Master Class Production Code Generation (PCG)

By Simon Eriksson
Case Study Introduction

- We are design or software engineers who inherited a Simulink model.
- Our Tasks:
  - Implement the algorithm in C code.
  - Verify that the generated code produces the same output as the original model.
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Demonstration

I will…

1. Introduce the Simulink model
2. Introduce the test harness model
3. Setup the model for code generation
4. Generate code from the model
### What Files are Generated?

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>model.c</code></td>
<td>Model algorithm code</td>
</tr>
<tr>
<td><code>model_data.c</code></td>
<td>Parameter definitions</td>
</tr>
<tr>
<td><code>ert_main.c</code></td>
<td>Example main function (optional)</td>
</tr>
<tr>
<td><code>model.h</code></td>
<td>Public interface to model code</td>
</tr>
<tr>
<td><code>model_private.h</code></td>
<td>Private macros, private data, etc.</td>
</tr>
<tr>
<td><code>model_types.h</code></td>
<td>Data types, structures and macros</td>
</tr>
</tbody>
</table>
What Functions are Generated?

**model_step**: Contains your algorithm code

```c
void model_step(void)
```

**model_initialize**: Initializes the model variables and states

```c
void model_initialize(void)
```

**model_terminate (optional)**: Hardware shutdown

```c
void model_terminate(void)
```
Optimizing Generated Code based on Specified Objectives
Questions to be Answered Today

How do I...

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. **Partition the code into sub-functions?**
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Subsystems

- A subsystem is a block containing other groups of blocks
- A subsystem can be either **virtual** or **atomic**
- Virtual or atomic can influence the execution sequence of the blocks within it
Making a virtual subsystem atomic
Influence on the Structure of the Generated Code

- *Virtual* subsystems do not influence the structure of the code of the blocks.

- The code for blocks in an *atomic* subsystem will be contiguous in the generated code.

- *Atomic* subsystems can be used in three ways:
  - Inline
  - Nonreusable function
  - Reusable function
Atomic Subsystems

- **Inline**: Inlined contiguous code

/* Start of Subsystem1 */

code line 1

code line 2

...  

/* End of Subsystem1 */
Atomic Subsystems

- **Nonreusable function**: Contiguous code inside a void-void function or a void function with arguments

```c
/* Start of Subsystem1 */

void Subsystem1(void)
{
    code line 1
    code line 2
    ...
}

/* End of Subsystem1 */
```
Atomic Subsystems

- **Nonreusable function**: Contiguous code inside a void-void function or a void function with arguments

```c
/* Start of Subsystem1 */

void Subsystem1(arg1, arg2, ...)
{
    code line 1
    code line 2
    ...
}

/* End of Subsystem1 */
```
Atomic Subsystems

- **Reusable function**: Contiguous code inside a void function with arguments (reused across model)

```c
/* Start of Subsystem1 */

void Subsystem1(arg1, arg2, ...)
{
    code line 1
    code line 2
    ...
}

/* End of Subsystem1 */
```
Demonstration

I will…

1. Make PI_ctrl a reusable function
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. **Partition the code into sub-functions?**
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. **Change the code interface?**
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Default Function Interface

- Algorithm code called at base sample rate
- Initializes variables ones
- Optional (hardware shut down)

```c
#include "my_add.h"
#include "my_add_private.h"

int32_T a;
int32_T b;
int32_T c;

void my_add_step(void)
{
    c = a + b;
}

void my_add_initialize(void)
{
}

void my_add_terminate(void)
{
}
```
Function Prototype Control

- Pass arguments by value or pointer
- Control argument names and order
- Simplifies code integration and testing
- Reduces global RAM usage
Demonstration

I will…

1. Show how to change the code interface
   - Change the function prototype of `model_step`
   - Change the name of `model_initialize`
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. **Change the code interface?**
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Data Characteristics

- What information could be contained in a variable definition and declaration?
  - Name
  - Data type
  - Dimension
  - Initial value
  - Comment (description, units, etc.)
  - Scope (global, local, static, extern)
  - Type (const, pointer)
  - Placement (definition file, header file)
Creating Data Dictionaries using Simulink Data Objects

Excel sheet

Data Object Wizard

MATLAB script

Simulink Data Objects
Simulink Data Objects

Package (Simulink)

Class (Signal)
- Property 1
  - Property value
- Property 2
  - Property value

Class (Parameter)
- Property 1
  - Property value
- Property 2
  - Property value
Control Storage Class of Data using Simulink Data Objects

Standard storage classes:

- Global
- Bit Field
- Volatile or Const Volatile
- Export to File
- Import From File
- Structure
- GetSet
- Simulink Global
- External Global
- Imported External Pointer
- Imported External
- Etc.
Demonstration (1)

I will…

1. Use Data Object Wizard to:
   a. Create parameter objects for all parameters
   b. Create signal objects for selected signals

2. Change code generation properties for:
   a. Signal objects
   b. Parameter objects
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. **Include my own legacy code?**
6. Verify the generated code?
Including Legacy Code

Generated code

```c
void model_step(void)
{
    uint32 u1;
    uint32 y1;
    u1 = DevDrvFcn();
    y1 = UtilFcn(u1);
    ExtSetFcn(y1);
}
```

Device driver function

```c
uint32 DevDrvFcn(void)
{
    uint32 out;
    ...
    return out;
}
```

Utility function

```c
uint32 UtilFcn(uint32 u)
{
    uint32 out;
    ...
    return out;
}
```

External set function

```c
void ExtSetFcn(uint32 y)
{
    ...
}
```
How could I integrate code with Simulink?

- S-Function
  - Legacy Code Tool
  - S-Function Builder
  - Hand-Crafted S-function
- Custom Code
- Stateflow
- Target Function Library
Legacy Code Tool

```matlab
specs = legacy_code('initialize')
```
Demonstration (1)

I will…

1. Present external utility function (look-up table)
   - SimpleTable.c
   - SimpleTable.h

2. Run Legacy Code Tool script that:
   - Creates S-function block (interface to SimpleTable)

3. Replace model look-up tables with S-function blocks
   - Verify that model behavior is correct
   - Generate and inspect code
Data Type Replacement

- Using your own defined data types in the generated code
- Built-in types
- Fixed-point types
Creating Reconfigurable Data Types

- Simulink data type classes:
  - `Simulink.AliasType`
    Supports only built-in types
  - `Simulink.NumericType`
    Expanded support to fixed-point types
Creating Reconfigurable Data Types

- With `Simulink.AliasType` you can:
  - DBL $\iff$ double
  - FLT $\iff$ single
  - INT $\iff$ int32
  - UINT $\iff$ uint32
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. **Include my own legacy code?**
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. **Verify the generated code?**
Why Perform Testing on the Generated Code?

- Often, behaviour in simulation and on-target is identical

- Some small differences may be acceptable
  - Non-IEEE floating-point on the target
  - Implementation of libraries, e.g. for trigonometric functions
  - Differences due to hardware-specific optimised code
Why Perform Testing on the Generated Code?

- Other differences may indicate a problem
  - Unintended side effect of compiler settings or optimizations
  - Unintended side effect of code generator settings and optimizations
  - Defects in the hardware, the compiler, the linker, or the code generator

- The later problems are identified, the more costly they are to correct
Software-in-the-Loop (SIL) Testing

Execution:
- Host/host
- Non-real-time

Compiled C Code (S-Function)
Demonstration

I will…

1. Show how to create a SIL S-function block
2. Use the SIL block to perform SIL testing
Processor-in-the-Loop (PIL) Testing

Execution
- Host/target
- Non-real-time

Exercise the same *object code* that will be used in production software.
Verify behavior and execution time of compiled code with processor-in-the-loop testing

- **Processor-in-the-loop (PIL)** testing generates, cross-compiles, and executes production object code on a target processor or simulator
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?
2. Partition the code into sub-functions?
3. Change the code interface?
4. Affect variable naming and properties?
5. Include my own legacy code?
6. Verify the generated code?
Questions to be Answered Today

How do I…

1. Setup my model for production code generation?

2. Partition the code into sub-functions?

3. Change the code interface?

4. Affect variable naming and properties?

5. Include my own legacy code?

6. Verify the generated code?
Thank you!