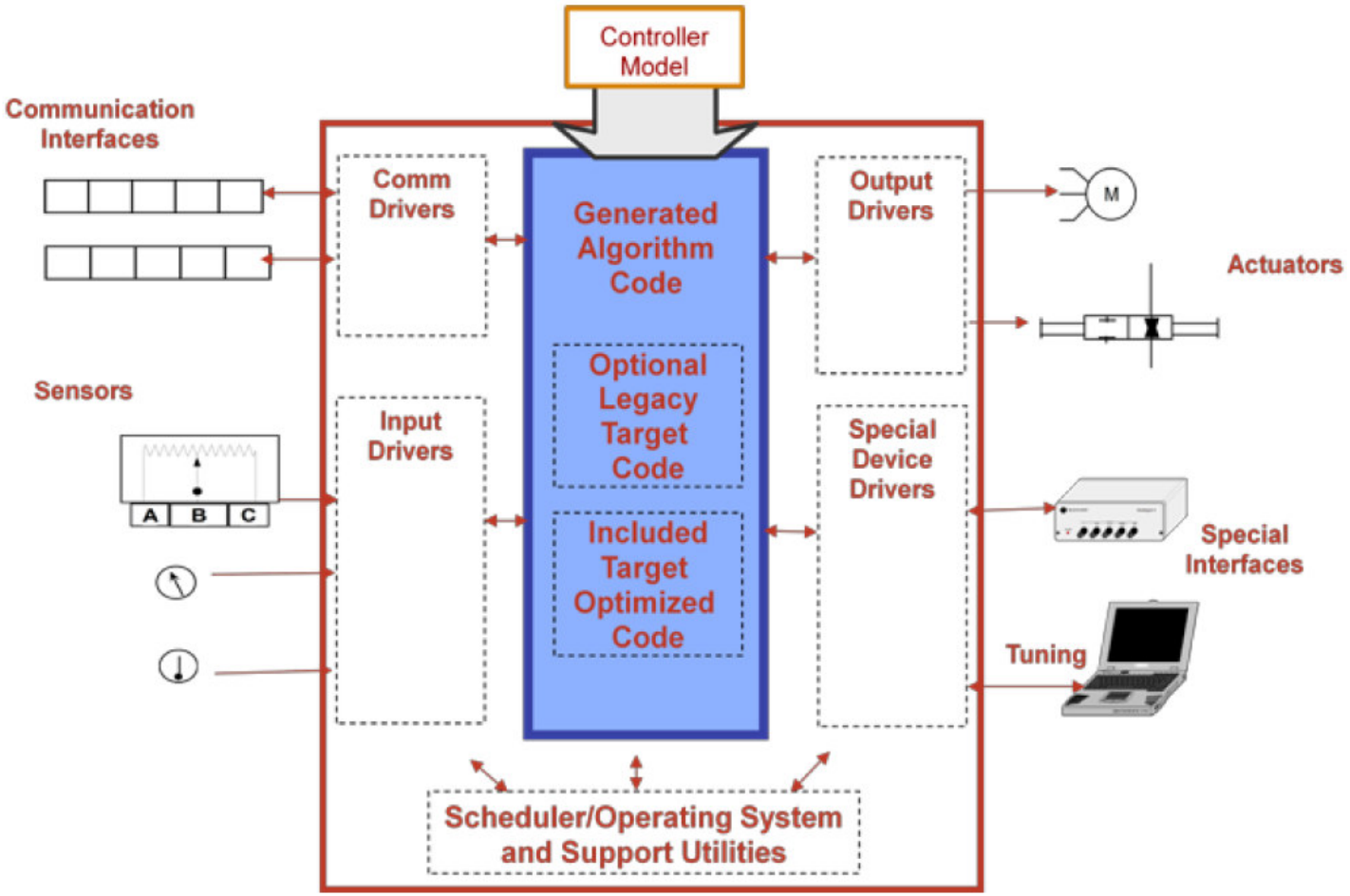
- Lack of consideration for the automatic code generation target approach can lead to the following issues:
  - Lack of portability between multiple hardware platforms
  - Limited reuse of existing software base
  - Committed resources focused on custom target creation and not product development

- **Recommendation 13 -** Determine the PCG deployment strategy along with architecture design

Full Real-Time Target



Algorithm Export Target



Next Steps

- This presentation only provides an introduction to the SAE paper 2010-01-0938, SAE 2010: Large-Scale Modeling for Embedded Applications, http://www.mathworks.com/automotive/technicalliterature.html
- This paper may provide enough detail for your organization to deploy to an embedded target using Model-Based Design
- MathWorks Consulting has experience working with customers who deploy Large Scale Embedded Applications so talk to your MathWorks account manager to see how we can support you

Thank You!