

The MATLAB logo is a stylized 'M' composed of five overlapping triangles: a blue triangle at the top, a red triangle at the bottom, a green triangle on the left, a yellow triangle on the right, and a cyan triangle in the center. The background of the entire image is a close-up, shallow depth-of-field photograph of several hands being clasped together, with a person's wrist and a gold bracelet visible in the lower right.

MATLAB EXPO 2018

KOREA

MATLAB EXPO 2018

New Perspective for Large and Complex Production Software Development

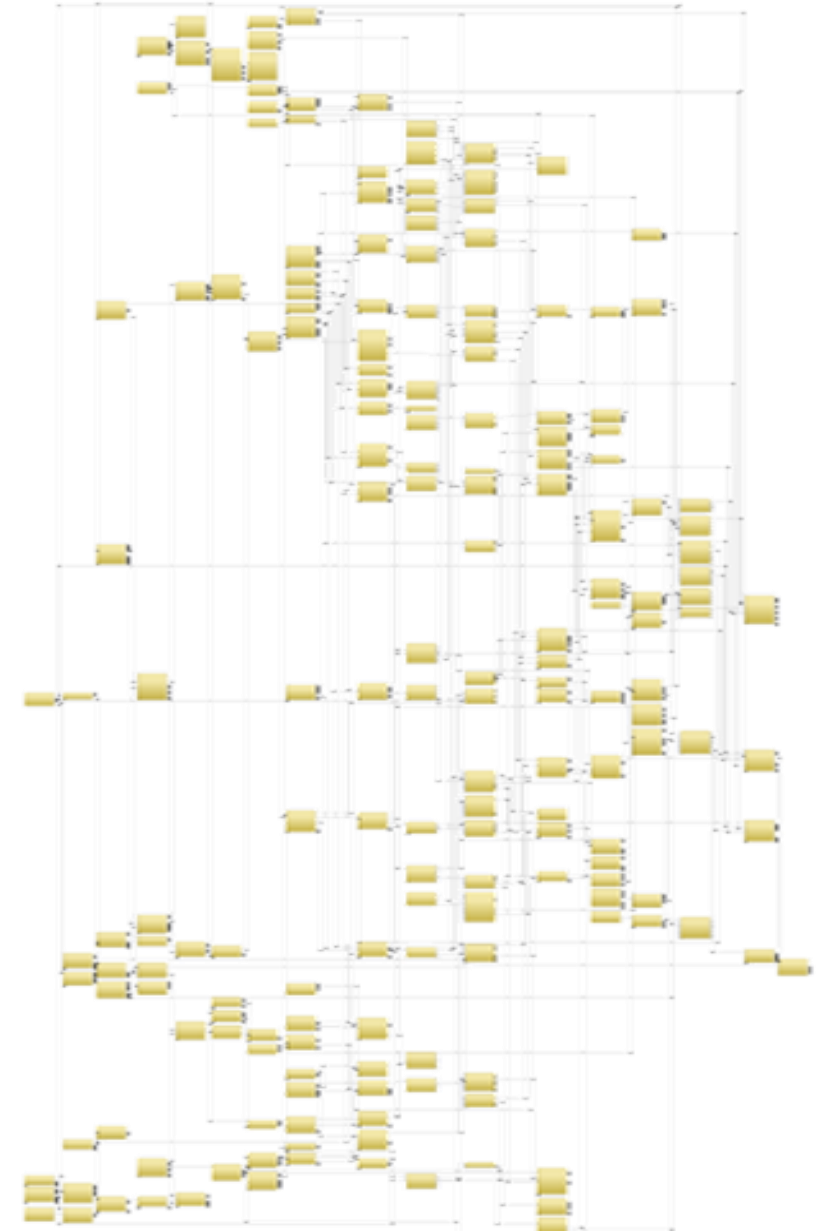
대규모 **SW** 개발에 적합한 모델링 패턴 및 코드 생성 방안

류성연 차장



Issues for Large-scaled Embedded Software Development

Issues \ Work Phase	Modeling	Code Generation
1. Complexity	✓	
2. Integration (Reusability + Scalability)	✓	✓
3. Scheduling	✓	
4. Multi-instantiation		✓

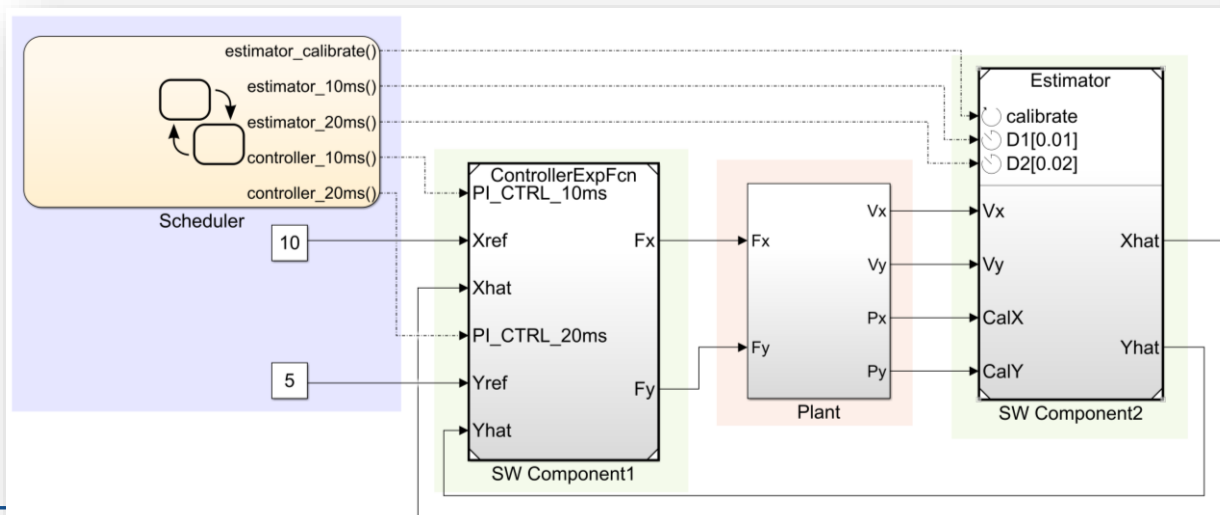


What to Consider for Model-Based Design

- Component-based design
- Integration in a composition level
- Component scheduling
- Code generation on SW frameworks
- Generated code customization

SW modeling patterns

Code generation workflow



AUTOSAR

ARINC

ROS

Your-custom Framework

For Software Modeling Patterns

Example: Throttle body control system

Throttle Control Model

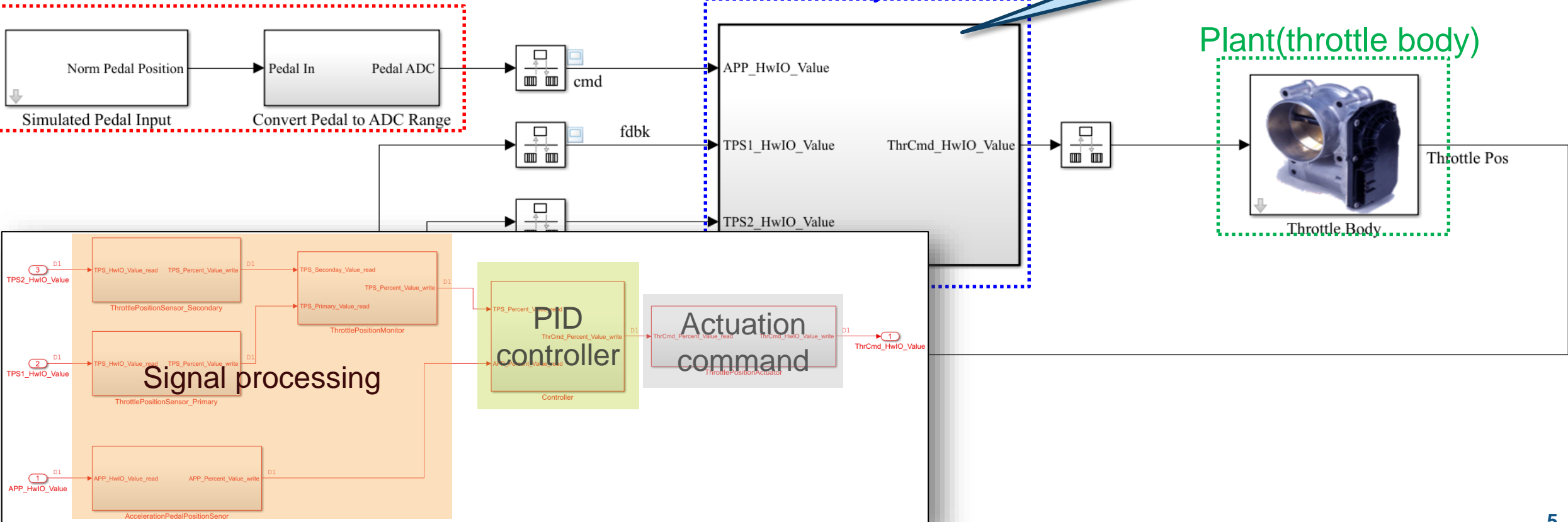
Copyright 2017 The MathWorks, Inc.

Code generation model

Acceleration pedal Input

Throttle body controller

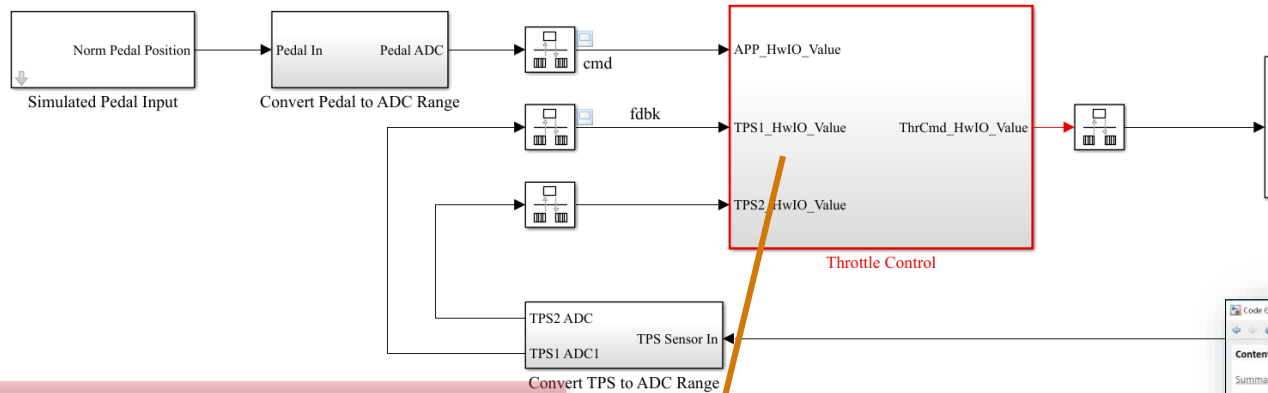
Plant(throttle body)



Inadequate Software Modeling & Code Generation

Throttle Control Model

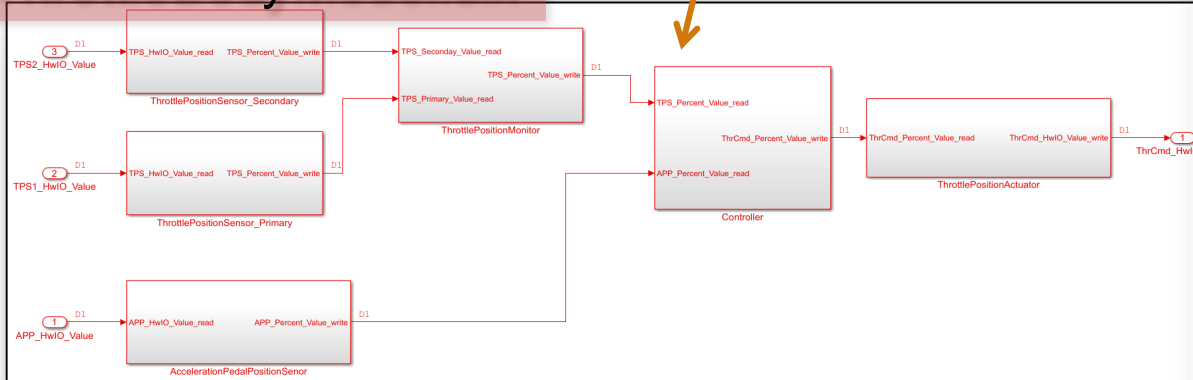
Copyright 2017 The MathWorks, Inc.



❖ Not reflecting SW architecture

- 1) Modeling in one Simulink file
- 2) Generated code in one function and one file
- 3) Hard to analyze interfaces among units
- 4) Unit execution orders are predefined

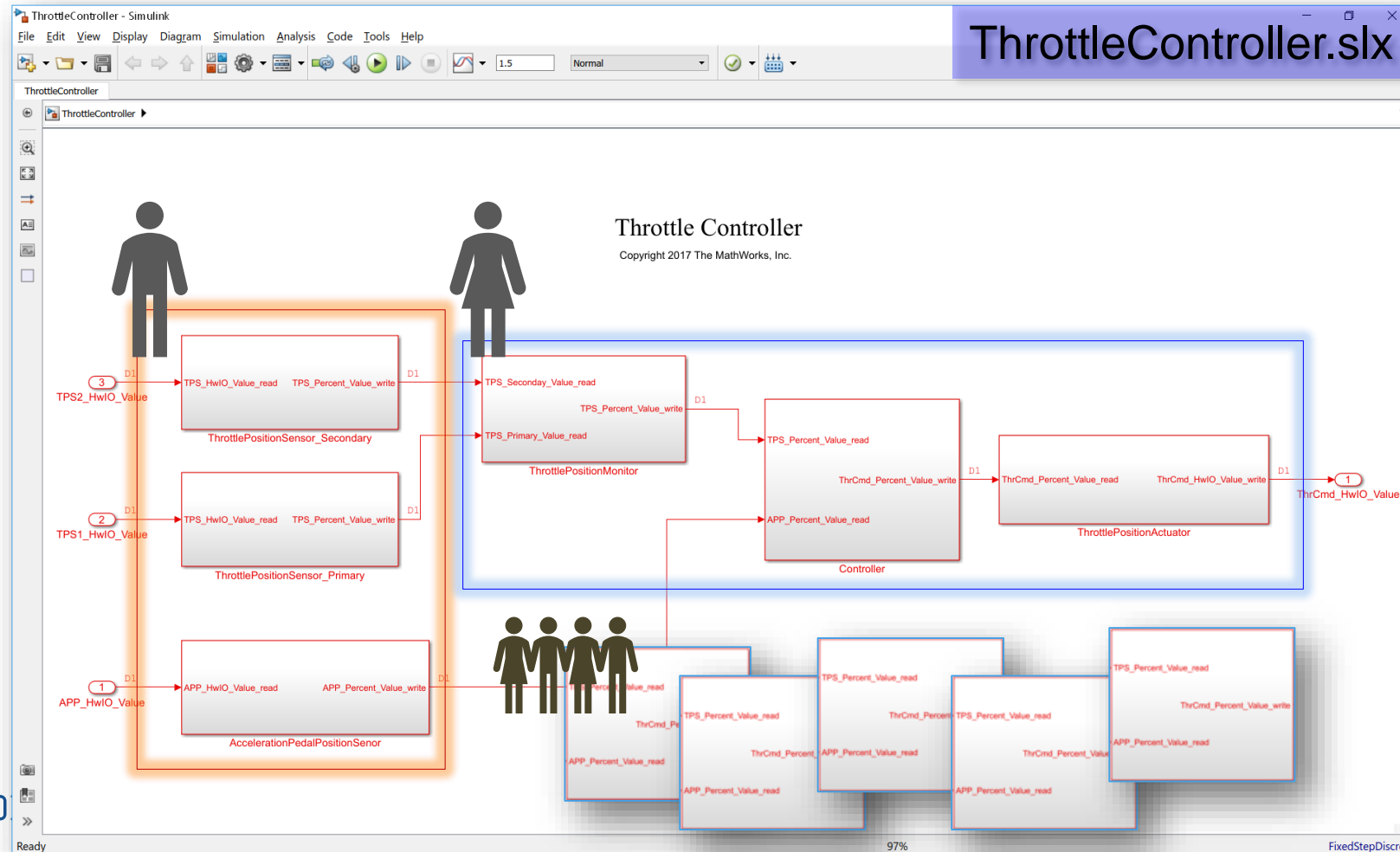
ThrottleBodyModel.slx



Not adequate for larger-scale software !!

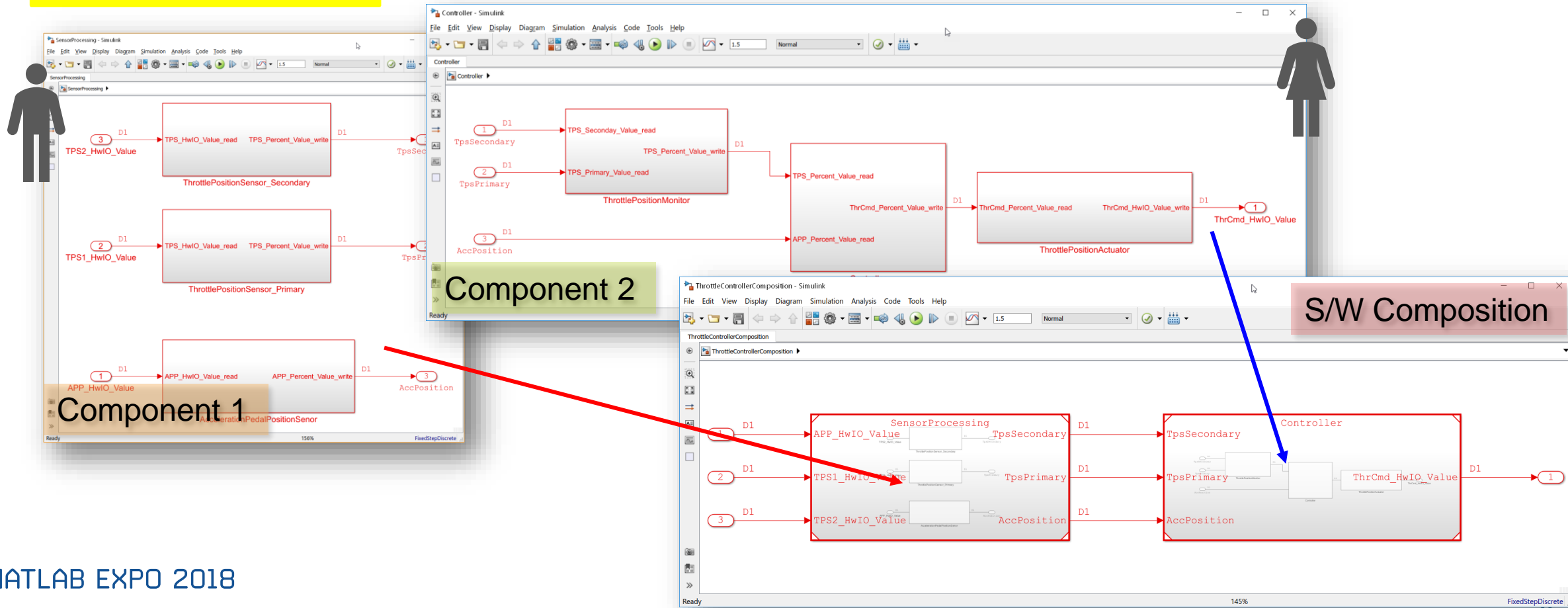
Let's Start from Software Architecture

- If there are many models from other developers or teams...



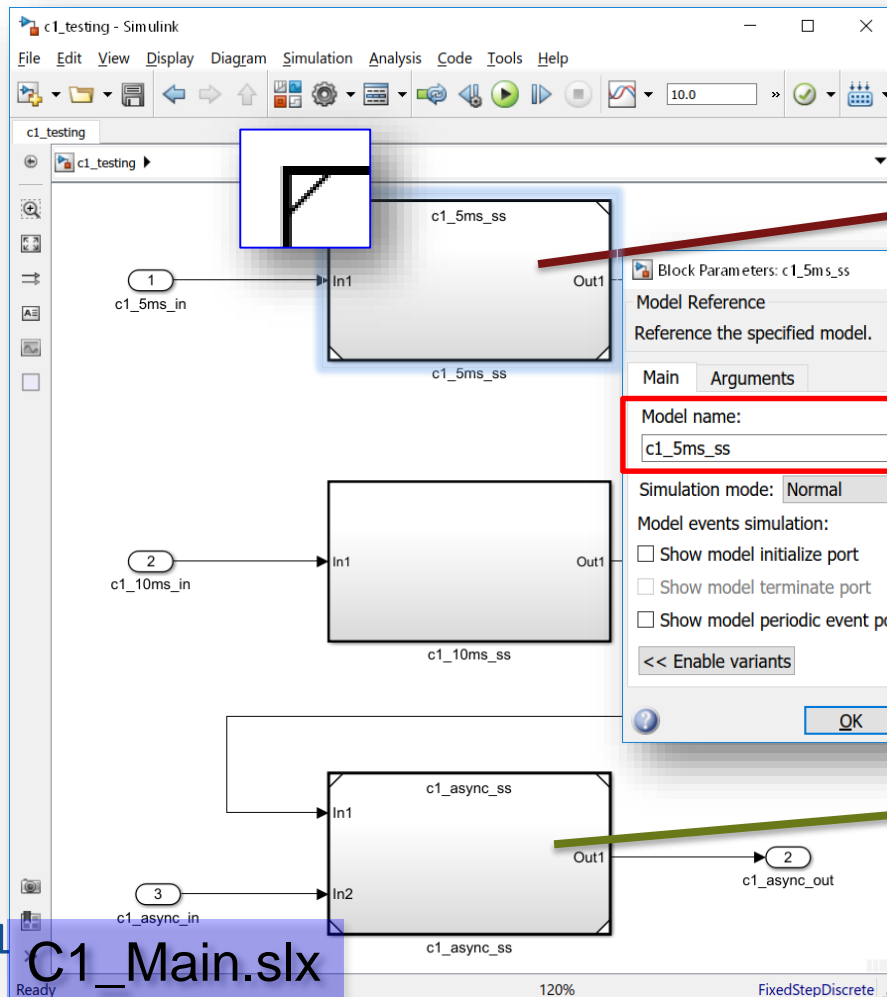
Integration in a Composition Level

- Modeling based on component and integration as a composition using **Model Reference**



What the Model Reference...?

- Model Reference enables to design models based on SW component



Block Parameters: c1_5ms_ss

Model Reference
Reference the specified model.

Main Arguments

Model name:
 Browse... Open Model

Simulation mode: Normal

Model events simulation:

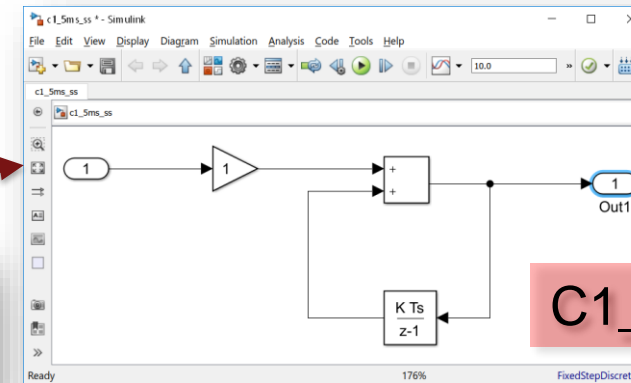
Show model initialize port

Show model terminate port

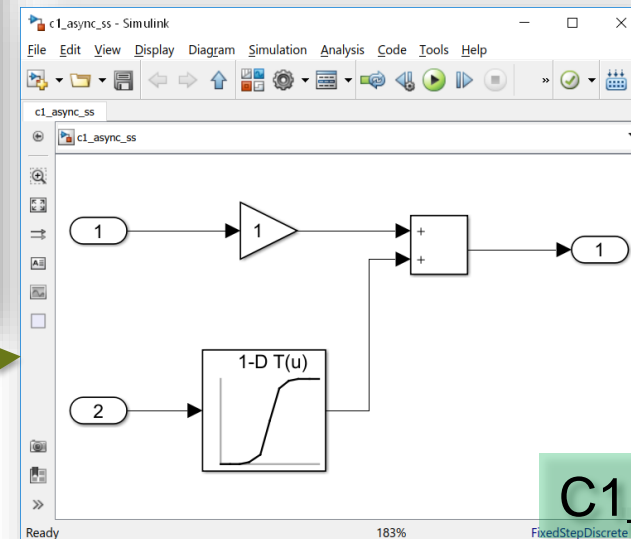
Show model periodic event ports

<< Enable variants

OK Cancel Help Apply



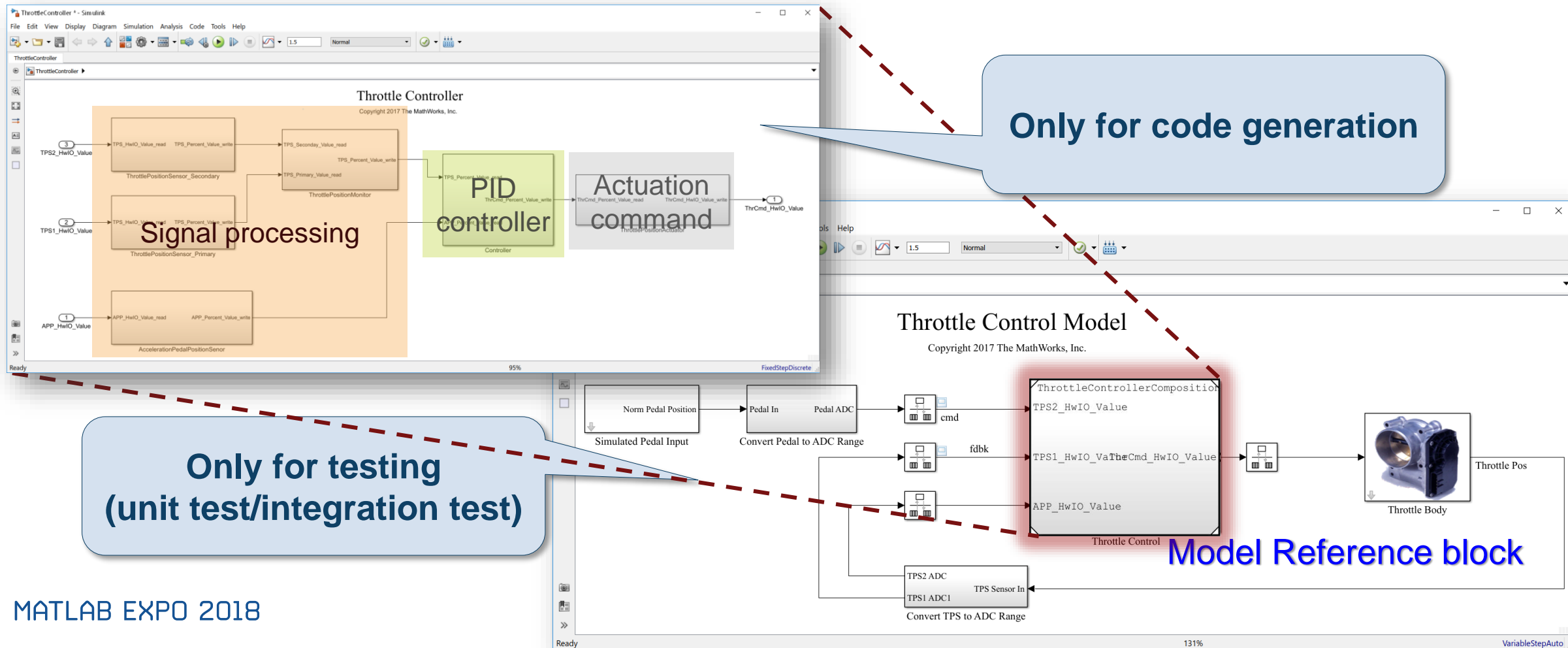
C1_5ms.slx



C1_async.slx

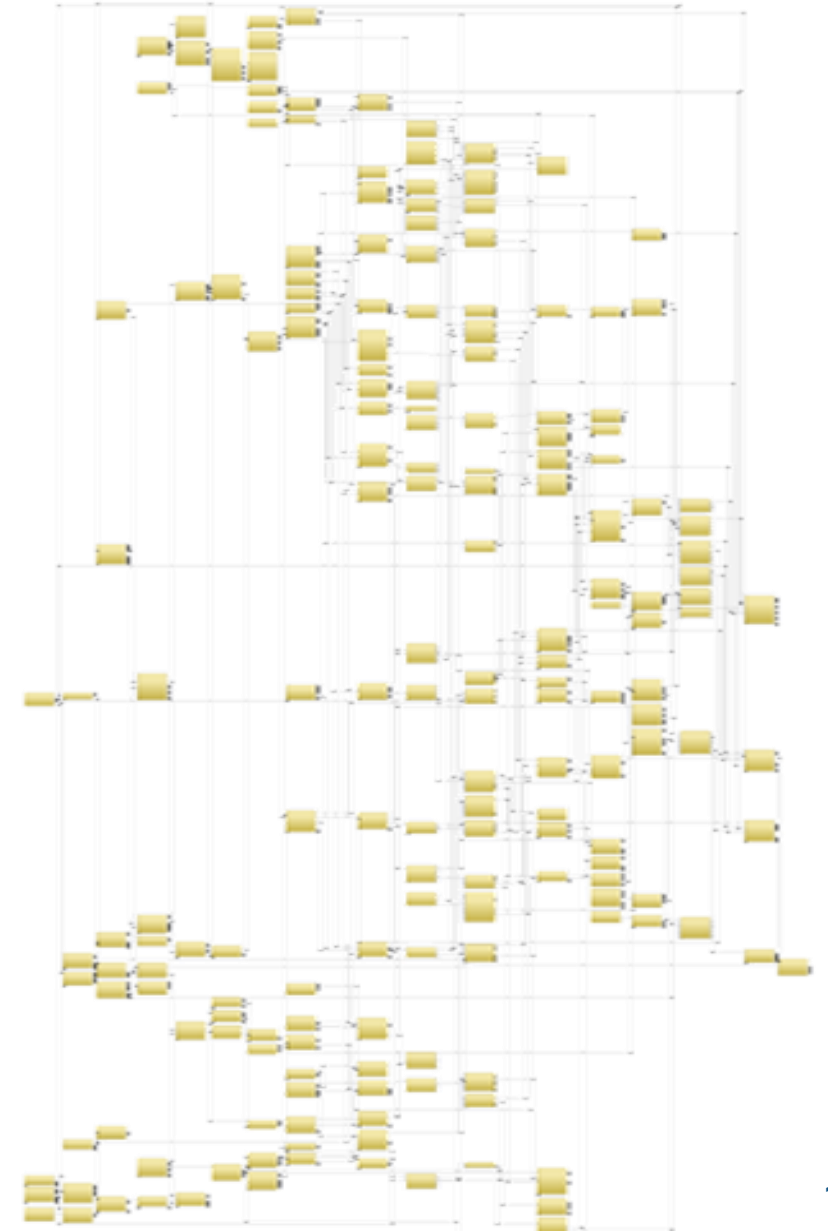
Creating Separate Test Harness Model

- Your model for code generation is separate from test harness model



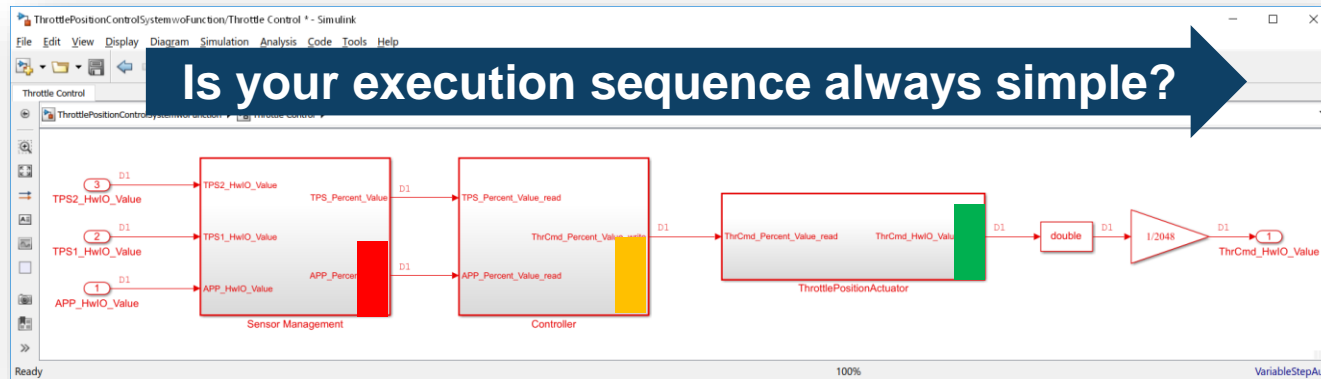
Issues for Large-scaled Embedded Software Development

Issues \ Work Phase	Modeling	Code Generation
1. Complexity	✓	
2. Integration (Reusability + Scalability)		
3. Scheduling	✓	
4. Multi-instantiation		

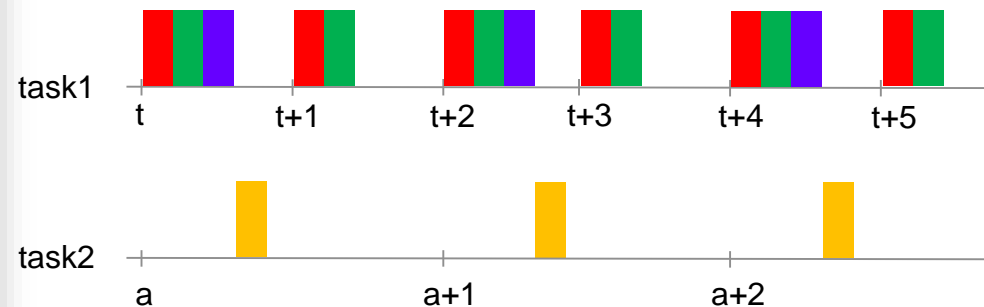
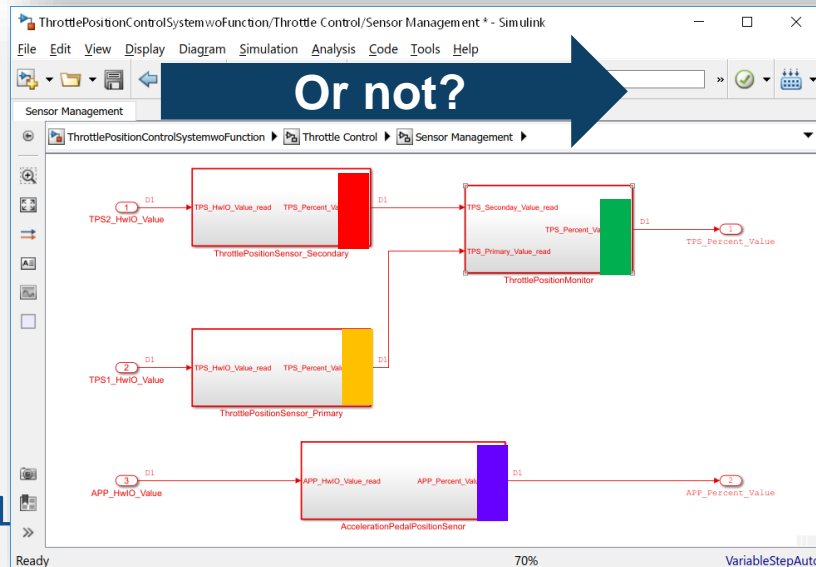
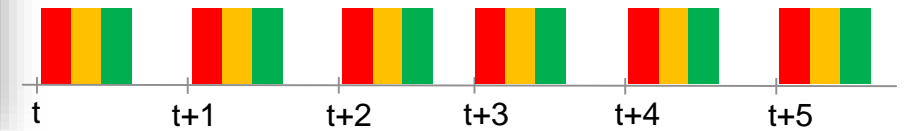


SW Scheduling for Larger-scale Software

- Requirement to analyze the results according to scheduling



OS/ scheduler
 Multicore execution
 Multi tasking
 ...

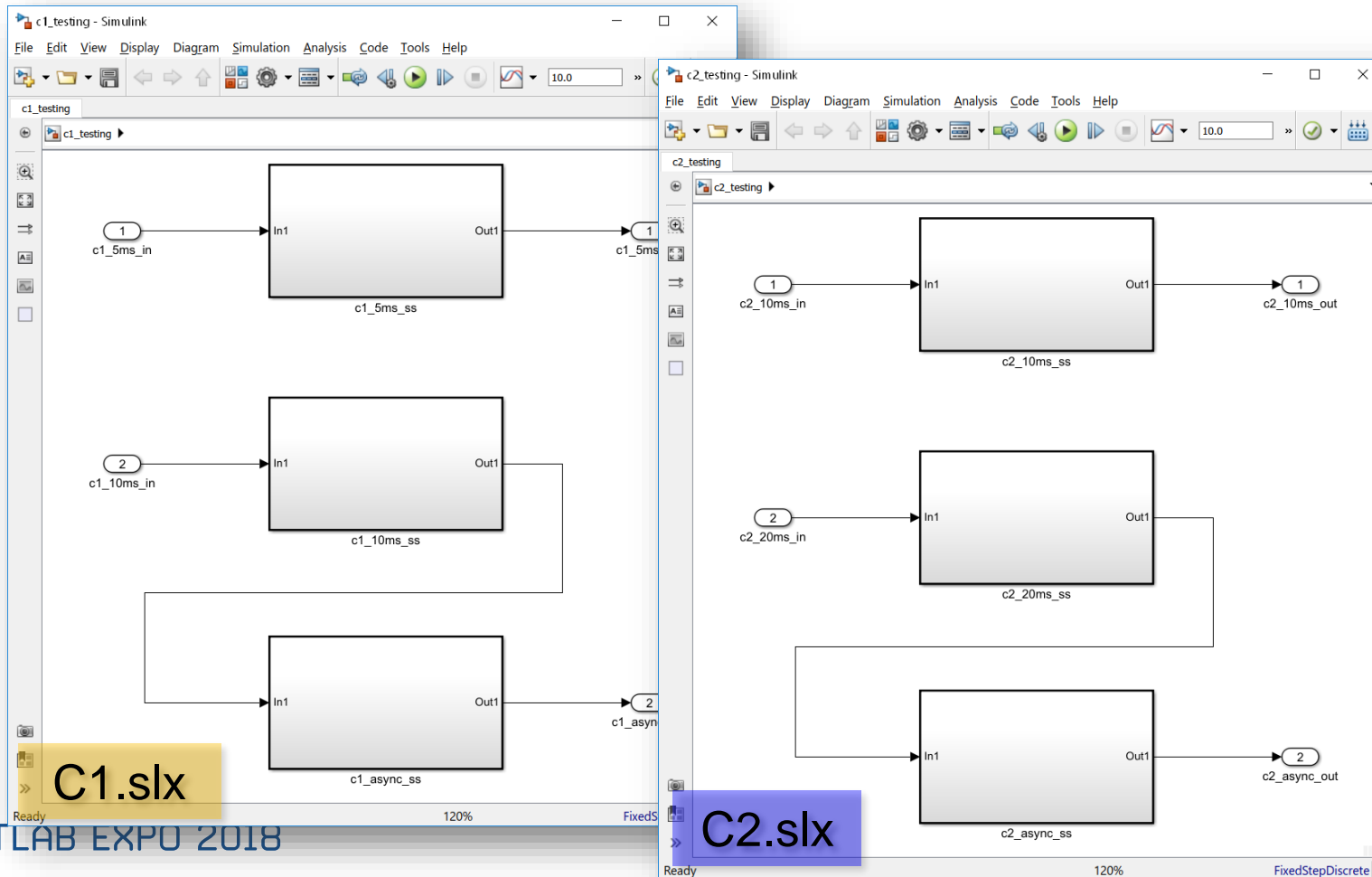


How guarantee ...?

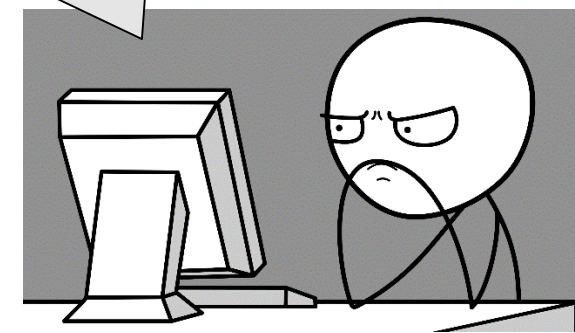


Typical Workflow for Software Integration and Scheduling

- Collecting models for code generation with considering scheduling



Now how do I integrate to base code?



Collect Entry Point Functions for Each Component

File: **c1.c**

```

1  #include "c1.h"
2
3  DW rtDW;
4  ExtU rtU;
5  ExtY rtY;
6  void c1_step0(void)
7  {
8      rtY.c1_10ms_out = 2.0 * rtU.c1_10m
9  }
10
11 void c1_step1(void)
12 {
13     real_T rtb_RateTransition;
14     rtDW.RateTransition_semaphoreTaken
15     rtb_RateTransition =
16     rtDW.RateTransition_Buffer[rtDW.
17     rtY.c1_20ms_out = rtb_RateTransiti
18 }
19
20 void c1_async(void)
21 {
22     rtDW.RateTransition_Buffer[rtDW.Ra
23     rtU.c1_async_in;
24     rtDW.RateTransition_ActiveBufIdx =
25     == 0);
26 }
27
28 void c1_initialize(void)
29 {
30 }
31
32 void c1_terminate(void)
33 {
34 }
35

```

File: **c2.c**

```

1  #include "c2.h"
2
3  DW rtDW;
4  ExtU rtU;
5  ExtY rtY;
6  void c2_step0(void)
7  {
8      rtY.c2_10ms_out = rtU.c2_10ms_in +
9  }
10
11 void c2_step1(void)
12 {
13     real_T rtb_In1;
14     rtb_In1 = rtU.c1_20ms_in;
15     rtDW.RateTransition_Buffer[rtDW.Rat
16     rtDW.RateTransition_ActiveBufIdx =
17     0);
18 }
19
20 void c2_async(void)
21 {
22     rtDW.RateTransition =
23     rtDW.RateTransition_Buffer[rtDW.R
24     rtY.c2_async_out = rtDW.UnitDelay_D
25     rtDW.UnitDelay_DSTATE = rtDW.RateTr
26 }
27
28 void c2_initialize(void)
29 {
30 }
31
32 void c2_terminate(void)
33 {
34 }

```

```

c1_step0()
c1_step1()
c1_async()

```

c1

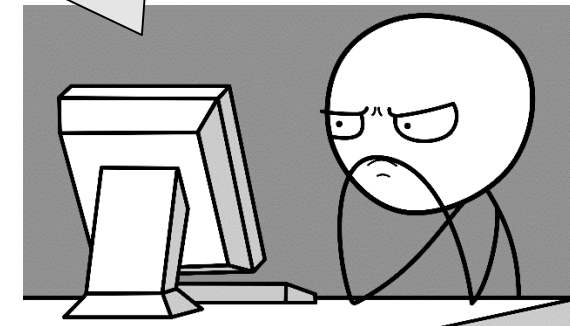
```

c2_step0()
c2_step1()
c2_async()

```

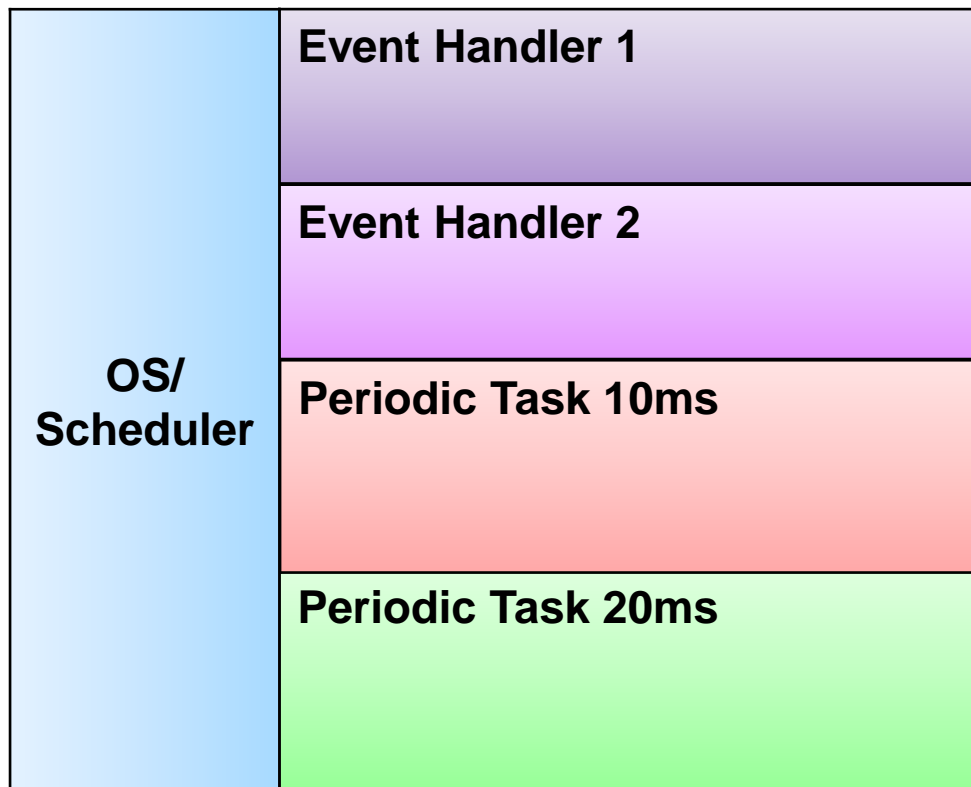
c2

And, how do I create scheduling orders?



Application Integrated to Base Software

- Integrate entry point functions from components with run-time environment



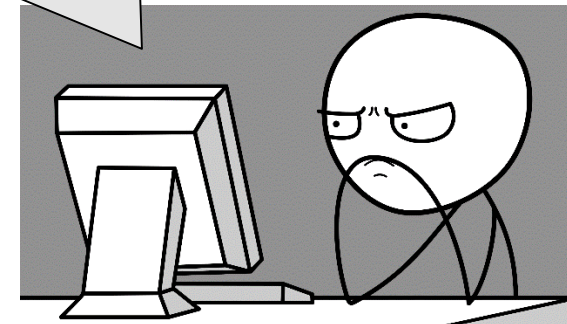
```
c1_step0 ()  
c1_step1 ()  
c1_async ()
```

c1

```
c2_step0 ()  
c2_step1 ()  
c2_async ()
```

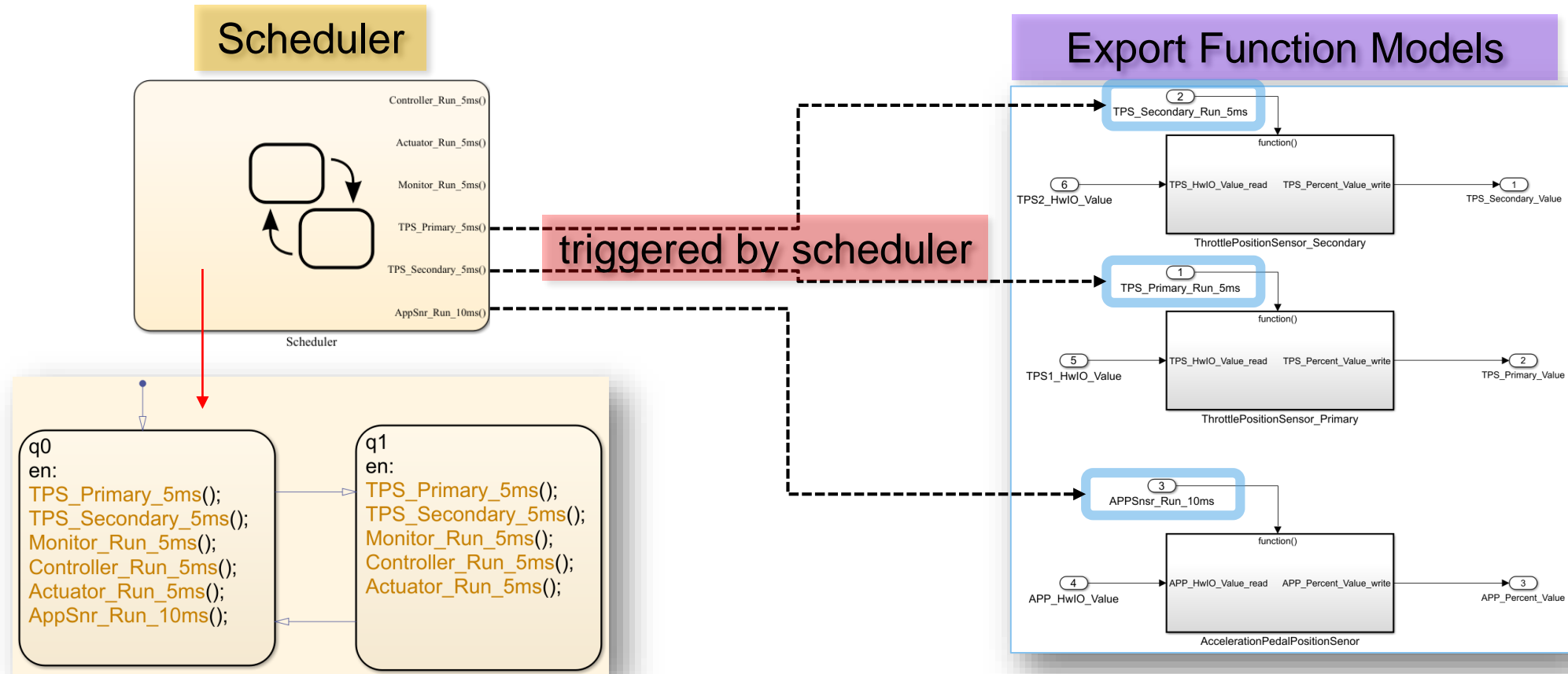
c2

But, I want to know
scheduling effects
before integration!



Software Testing with Scheduling Effects

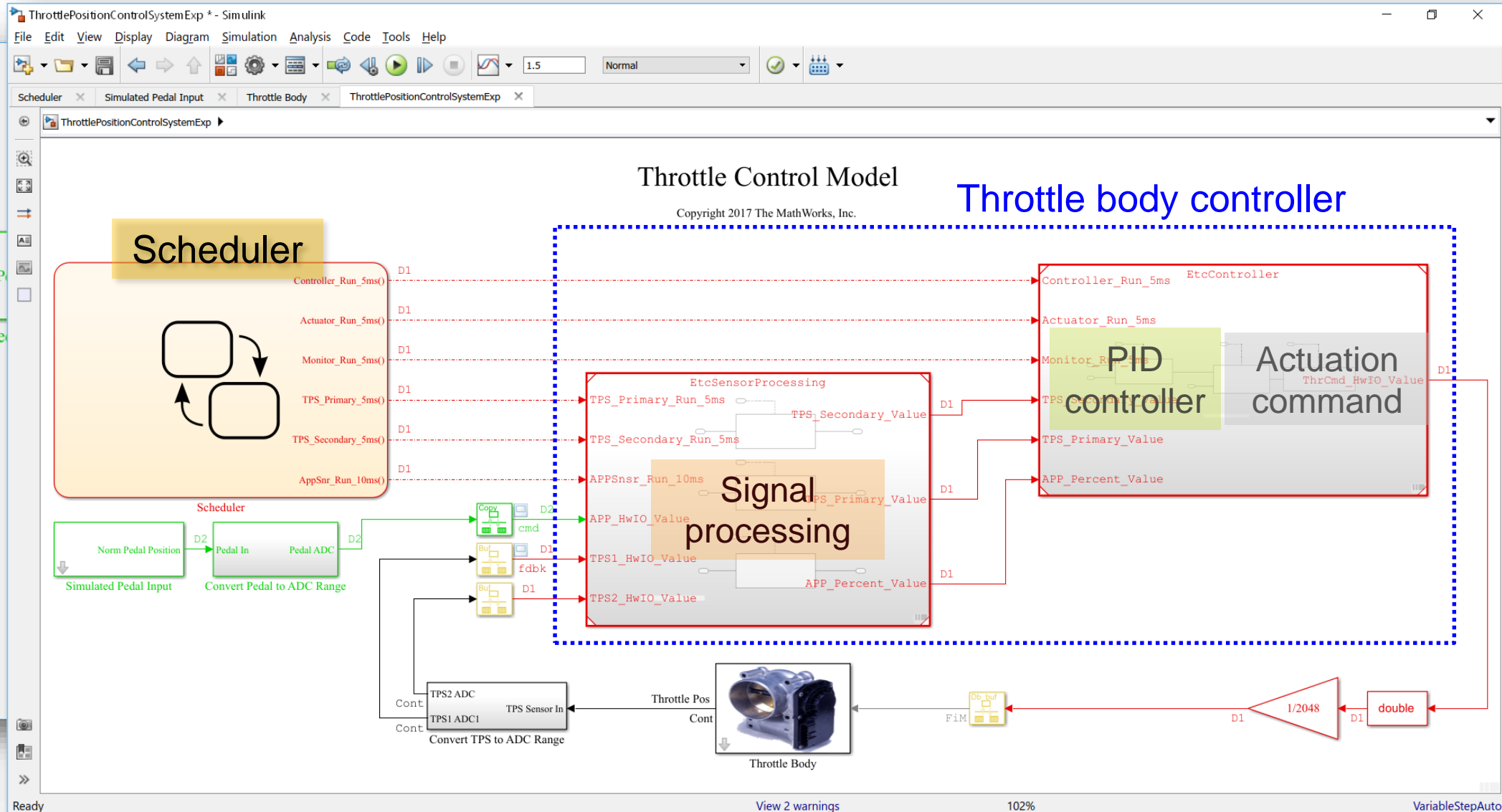
Export Function



→ Scheduler makes periodic events (ex. 5ms/10ms)

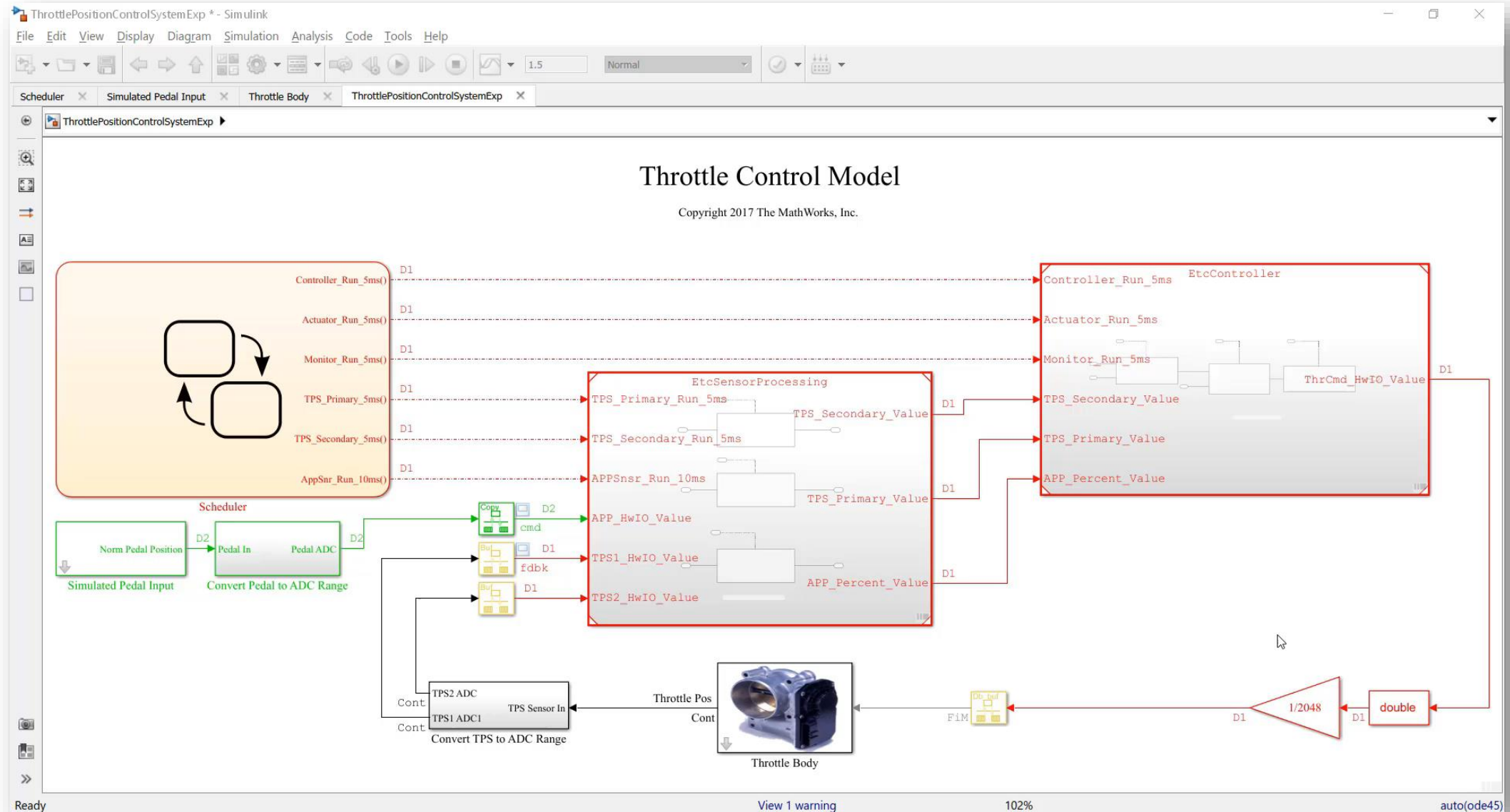
Redesigned Model with Scheduler and Export Functions

Export Function



Demo: SW Modeling with Export Functions

Export Function

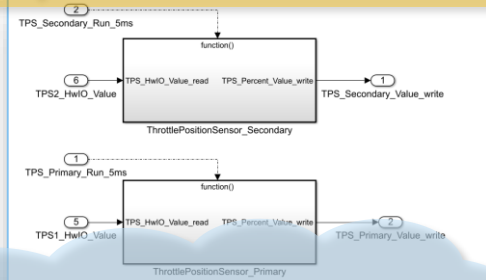


Testing Scheduling Effects from Different Patterned Models

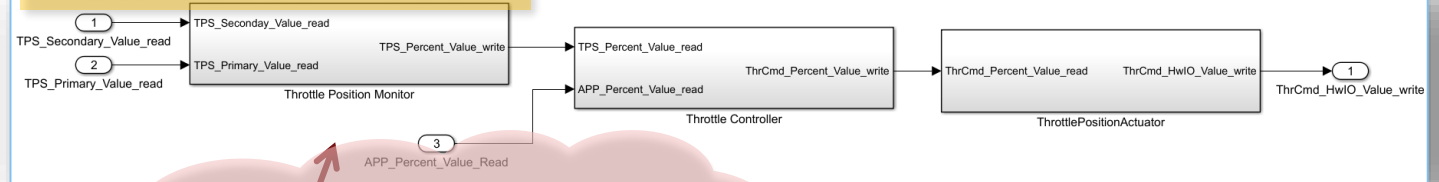
Schedulable Component

- What if there are any other models with different modeling patterns ?

Export-function models



Rate-based models

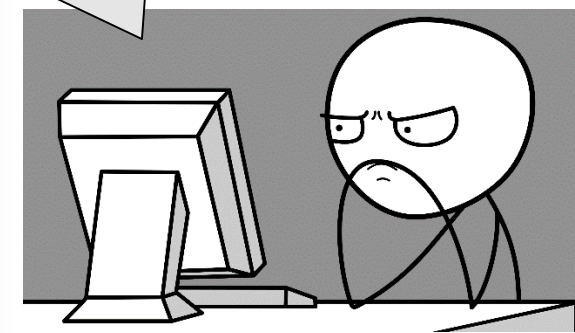
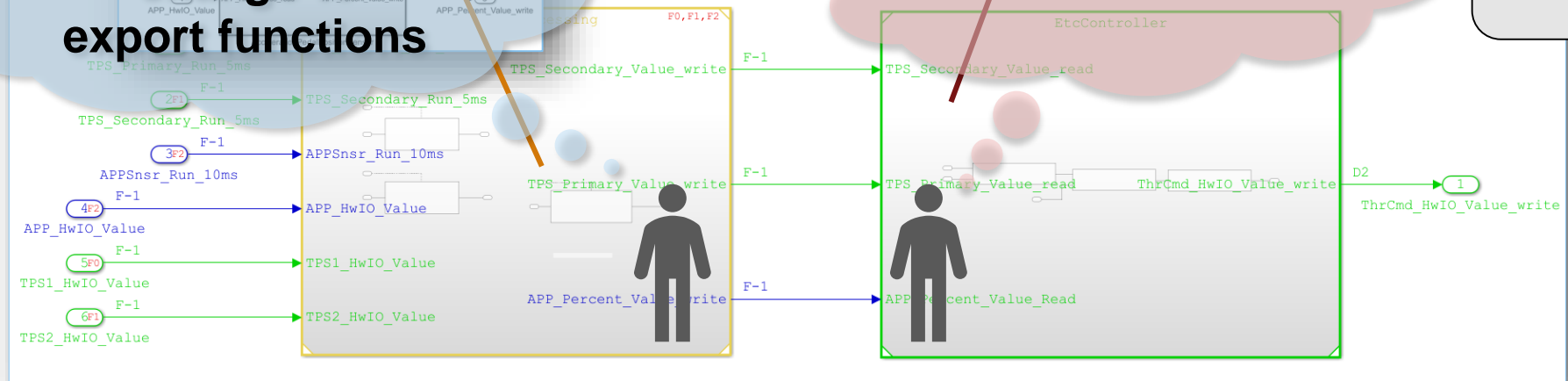


To integrate, change to export functions

Throttle Control Model
Copyright 2017 The MathWorks, Inc.

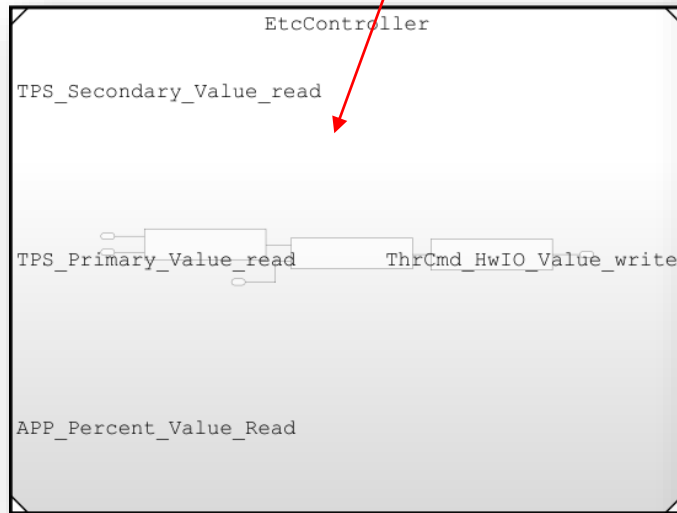
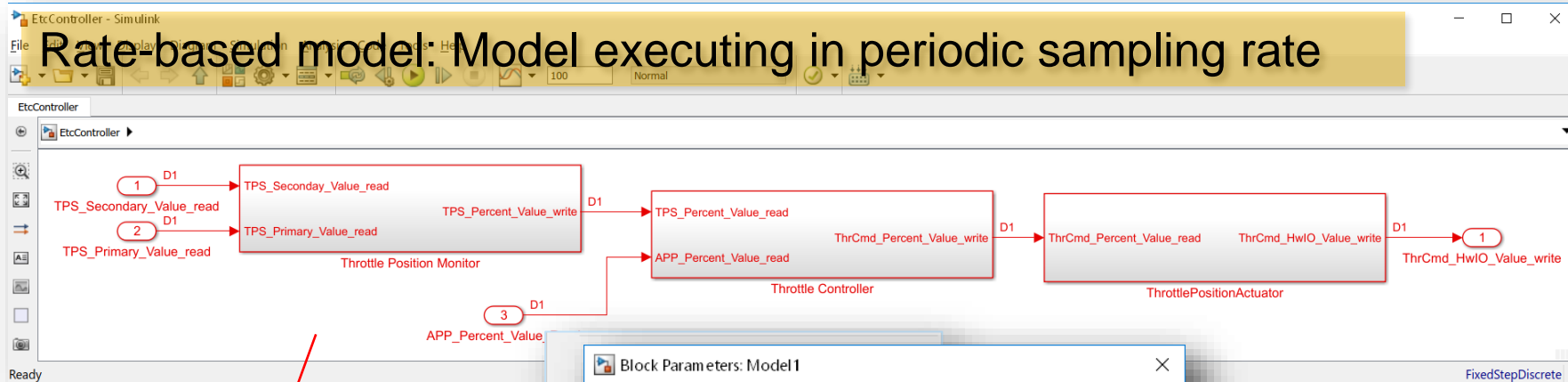
No, impossible...

Wow, How do I resolve this struggles?



Creating Schedulable Component from Model Reference

Schedulable Component



Block Parameters: Model1

Model Reference
Reference the specified model.

Main Arguments

Model name: EtcController.slx [Browse... Open Model]

Simulation mode: Normal

Model events simulation:

- Show model initialize port
- Show model terminate port
- Show model periodic event ports

<< Enable variants

OK Cancel Help Apply

Block Parameters (ModelReference)
Properties...
Help

Schedulable Component

EtcController

D1 [0.005]

TPS_Secondary_Value_read

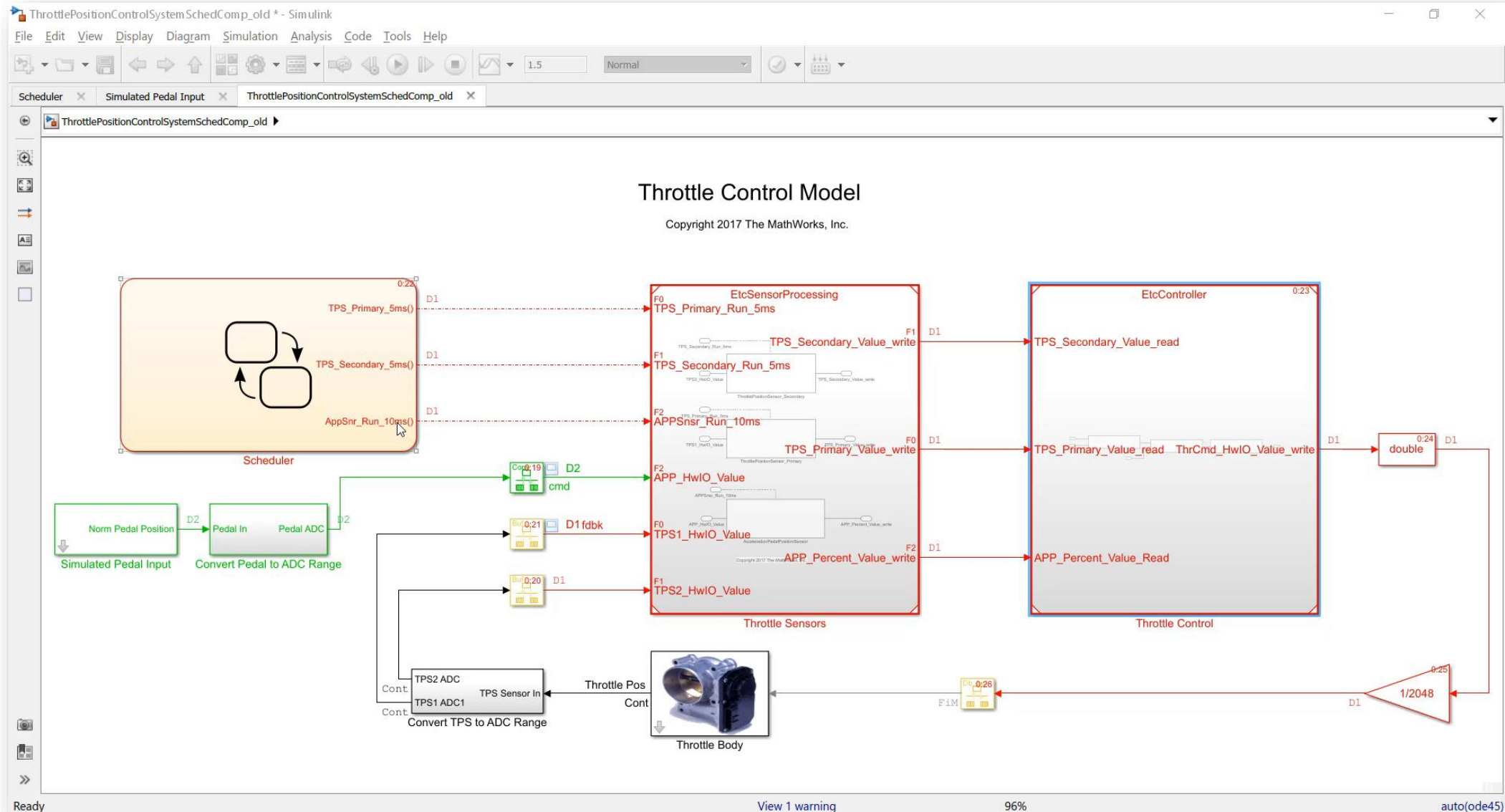
TPS_Primary_Value_read ThrCmd_HwIO_Value_write

APP_Percent_Value_Read

*Event port for scheduling
→ This port is not for code generation but only for simulation*

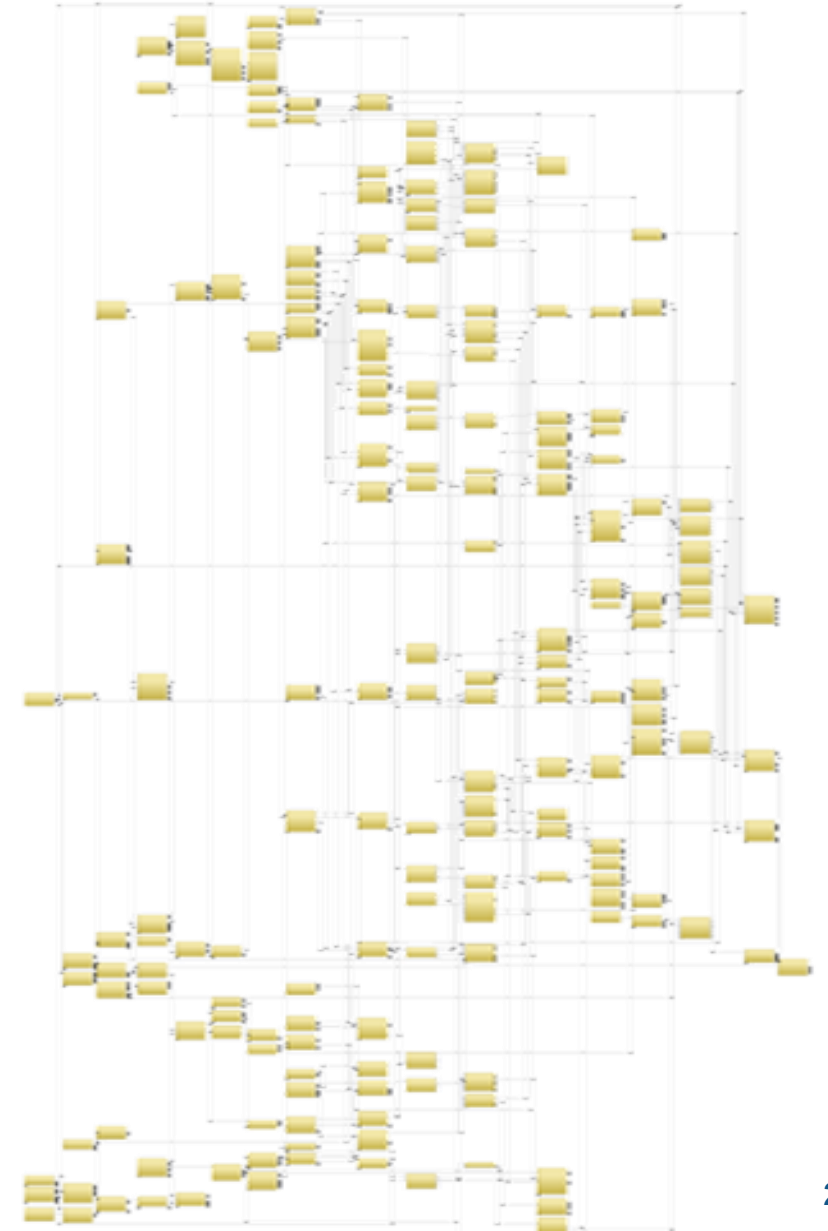
Demo: SW Modeling with Schedulable Components

Schedulable Component



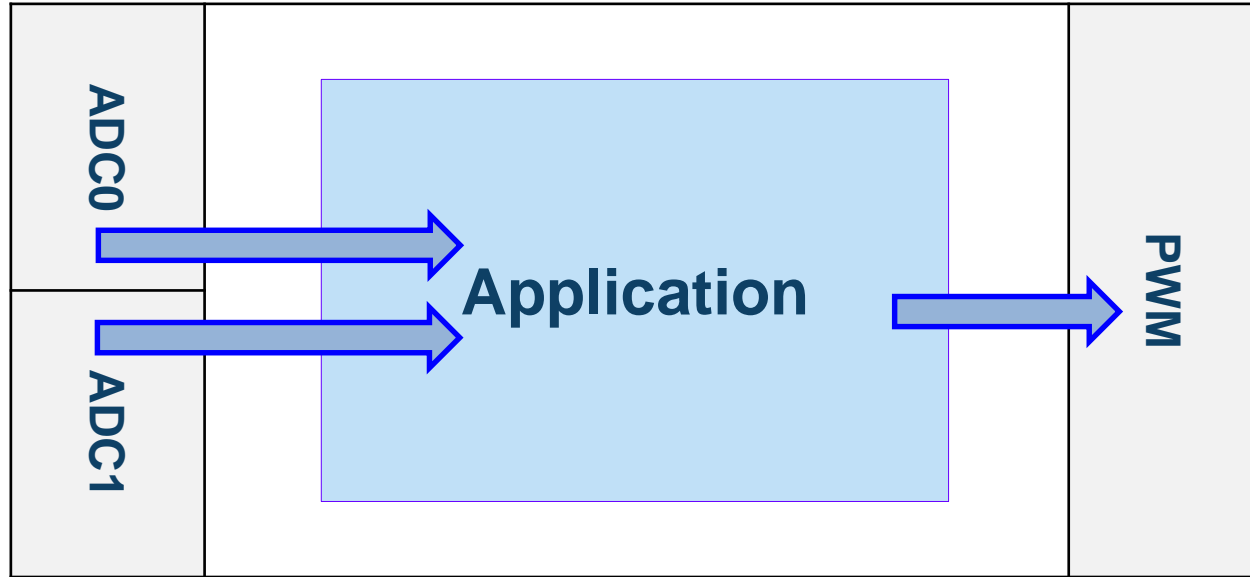
Issues for Large-scaled Embedded Software Development

Issues \ Work Phase	Modeling	Code Generation
1. Complexity	✓	
2. Integration (Reusability + Scalability)	✓	
3. Scheduling	✓	
4. Multi-instantiation		



Modeling for Access to Hardware Resources

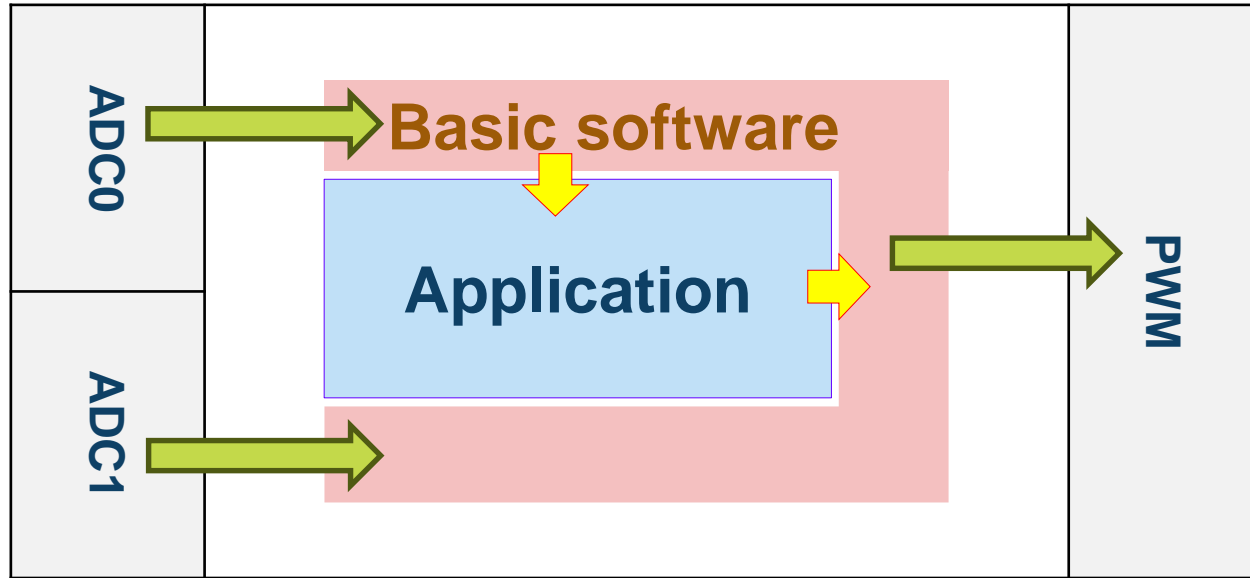
Simulink Function



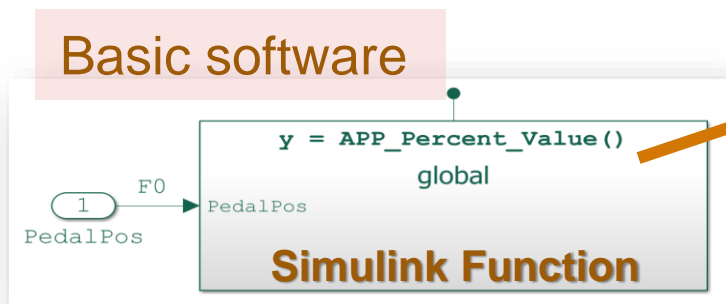
- Some application SW does not process external signals directly. Or...

Modeling for Access to Hardware Resources

Simulink Function

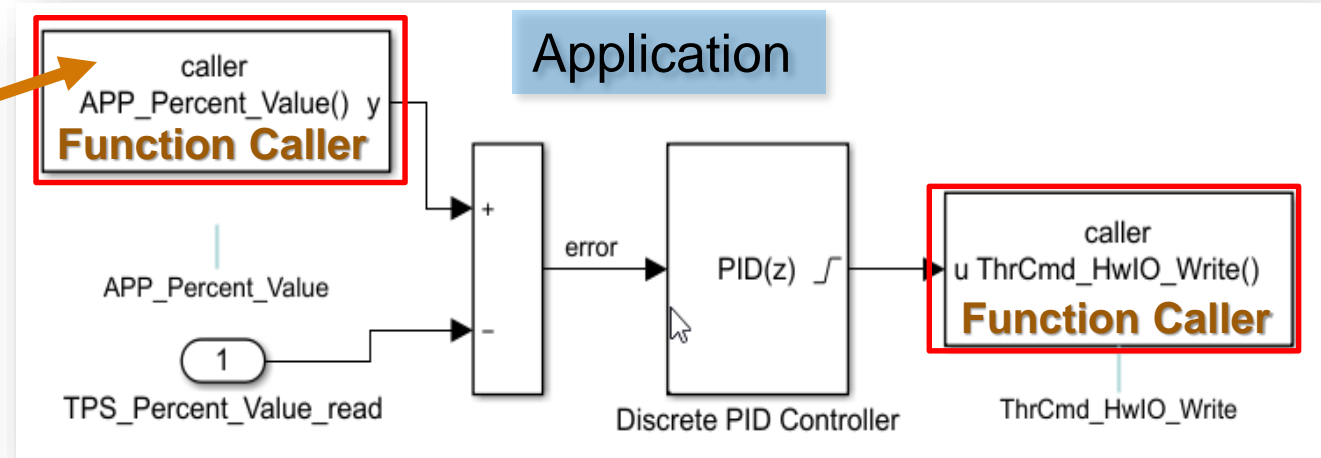


- Some application SW does not process external signals directly. Or...
- External signals are processed in BSW or HAL and accessed by applications
- Application software use APIs to request or send data



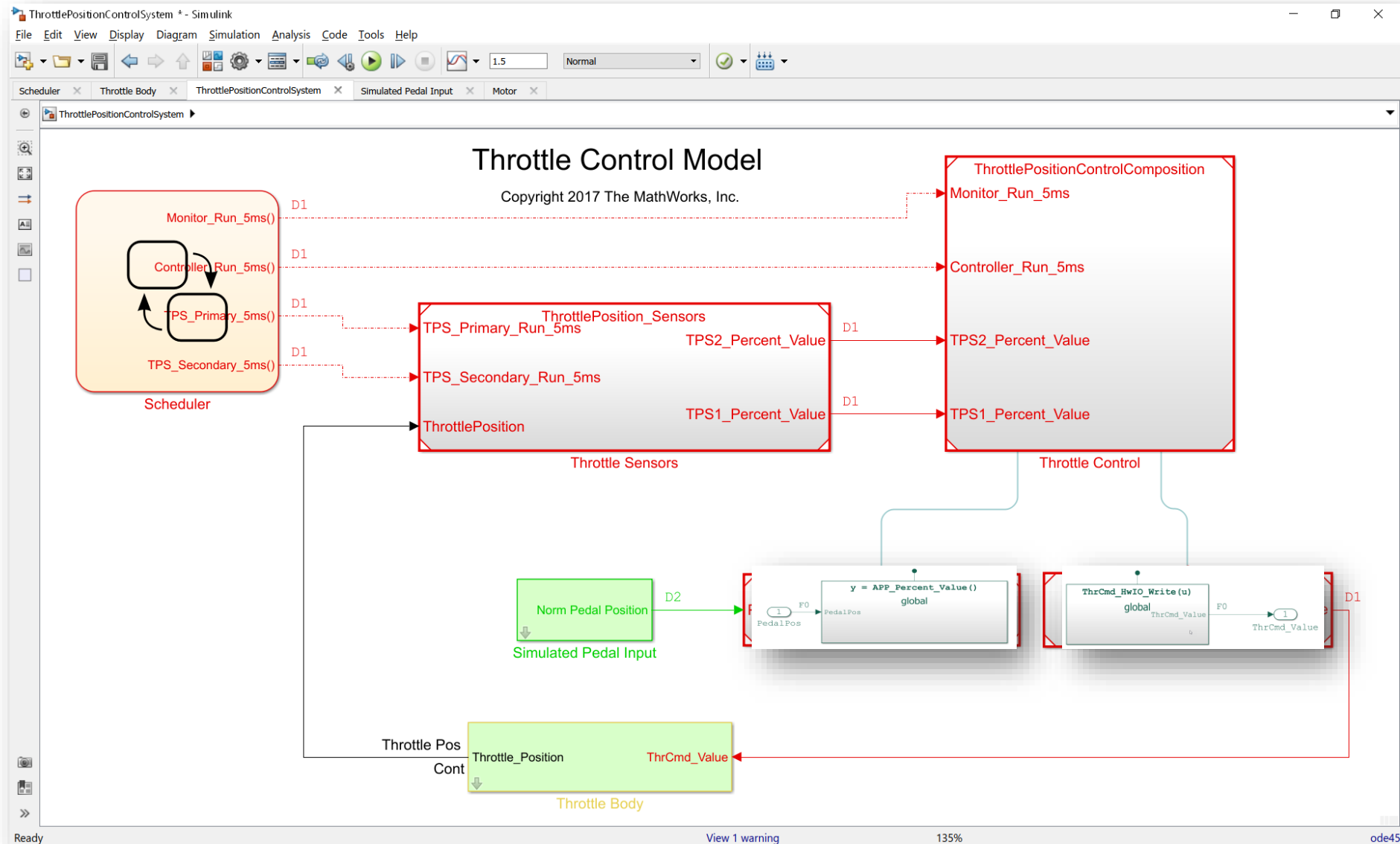
Only for simulation

request



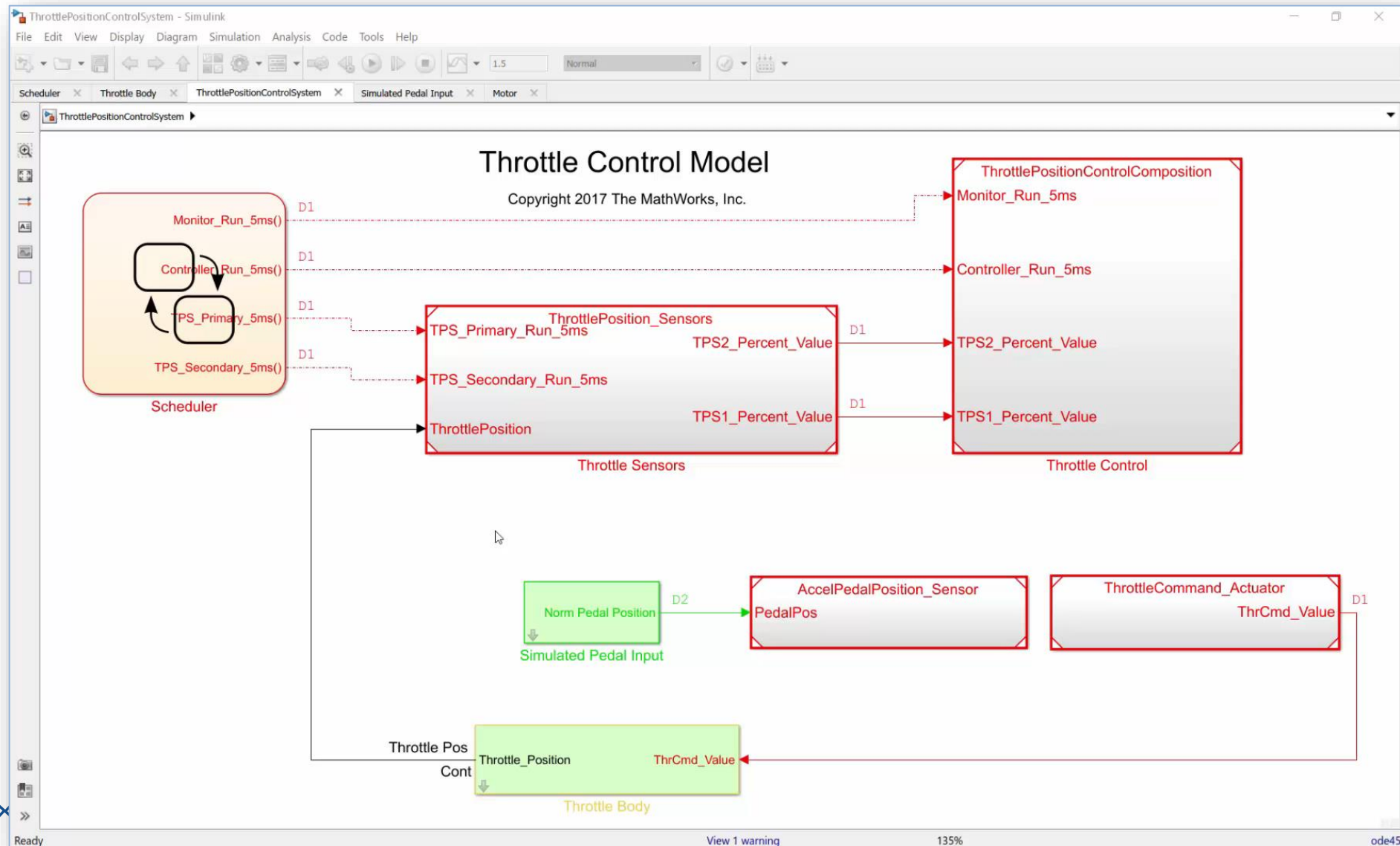
Access to Shared Resources with Simulink Functions

Simulink Function



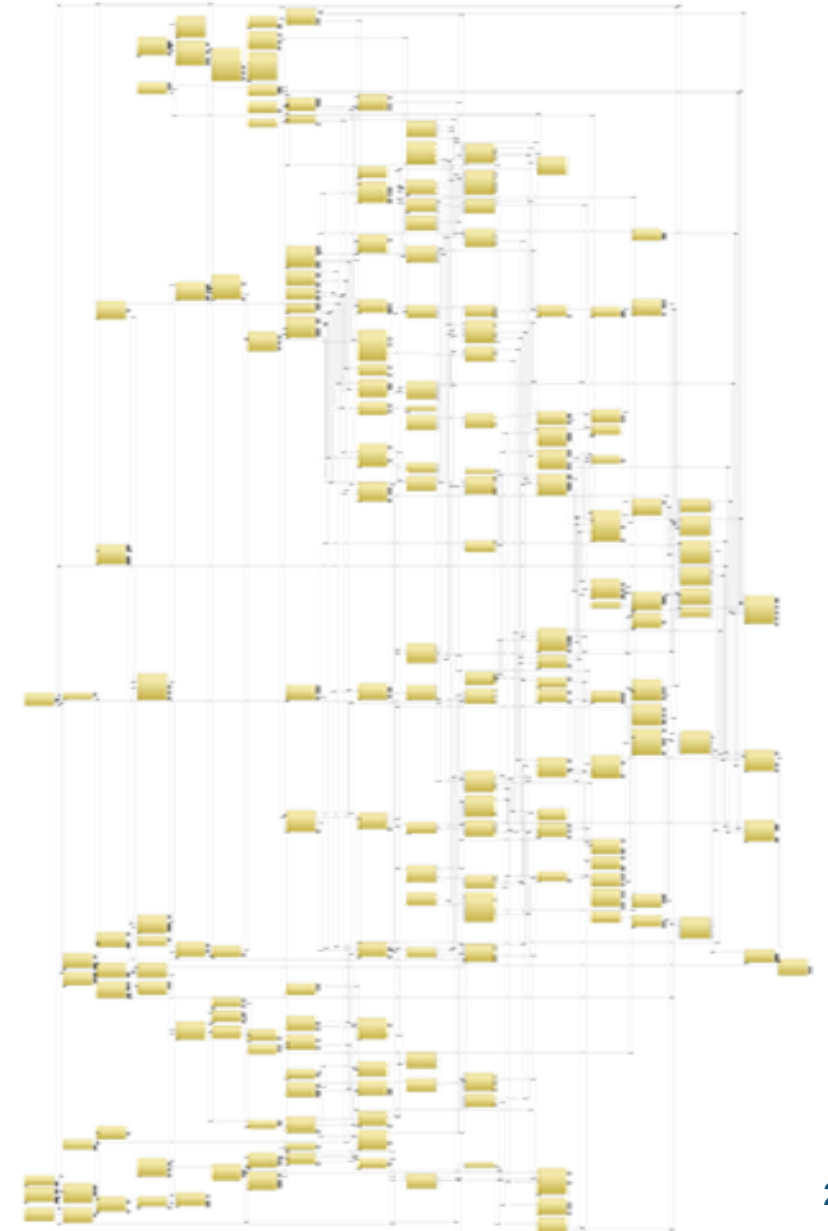
Demo: SW Modeling with Simulink Functions

Simulink Function



Issues for Large-scaled Embedded Software Development

Issues \ Work Phase	Modeling	Code Generation
1. Complexity	✓	
2. Integration (Reusability + Scalability)	✓	
3. Scheduling	✓	
4. Multi-instantiation		✓



Self-Study Resources for Embedded Code Generation

- Embedded Coder Quick Start Video
https://kr.mathworks.com/videos/coder-summit-2018-how-to-generate-production-code-in-5-minutes--1522057622892.html?s_tid=srchtitle
- Simulink와 Embedded Coder를 이용한 최적 코드 생성 (MATLAB Expo 2017)
<http://www.matlabexpo.com/kr/2017/proceedings/better-than-hand-generating-highly-optimized-code-using-simulink-embedded-coder.pdf>
- C code generation from Simulink model (webinar)
https://kr.mathworks.com/videos/software-design-and-c-code-generation-using-simulink-116860.html?elqsid=1524127259550&potential_use=Commercial
- Other Embedded Coder Videos
https://kr.mathworks.com/products/embedded-coder/videos.html?s_tid=srchtitle

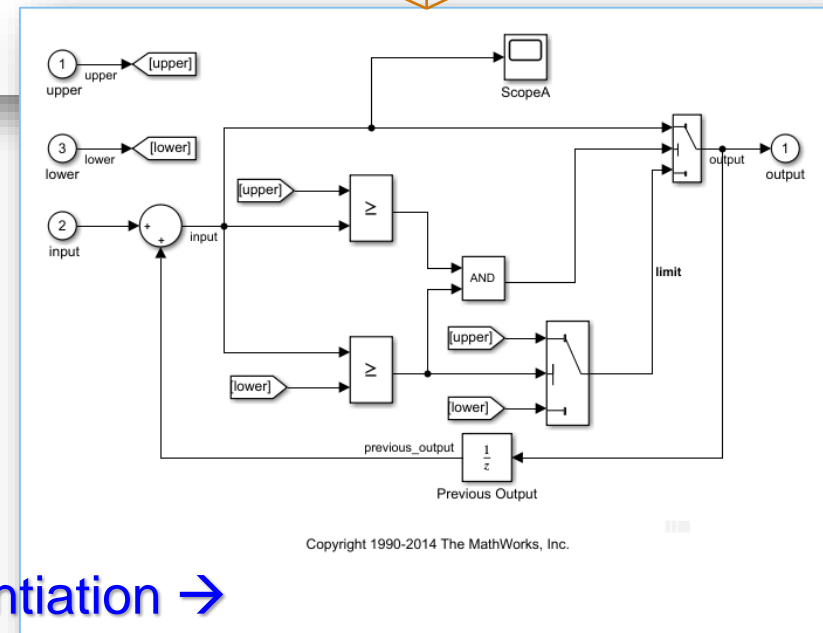
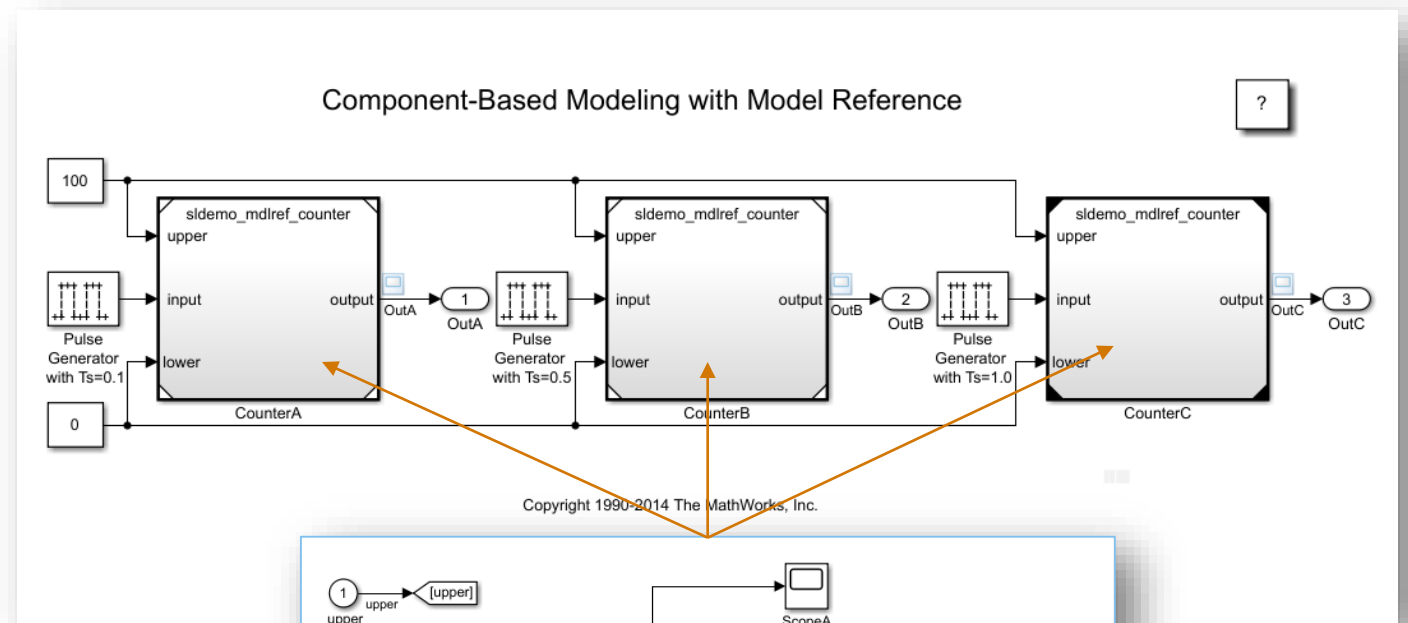
Multi-instantiation for Large-scaled Software

■ Issues

- Limited resources for code size
- Maintenance problem

■ Solution

- Calling the same functions through multi-instantiation when generating code



Configuration for Multi-instantiation

C

Code interface

Code interface packaging: Reusable function Multi-instance code error diagnostic: Error

Pass root-level I/O as: Structure reference

Remove error status field in real-time model data structure

Configure Model Functions

Before(non-reusable)

```
/* Model step function */
void sldemo_mdhref_counter_r_step(void)
{
```



After(reusable)

```
/* Model step function */
void sldemo_mdhref_counter_r_step(RT_MODEL_sldemo_mdhref_counter_r_T *const
sldemo_mdhref_counter_r_M, ExtU_sldemo_mdhref_counter_r_T
*sldemo_mdhref_counter_r_U, ExtY_sldemo_mdhref_counter_r_T
*sldemo_mdhref_counter_r_Y)
{
```

C++

Code interface

Code interface packaging: C++ class Multi-instance code error diagnostic: Error

Remove error status field in real-time model data structure

Data Member Visibility/Access Control

Parameter visibility: private Parameter access: None

External I/O access: None

Configure C++ Class Interface

Creating instances

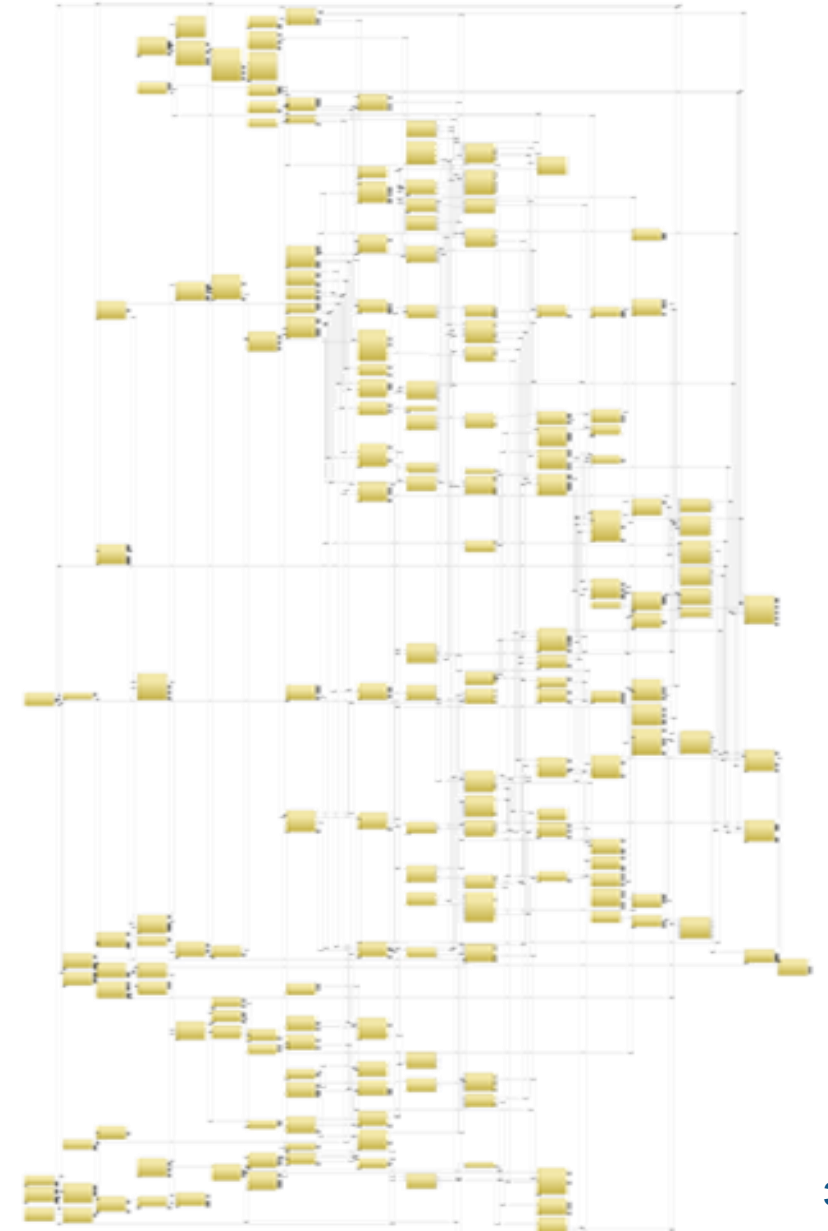
```
// model instance variable for '<Root>/CounterA'
sldemo_mdhref_counter_rModelClass CounterAMDLOBJ1;

// model instance variable for '<Root>/CounterB'
sldemo_mdhref_counter_rModelClass CounterBMDLOBJ2;

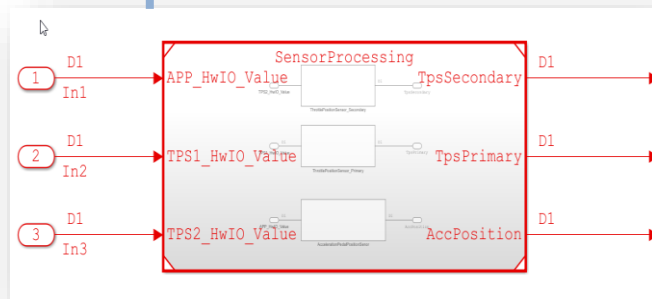
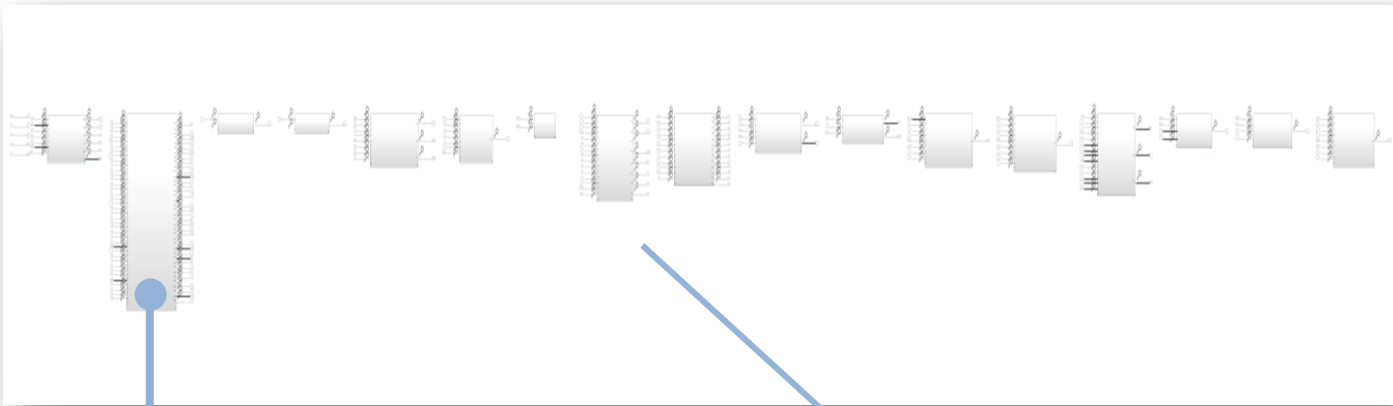
// model instance variable for '<Root>/CounterC'
sldemo_mdhref_counter_rModelClass CounterCMDLOBJ3;
```

Issues for Large-scaled Embedded Software Development

Issues \ Work Phase	Modeling	Code Generation
1. Complexity	✓	
2. Integration (Reusability + Scalability)	✓	✗
3. Scheduling	✓	
4. Multi-instantiation		✓



Emergence of the Software Framework



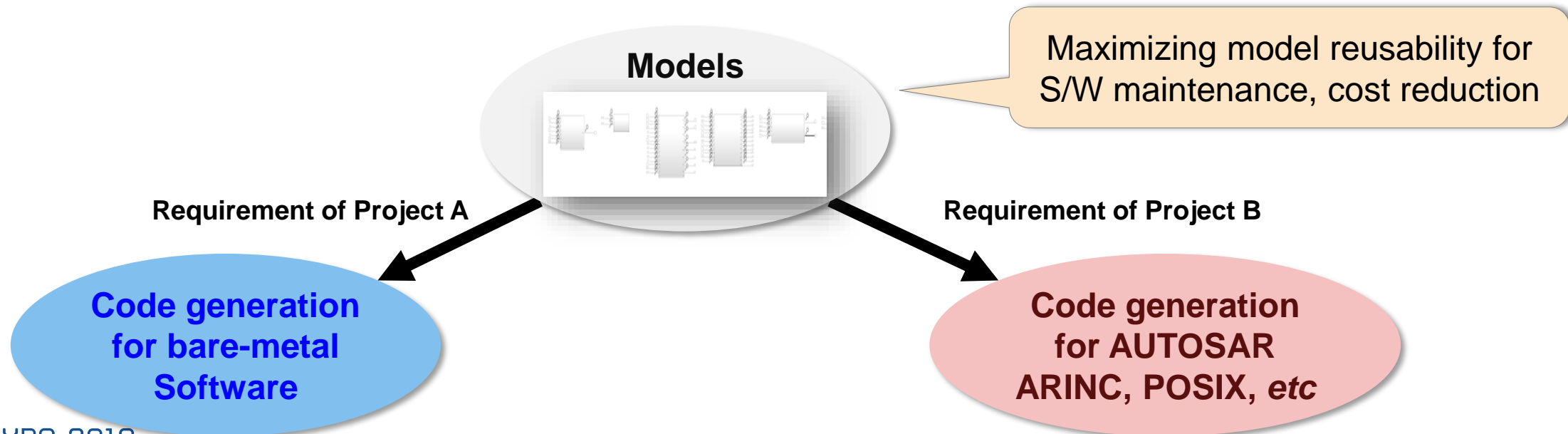
Conform to a (standard) framework (ex. AUTOSAR)

Deploy



Issue 1: Mapping Generated Code to Software Frameworks

- There are many frameworks (ex. AUTOSAR, ARINC, etc) including bare-metal software
- Solution : Configuration management of code mapping information apart from S/W frameworks
 - Just using code mapping information according to requirements



Issue 2: Code Packaging for Efficient Code Management

- There are needs to manage efficient tuning parameters in large-scaled S/W
 - To change only tuning parameters according to requirements
 - Efficient code maintenance
- Solution
 - Configuring storage class for code generation
 - Easy customization using GUI

**Code for
algorithm**



Controller.h



Controller.c

**Code for
parameters**



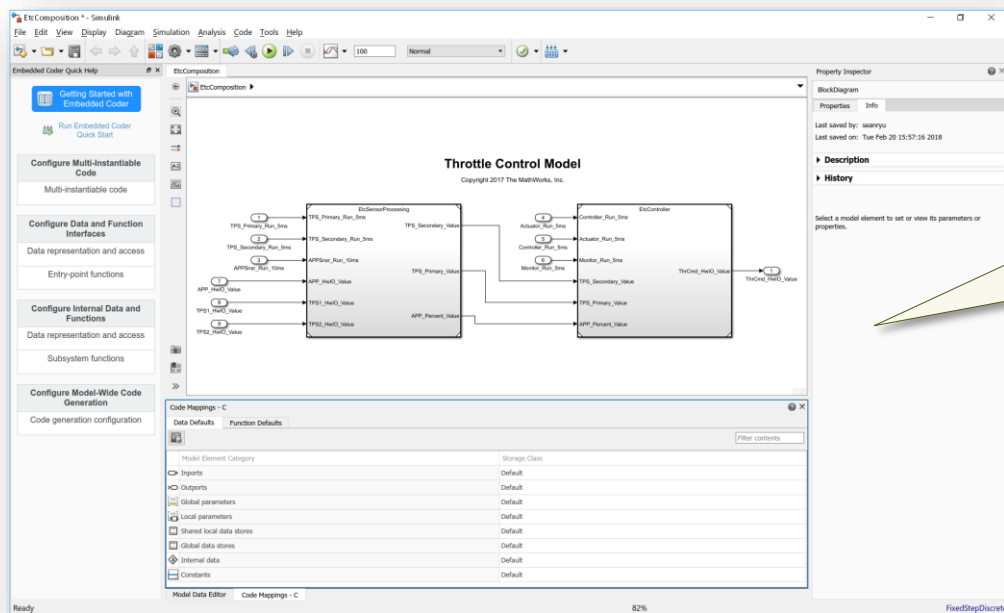
Parameter.h



Parameter.c

Code Perspective & Embedded Coder Dictionary

- Effective code generation customization as to SW frameworks

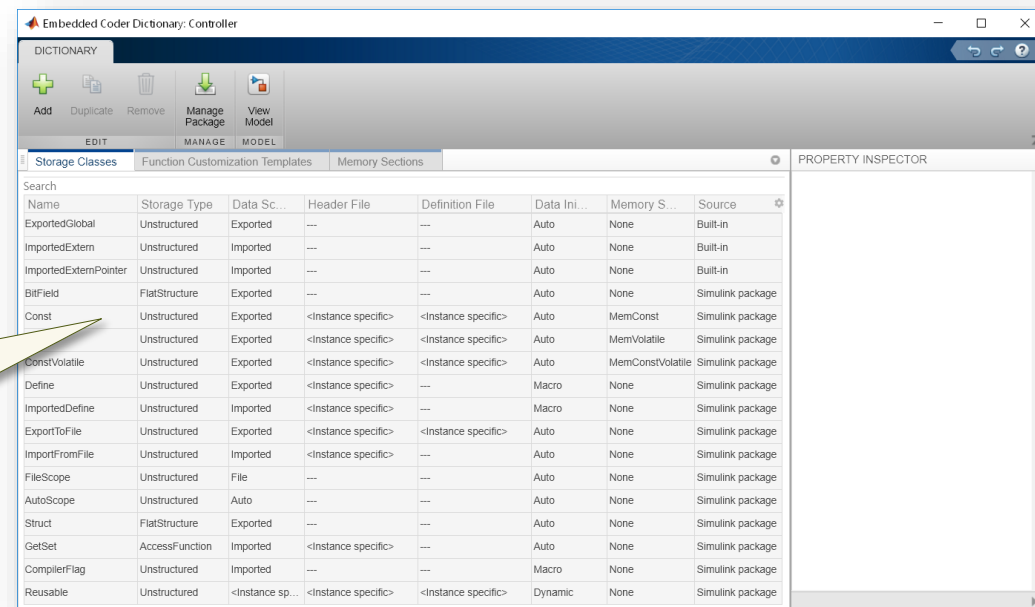


Code Perspective

- Easy configuration for generated code into any C/C++ SW framework

Embedded Coder Dictionary

- GUI for custom code definitions
 - Function template
 - Storage class
 - Memory section



Embedded Coder Dictionary

Embedded Coder Dictionary: EtcComposition
[-] [x]

DICTIONARY

Add
 Duplicate
 Remove
 Manage Package
 View Model

EDIT
MANAGE
MODEL

- **Storage classes**
 - Control the code generated for model data (I/O, signals, data stores, states, parameters)

Storage Classes | Function Customization Templates | Memory Sections

Search

Name	Storage Ty...	Data Sc...	Header File	Definition File	Data Ini...	Memory S...	Source
ExportedGlobal	Unstructured	Exported	---	---	Auto	None	Built-in
ImportedExtern	Unstructured	Imported	---	---	Auto	None	Built-in
ImportedExternPointer	Unstructured	Imported	---	---	Auto	None	Built-in
BitField	FlatStructure	Exported	---	---	Auto	None	Simulink package
Const	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	MemConst	Simulink package
Volatile	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	MemVolatile	Simulink package
ConstVolatile	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	MemConstVolatile	Simulink package
Define	Unstructured	Exported	<Instance specific>	---	Macro	None	Simulink package
ImportedDefine	Unstructured	Imported	<Instance specific>	---	Macro	None	Simulink package
ExportToFile	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	None	Simulink package
ImportFromFile	Unstructured	Imported	<Instance specific>	---	Auto	None	Simulink package
FileScope	Unstructured	File	---	---	Auto	None	Simulink package
AutoScope	Unstructured	Auto	---	---	Auto	None	Simulink package
Struct	FlatStructure	Exported	---	---	Auto	None	Simulink package
GetSet	AccessFunction	Imported	<Instance specific>	---	Auto	None	Simulink package
CompilerFlag	Unstructured	Imported	---	---	Macro	None	Simulink package
Reusable	Unstructured	<Instance sp...	<Instance specific>	<Instance specific>	Dynamic	None	Simulink package

PROPERTY INSPECTOR

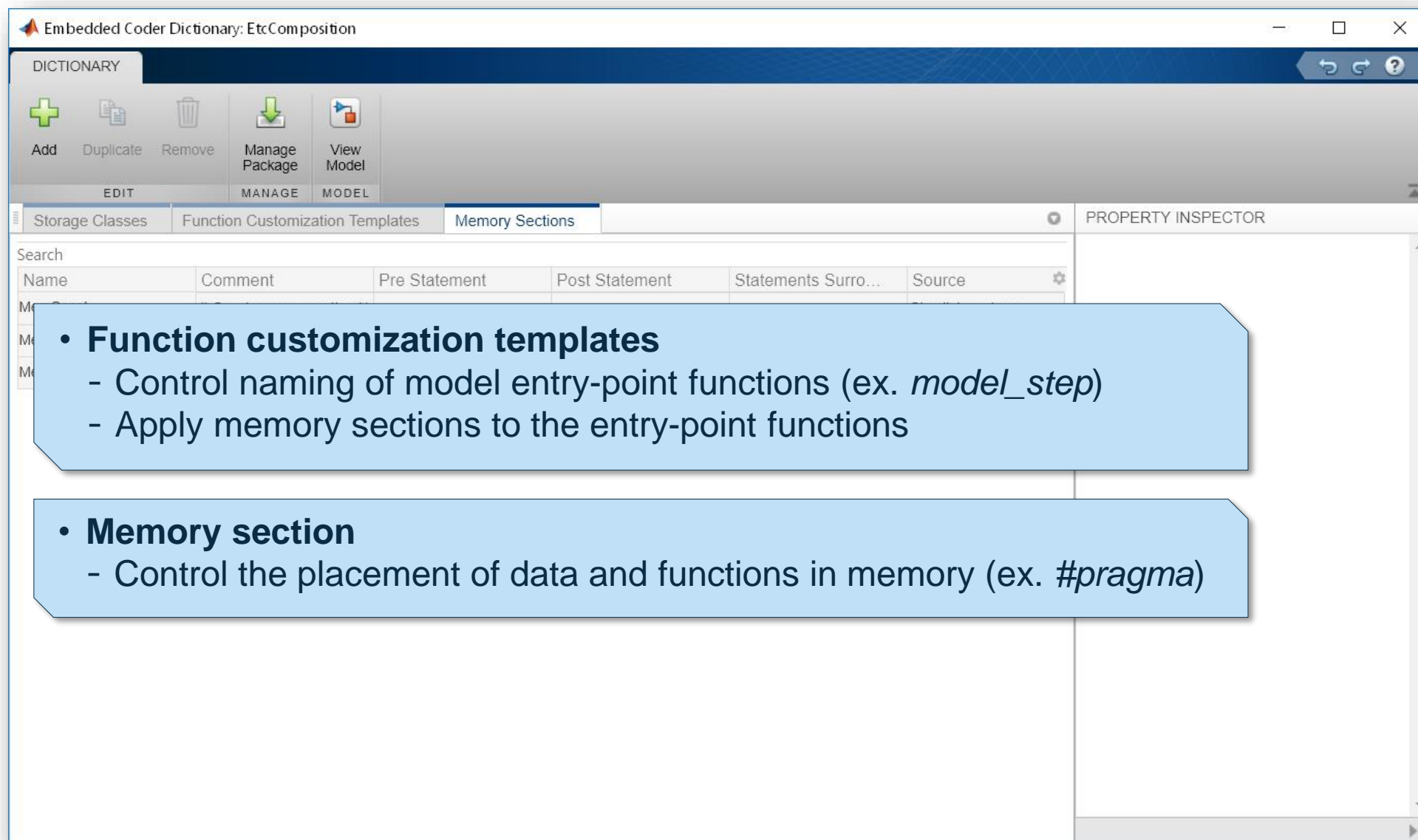
Storage allocation and scope
(ex, global, extern, static, register, pointer ...)

Bitfield, Constant, Pre-processor, ...

Export to or import from external files, ...

Etc.: Structure type, Get/Set APIs, ...

Embedded Coder Dictionary



The screenshot shows the 'Embedded Coder Dictionary: EtcComposition' window. The 'MEMORY SECTIONS' tab is selected, displaying a table with columns: Name, Comment, Pre Statement, Post Statement, Statements Surro..., and Source. A search bar is located above the table. The interface includes a toolbar with icons for Add, Duplicate, Remove, Manage Package, and View Model, and a 'PROPERTY INSPECTOR' panel on the right.

- **Function customization templates**
 - Control naming of model entry-point functions (ex. *model_step*)
 - Apply memory sections to the entry-point functions
- **Memory section**
 - Control the placement of data and functions in memory (ex. *#pragma*)

Code Perspective

1) Embedded Coder Quick Help

- Embedded Quick Start
- Hyperlink to configuration and documents
- Help video clips

2) Property Inspector

- Configure model properties

3-1) Model Data Editor

- Inspect and edit data items
- Configure storage class of each blocks or signals

3-2) Code Mapping Editor

- Configuring model data elements and entry-point functions for code generation comprehensively

The screenshot illustrates the process of switching to the Code Perspective in Simulink. The 'Code' menu is open, and the option 'Configure Model in Code Perspective' (with the keyboard shortcut Ctrl+Shift+C) is highlighted. Below the menu, the 'Throttle Control Model' block diagram is shown. At the bottom, the 'Code Mappings - C' editor is open, displaying a table of model elements and their storage classes.

Model Element Category	Storage Class
Inports	Default
Outports	Default
Global parameters	Default
Local parameters	Default
Shared local data stores	Default
Global data stores	Default
Internal data	Default
Constants	Default

Example on Issue 1: Code mapping implementation

- Code mapping to embedded S/W frameworks
 - Entry-point functions and interfaces can be customized according to SW architecture

Code mapping to C

Code mapping to AUTOSAR

Throttle Control Model
Copyright 2017 The MathWorks, Inc.

The diagram shows two main blocks: **EtcSensorProcessing** and **EtcController**. **EtcSensorProcessing** has inputs 1-3 (TPS and APPSnr) and outputs 7-9 (APP and TPS HwIO). **EtcController** has inputs 4-6 (Actuator, Controller, Monitor) and outputs 1 (ThCmd_HwIO_Value).

Source	DataAccessMode	Port	Element
APP_HwIO_Value	ImplicitReceive	APP_HwIO_Value	APP_HwIO_Value
TPS1_HwIO_Value	ImplicitReceive	TPS1_HwIO_Value	TPS1_HwIO_Value
TPS2_HwIO_Value	ImplicitReceive	TPS2_HwIO_Value	TPS2_HwIO_Value

AUTOSAR Dictionary

The dictionary shows a tree view of components under AUTOSAR, including ReceiverPorts, SenderPorts, and various interfaces. A table below lists the interfaces:

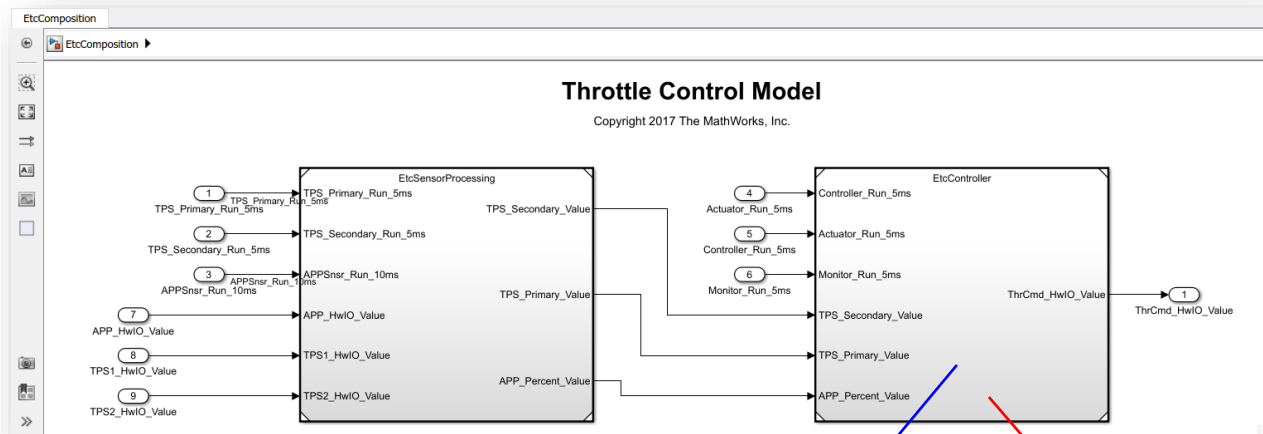
Name	Interface
APP_HwIO_Value	APP_HwIO_Value
TPS1_HwIO_Value	TPS1_HwIO_Value
TPS2_HwIO_Value	TPS2_HwIO_Value

Embedded Coder Dictionary

This window displays a list of storage classes with columns for Name, Storage Type, Data Src, Definition File, Data Init, Memory Size, and Source. A red arrow points from the 'Code Mappings - C' table to this dictionary.

Example on Issue 2: Partition and Modularize Generated Code

- Tuning parameter modularization example with customizing storage class



customization:
Struct + ExportToFile

Algorithm code

Controller.h

Controller.c

Tuning parameters

Parameter.h

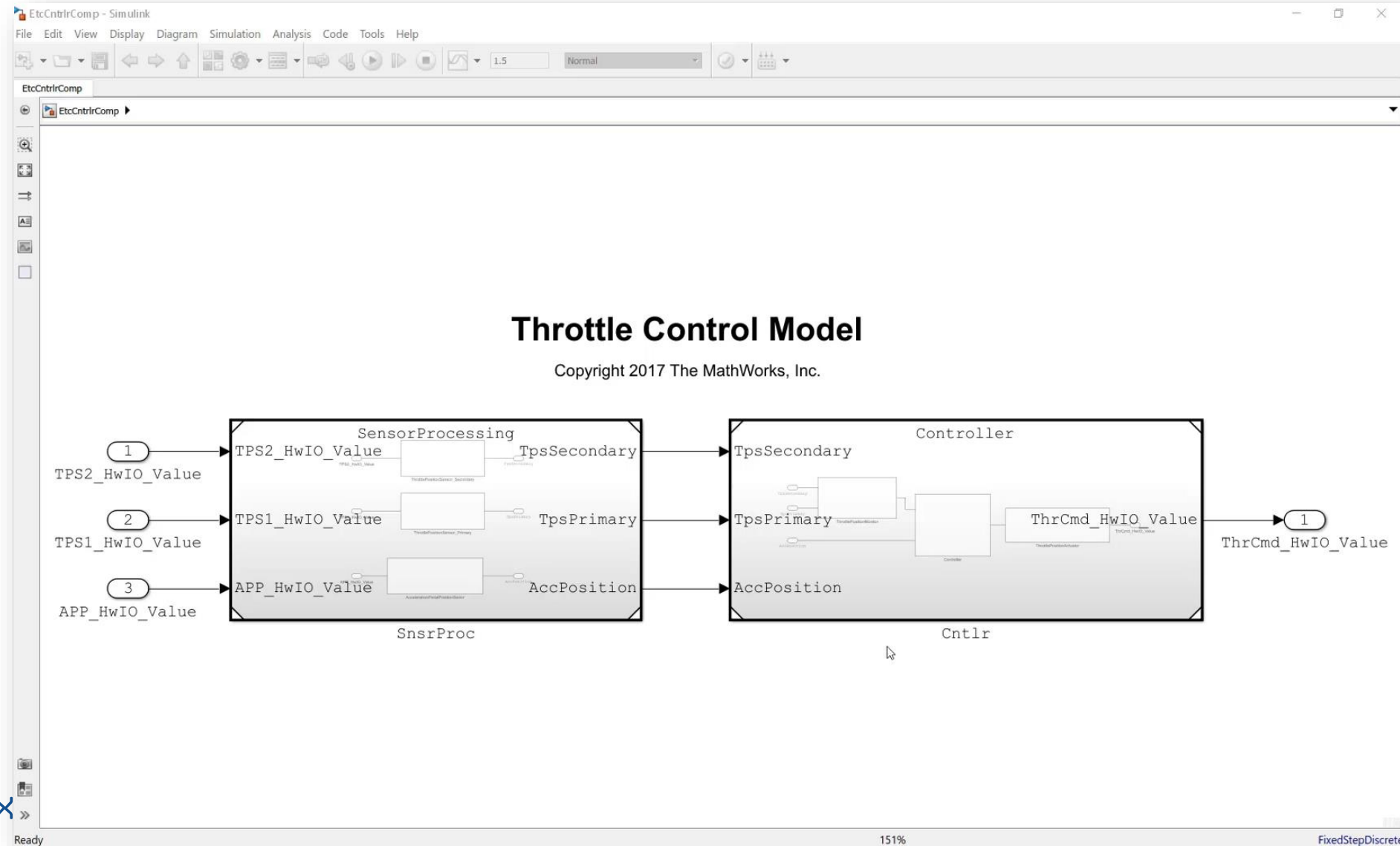
Parameter.c

```

Typedef struct
{
    int D_Gain;
    int I_Gain;
    int P_Gain;
} rt_SI_Struct_type;
    
```

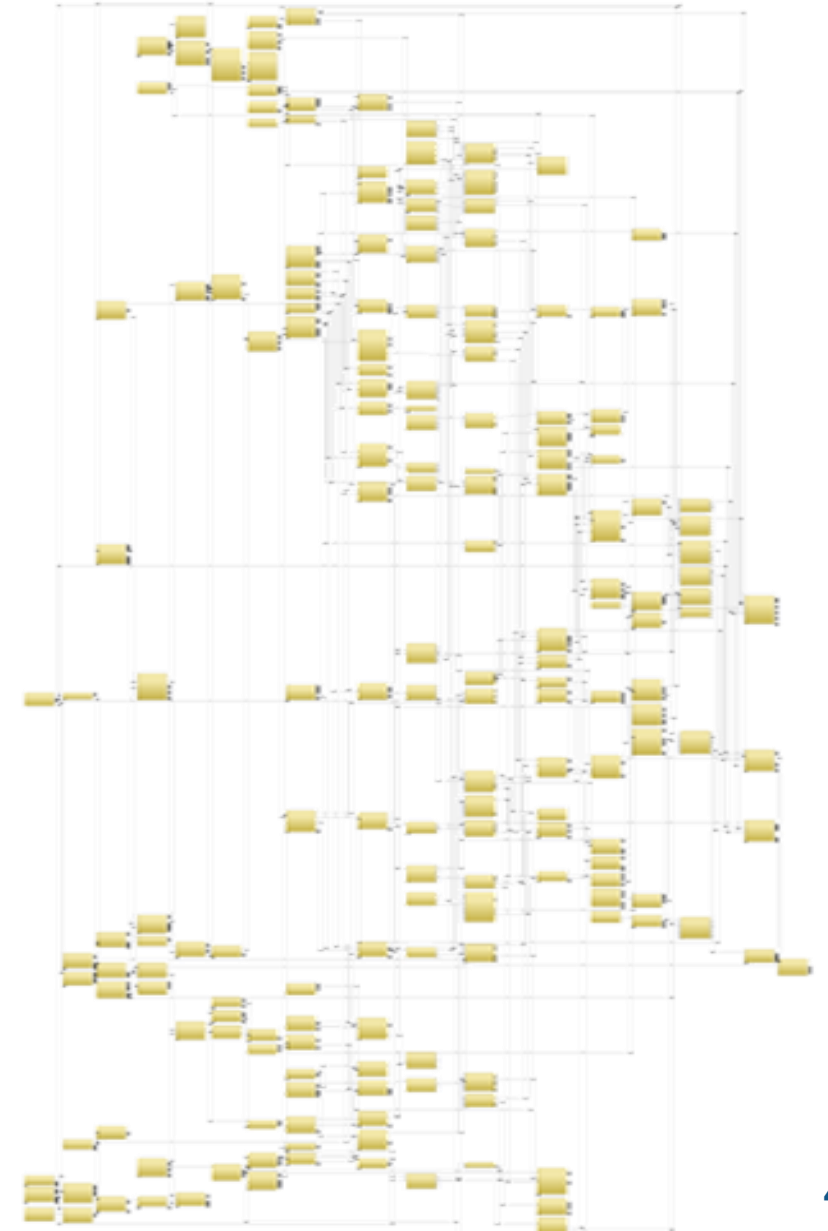

Partition and Modularize Generated Code

Example: Tuning parameter modularization



Issues for Large-scaled Embedded Software Development

Issues \ Work Phase	Modeling	Code Generation
1. Complexity	✓	
2. Integration (Reusability + Scalability)	✓	✓
3. Scheduling	✓	
4. Multi-instantiation		✓



Key Takeaway

- SW modeling pattern importance for effective code generation
 - Component-based modeling
 - Integration in a composition level using Model Reference
 - Export functions/ scheduling components modeling patterns
 - Simulink Function models for access to hardware resources
- Code generation customization framework
 - Code Perspective
 - Embedded Coder Dictionary