MF2004 Lab Exercise 2

Code Generation with Simulink and Arduino

1 Background

In lab 1, you have already tried to tune your controller in Simulink and re-implemented it on the microcontroller. However, you might find at least the following two weaknesses of this development approach.

1. It is difficult to guarantee the consistency between the manual implementation of the PI controller and the MATLAB implementation. Therefore, the re-implemented controller may lead to slightly different control performance.

2. Hand writing code is time consuming and error-prone especially for the implementation of complicated control algorithms.

The idea of model based development (MBD) gets rid of these weaknesses and also introduces a lot of other beneficial features. Simulink is the most commonly used MBD tool for control engineers to design and verify their controller. With the help of Simulink coder, control strategies modeled in Simulink can be directly downloaded to microcontroller by one click. By running the model in Simulink's external mode, developers can also monitor signals and tune control parameters during runtime. If you are interested in these amazing features, now is the time to lift the veil from code generation.

Learning Goals

In this lab exercise, you will be guided to learn

- basic concepts of Simulink code generation;

- how to customize the generated code by inserting user defined functions.

2 Simulink Code Generation

Simulink coder can automate the following process

1. transform Simulink model to corresponding C/C++ code;

2. compile these C/C++ files and link them with other pre-programed and hardware specific files to build an executable;

3. download this executable to the micro controller and run it;

All the steps within this process are configurable, and the configuration method is comprehensively elaborated in MATLAB help. In this lab, we only focus on the C code generation from a single rate model (the model contains only one sample time).

2.1 Generation Mechanism

Please refer to the following hints in MATLAB (2014b) help to figure out (1) how Simulink Coder works to generate real-time C code for single chip microcontroller from single rate Simulink model and (2) how code generation is used in industry.

- Help -> Simulink Coder -> Get Started with Simulink Coder -> Simulink Coder Product Description
- Help -> Simulink Coder -> About Code Generation from Simulink Models -> All

2.2 User-defined Blocks

Simulink Coder is good enough to generate C/C++ code for whatever control strategy you modeled in Simulink, however, it is not intelligent enough to generate driver code for an unknown motor or other devices. To handle this, Simulink provides an approach to insert user written code into the generated code from Simulink model. The inserted code can be allocated at specific positions and interact with other code. Please refer to the following hints in MATLAB help to figure out (1) how to insert user code into the generated code with C S-Function block and (2) how S-Functions work.

- Help -> Simulink -> Block Creation -> Host-Specific Code -> S-Function Basics -> Concepts -> What Is an S-Function?
- Help -> Simulink -> Block Creation -> Host-Specific Code -> S-Function Basics -> Concepts -> How S-Functions Work?
- Help -> Simulink -> Block Creation -> Host-Specific Code -> C/C++ S-Function -> Create C/C++ S-Function -> Concepts -> Simulink Engine Interaction with C S-Functions

2.3 External Mode

In external mode, developers can monitor signals in the Simulink model with Simulink scope and modify parameters during runtime. The word external is used to describe that the simulation is running on target (the microcontroller) not in Simulink on your computer (as it is in normal mode) and the Simulation is running in real-time but not as fast as possible (as it is in normal mode). In external mode the Simulink model is just a human machine interface (HMI) to help the developers to calibrate their controller. The way to switch the simulation into External mode is shown in Figure 1.

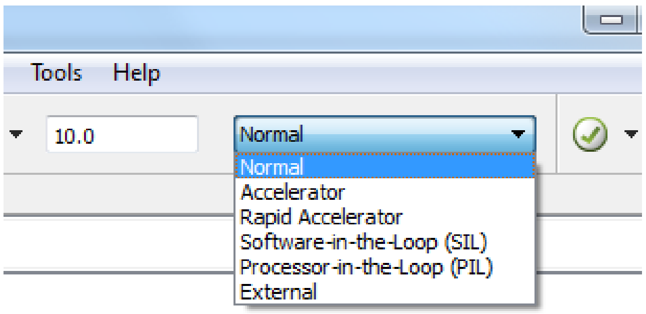


Figure 1. Switch to External Mode

If you are interested in how external mode is implemented and how to customize it, you can find more related information in MATLAB help. But it is not compulsory since this is not the emphasis of this lab.

3 Hands-on Exercises

Get your motor ready and everything connected. Now you will be guided to control the motor through Arduino in Simulink. As learned from Lab 1, we know that in order to control the motor velocity with Arduino, we need to read the velocity of the motor from the encoder and give control value to the PWM module. **Let's firstly try to make a driver block for PWM with S-Function**. The final driver block would look like the one shown in Figure 6. If we change the value in the constant block when the model is run in external mode, we can see the change of the motor velocity.

Open your MATLAB (2014b), create a new Simulink model and save it in an empty folder. Drag an S-Function Builder (it can be found at the location shown in Figure 2) into the model and double click on it. You will see the dialog as shown in Figure 3. Please click the help button at the bottom right corner and go through the instructions. First give this S-function block a name at the top left corner as circled in red in Figure 3. At the upper middle part of the dialog, you will see several tabs for different panels. In Initialization panel, set "Number of discrete states" to 1. Switch to Data Properties panel where you will see a set of sub-panels. In the Input ports sub-panel, change the port name to "duty". In the Output ports sub-panel, remove the output port "y0". In data type attributes sub-panel, set the Data type of the input port duty to float.

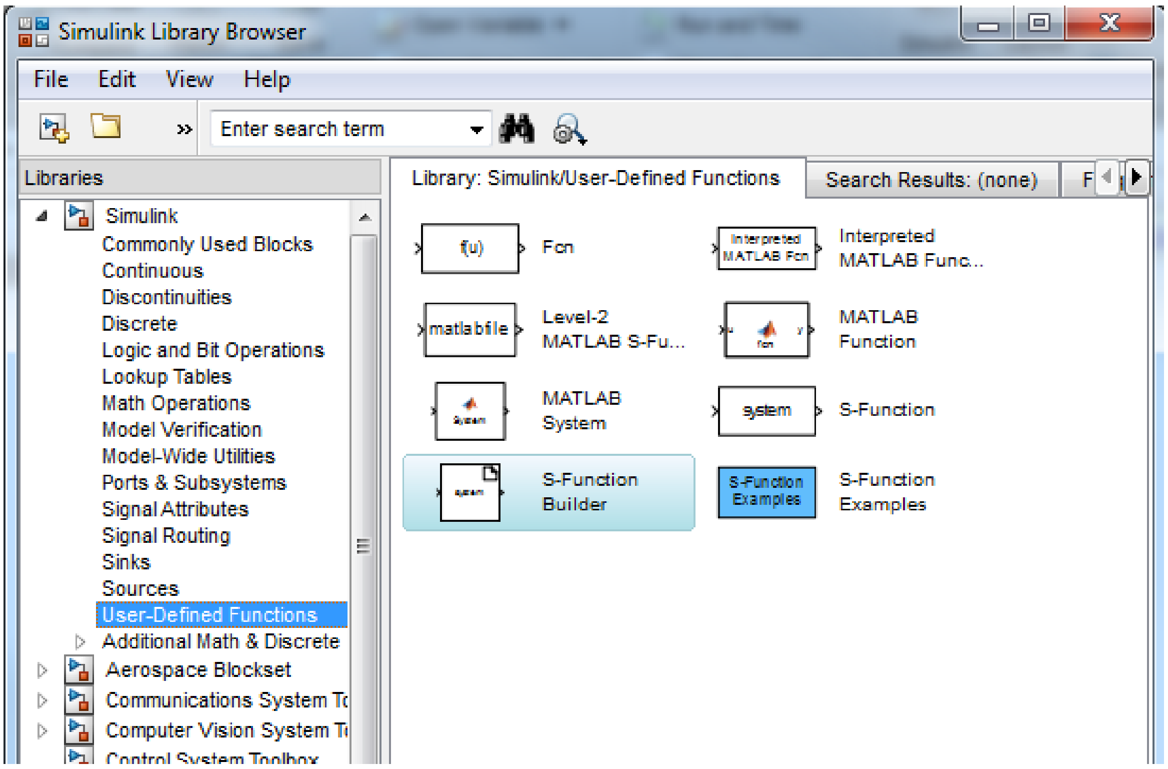
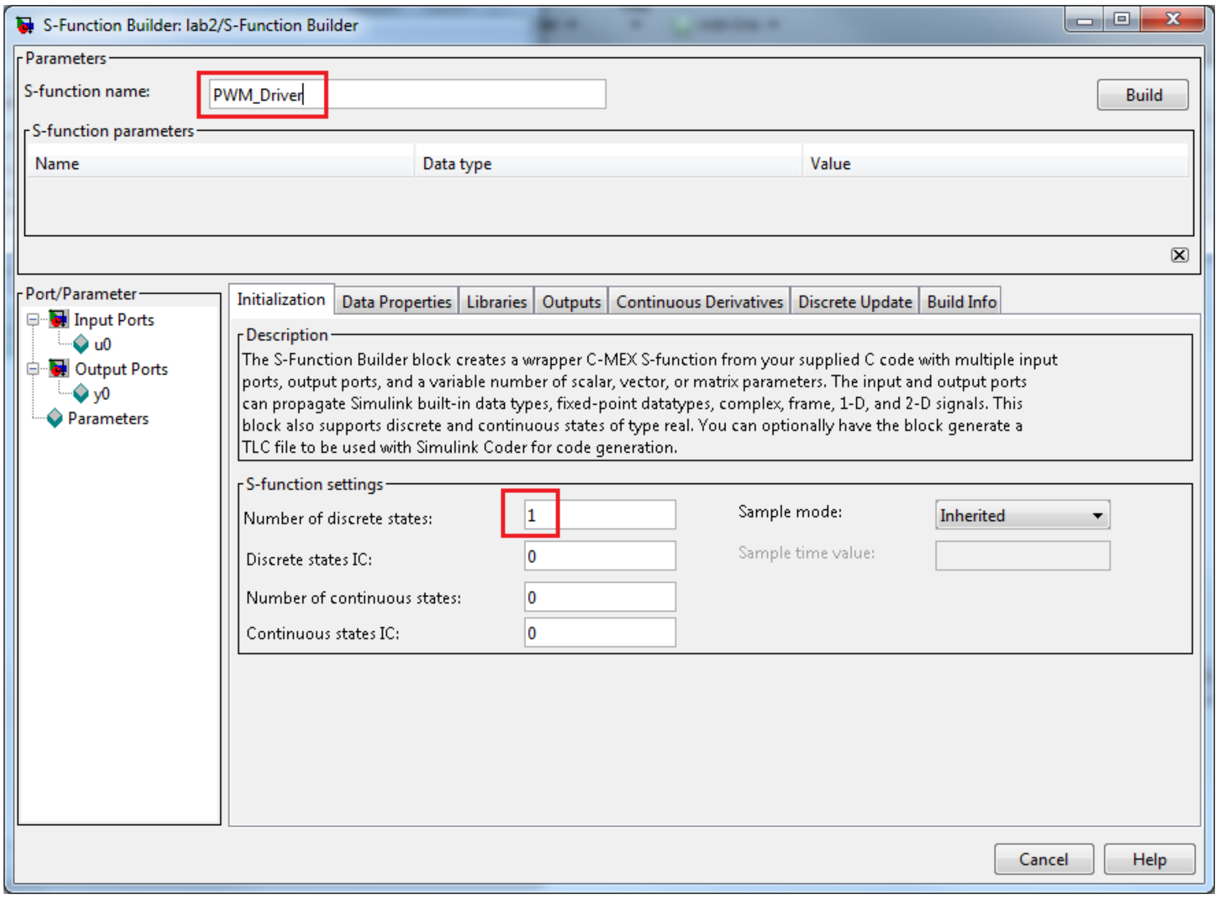


Figure 2. Location of S-Function Builder


S-Function Builder: lab2/S-Function Builder

Parameters

S-function name: PWM_Driver

Description

The S-Function Builder block creates a wrapper C-MEX S-function from your supplied C code with multiple input ports, output ports, and a variable number of scalar, vector, or matrix parameters. The input and output ports can propagate Simulink built-in data types, fixed-point datatypes, complex, frame, 1-D, and 2-D signals. This block also supports discrete and continuous states of type real. You can optionally have the block generate a TLC file to be used with Simulink Coder for code generation.

S-function settings

Number of discrete states: 1

Discrete states IC: 0

Number of continuous states: 0

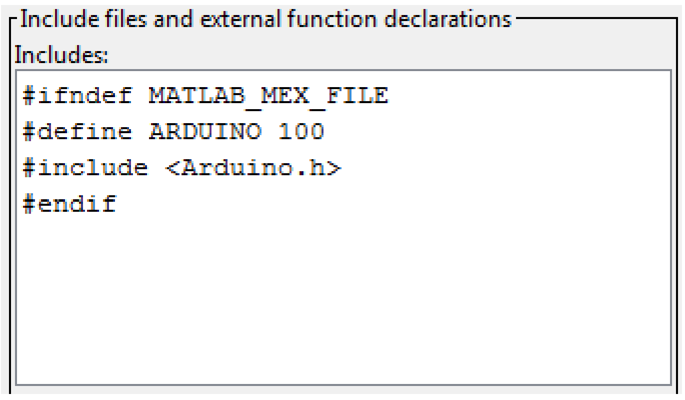
Continuous states IC: 0

Sample mode: Inherited

Sample time value:

Figure 3. Main Dialog of S-Function Builder

Click the tab Libraries. This panel is used to manage all the libraries used in this S-Function block. Fill the Includes box as shown below. Now you need to find out the function of the #ifndef statement in MATLAB help. **It is important for the following exercises**. You can also add any necessary header files within #ifndef for your block.


Include files and external function declarations

Includes:

```
#ifndef MATLAB_MEX_FILE
#define ARDUINO 100
#include <Arduino.h>
#endif
```

Figure 4. Includes box of Libraries panel

Next, let's switch to Discrete Update panel. We need to put all the code regarding PWM initialization (the code you wrote in the setup() function in Arduino IDE) in the white box. Please firstly have a look at the code description and the commented code example. The PWM initialization code should be run only once at the beginning of the program execution, however, the code programed in this section will be run periodically. Therefore, we can use the following if condition to guarantee that the code can be only executed once. Add the following code to the white box and add your own code to the specified position. You can reuse your code in lab 1.

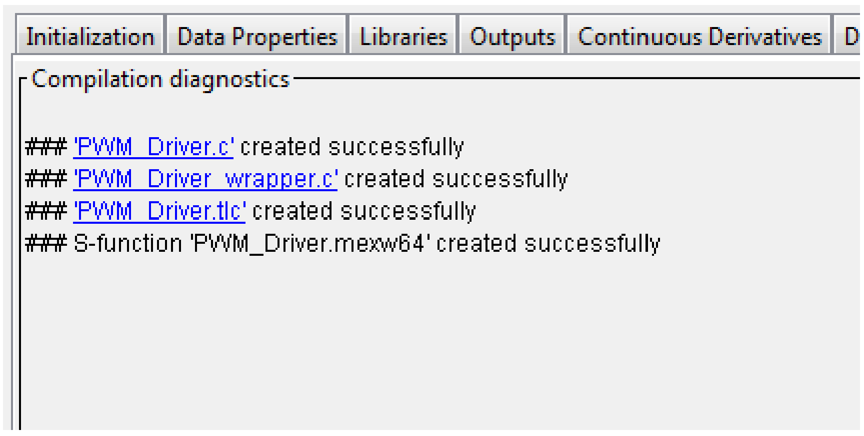
```
if(xD[0] != 1)
{
    //add your code here
    //code for lab1 can be reused

    xD[0] = 1;
}
```

In the Outputs panel, what we need to do is to provide the necessary code to transmit the input signal to PWM hardware. Please again, have a look at the code description and the commented code example first. The purpose of the code here is to read the input duty cycle and give it to PWM. All the code here will also be run periodically. Please check MATLAB help to figure out the differences between this section and the Discrete Update section. In order to guarantee that all the code here is executed after the initialization, your code need to be wrapped within the following if condition. Add the following code to the white box and add your own code to the specified position. You can reuse your code for setPWM() function in lab 1.

```
if(xD[0] == 1)
{
    //add your codes here.
    //Code fro lab_1 can be reused.
}
```

After you finish all the implementation, press the Build button at the top right corner. If the block is successfully built, you will receive the following messages.



```
### 'PWM_Driver.c' created successfully
### 'PWM_Driver_wrapper.c' created successfully
### 'PWM_Driver.tlc' created successfully
### S-function 'PWM_Driver.mexw64' created successfully
```

Figure 5. Successive message when S-Function block is successively built

If all the code you wrote in the boxes is C code, this is everything for the editing of the block. If any C++ features are used in the code, for example, Boolean typed variable or classes, you still need to do the some other changes. Please refer to the following video for the changes:

https://www.youtube.com/watch?v=_OLctOFjjYQ

Even though you have successfully built the S-Function block, it is still too early to say congratulations. Before that, we need to verify the block. The verification method is as simple as connecting your block to a constant block as shown below.

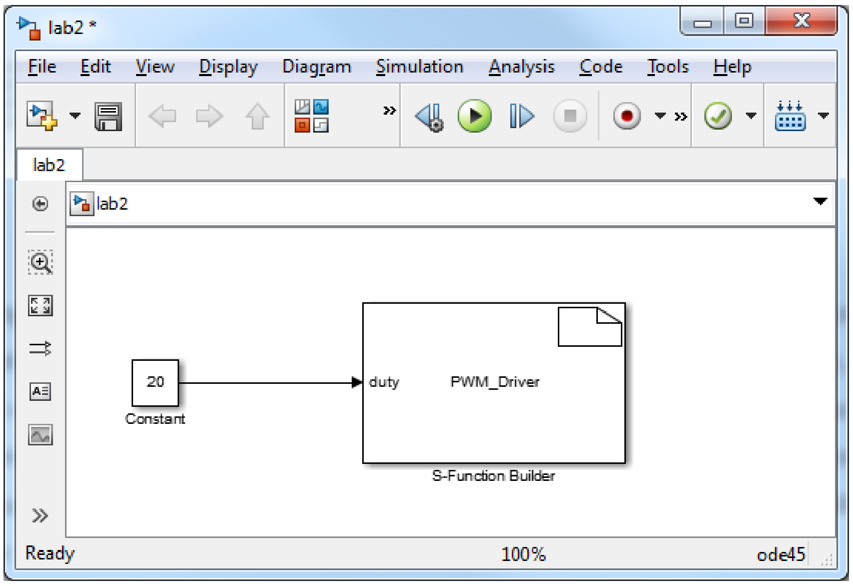


Figure 6. Verify your S-Function block for PWM

Before running the model, we need to do some configuration. Go to Tools -> Run on Target Hardware -> Prepare to run… Select target hardware as Arduino Mega 2560. Select Solver from the left side, set sample time to 0.05. Press OK. Set the running time of this Simulink model to inf and select External mode.

Now, take a deep breath, cross your fingers and press the Run button. Good luck! If the model is successfully compiled, downloaded and run on board and the motor is spinning at the given speed, try to change the input value during runtime to see whether this will change the velocity of the motor. If it is changed, congratulations! Now you can try to edit another S-Function block for encoder reading all by yourself.

The final task is to control the motor through Simulink code generation and external mode with the two driver blocks that you just implemented. The Discrete PID Controller block provided by Simulink can be directly used as the core controller. The final controller model should be similar as the one shown in Figure 7. The velocity of the motor can be monitored in real-time by a scope in Simulink. If the controller parameters are well tuned, the control performance should be better than the one shown in Figure 8.

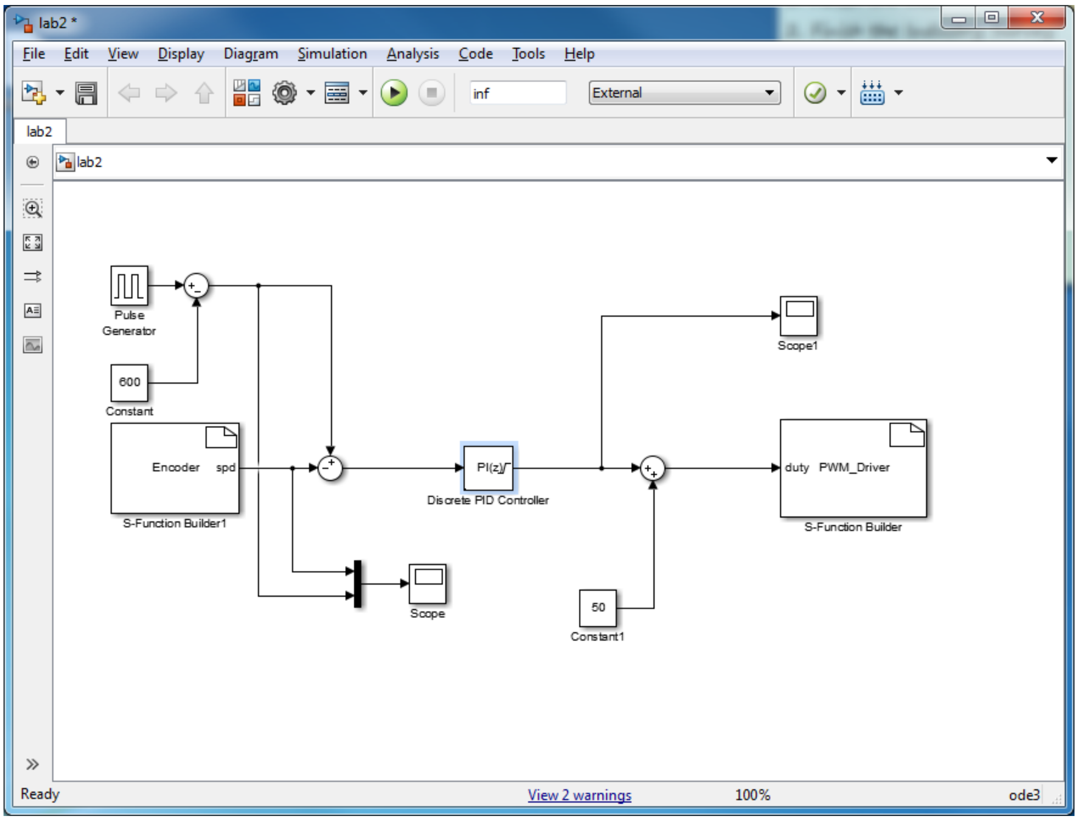


Figure 7. Simulink model of motor control strategy

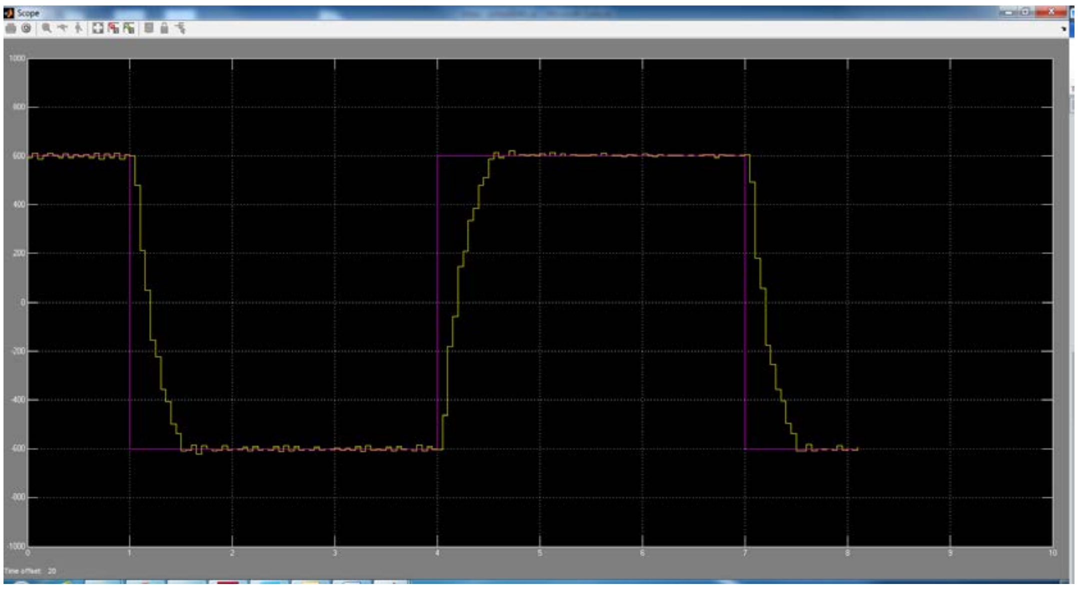


Figure 8. Control performance monitored in real-time

4 Questions

1) Comparing to traditional development process of embedded systems, what are the advantages of MBD?

2) Programing and debugging the driver blocks in Simulink seem to be more complicated than programing directly in C in Arduino IDE. What are the benefits of doing so?

3) What are the differences between software-in-the-loop (SIL) testing, processor-in-the-loop (PIL) testing and hardware-in-the-loop (HIL) testing? Why do we need all these types of testings? Is what we have done in this lab an SIL, PIL, HIL or something else?

4) Consider the case where we have multiple motor types with different specifications. It means that those motors need different drivers on code level but the same interface (S-Function block) for Simulink. Can you think of a simple and standard approach to implement and manage those drivers?

5) In the project folder, you can find a sub-folder named ”lab2_rtt” (if your project is named lab2). In this folder, there is a file named “ert_main.c” which contains the implementation of the main function of the project. Please go through this file and try to figure out the basic structure of the generated code. (Other related files may also need to be referred to .)