Simulink를 이용한 효율적인 레거시 코드 검증 방안

류성연
Agenda

▪ Overview to V&V in Model-Based Design
▪ Legacy code integration using Simulink
▪ Workflow for legacy code verification
Model-Based Design With Legacy C/C++ Code?

Hand Coding

Full MBD

Legacy code verification using Simulink?

MD with C/C++ code?

Model-Based Design
Why Using Simulink for Legacy Code Testing?

- Input
- Display
- Algorithm

Simulink, Stateflow, MATLAB, C/C++, Vehicle Dynamics Blockset (R2018a)
ISO26262 “Road Vehicles - Functional Safety”

- **Functional safety standard** for passenger cars
  - Concerned with avoidance of unreasonable risks due to hazards caused by E/E systems
  - Recommends tool certification, but offers little guidance

- Serves as an umbrella standard for industry specific adaptions including:
  - ISO 26262 - Automotive
  - EN 50128 - Rail
  - IEC 62304 - Medical
  - IEC 61511 - Process Control
Software Development Workflow for Embedded Applications

Textual Requirements → Executable Specification → Model used for production code generation → Generated C/C++ code → Object code

Requirements Authoring → Modelling → Software architecture and design

Code Generation → Compilation and Linking → Handwritten C/C++ code

Requirements Trace Documentation
Version Control
Tool Qualification
Legacy Code Verification Overview

**Textual Requirements**

**Executable Specification**

**Model used for production code generation**

**Generated C/C++ code**

**Object code**

**Requirements Authoring**

**Modelling**

**Code Generation**

**Compilation and Linking**

**Model used for production code generation**

**Generated C/C++ code**

**Object code**

**Software architecture and unit design**

**S-functions from Handed C/C++ code**

**Coverage analysis / Test case generation**

**Unit(Integration) testing**

**Code review and Static analysis**

For Model-Based Design

For legacy code development
Agenda

- Overview to V&V in Model-Based Design
- Legacy code integration using Simulink
- Workflow for legacy code verification
How to Import Legacy Code

- Legacy Code Tool
- C Caller Block
- Legacy code integration in Stateflow
What legacy C code integration in Simulink means?

- Legacy Code Tool enables existing C code to be used in Simulink models
How to use Legacy Code Tool?

- General procedure for using Legacy Code Tool

![Diagram showing the process of using Legacy Code Tool](image)
Prerequisite to use Legacy Code Tool

- What is wrapper code?
  - Root-level C function having in/output variables for S-Function block’s in/out ports

```c
double doubleit(double u1)
{
    MainInp = u1;
    double_main();
    y1 = MainOutp;
    return y1;
}
```

or

```c
void doubleit(double u1, double *y1)
{
    MainInp = u1;
    double_main();
    *y1 = MainOutp;
}
```
MATLAB Script to Build and Generate S-Function Block

- **m-script file**: compiling C files and generate a S-Function Block

```matlab
Simulink.importExternalCTypes('ex_myTypes_LCT.h');

def = legacy_code('initialize');

def.SFunctionName = 'sfun_ex_mySrc_LCT';
def.SourceFiles = {'ex_mySrc_LCT.c'};
def.HeaderFiles = {'ex_myTypes_LCT.h'};
def.OutputFcnSpec = 'void myFcn(sigStructType u1[1], paramStructType p1[1], sigStructType y1[1])';
def.IncPaths = {'rtwdemo_lct_src'};
def.SrcPaths = {'rtwdemo_lct_src'};

legacy_code('sfcn_cmex_generate', def);

legacy_code('compile', def);

legacy_code('slblock_generate', def);
```

① **C files to integrate in Simulink**

② **S-Function block specification**

③ **Include folders**

④ **Compile and s-function generation**
Generate Simulink Representations from C or C++ Code

- Import external C header file and generate available Simulink data types

```matlab
Simulink.importExternalCTypes('ex_myTypes_LCT.h');
```

Automatically generating to Simulink Bus

Selecting generated Simulink Bus
Issues for Legacy Code Tool

- There are still technical challenges to make S-Function Block

  - Difficult to build test cases
  - Limited input variation
  - Long lead time from development to test
  - Hard work to improve test results

Conventional C code verification

Legacy Code Tool

C code verification with Simulink

- No unified interfaces to interact with legacy code
- Hard to build S-Function Block
- No auto sync with custom C code change
- Still maintenance problem
Example Issue: Too Many Function Arguments

Legacy code

```
#include "DataTypes.h"
#include "CruiseControllerTypes.h"

/* Cruise controller input */
extern int16_t s16BrakeP;
extern boolean_t u3CncSlw;
extern boolean_t u1Decslw;
extern boolean_t u1Lbnb3sw;
extern boolean_t u1ChecSw;
extern boolean_t u1LbrmSw;
extern boolean_t u1SameSw;
extern boolean_t u1LResSw;
extern boolean_t u1IncSw;
extern boolean_t u1DecSw;
extern boolean_t u1BrakeP;
extern boolean_t u1xKey;
extern variable_t u32ThrotDrv;
extern variable_t u32VehSpd;

/* Cruise controller output */
extern enumReqDrvOut;  
extern boolean_t u1StatusOut;
extern int32_t s32TargetSpOut;
extern int32_t s32ThrotCcoOut;

#define KeyOn 2
#define ShiftDrive 2
#define BrakeEmcThrsP 5

#endif /* CRUISECNTLRLR_H */
```

Wrapper code

```
#include "CrsCntrl_Wrapper.h"

void CrsCntrl(booleans_t u1, booleans_t u2, booleans_t u3, booleans_t u4, booleans_t u5, booleans_t u6, 
            int16_t u7, uint8_t u8, uint8_t u9, int32_t u10, int32_t u11, 
            uint8_t *y1, booleans_t *y2, uint8_t *y3, int32_t *y4, int32_t *y5)
{
    u1Enbl3sw = u1;
    u1Cnc3l3w = u2;
    u1Set3sw = u3;
    u1Res3w = u4;
    u1Inc3w = u5;
    u1Dec3w = u6;
    u1BrakeP = u7;
    u1xKey = u8;
    s321 = u9;
    s322 = u10;
    s323 = u11;
    def.SourceFiles = {'CrsCtrl_Wrapper.c', 'CruiseCtrl.h'};
    def.HeaderFiles = {'CrsCtrl_Wrapper.h', 'CruiseCtrl.h'};
    def.IncPaths = {'defaultDir', 'files/legacycode'};
    CrsCtrl.SrcPaths = {'defaultDir', 'files/legacycode'};

    /* y1 */
    /* y2 */
    /* y3 */
    /* y4 */
    /* y5 */
```

Script file

```
def.SrcPaths = {'defaultDir', 'files/legacycode'};

#define StartFcnSpec - 'void sbr_initialize(void)';
#define OutputFcnSpec = 'void CrsCntrl(boolean_t u1, boolean_t u2, boolean_t u3, boolean_t u4, boolean_t u5, boolean_t u6, 
                              int16_t u7, uint8_t u8, uint8_t u9, int32_t u10, int32_t u11, 
                              uint8_t y1[y][1], boolean_t y2[y], uint8_t y3[y], int32_t y4[y], int32_t y5[y]);'
def.Options.supportsCoverageAnalysis = true;
def.Options.optimize = true;
def.Options.isMacro = true;
Generate the C-MEX S-function

legacy_code('sfcn_cmxex_generate', def);
legacy_code('rtwmakecfa_generate', def);
```
Maintenance Problem...

Legacy code

```c
void CtrlMainMain(int argc, char **argv)
{
    uLong uNum; 
    uNum = 0;
    if (uNum < 10)
    {
        if (uNum % 2 == 0)
            uNum = uNum + 1;
        else
            uNum = uNum + 2;
    }
    else
    {
        if (uNum % 2 == 0)
            uNum = uNum + 1;
        else
            uNum = uNum + 2;
    }
    if (uNum == 0)
        uNum = 0;
    else
        uNum = 0;
}
```

Wrapper code

```c
#include "CtrlMainWrapper.h"

void CtrlMainMain(void)
{
    uLong uNum = 0;
    uNum = uNum + 1;
    if (uNum % 2 == 0)
    {
        uNum = uNum + 1;
    }
    else
    {
        uNum = uNum + 2;
    }
    if (uNum == 0)
        uNum = 0;
    else
        uNum = 0;
}
```

Modeling

Script file
Introducing C Caller Block

C Caller Block makes it easier to call C Functions in Simulink → It works for simulation and Code Generation
Key Features

- Automate the process

  - Define Block Interface
  - Build Simulation MEX
  - Write Codegen TLC

  Automate

  - Tedious
  - Error prone
  - Hard to maintain

- Synchronize with custom code changes

  C/C++ Code
1. Specify Custom Code in the Configuration Parameters

- Custom code is specified on the Configuration Parameters.
  - **The Header file section**: Any code that needs to be inserted into the header file
  - **The Source files section**: List of source files that needs to be compiled
Using C Caller Block

2. Select the function that you want to call
Using C Caller Block

3. Customize the function that you want to call

- Mapping inputs, outputs or parameters to C Caller Block

1) Change argument scope to “Output”
2) (Optional) Override with a better output name
3) Complete the test model with connecting signal ports
Demo: Simple C Caller
Library Workflow

- C Caller block can be configured as a library model
  - Custom Code Settings can be accessed from View Menu → Library Custom Code Settings

- Include custom header file
- Add custom source file
Demo: Reusable Library Workflow with OpenCV
Legacy Code Evaluation in Stateflow

- Using legacy code in Stateflow chart

```c
#include "custom_code.h"

double c_multiple = 0.8;

double c_fcn(double in1)
{
    return in1 * c_multiple;
}

void set_c_multiple(double in)
{
    c_multiple = in;
}
```

Step 1: Have C code

Step 2: Put on Config. Set

Step 3: Use in Stateflow
Agenda

▪ Overview to V&amp;V in Model-Based Design

▪ Legacy code integration using Simulink

▪ Workflow for legacy code verification
Legacy Code Verification using Simulink V&V

- Test harness model
- S-Function or C Caller
- Test case generation
- Code coverage analysis

Test Cases
Demo: Legacy C Code Verification
Needs for Test Automation

- Test automation/management
- Code coverage analysis
- Function/Function call coverage
- Report generation
Test Automation with Test Manager

- Test automation
- Test case creation from template
- Customization
- View, share, report results
Static Code Analysis

- Run time error / MISRA rule check
- Polyspace report from Simulink
- Reducing Polyspace set-up efforts
Key Takeaways

Textual Requirements → Executable Specification → Model used for production code generation → Generated C/C++ code → Object code

Requirements Authoring

Modelling

Software architecture and unit design

Code Generation

Compilation and Linking

Polyspace

Review and Static analysis

Coverage analysis / Test case generation

Simulink Requirement

Simulink Test

Simulink Coverage

Simulink Design Verifier

Simulink Coder

Embedded Coder

SL Requirement
Thank You!