The MathWorks™

MATLAB® & SIMULINK®

Latest Features in Real-Time Workshop Embedded Coder

October 2008

R2008b

© 2008 The MathWorks, Inc.

Encapsulated C++ Class Generation

Challenge

- C++ classes are needed in generated code for integrating with existing code or complying with company standards.

Solution

- Provide option to generate C++ classes for model-level interfaces, encapsulating data and methods
- Offer flexible C++ class configuration options

Benefit

- Facilitates code integration with object-oriented frameworks
- Enforces software modularity through C++ encapsulation

Target selection
System target file: ert.tlc
Language: C++ (Encapsulated)
Description: C / C++ / C++ (Encapsulated)
Build process

Code interface
Parameters and states members private | Parameters and states access methods
I/O access methods | Inline access methods
Configure C++ Encapsulation Interface
Verification
Configure C++ Encapsulation Interface for the model.

Configure C++ encapsulation interface: rtwdemo_counter

```
62
63   /* Class declaration for model rtw
64   class rtwdemo_cppModelClass {
65     /* public data and function memb
66    public:
67     ExternalOutputs_rtwdemo_cpp rtwd
68
69     /* Model entry point functions *
70
       /* model initialize function */
       void initialize(void);

       /* model step function */
       void step(void);

       /* model terminate function */
       void terminate(void);

       /* Constructor */
       rtwdemo_cppModelClass(void);

       /* Destructor */
       ~rtwdemo_cppModelClass();
       RT_MODEL_rtwdemo_cpp * getRTM(vo

       /* private data and function mem
      private:
       BlockIO_rtwdemo_cpp rtwdemo_cpp_
90     D_Work_rtwdemo_cpp rtwdemo_cpp_D
91     RT_MODEL_rtwdemo_cpp rtwdemo_cpp
92     RT_MODEL_rtwdemo_cpp *rtwdemo_cp
93   };
```

C++ Configuration Options

Demo: `>> rtwdemo_cppencap`

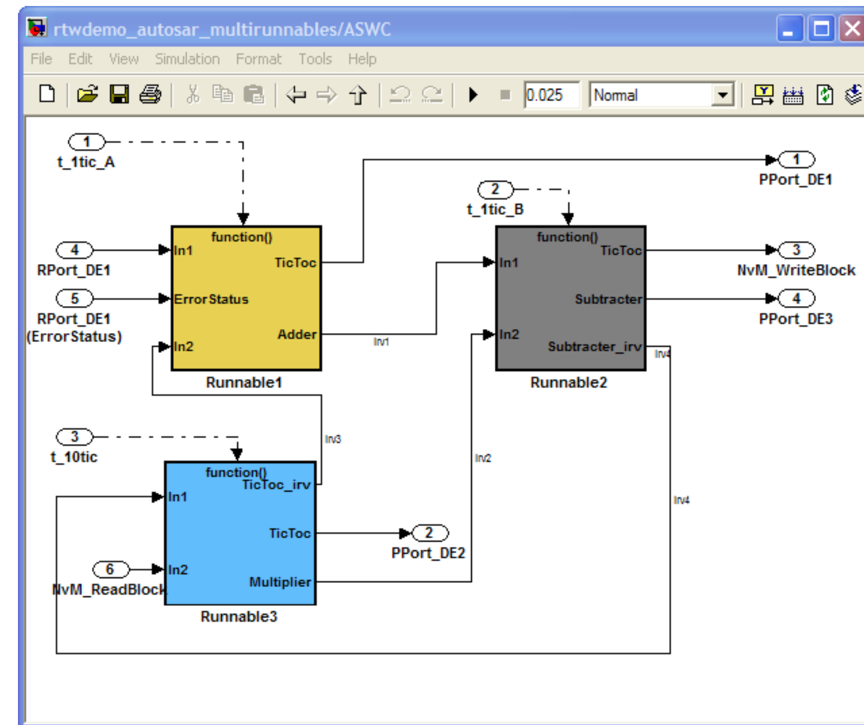
AUTOSAR Target Multiple Runnables

Challenge

- Related functionality with multiple rates could not be combined as runnables in a single software component.

Solution

- Enhance existing AUTOSAR target to:
  - Map virtual subsystem to AUTOSAR Atomic Software Component
  - Model AUTOSAR runnables as function-call subsystems
  - Create `InterRunnableVariables` to communicate data between runnables and ensure data integrity



Benefit

- Cleaner mapping of AUTOSAR runnable concept to Simulink
- Improved automotive production code generation with AUTOSAR

Demo: `>> rtwdemo_autosar_multirunnables`

R2008b

AUTOSAR Target Calibration Parameters

Challenge

- Direct access to calibration parameters was difficult using Simulink with AUTOSAR.

Solution

- Import AUTOSAR calibration parameter objects into base workspace and assign to Simulink block parameters
- Convert existing parameters easily into AUTOSAR

Benefit

- Cleaner mapping of AUTOSAR calibration concept to Simulink
- Improved automotive production code generation for AUTOSAR

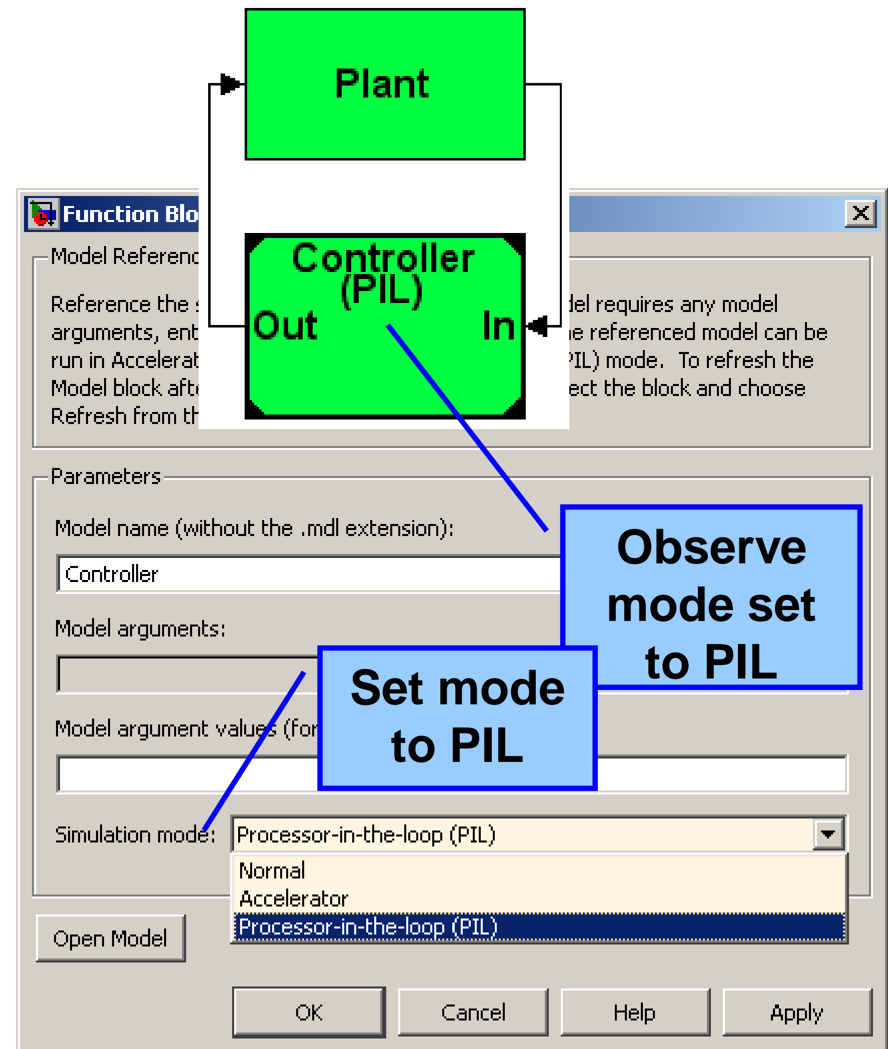
Demo: `>> rtwdemo_autosar_multirunnables`

PIL Simulation Mode for Model Blocks

Challenge
- Working outside Simulink for testing can reduce productivity.
- Adding a new block for testing is cumbersome.

Solution
- Provide processor-in-the-loop (PIL) testing using a PIL simulation mode intrinsic to the model block
- Allow easy switching between Normal, Accelerator, and PIL simulation modes
- Use PIL API (next slide)

Benefit
- Automated and reusable testing
- Seamless integration in Simulink



Demo: `>> rtwdemo_pil`

R2008b

PIL API

Challenge

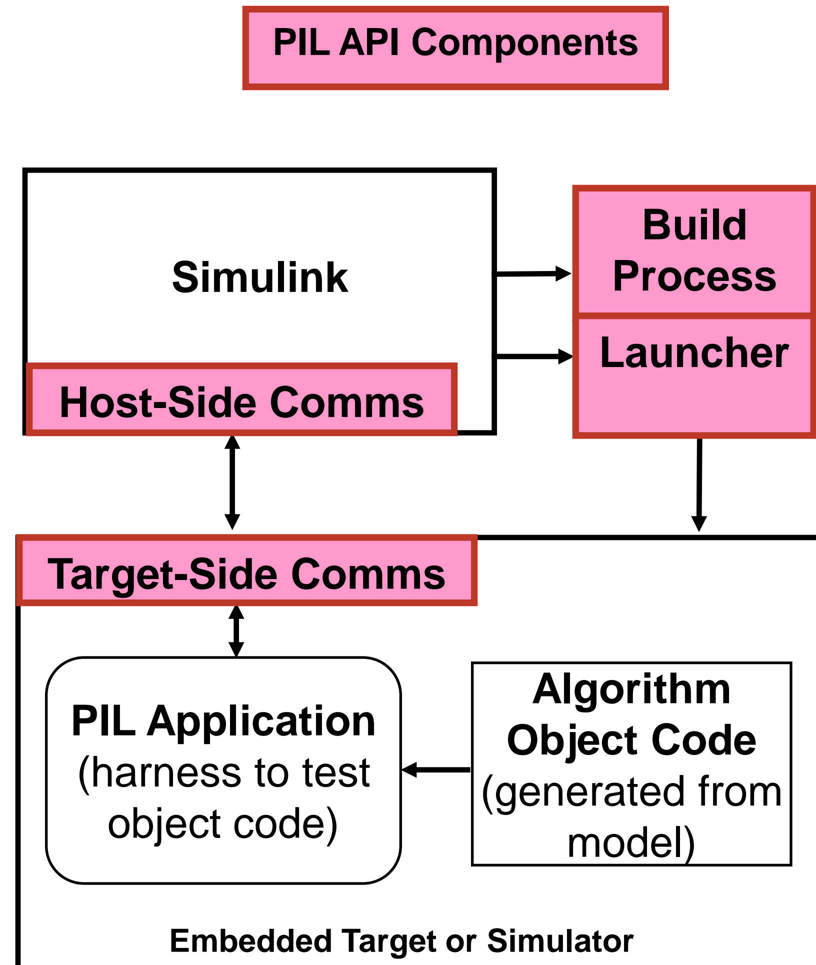
- Existing solutions do not support PIL for an arbitrary combination of:
  - Processor
  - Compiler
  - Debugger or download utility
  - Communications channel

Solution

- Provide API that enables integration of third-party or additional tools for:
  - Building a PIL application
  - Downloading and running the application
  - Communicating with the application

Benefit

- Easily adaptable PIL verification for any environment
- Fully documented and stable API across MathWorks product releases



Demo: `>> rtwdemo_custom_pil`

Optimized Inlining of Embedded MATLAB Functions

Challenge

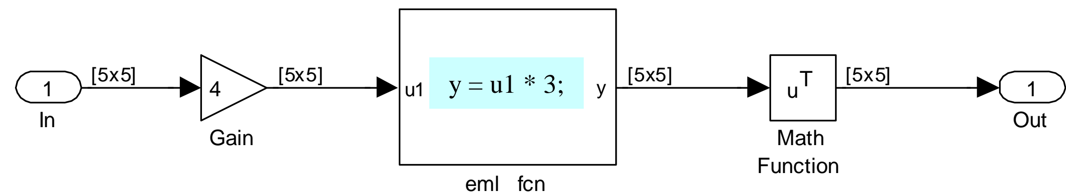
- Limited code optimizations were supported across Simulink and Embedded MATLAB functions.

Benefit

- Improved readability and efficiency

Solution

- Enable optimizations across multiple domains



R2008a

```
real32_T rtb_Gain[25];
real32_T rtb_y[25];
for (i = 0; i < 25; i++) {
  rtb_Gain[i] = 4.0F * m_U.In[i];
}
for (eml_i0 = 0; eml_i0 < 5; eml_i0++) {
  for (eml_i1 = 0; eml_i1 < 5; eml_i1++) {
    rtb_y[eml_i1 + 5 * eml_i0] = rtb_Gain[eml_i1 + 5 * eml_i0] * 3.0F;
  }
}
for (i = 0; i < 5; i++) {
  for (tmp = 0; tmp < 5; tmp++) {
    m_Y.Out[tmp + 5 * i] = rtb_y[5 * tmp + i];
  }
}
```

R2008b

```
for (tmp_0 = 0; tmp_0 < 5; tmp_0++) {
  for (tmp = 0; tmp < 5; tmp++) {
    m_Y.Out[tmp + 5 * tmp_0] = m_U.In[5 * tmp + tmp_0] * 4.0F * 3.0F;
  }
}
```

Demo: `>> rtwdemo_fixpt1`

Expanded Target Function Library Support

Create and select custom TFL

Challenge

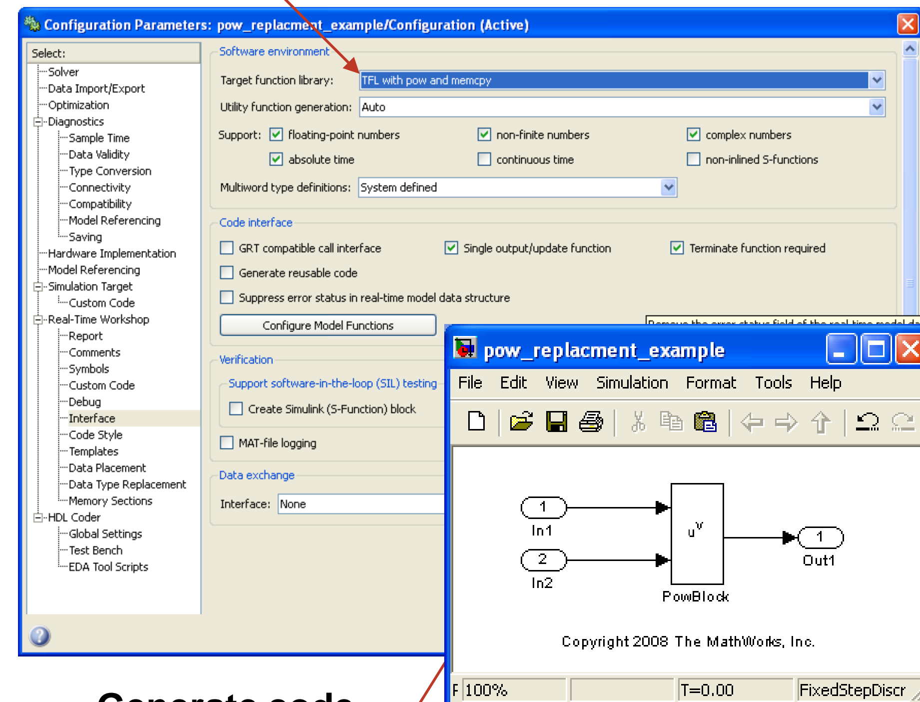
- Default implementations for `pow`, `power`, and `memcpy` are insufficient for some applications.

Solution

- Provide Target Function Library replacements for those functions

Benefit

- Allows use of custom functions for efficiency or company standards



Generate code

```
rtY.Out1 = my_pow_double(rtU.In1, rtU.In2);
```

Demo: `>> rtwdemo_tflmath`

Target Function Library Support for Embedded MATLAB Command (`emlc`)

Challenge

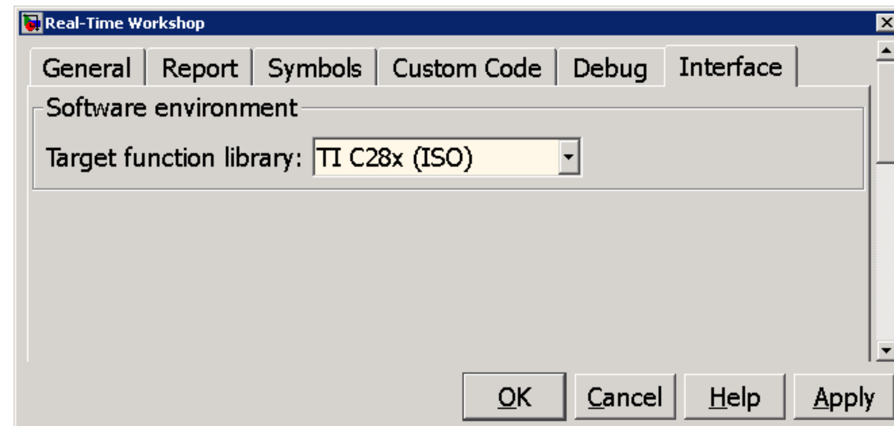
- Target Function Libraries were not supported for Embedded MATLAB when generating code directly from MATLAB using `emlc`.

Solution

- `emlc` now supports Target Function Libraries by using Real-Time Workshop Embedded Coder.
- Target Function Libraries can be shared across MATLAB (`emlc`) and Simulink (`rtwbuild`) generated code.

Benefit

- Allows use of custom functions for efficiency or company standards



Demo: `>> rtwdemo_tflmath`

Code Interface Report

Challenge

- Reviewers and integrators of generated code need to know the code interfaces.

Solution

- Automatically generate report for code interfaces including:
  - Entry point functions
  - Input/output data

Benefit

- Facilitates code integration and code review

ecdemo

File Edit View Simulation Format Tools Help

INC uint8 sum_out LIMIT

Real-Time Workshop Report

Back Forward

Contents

Summary
Subsystem Report
Code Interface Report
Traceability Report

Generated Source Files

ecdemo.c
ert_main.c
ecdemo.h
ecdemo_private.h
ecdemo_types.h
rtwtypes.h

Entry Point Functions

Function: ecdemo_initialize

Prototype	void ecdemo_initialize(boolean_T firstTime)
Description	Initialization entry point of generated code
Timing	Called once
Arguments	# Name Data Type Description 1 firstTime boolean_T Internal data
Return value	None
Header file	ecdemo.h

Step function

Function: ecdemo_step

Prototype	void ecdemo_step(void)
Description	Output entry point of generated code
Timing	Called periodically: Every 1 second
Arguments	None
Return value	None
Header file	ecdemo.h

Inports

Input data

Block Name	Code Identifier	Data Type	Dimension
<Root>/In	INPUT	int32_T	1

Outports

Block Name	Code Identifier	Data Type	Dimension

OK Cancel Help Apply

Demo: `>> rtwdemo_hyperlink`

Scheduling Code Separate from Algorithm

Challenge

- Multiple rate models interleaved scheduling and algorithm code, increasing software integration effort.

Solution

- Separate scheduler from algorithm code
- Move scheduler code from `modelname.c` to `ert_main.c`

Benefit

- Simplifies code integration
- Improves code efficiency because there is less code at base rate
- Improves memory usage because there is smaller or no `rtModel` data structure

R2008a

```
void m3ts_step0(void) {      /* Sample time: [0.001s, 0.0s] */
  rate_monotonic_scheduler();
  m3ts_Y.Out2 = m3ts_P.Gain_ts4_Gain * m3ts_U.In2;
}

void m3ts_step1(void)  {      /* Sample time: [2.0s, 0.0s] */
…..
}
void m3ts_step2(void)  {      /* Sample time: [4.0s, 0.0s] */
…..
}
```

R2008b

```
void m3ts_step0(void) {      /* Sample time: [0.001s, 0.0s] */
   m3ts_Y.Out2 = m3ts_P.Gain_ts4_Gain * m3ts_U.In2;
}

void m3ts_step1(void)  {      /* Sample time: [2.0s, 0.0s] */
…..
}
void m3ts_step2(void)  {      /* Sample time: [4.0s, 0.0s] */
…..
}
```

Demo: `>> rtwdemo_mrmtbb`