
MATLAB EXPO 2018
KOREA

MATLAB EXPO 2018

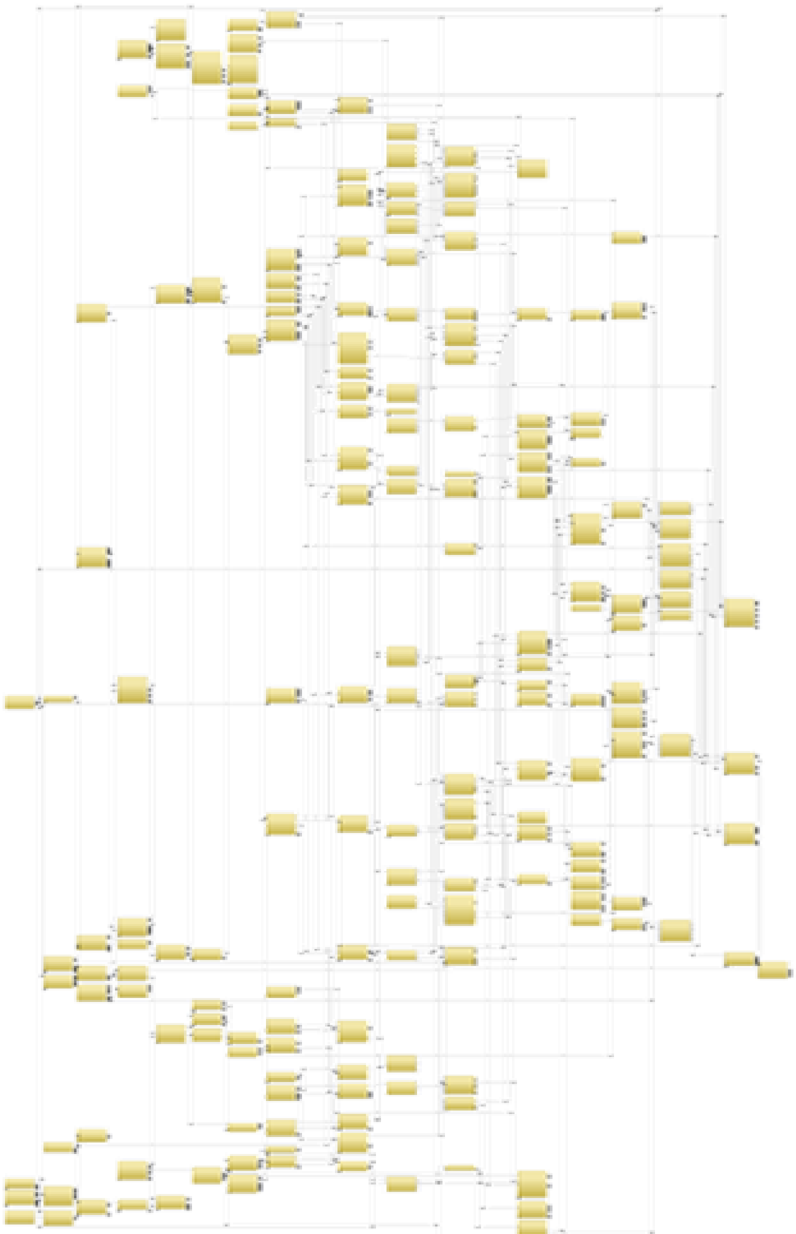
New Perspective for Large and Complex Production Software Development

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		✓

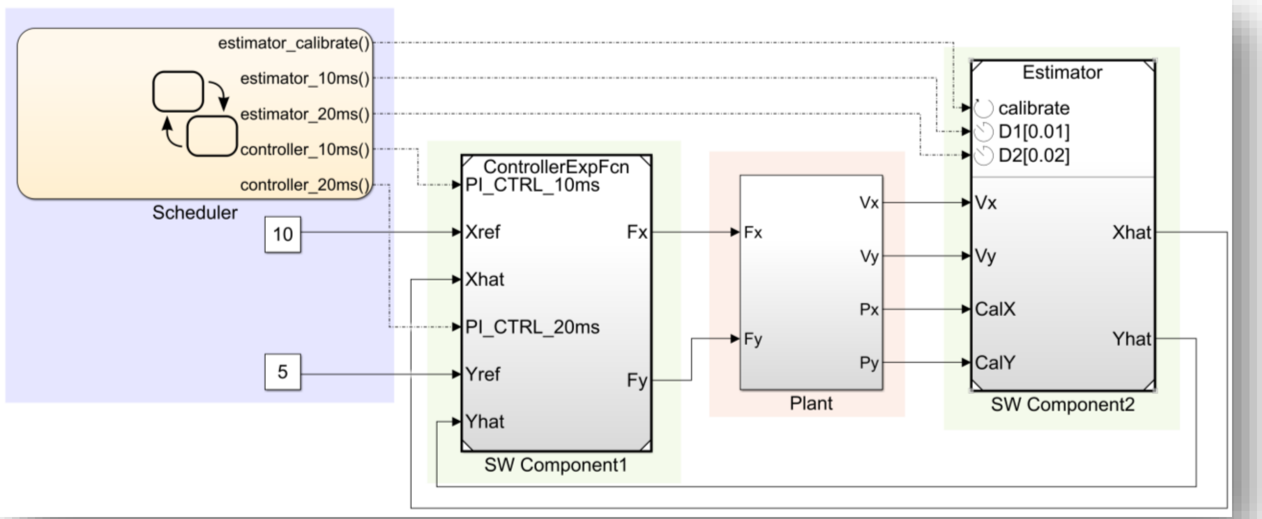


What to Consider for Model-Based Design

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

SW modeling patterns (Component-based design, Integration in a composition level, Component scheduling)

Code generation workflow (Code generation on SW frameworks, Generated code customization)



AUTOSAR

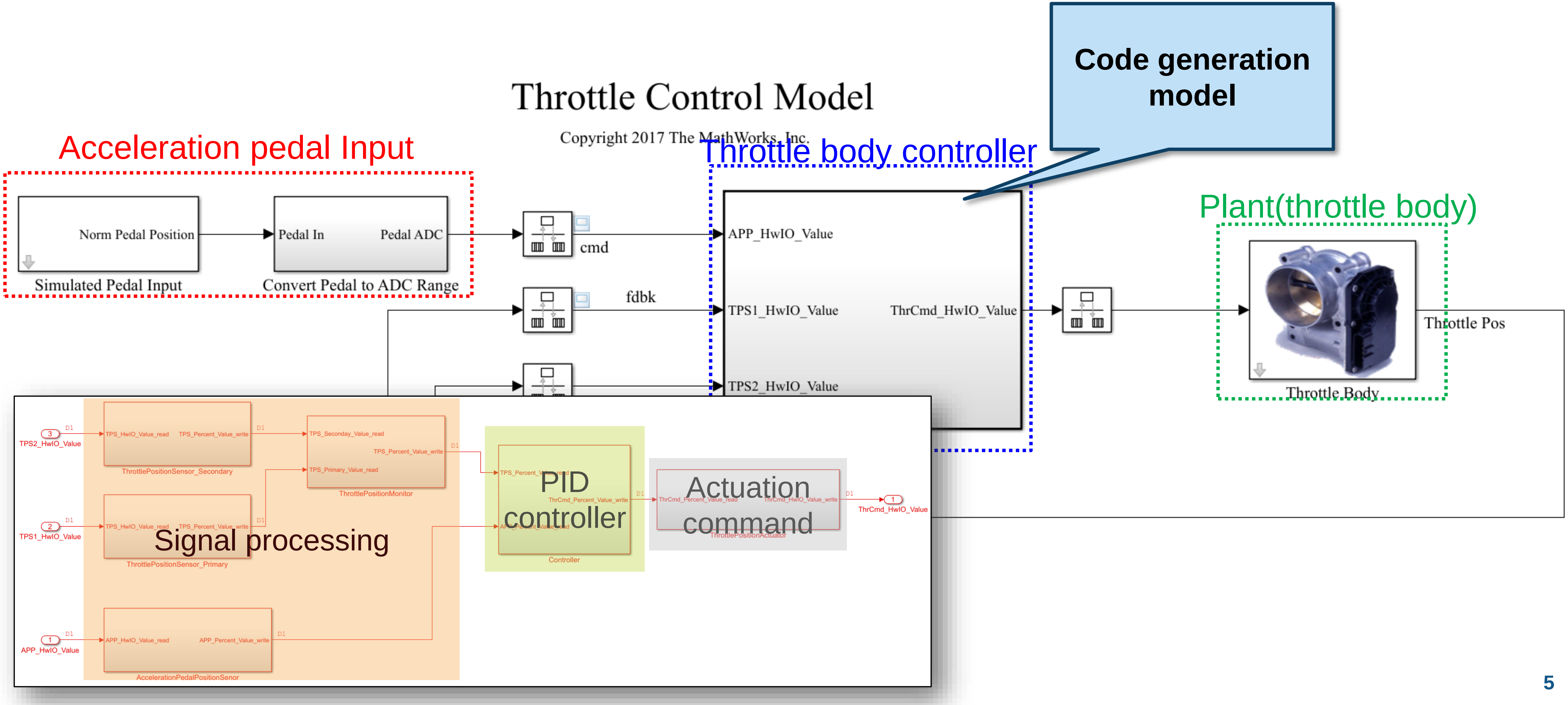
ARINC

ROS

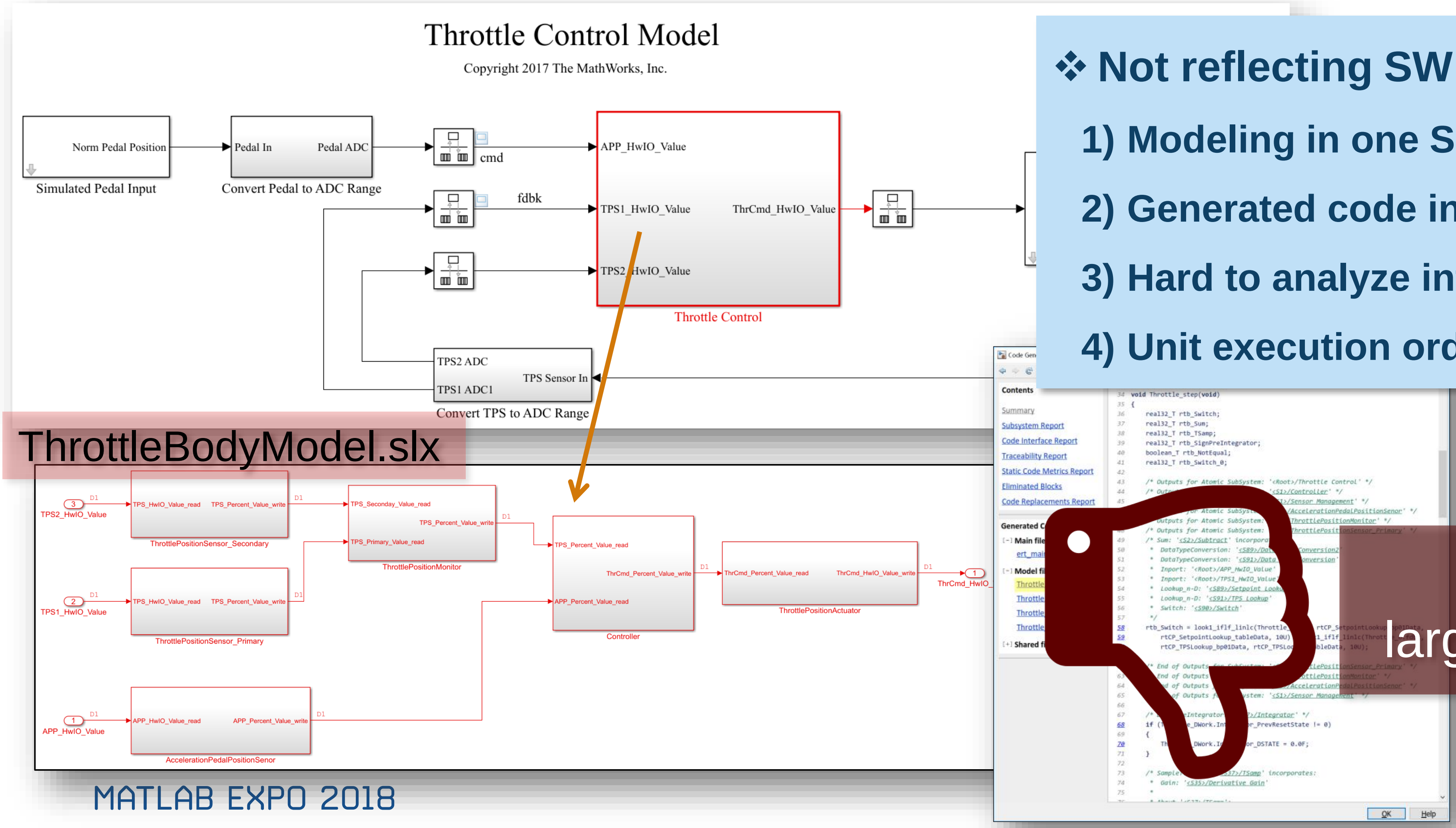
Your-custom Framework

For Software Modeling Patterns

Example: Throttle body control system



Inadequate Software Modeling & Code Generation



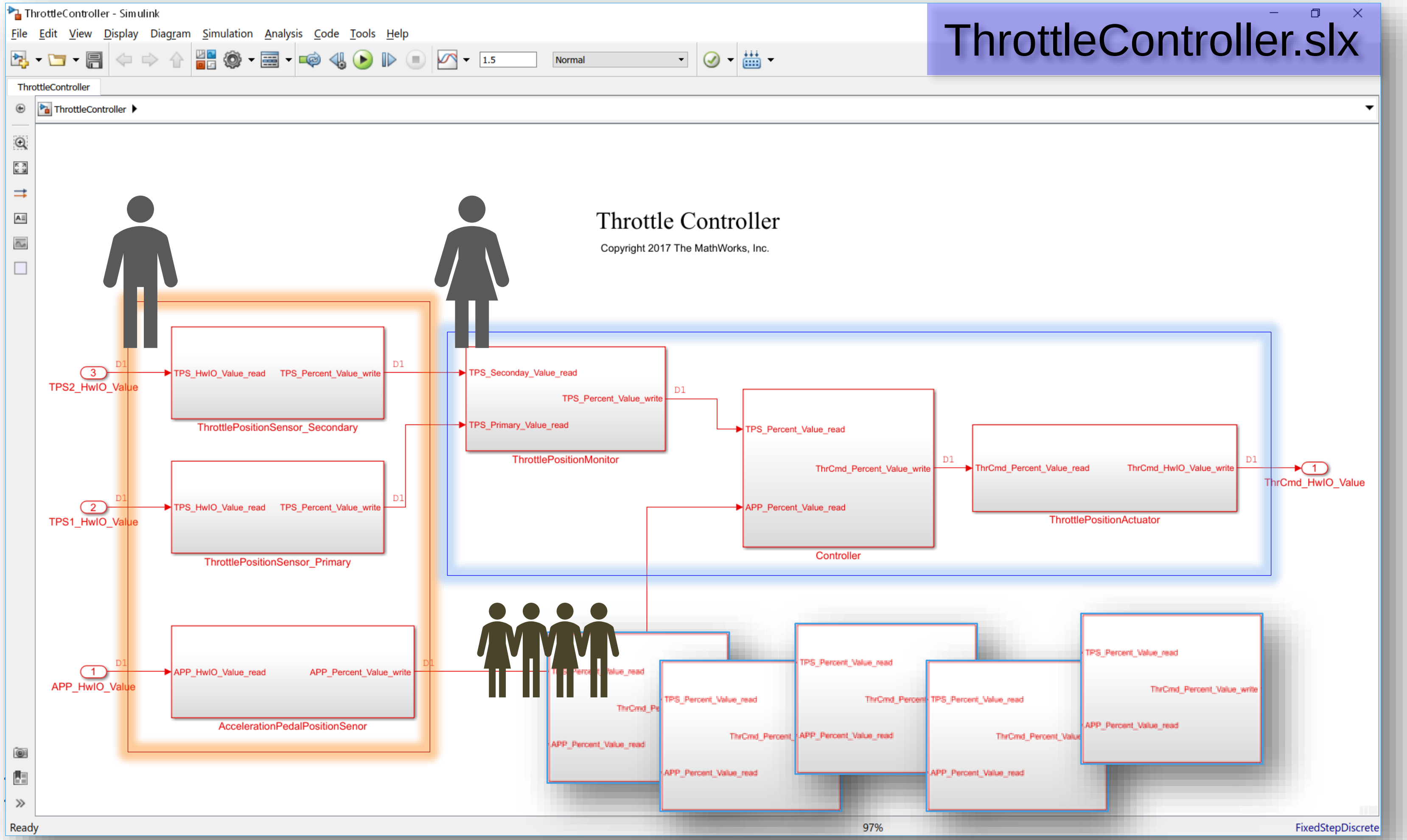
ThrottleBodyModel.slx

- ❖ **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

Not adequate for larger-scale software !!

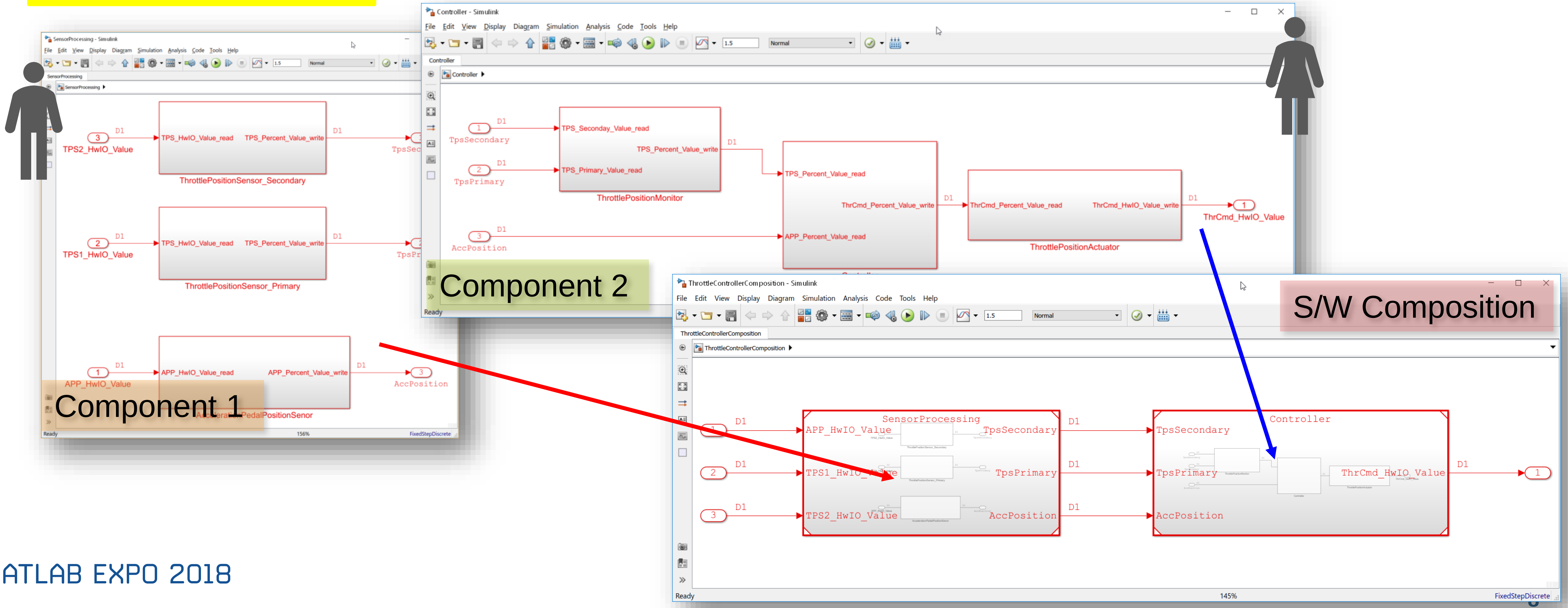
Let’s Start from Software Architecture

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



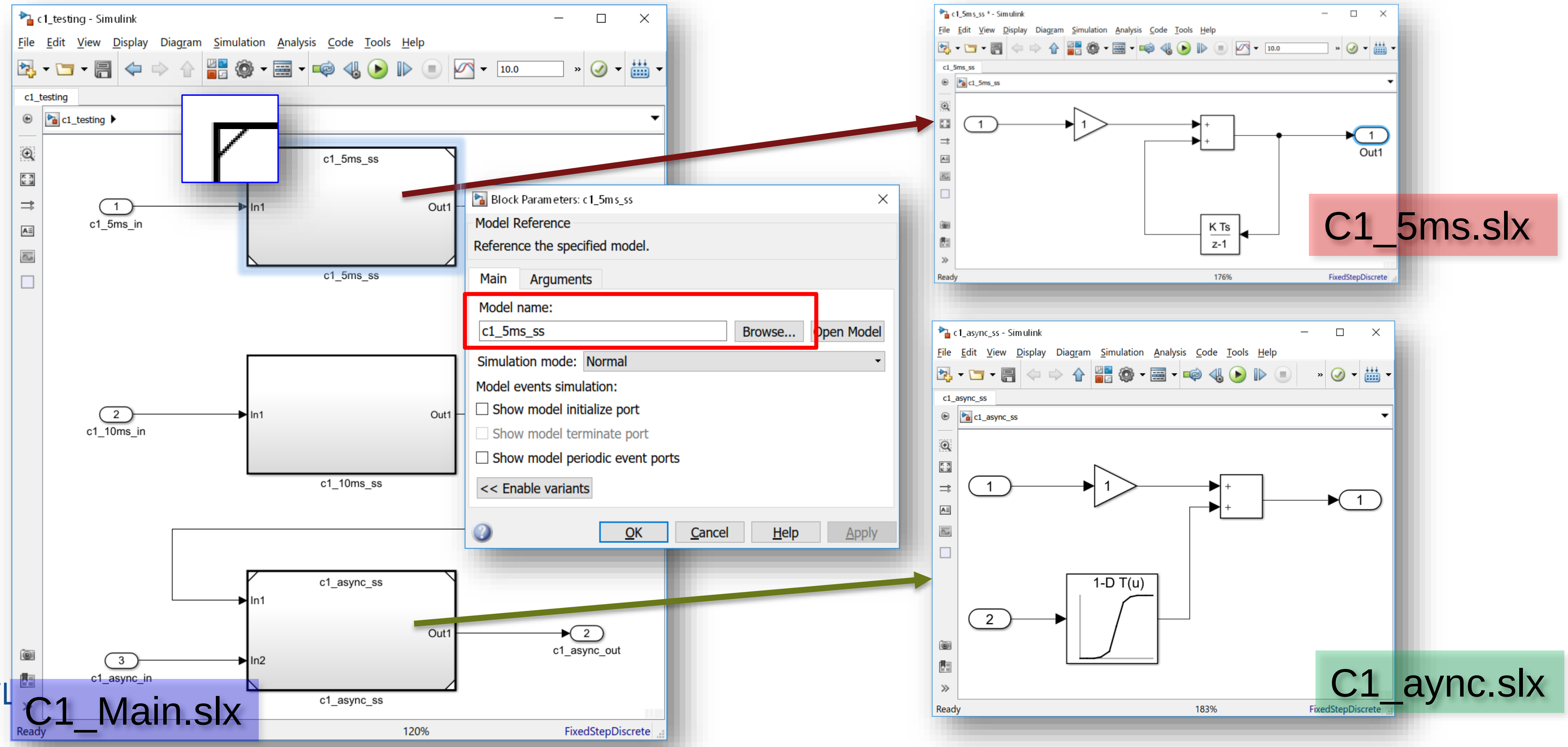
Integration in a Composition Level

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



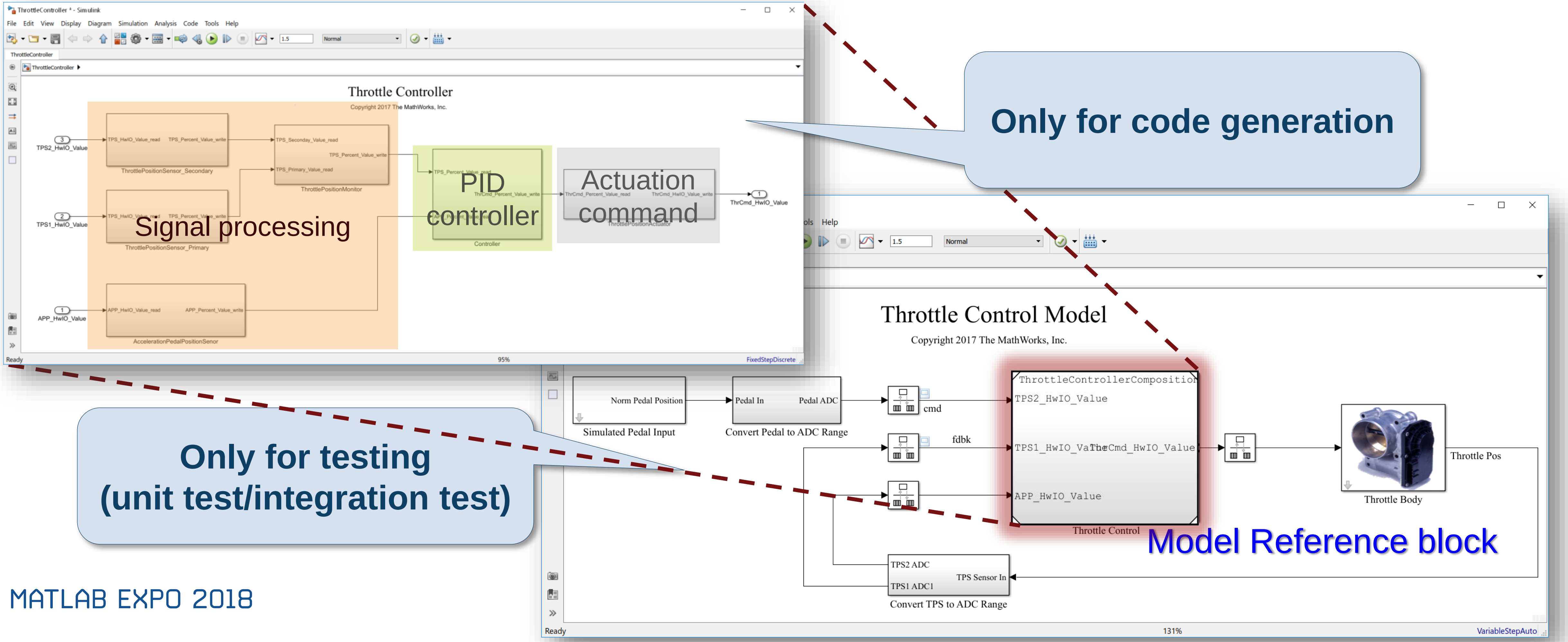
What the Model Reference…?

- Model Reference enables to design models based on SW component



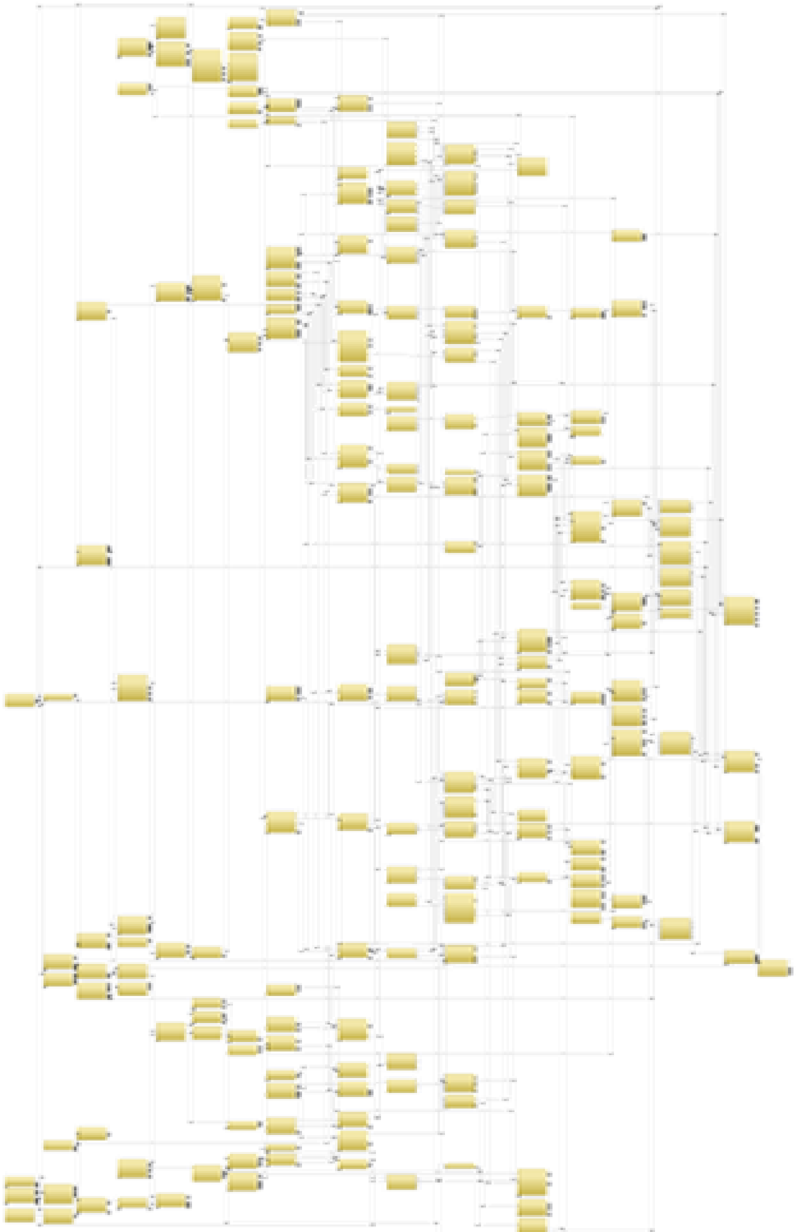
Creating Separate Test Harness Model

- 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		

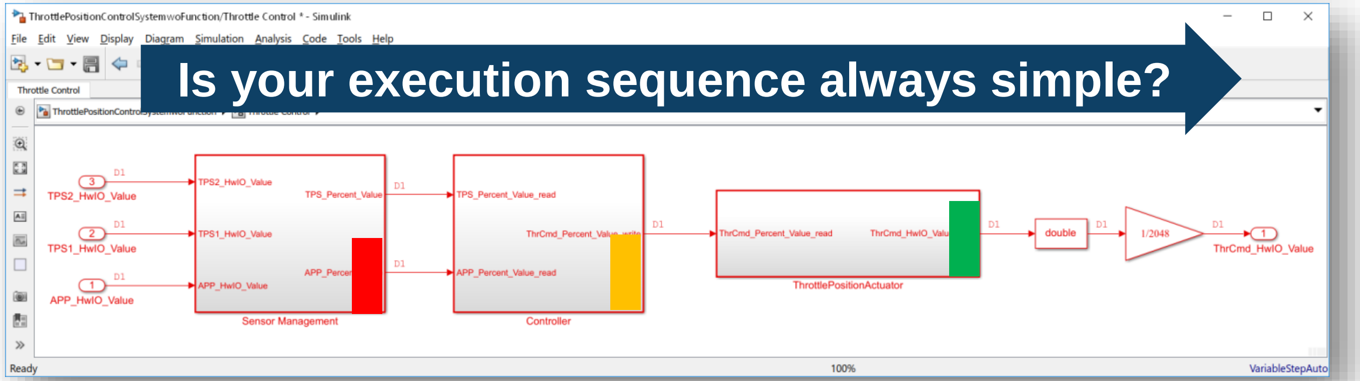


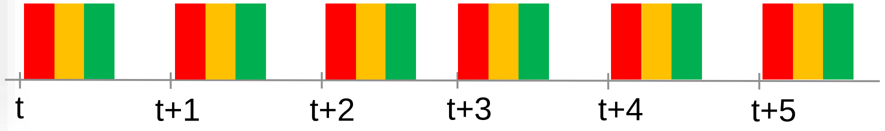
SW Scheduling for Larger-scale Software

- Requirement to analyze the results according to scheduling

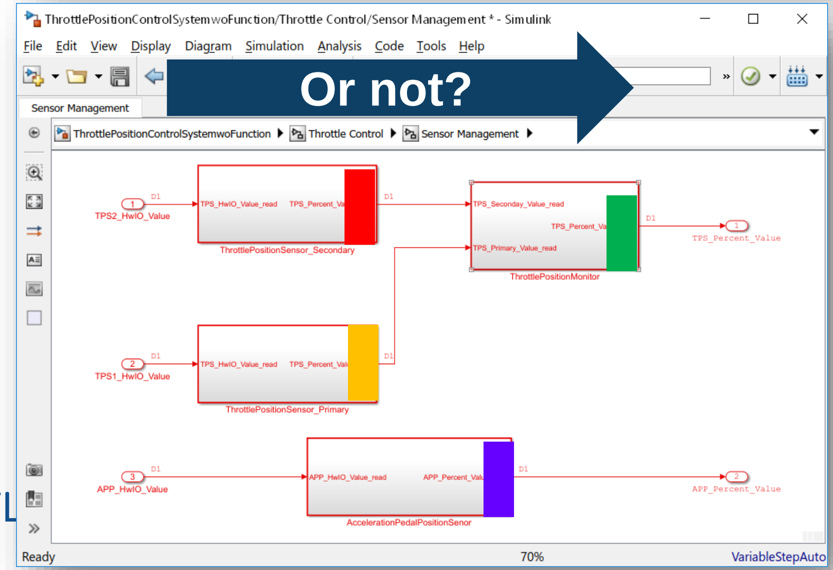
OS/ scheduler
Multicore execution
Multi tasking
…

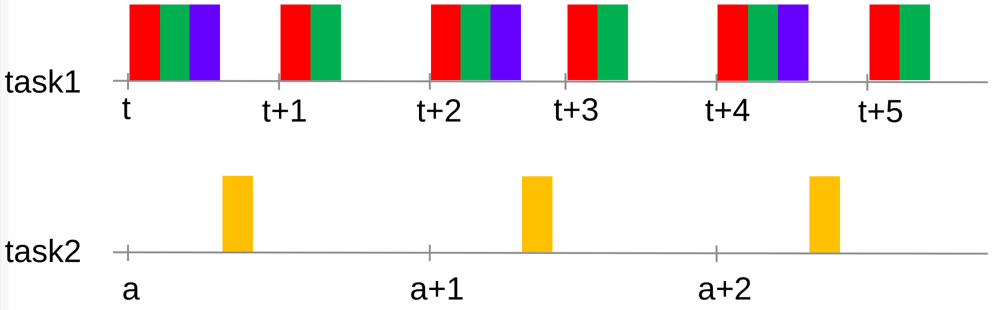
Is your execution sequence always simple?





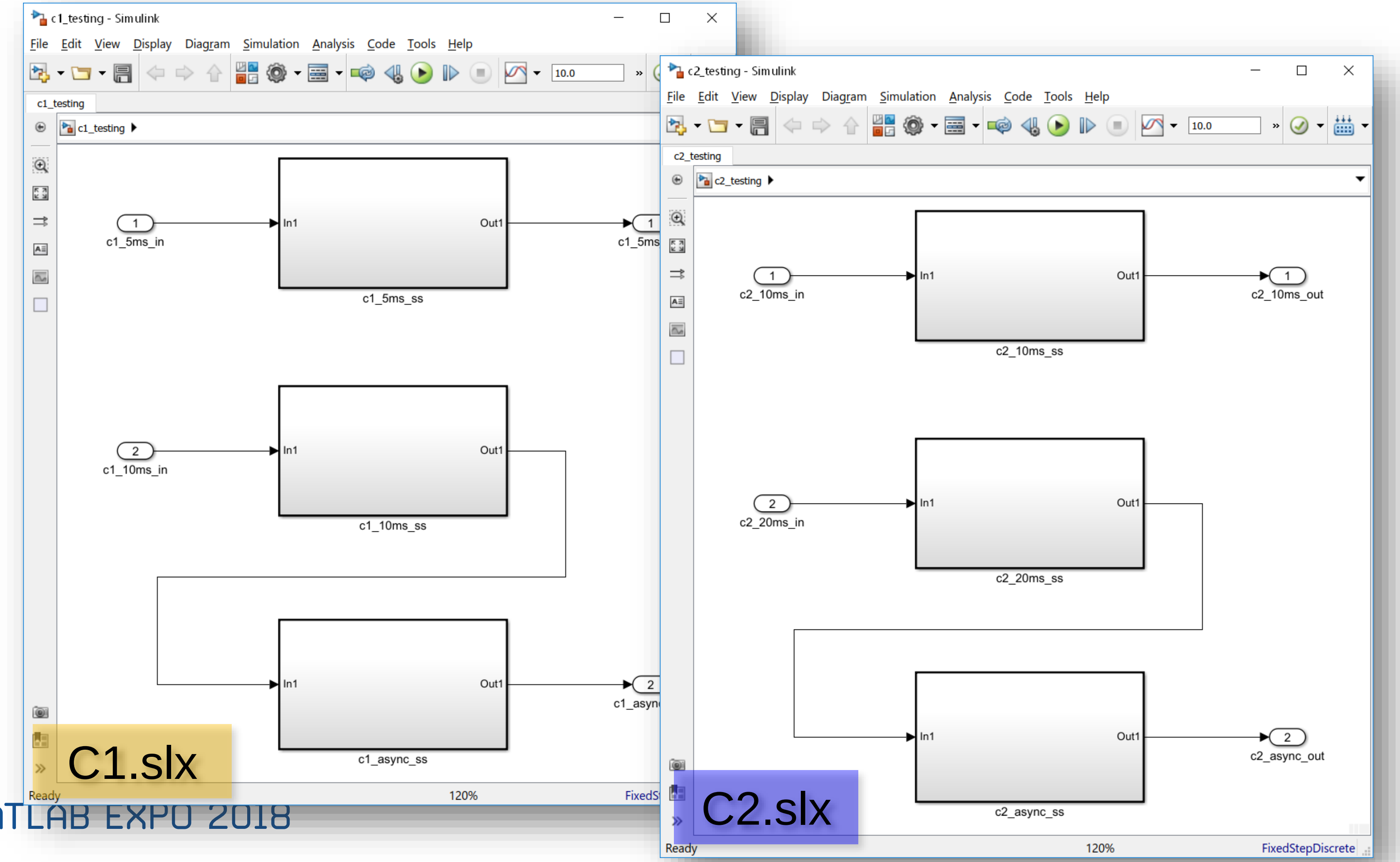
Or not?

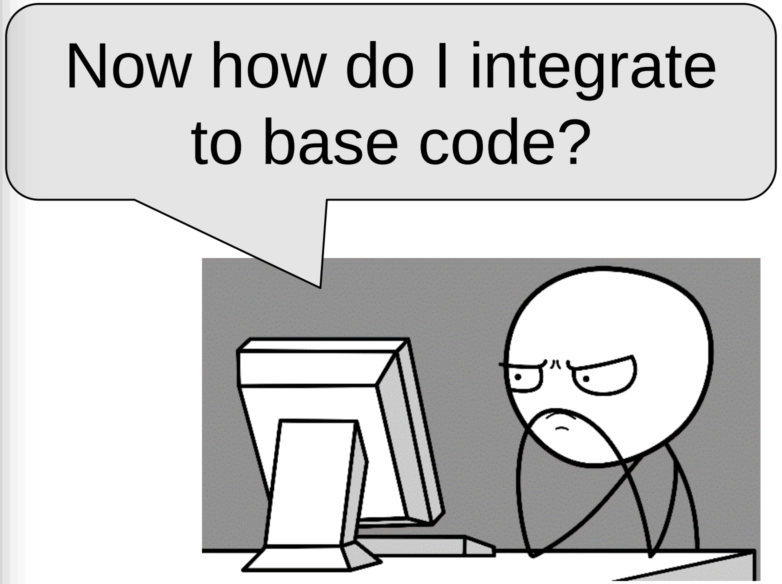




Typical Workflow for Software Integration and Scheduling

- Collecting models for code generation with considering scheduling





Collect Entry Point Functions for Each Component

File: c1.c

```c
#include "c1.h"

DW rtDW;
ExtU rtU;
ExtY rtY;
void c1_step0(void)
{
  rtY.c1_10ms_out = 2.0 * rtU.c1_10m
}

void c1_step1(void)
{
  real_T rtb_RateTransition;
  rtDW.RateTransition_semaphoreTaken
  rtb_RateTransition =
    rtDW.RateTransition_Buffer[rtDW.
  rtY.c1_20ms_out = rtb_RateTransiti
}

void c1_async(void)
{
  rtDW.RateTransition_Buffer[rtDW.Ra
    rtU.c1_async_in;
  rtDW.RateTransition_ActiveBufIdx =
    == 0);
}

void c1_initialize(void)
{
}

void c1_terminate(void)
{
}
```

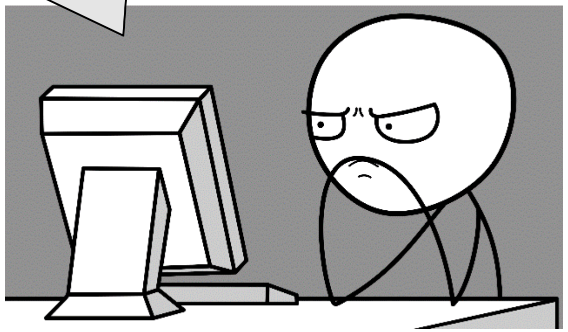
File: c2.c

```c
#include "c2.h"

DW rtDW;
ExtU rtU;
ExtY rtY;
void c2_step0(void)
{
  rtY.c2_10ms_out = rtU.c2_10ms_in +
}

void c2_step1(void)
{
  real_T rtb_In1;
  rtb_In1 = rtU.c1_20ms_in;
  rtDW.RateTransition_Buffer[rtDW.Rat
  rtDW.RateTransition_ActiveBufIdx =
    0);
}

void c2_async(void)
{
  rtDW.RateTransition =
    rtDW.RateTransition_Buffer[rtDW.R
  rtY.c2_async_out = rtDW.UnitDelay_D
  rtDW.UnitDelay_DSTATE = rtDW.RateTr
}

void c2_initialize(void)
{
}

void c2_terminate(void)
{
}
```

c1
c1_step0()
c1_step1()
c1_async()

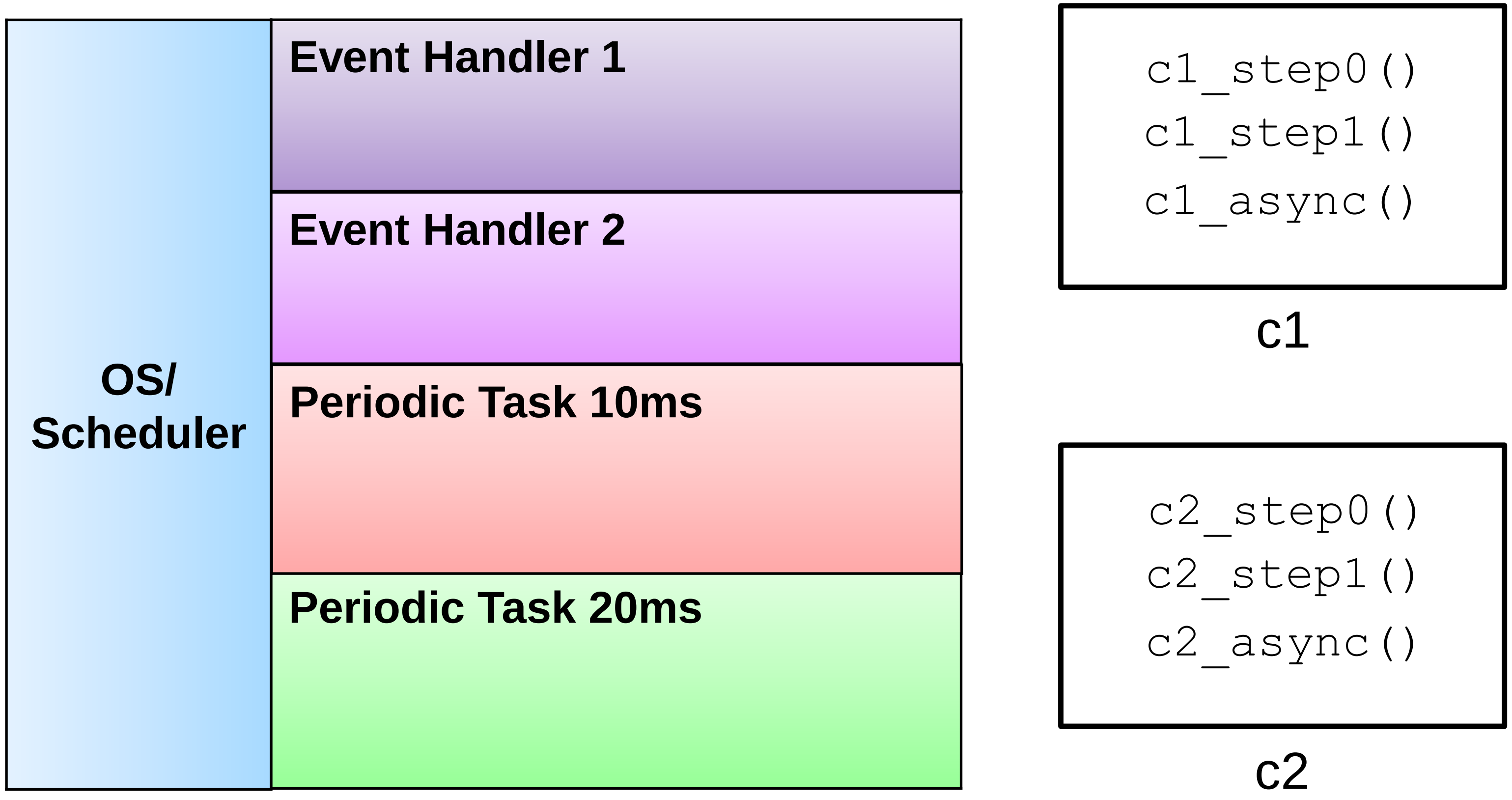
c2
c2_step0()
c2_step1()
c2_async()

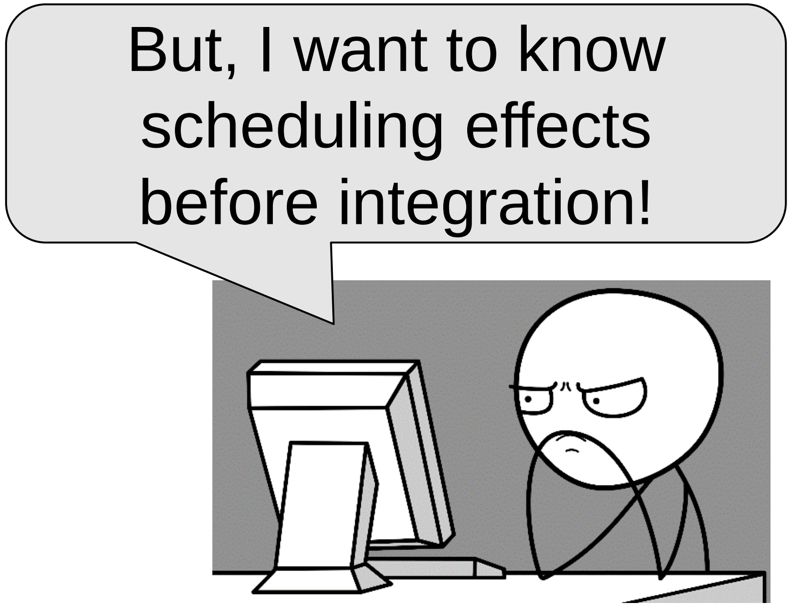
And, how do I create scheduling orders?



Application Integrated to Base Software

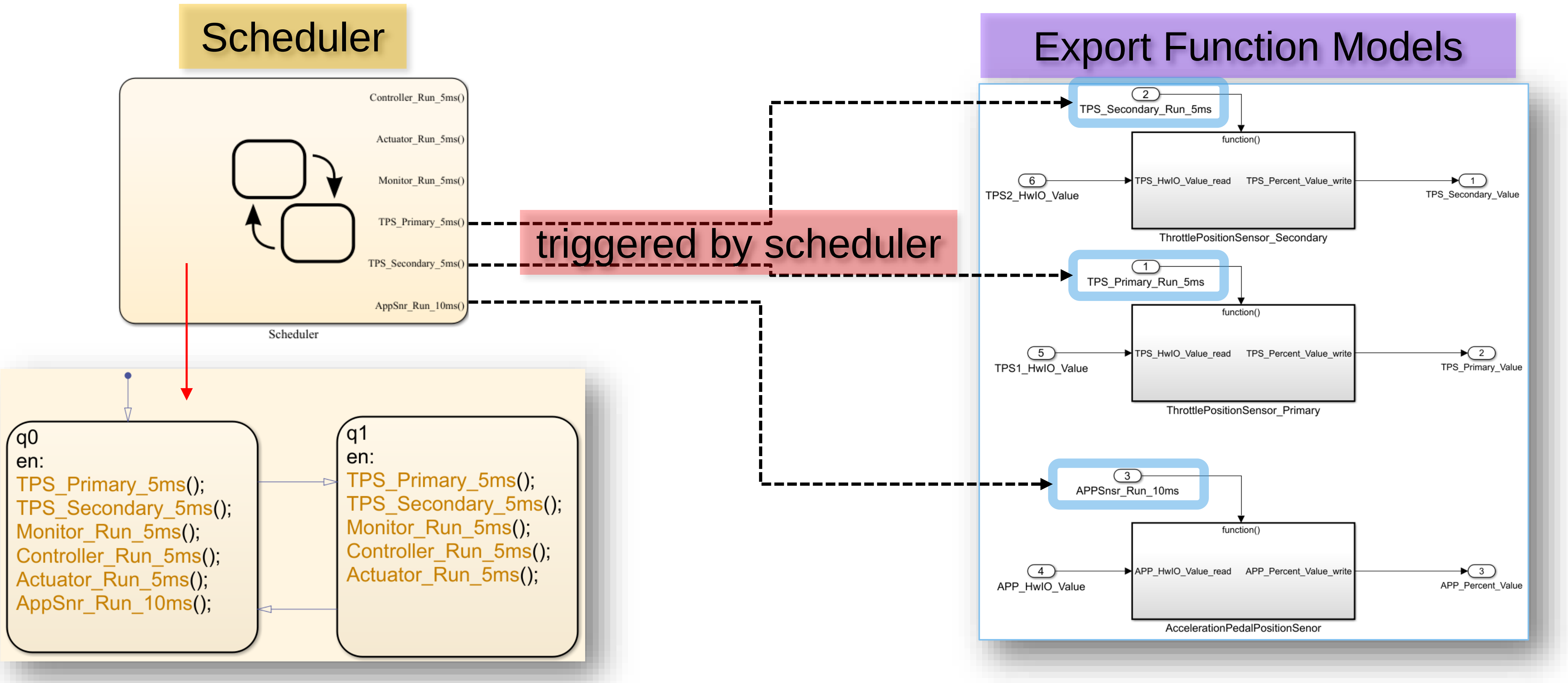
- Integrate entry point functions from components with run-time environment





Software Testing with Scheduling Effects

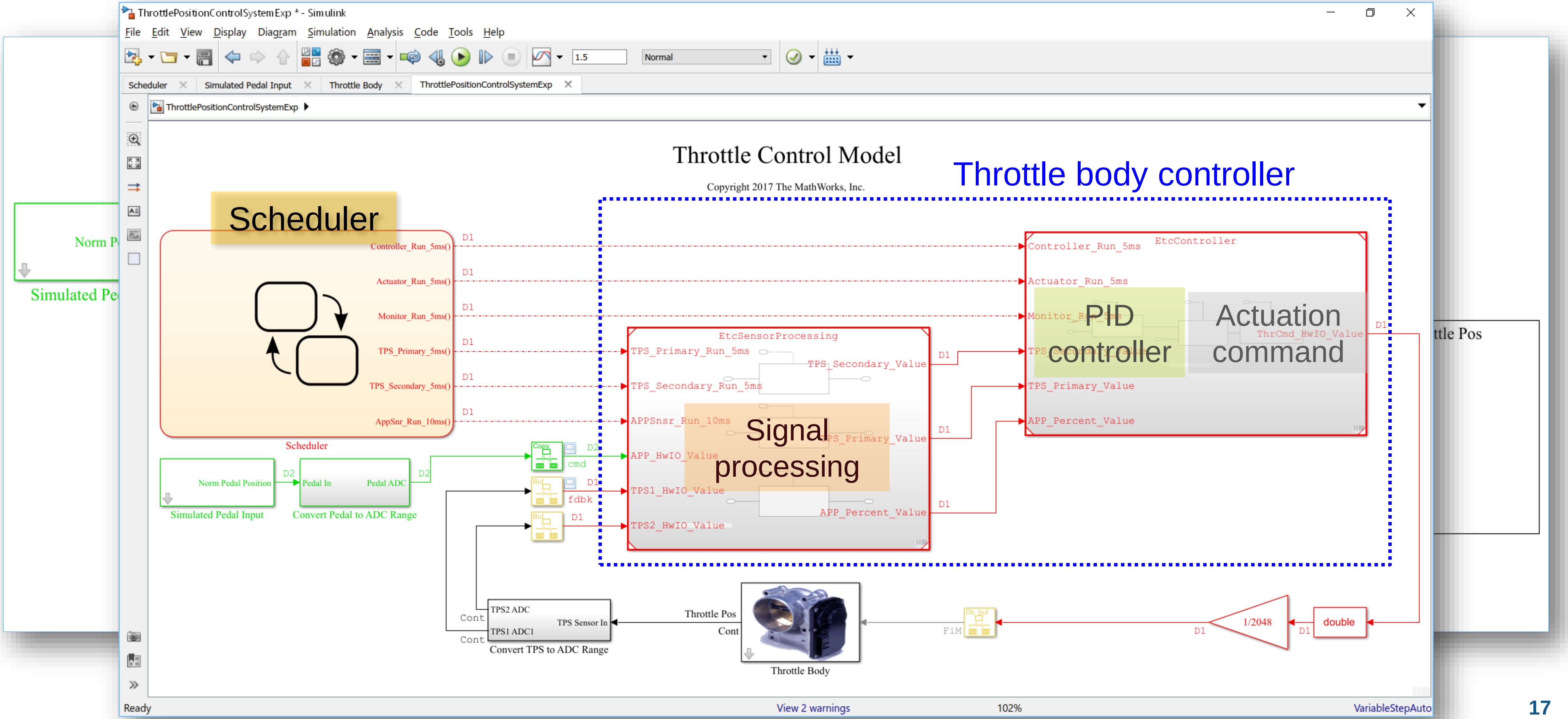
Export Function



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

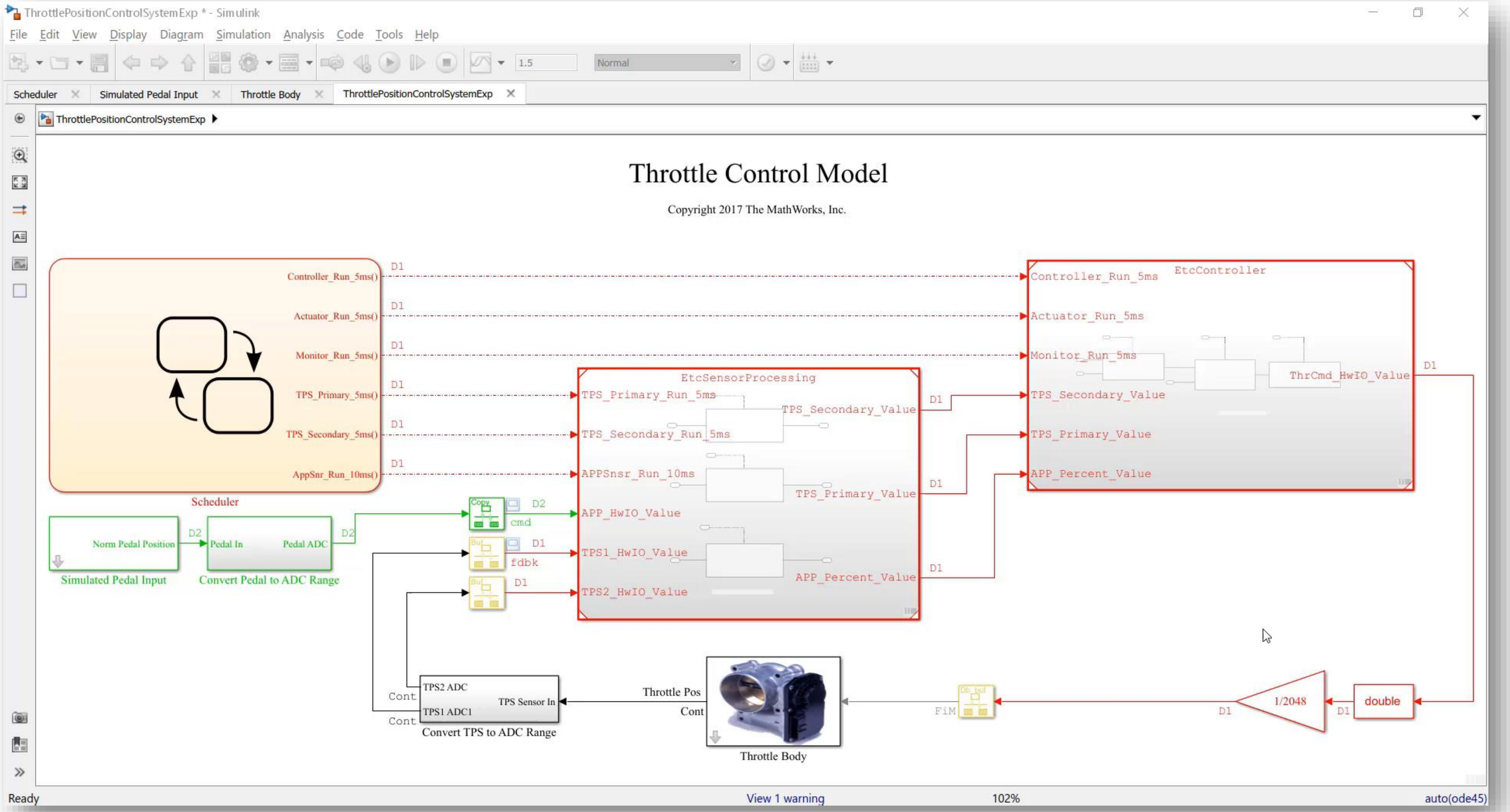
Redesigned Model with Scheduler and Export Functions

Export Function



Demo: SW Modeling with Export Functions

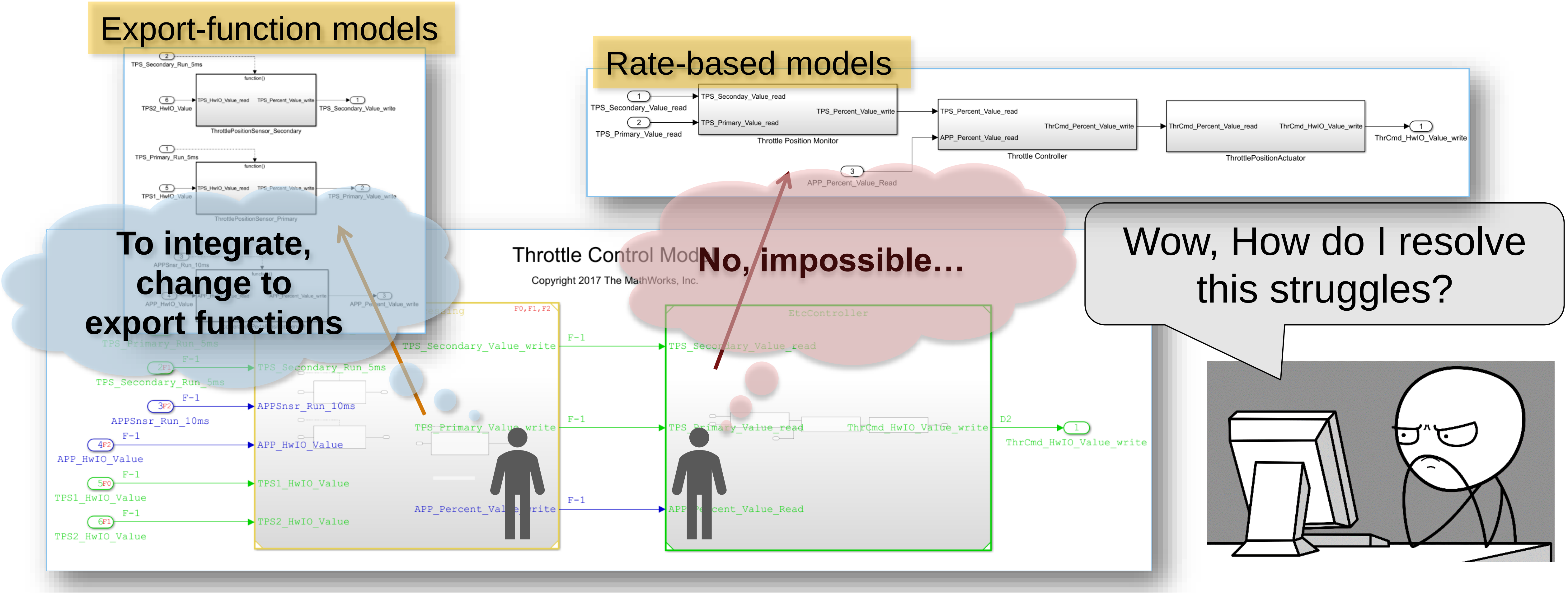
Export Function



Testing Scheduling Effects from Different Patterned Models

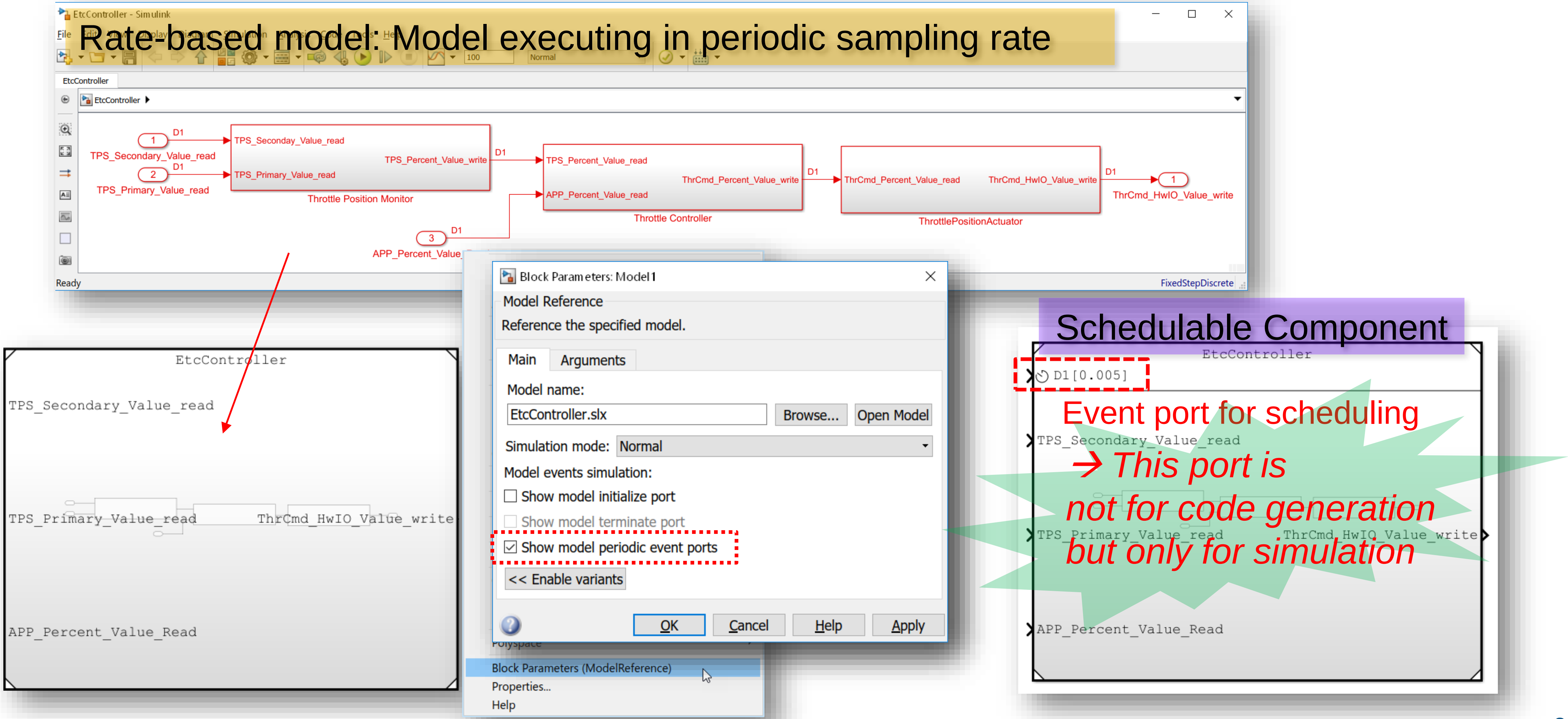
Schedulable Component

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



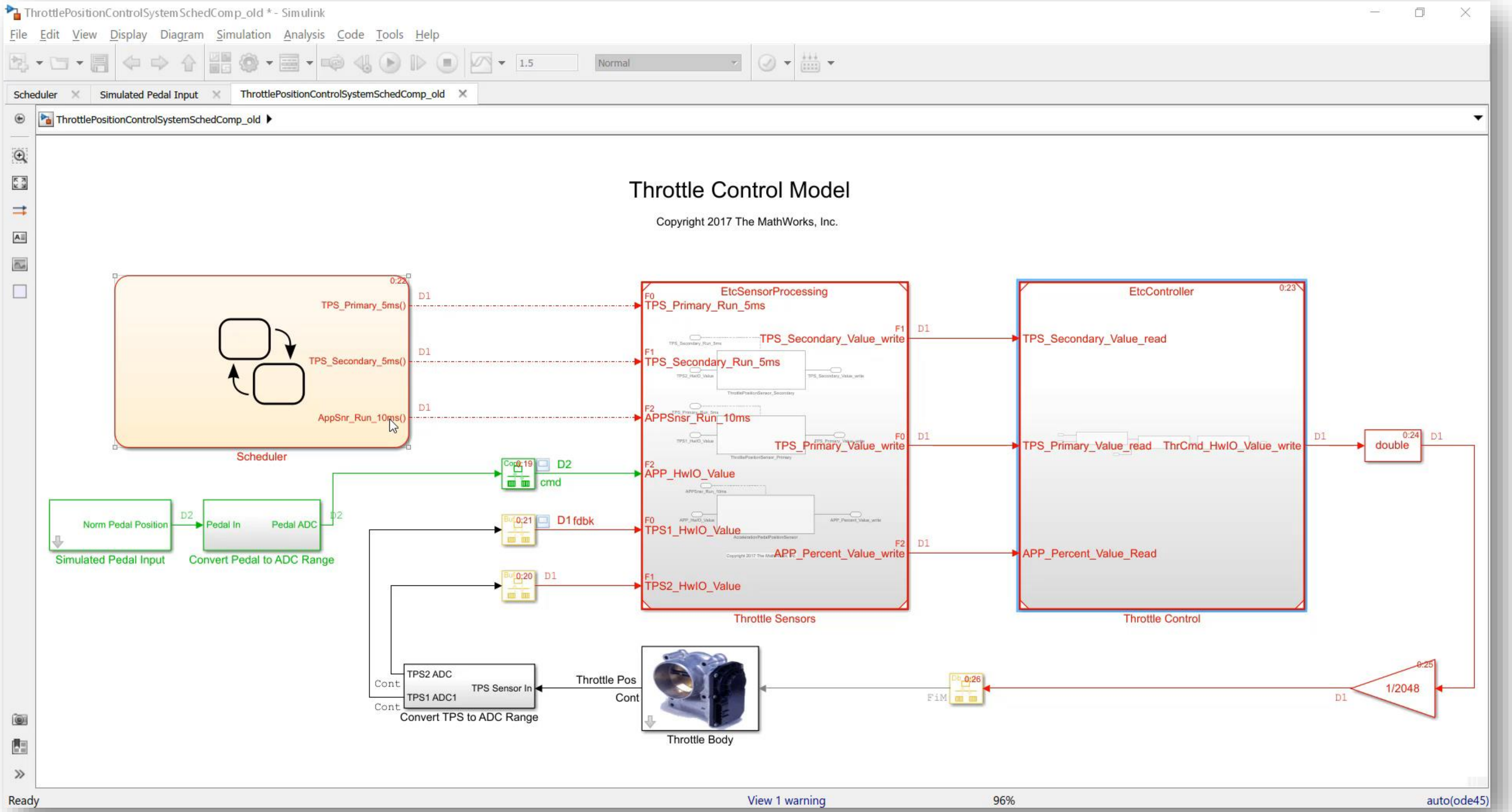
Creating Schedulable Component from Model Reference

Schedulable Component



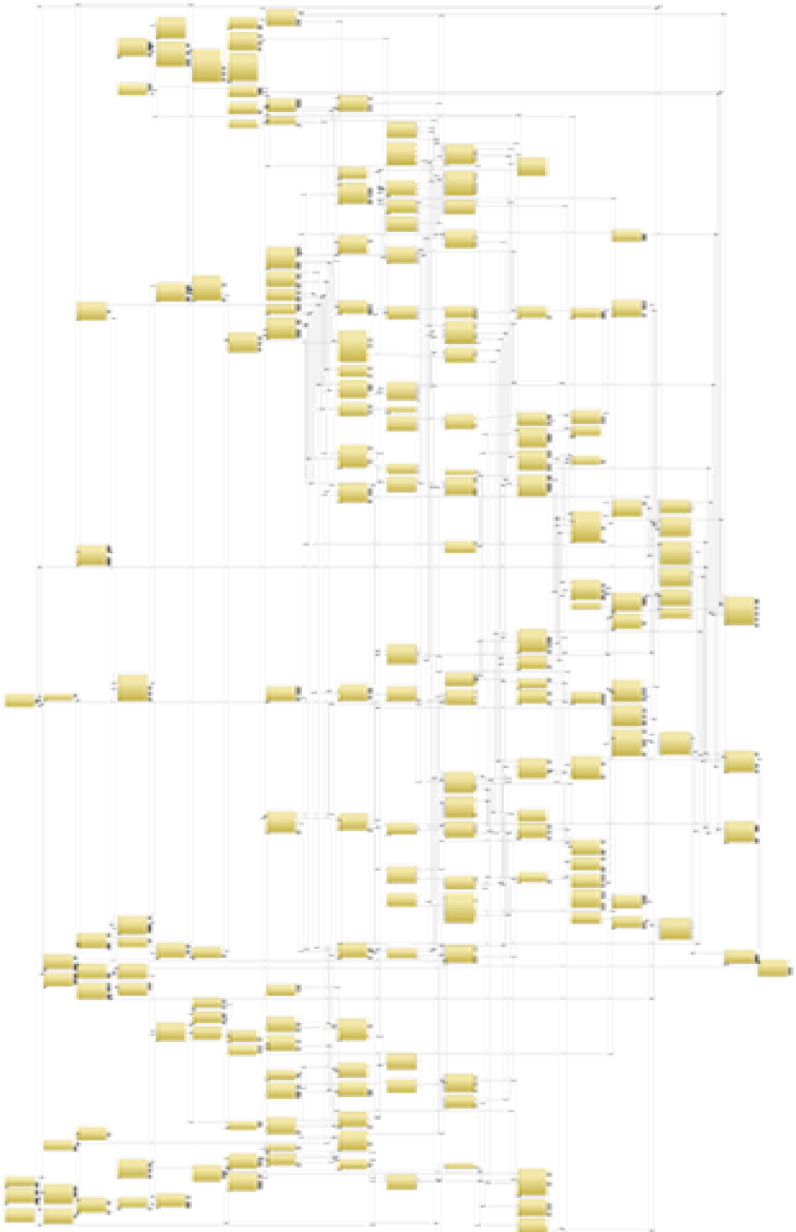
Demo: SW Modeling with Schedulable Components

Schedulable Component



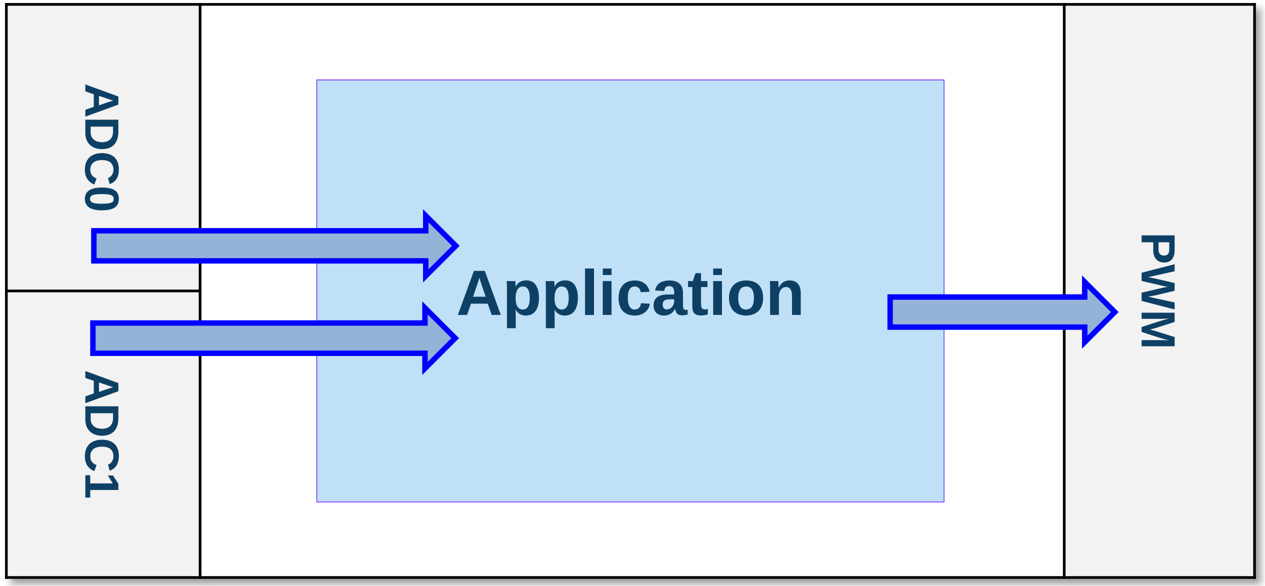
Issues for Large-scaled Embedded Software Development

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



Modeling for Access to Hardware Resources

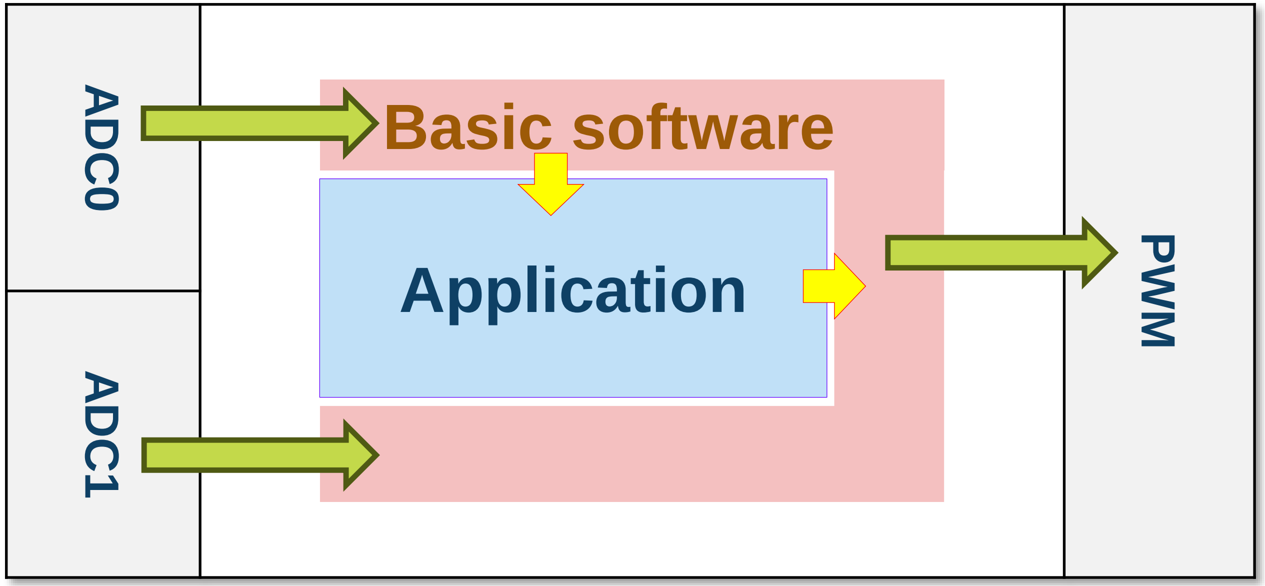
Simulink Function



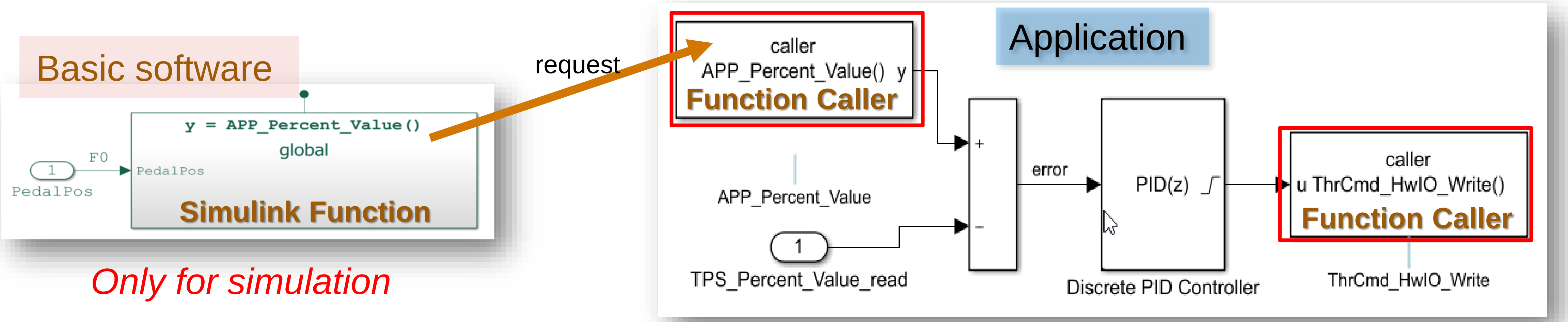
- Some application SW does not process external signals directly. Or…

Modeling for Access to Hardware Resources

Simulink Function

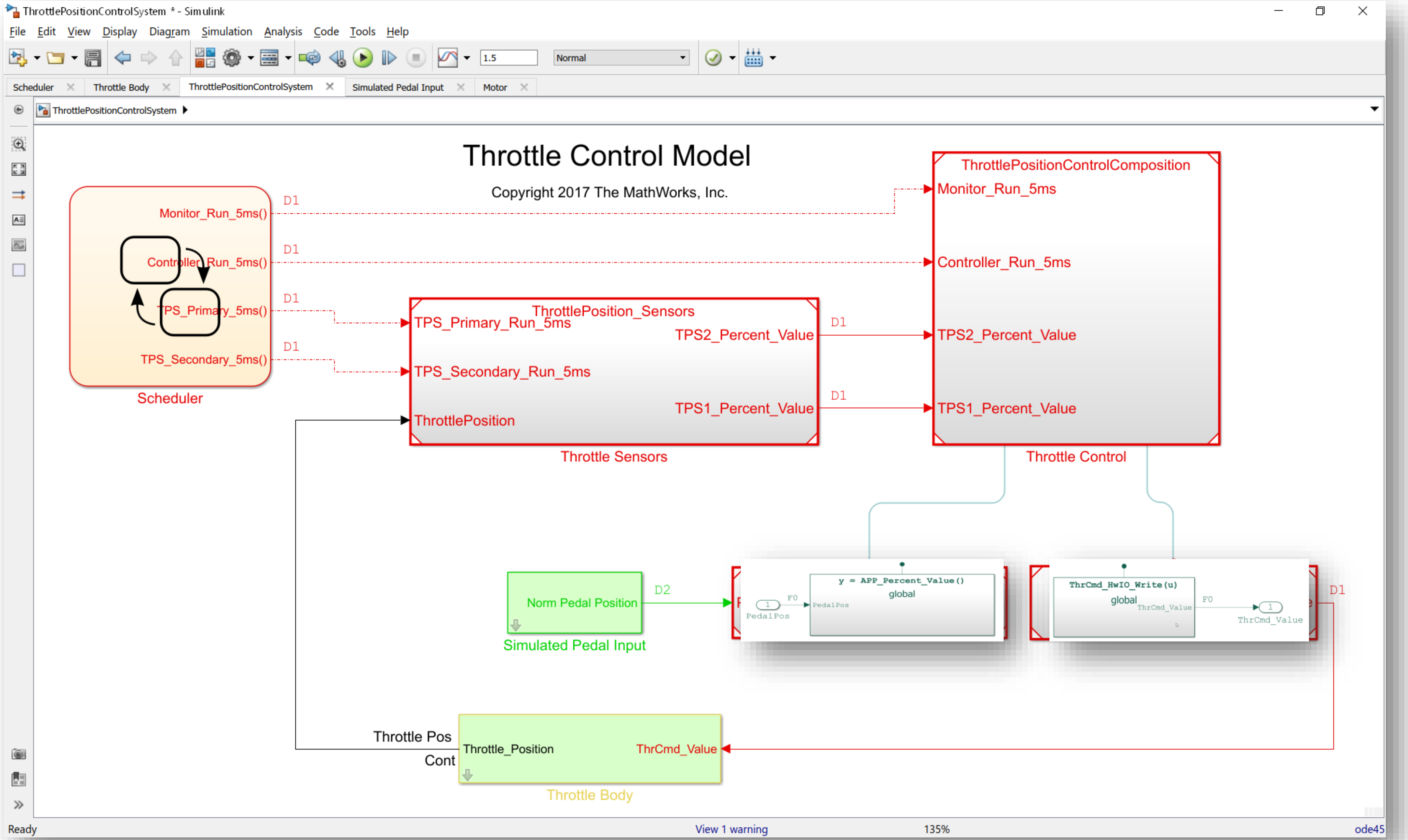


- Some application SW does not process external signals directly. Or…
- External signals are processed in BSW or HAL and accessed by applications
- Application software use APIs to request or send data



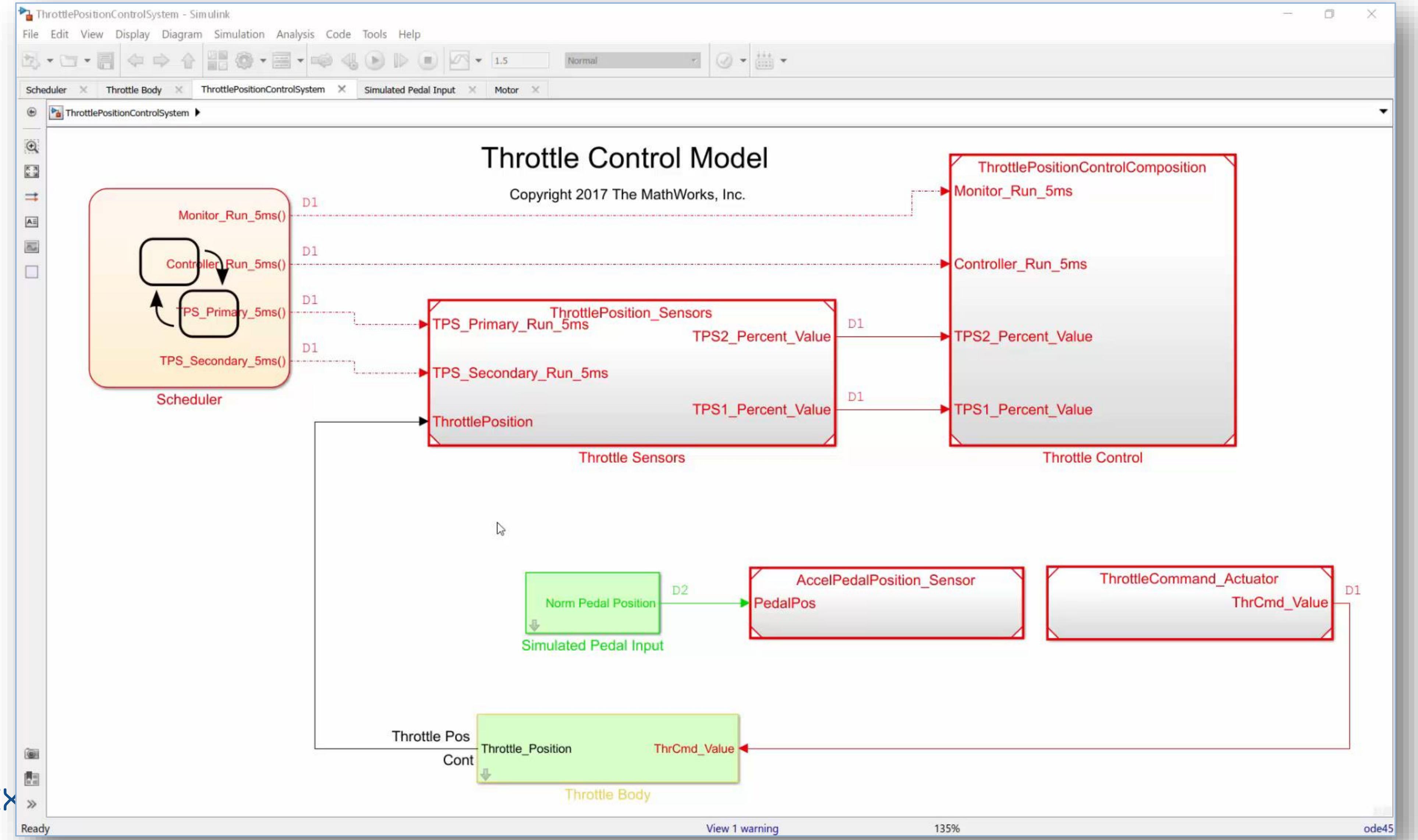
Access to Shared Resources with Simulink Functions

Simulink Function



Demo: SW Modeling with Simulink Functions

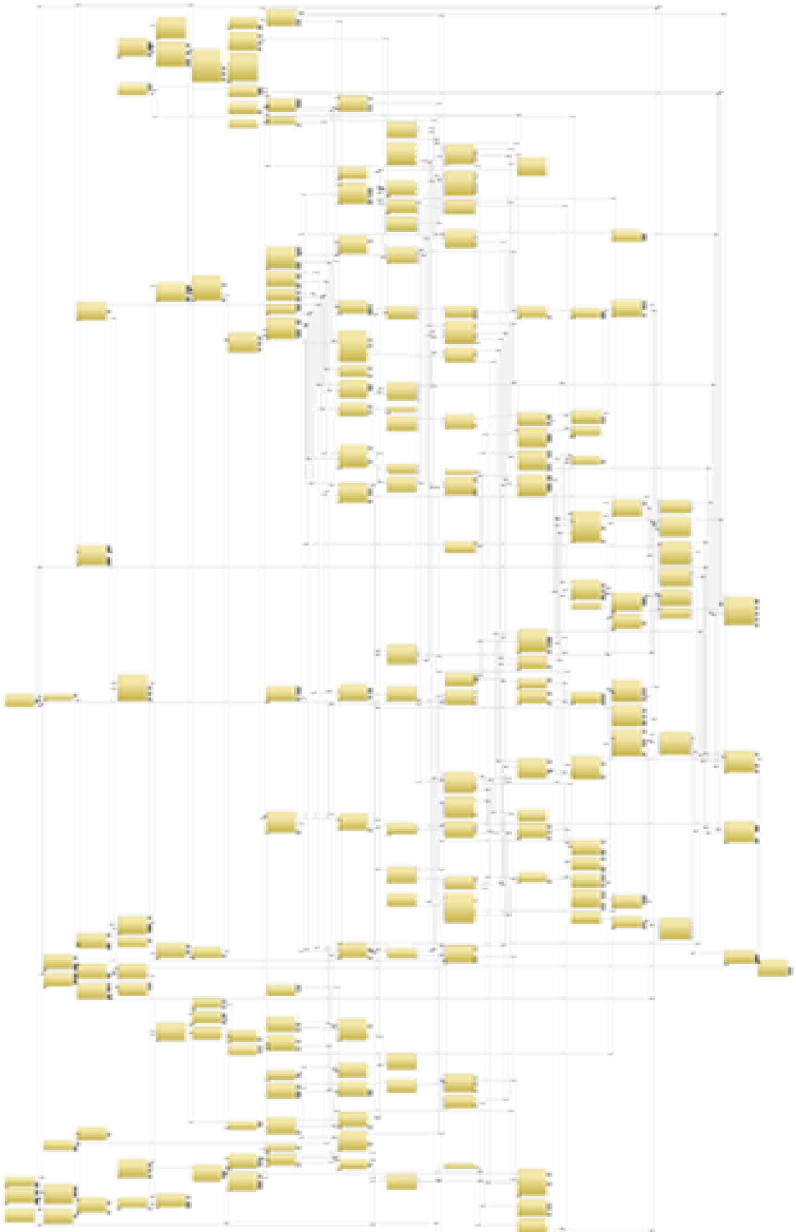
Simulink Function



MATLAB EX

Issues for Large-scaled Embedded Software Development

Work Phase Issues	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

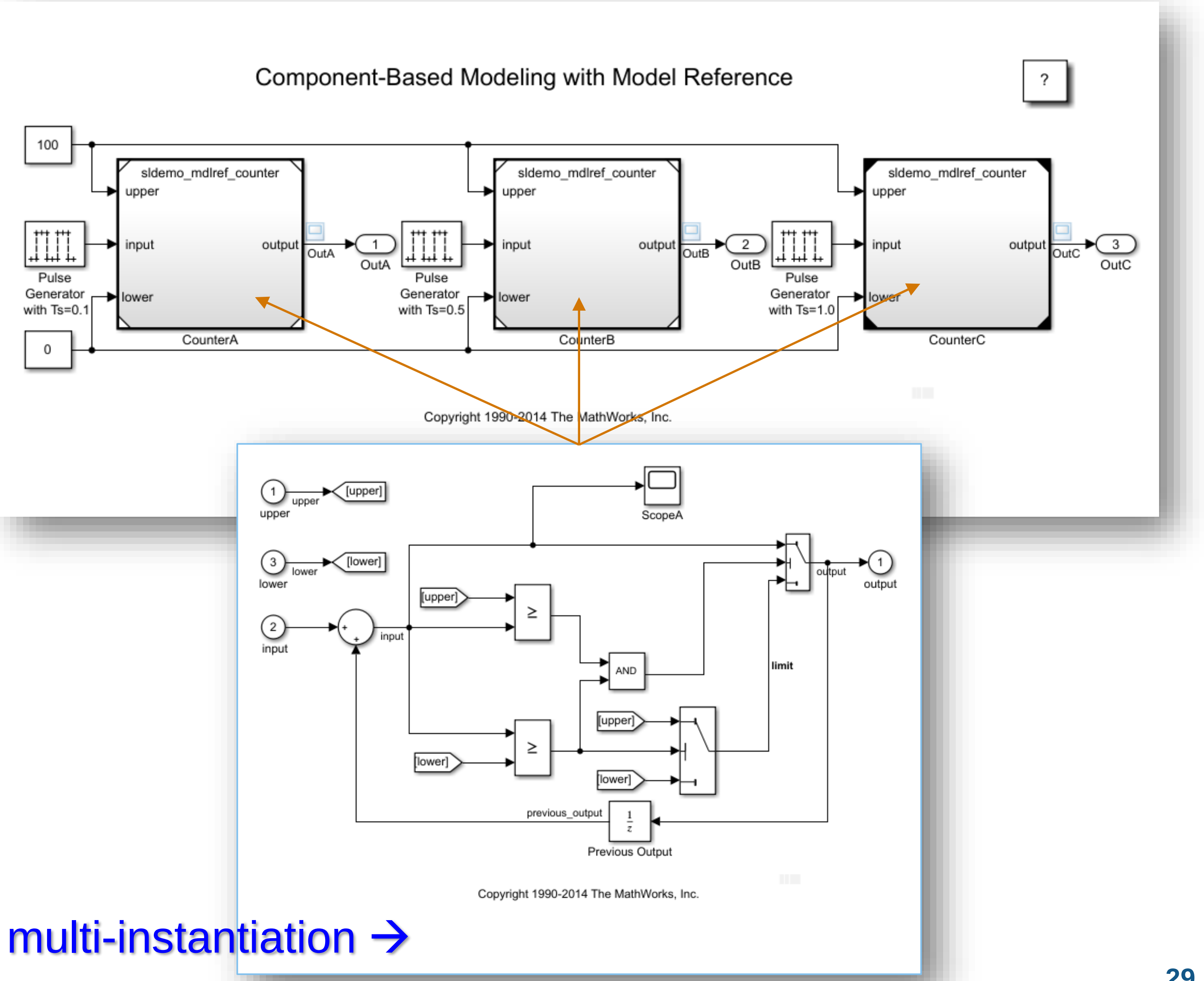
- C code generation from Simulink model(webinar)
  https://kr.mathworks.com/videos/software-design-and-c-code-generation-using-simulink-116860.html?elqsid=1524127259550&potential_use=Commercial

- 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



Generating code for multi-instantiation →

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_mdlref_counter_r_step(void)
{
```

After(reusable)

```
/* Model step function */
void sldemo_mdlref_counter_r_step(RT_MODEL_sldemo_mdlref_counter_r_T *const
  sldemo_mdlref_counter_r_M, ExtU_sldemo_mdlref_counter_r_T
  *sldemo_mdlref_counter_r_U, ExtY_sldemo_mdlref_counter_r_T
  *sldemo_mdlref_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

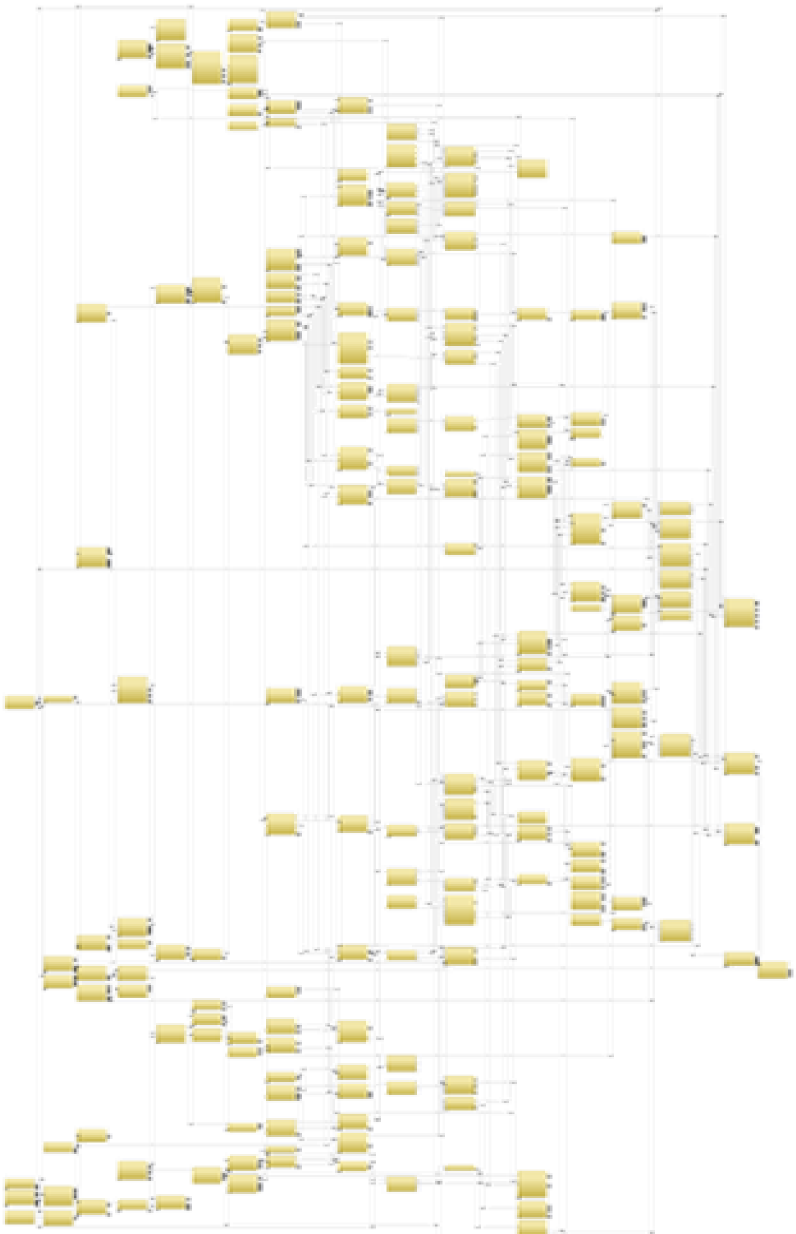
Configure C++ Class Interface

Creating instances

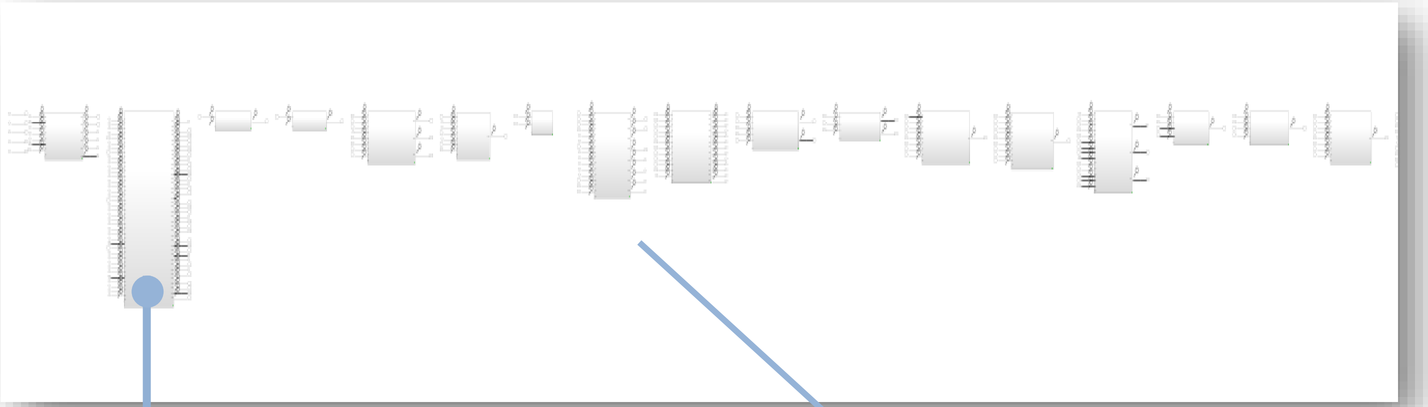
```
// model instance variable for '<Root>/CounterA'
sldemo_mdlref_counter_rModelClass CounterAMDLOBJ1;

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

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

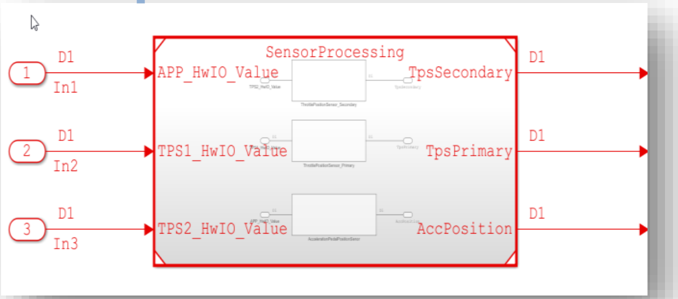
Issues for Large-scaled Embedded Software Development

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

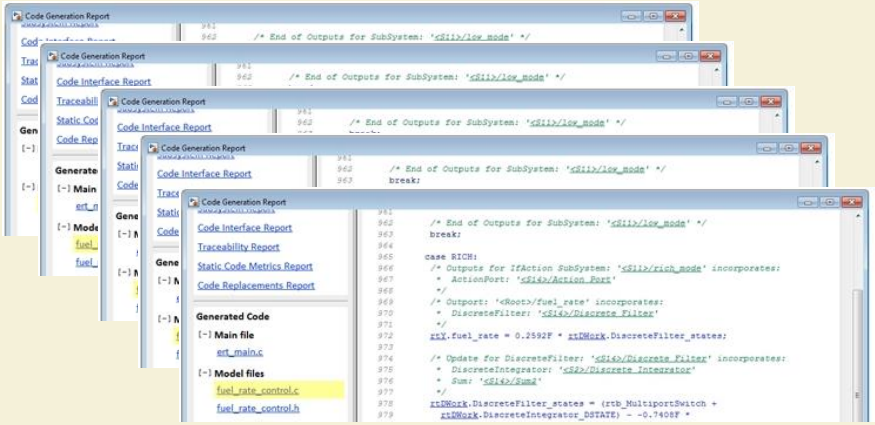


Emergence of the Software Framework





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

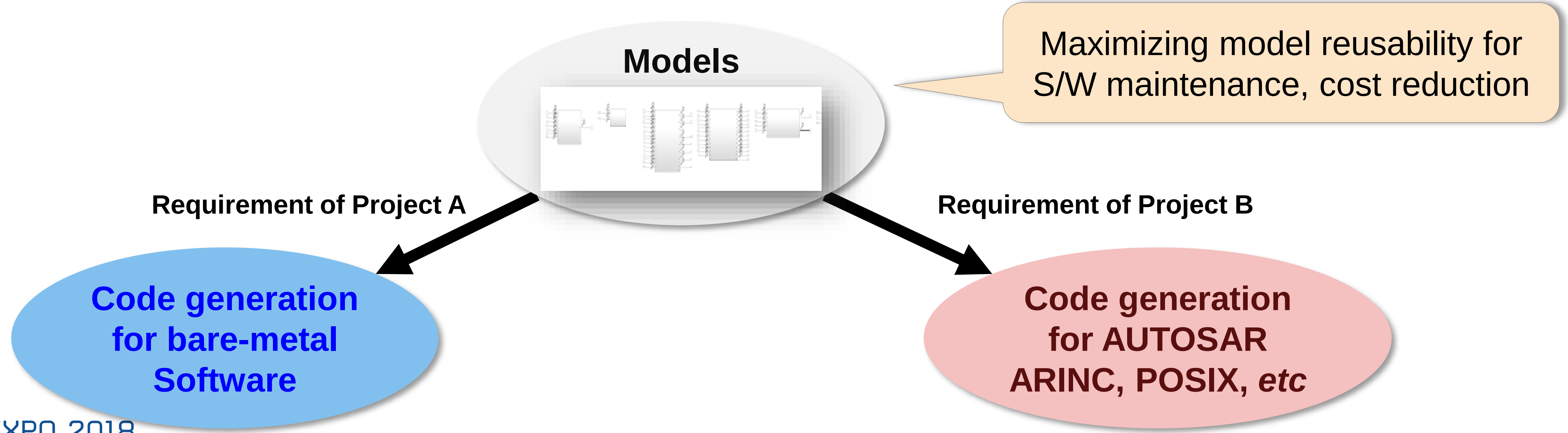


Scheduler | **Memory** | **COM** | **I/O**

Deploy

Issue 1: Mapping Generated Code to Software Frameworks

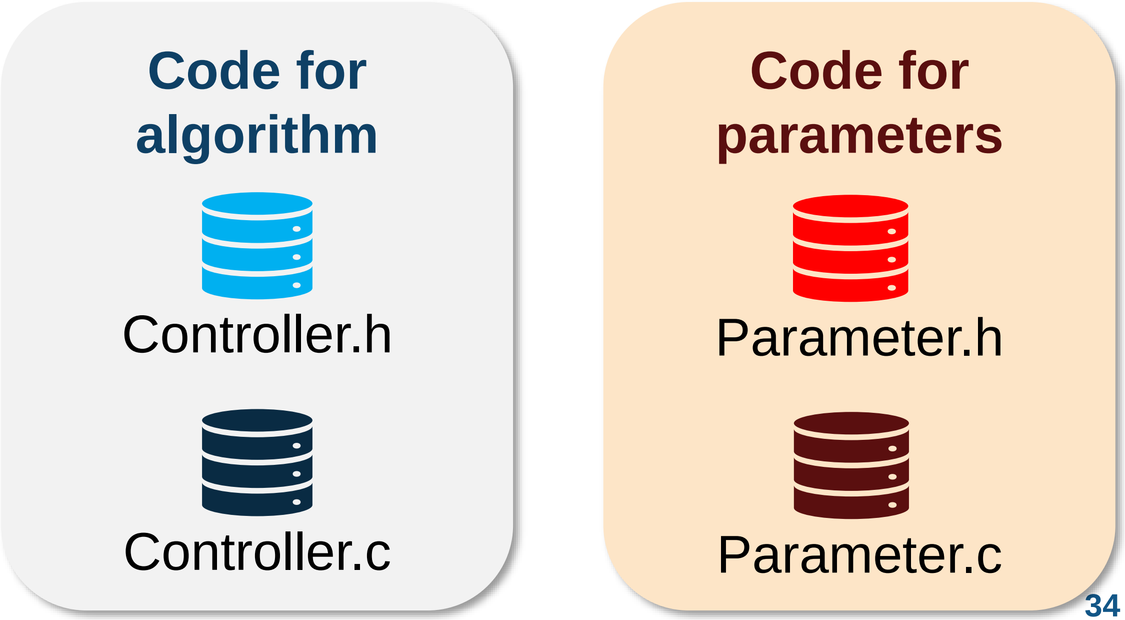
- There are many frameworks (ex. AUTOSAR, ARINC, etc) including bare-metal software
- Solution : Configuration management of code mapping information apart from S/W frameworks
  - Just using code mapping information according to requirements



Issue 2: Code Packaging for Efficient Code Management

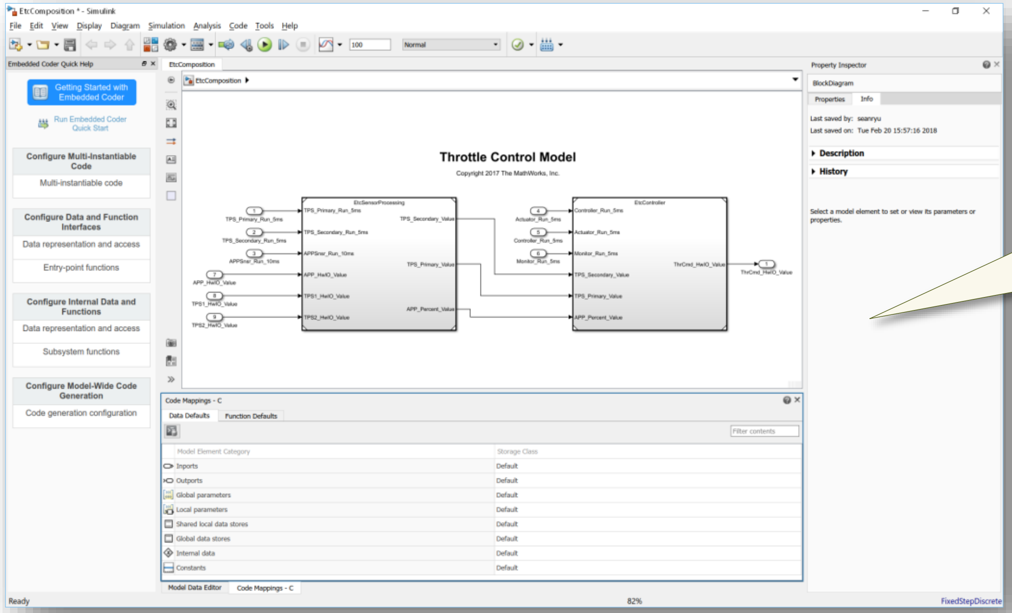
- There are needs to manage efficient tuning parameters in large-scaled S/W
  - To change only tuning parameters according to requirements
  - Efficient code maintenance

- Solution
  - Configuring storage class for code generation
  - Easy customization using GUI



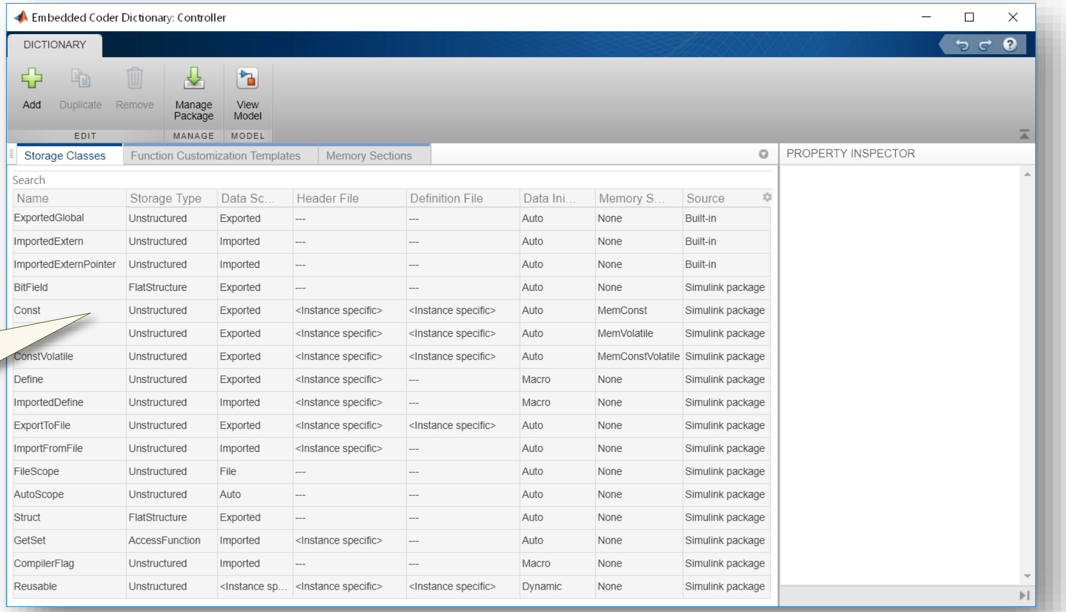
Code Perspective & Embedded Coder Dictionary

- Effective code generation customization as to SW frameworks



Code Perspective

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

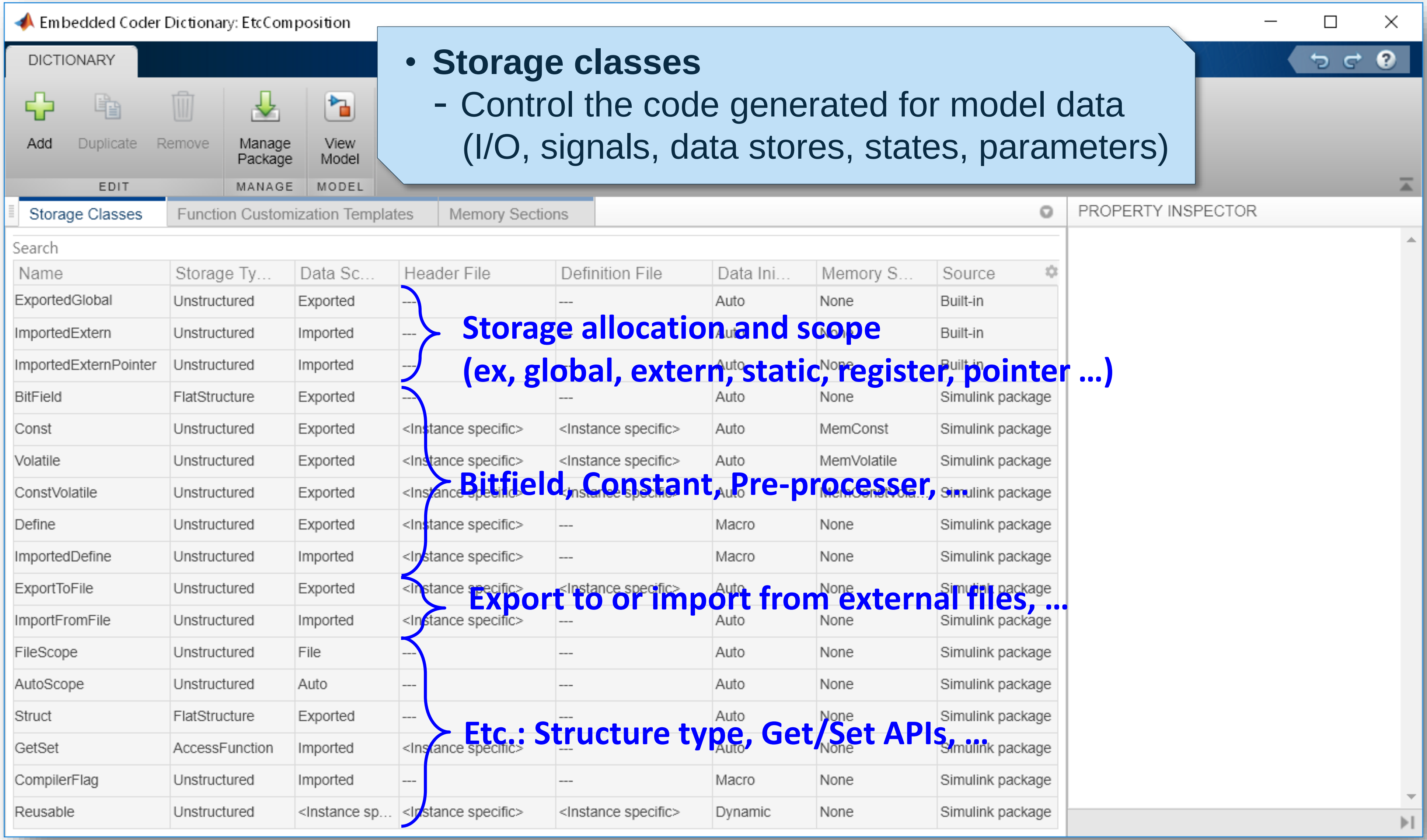


Embedded Coder Dictionary

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

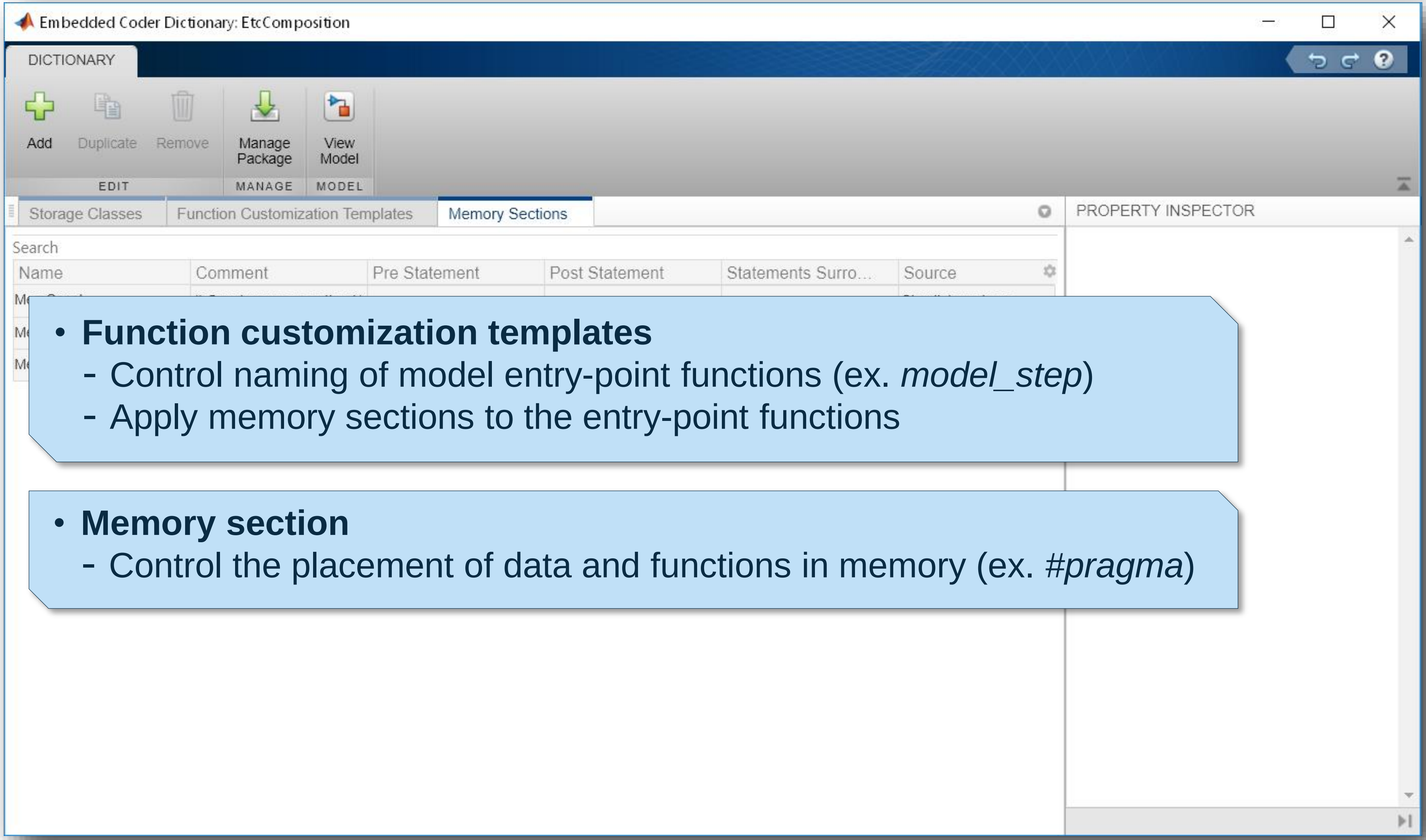
Embedded Coder Dictionary

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



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	MemConstVola…	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

Embedded Coder Dictionary



- **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*)

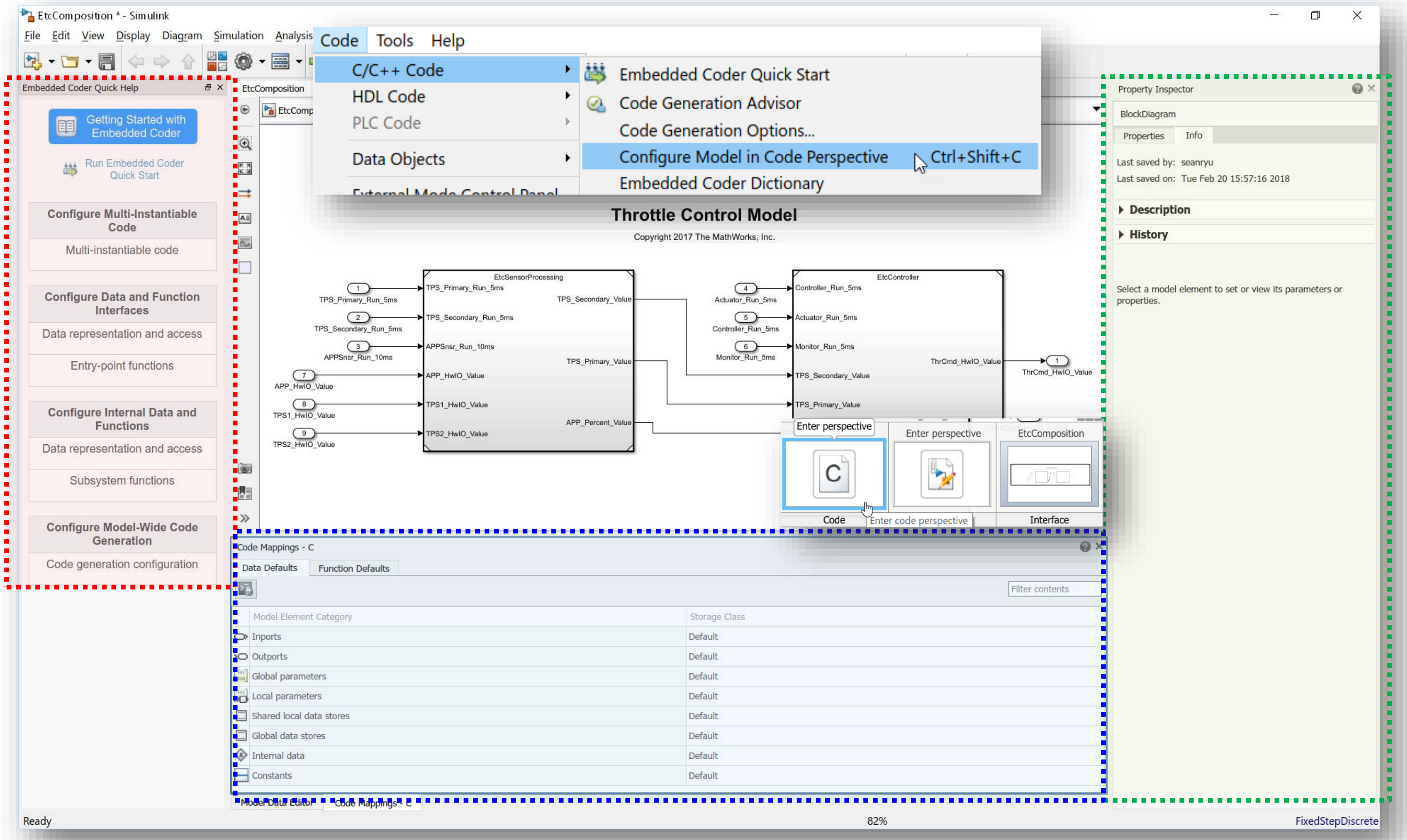
Code Perspective

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

2) Property Inspector
- Configure model properties

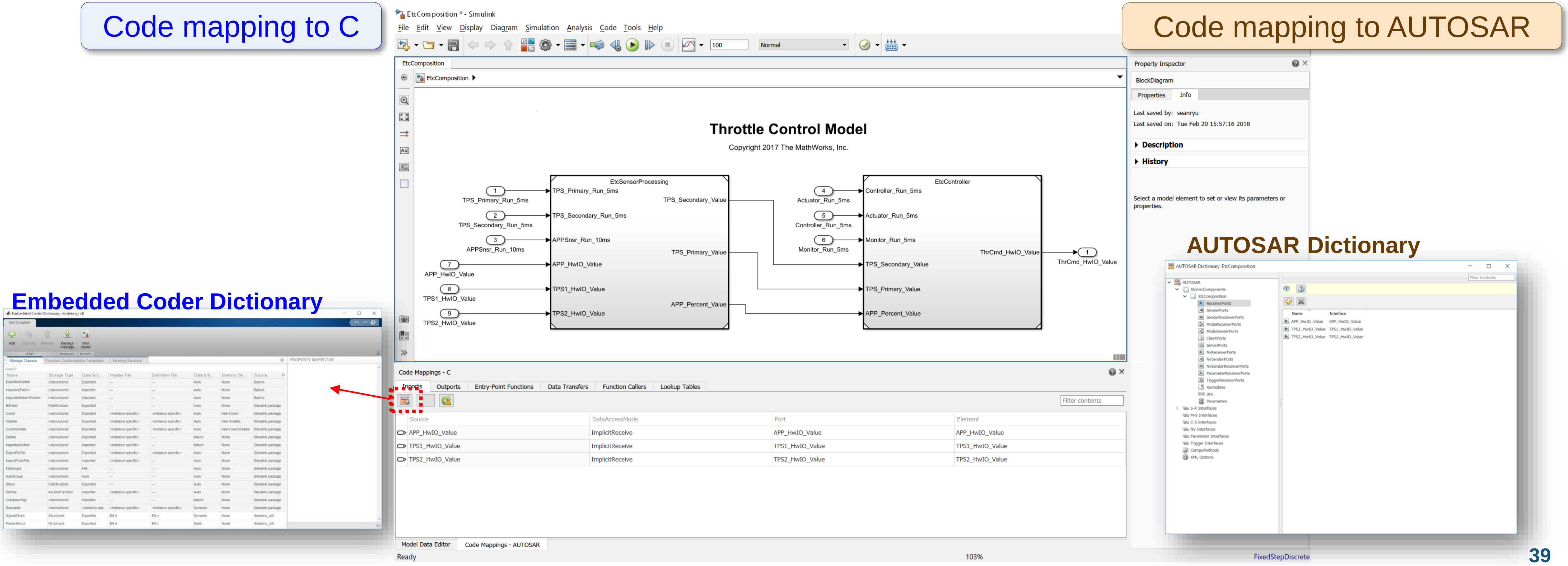
3-1) Model Data Editor
- Inspect and edit data items
- Configure storage class of each blocks or signals

3-2) <u>Code Mapping Editor</u>
- Configuring model data elements and entry-point functions for code generation comprehensively



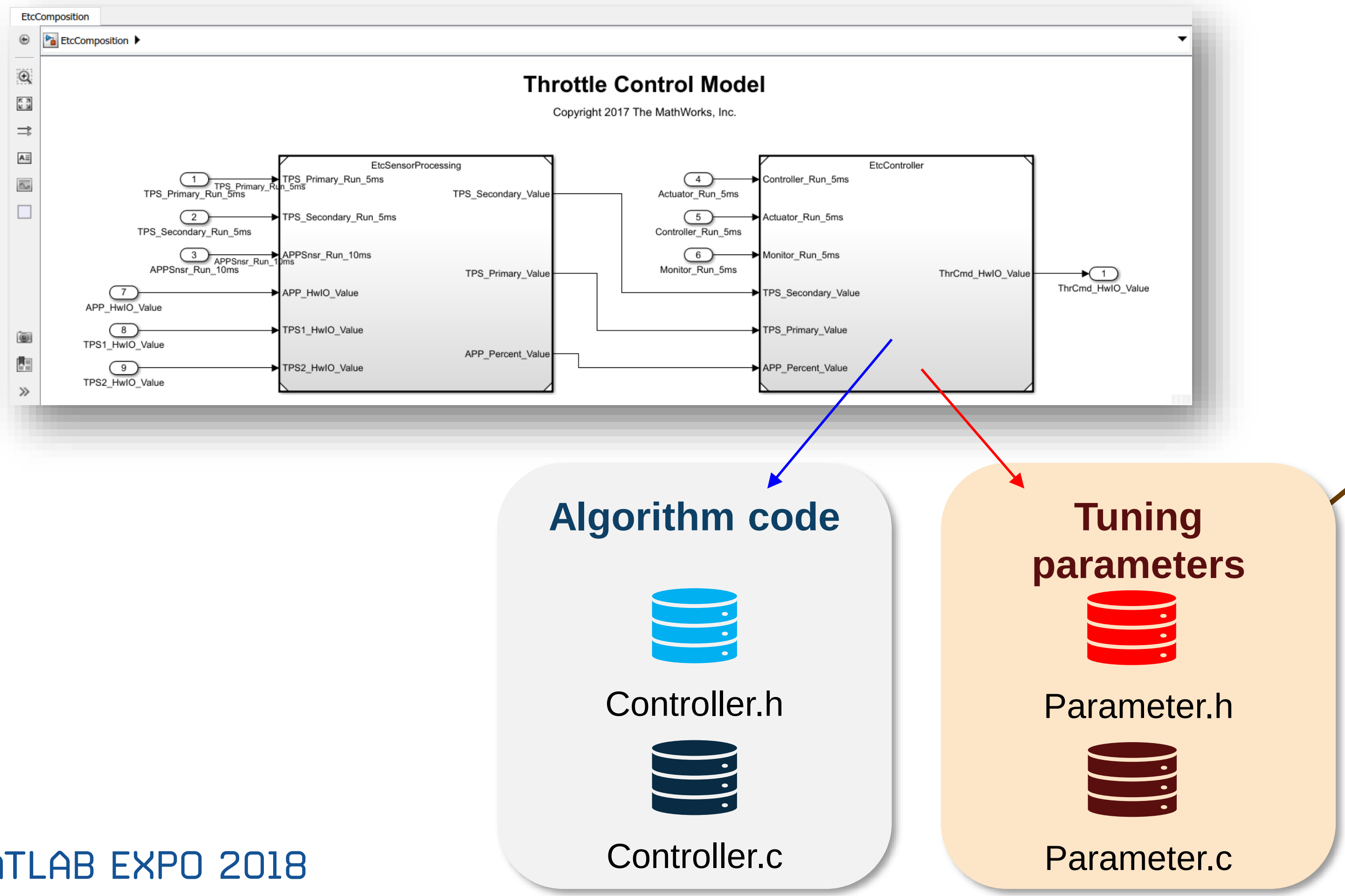
Example on Issue 1: Code mapping implementation

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



Example on Issue 2: Partition and Modularize Generated Code

- Tuning parameter modularization example with customizing storage class

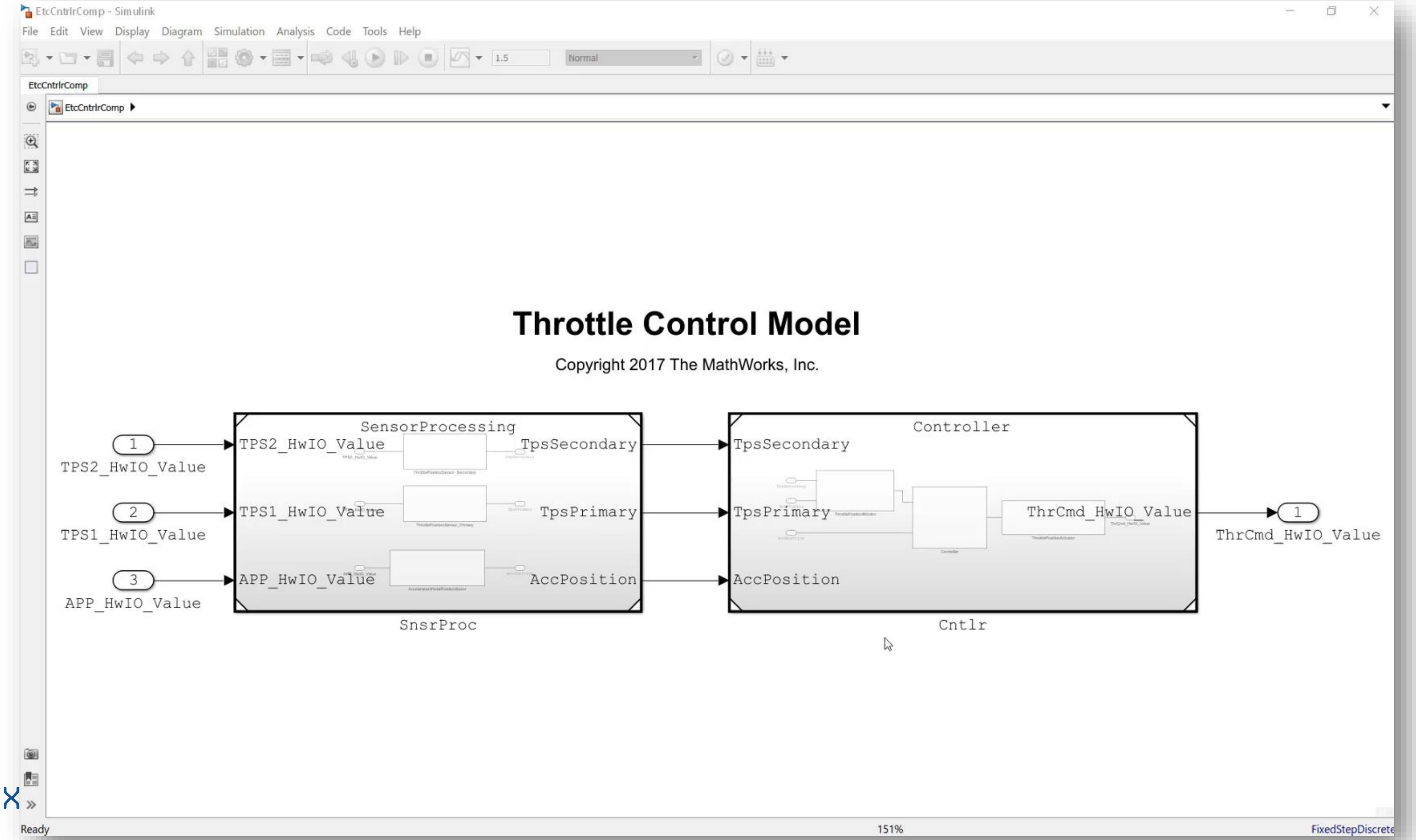


customization:
Struct + ExportToFile

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

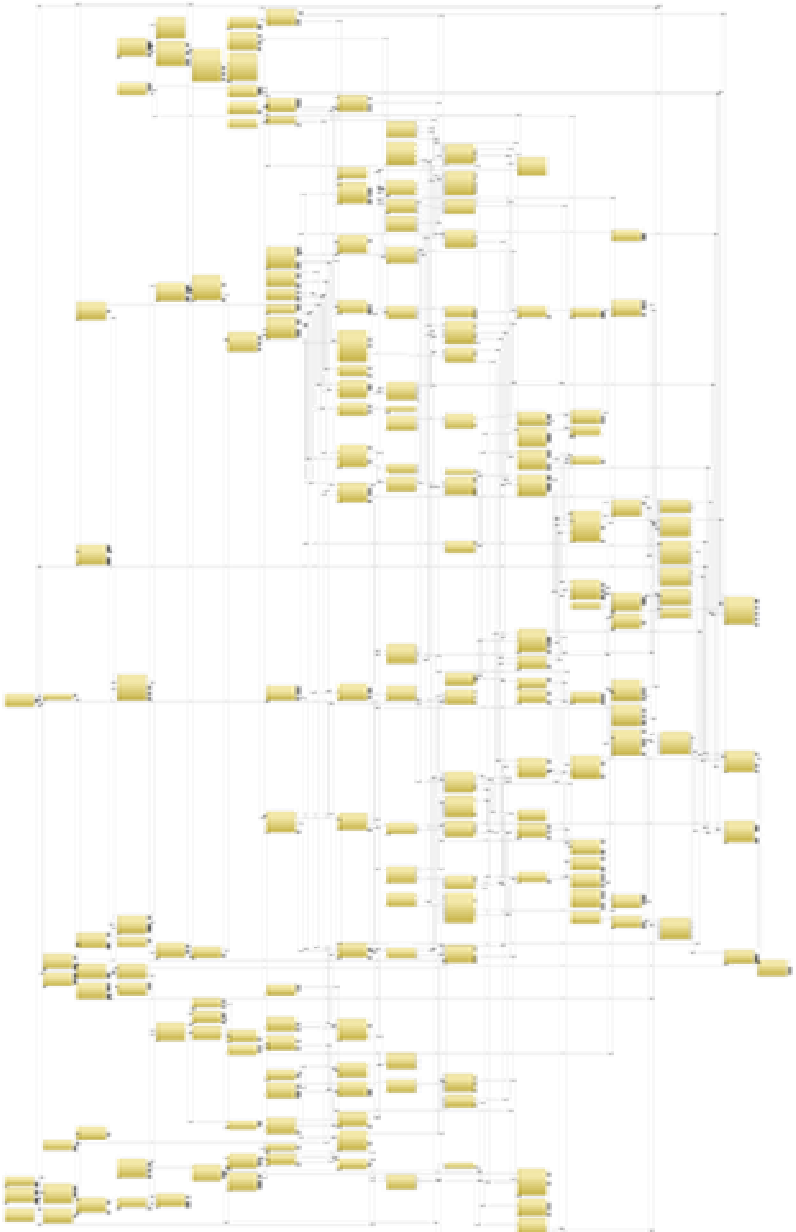
Partition and Modularize Generated Code

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