Multicopter Design and Control Practice Experiments

RflySim Advanced Courses
Lesson 02: Flight Control Experiments

Dr. Xunhua Dai, Associate Professor,

School of Computer Science and Engineering,
Central South University, China;

Email: dai.xh@csu.edu.cn ;

https://faculty.csu.edu.cn/daixunhua

北航可靠飞行控制研究组
BUAA Reliable Flight Control Group

Content

1. Course Learning

2. Platform Framework

3. Advanced Examples

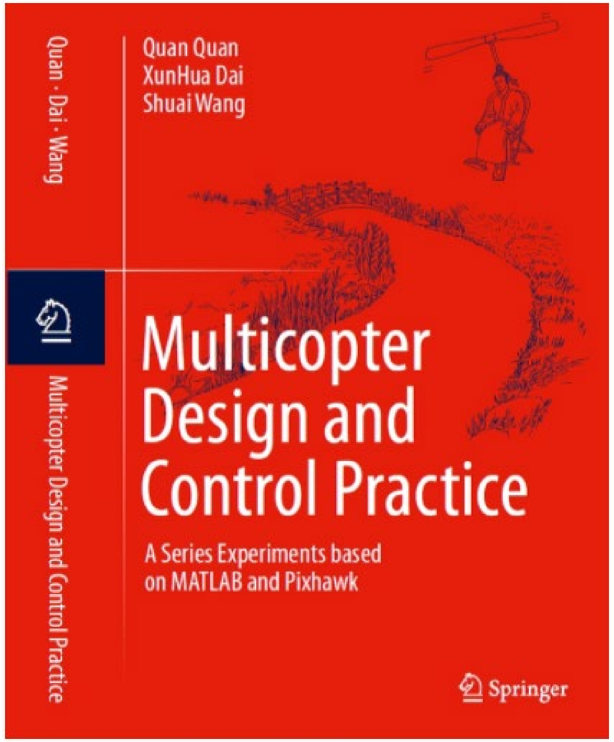
4. Summary

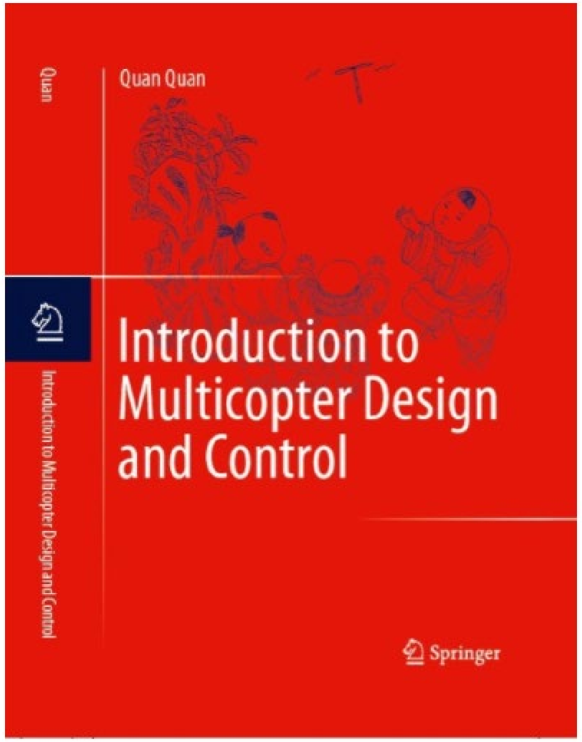
Path of demo and source code of this lesson:
“RflySimAPIs\FlightControlExpCourse”

1. Course Learning

1.1 Reference Books

- Quan Quan, Xunhua Dai, Shuai Wang. Multicopter Design and Control Practice. Springer, 2020. URL: https: //www.springer.com/gp/book/9789811531378





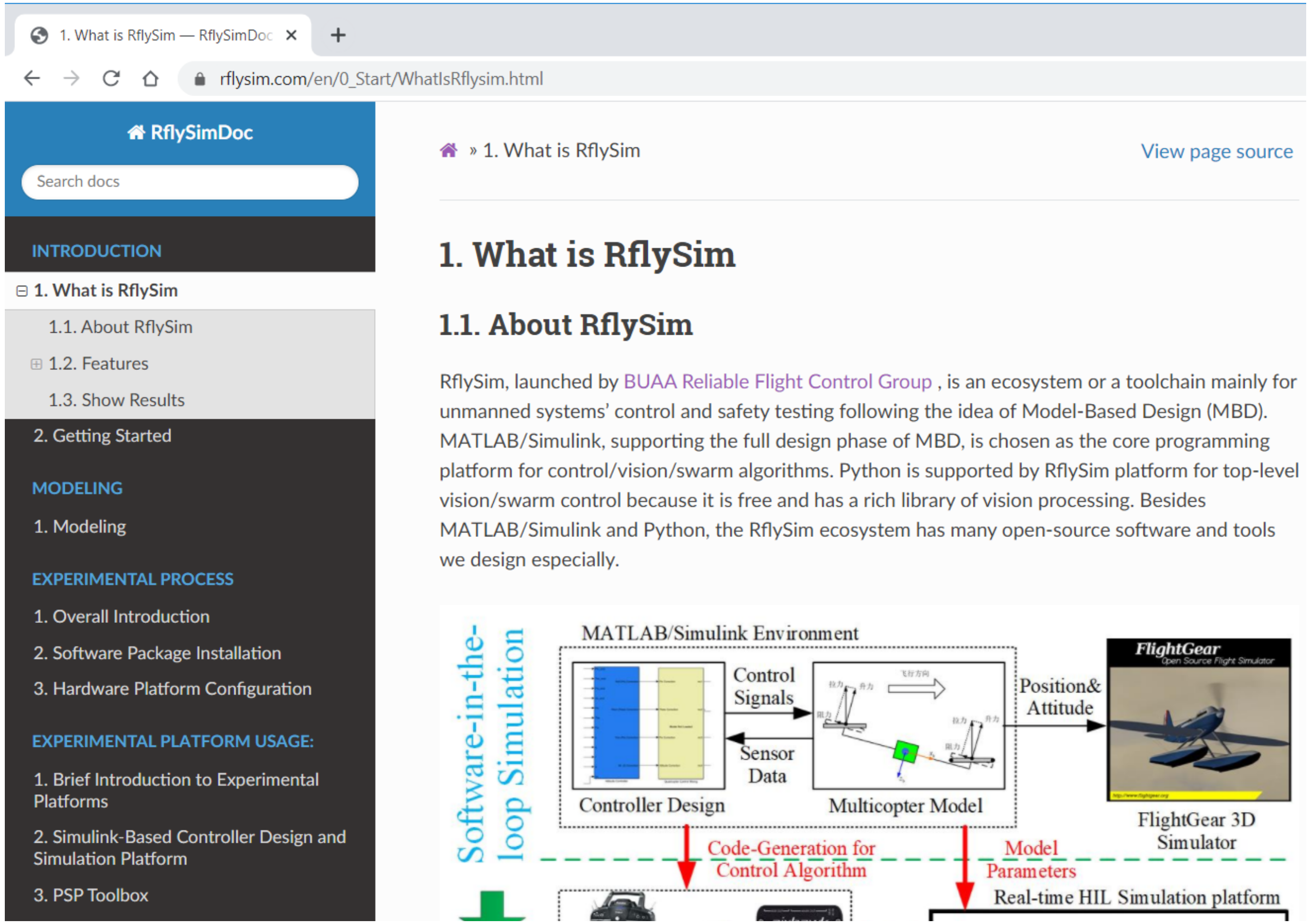
Note:

- The book *Multicopter Design and Control Practice* on the left is a practical course for flight control algorithm development launched in 2020. It contains some theoretical knowledge and a series of experiments, help readers quickly program their own algorithms in Simulink , and then auto generate C/C++ to Pixhawk hardware for flight experiment.
- The book *Introduction to Multicopter Design and Control* on the right is a tutorial launched in 2017, mainly for multicopter control theory.

1. Course Learning

1.2 Website: https: //rflysim.com/

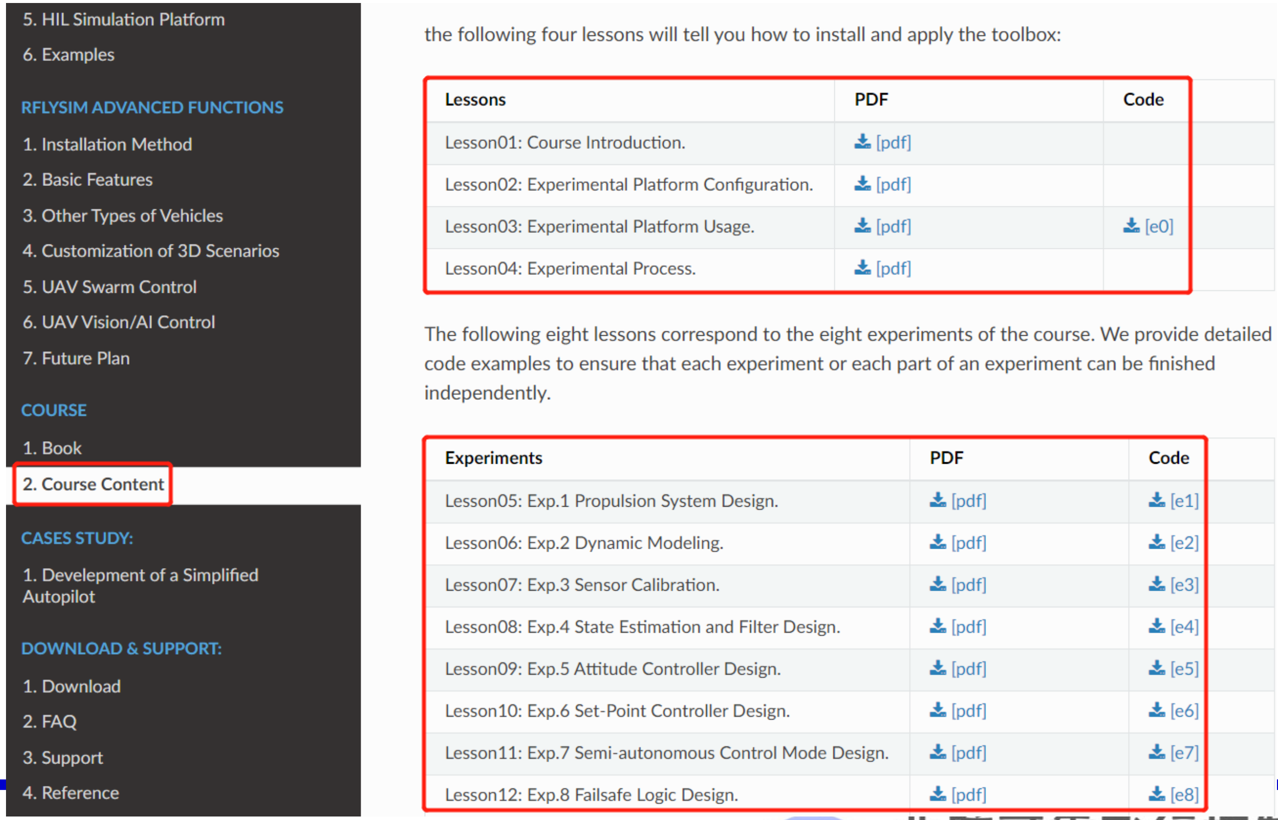
- include material of top four sections of the book about the platform
- provide download address for PPTs and source code
- provide some project demos developed with RflySim platform
- Provide basic introduction and instructions of uses of advanced functions
- Provide answers to common questions



1. Course Learning

1.3 Online latest PPTs/source code download address

- https: //rflysim.com/en/5_Course/CourseContent.html
- https: //github.com/RflySim/RflyExpCode

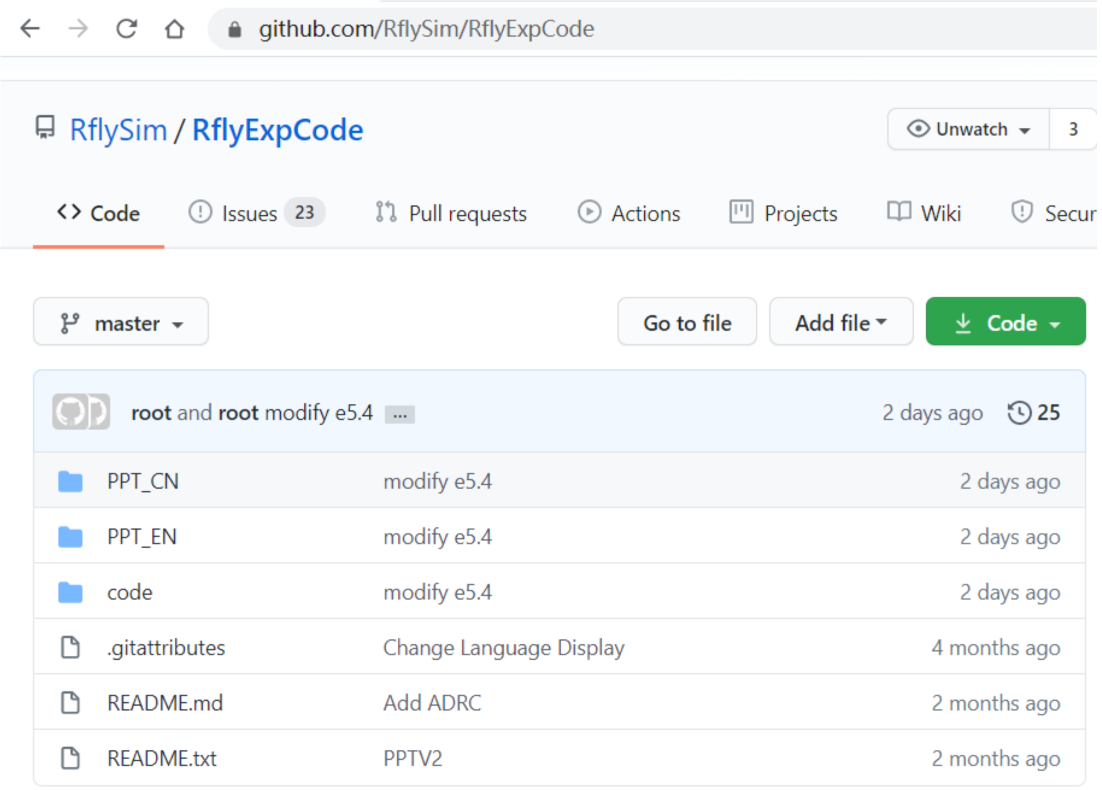


the following four lessons will tell you how to install and apply the toolbox:

Lessons	PDF	Code
Lesson01: Course Introduction.	[pdf]	
Lesson02: Experimental Platform Configuration.	[pdf]	
Lesson03: Experimental Platform Usage.	[pdf]	[e0]
Lesson04: Experimental Process.	[pdf]	

The following eight lessons correspond to the eight experiments of the course. We provide detailed code examples to ensure that each experiment or each part of an experiment can be finished independently.

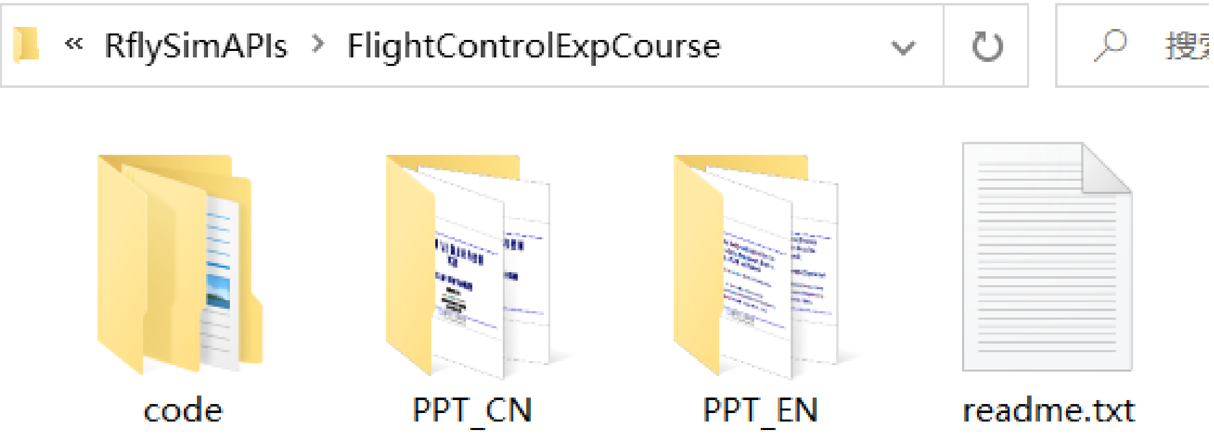
Experiments	PDF	Code
Lesson05: Exp.1 Propulsion System Design.	[pdf]	[e1]
Lesson06: Exp.2 Dynamic Modeling.	[pdf]	[e2]
Lesson07: Exp.3 Sensor Calibration.	[pdf]	[e3]
Lesson08: Exp.4 State Estimation and Filter Design.	[pdf]	[e4]
Lesson09: Exp.5 Attitude Controller Design.	[pdf]	[e5]
Lesson10: Exp.6 Set-Point Controller Design.	[pdf]	[e6]
Lesson11: Exp.7 Semi-autonomous Control Mode Design.	[pdf]	[e7]
Lesson12: Exp.8 Failsafe Logic Design.	[pdf]	[e8]

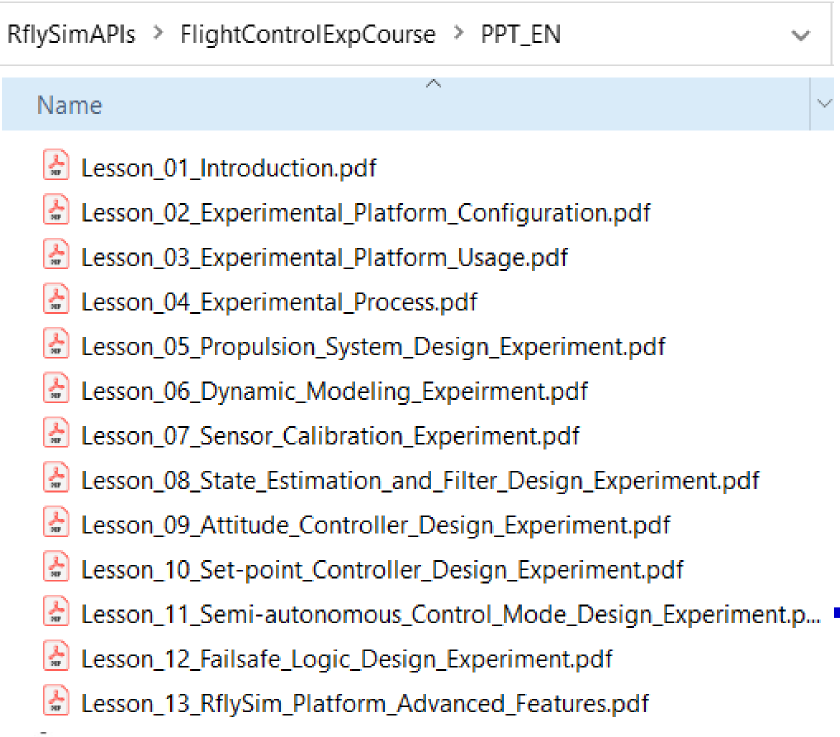


1. Course Learning

1.4 Local PPTs/source code address

- Directly entering “**RflySimAPIs\FlightControlExpCourse**” folder
- “**PPT_EN**” contains all English PPT files
- “**code**” contains source code correspond to the lesson
- Please connect with Website/PPTs/source code to learn courses related to flight control algorithm

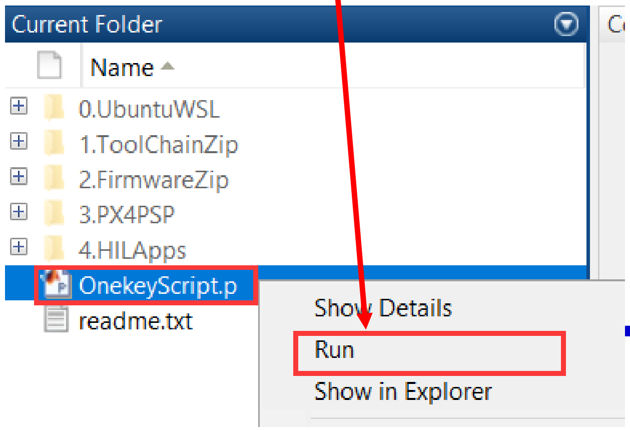


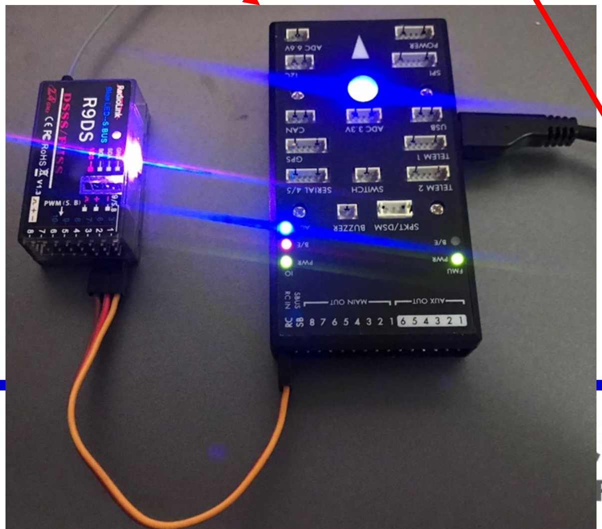


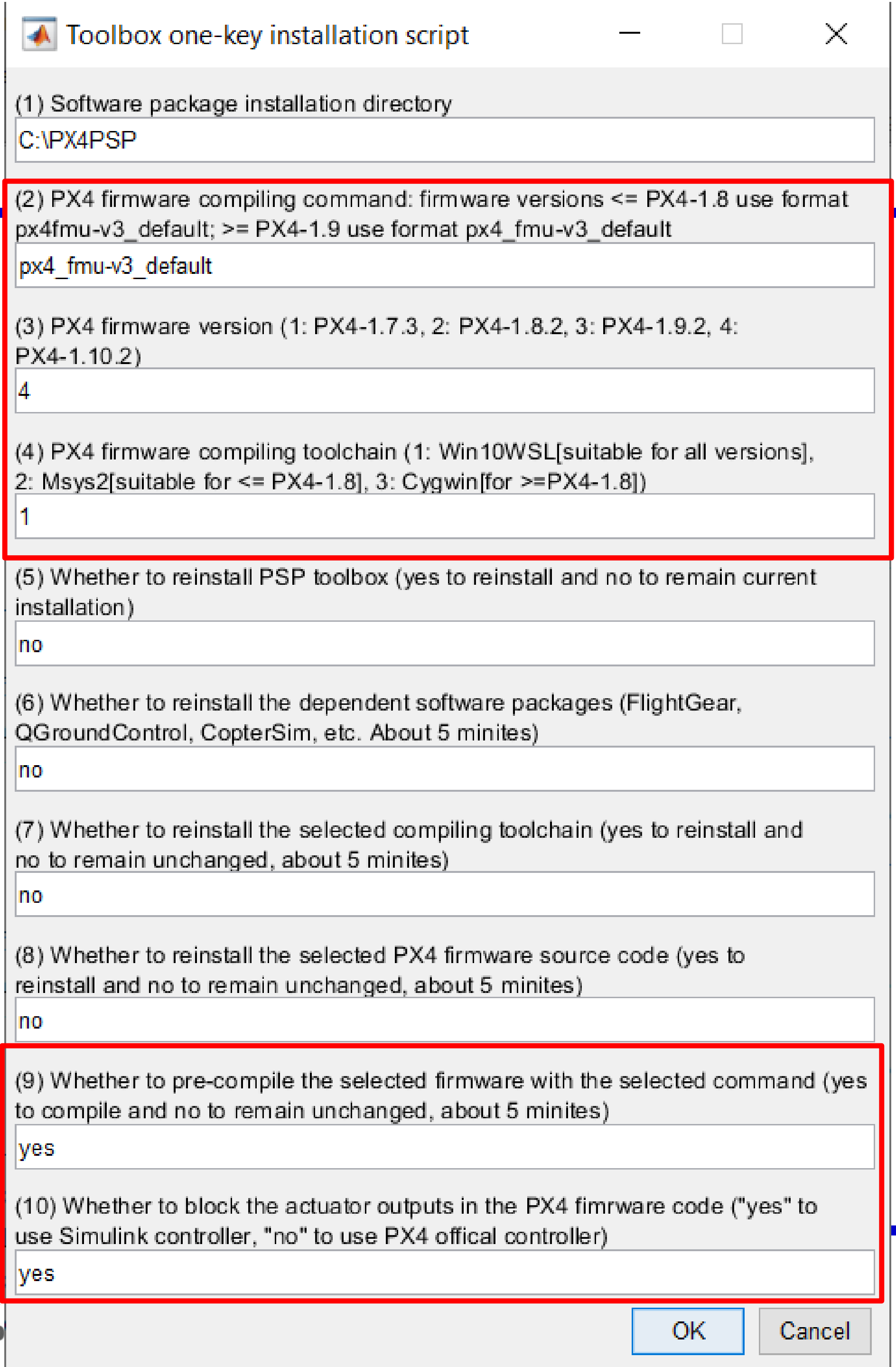
1. Course Learning

1.5 Installation Configuration

- Re-run the “**OnekeyScript.p**” script in installation package
- If you use the recommended Pixhawk1 (see below picture), the compile command ‘**px4_fmu-v3_default**’ from textbook
- Please enter “**yes**” for the option **(10)** on the right figure to block the PX4 output
- All the rest configuration shows on the right







1. Course Learning

(2) PX4 firmware compiling command: firmware versions <= PX4-1.8 use format px4fmu-v3_default; >= PX4-1.9 use format px4_fmu-v3_default

px4_fmu-v3_default

Different Pixhawk hardware has different compile order, normal compile order (PX4 1.9 and later firmware) shows as follow

- **Pixhawk 1**: px4_fmu-v2_default
- **Pixhawk 1 (2M flash):** px4_fmu-v3_default
- **Pixhawk 4**: make px4_fmu-v5_default
- **Pixracer**: make px4_fmu-v4_default
- **Pixhawk 3 Pro**: make px4_fmu-v4pro_default
- **Pixhawk Mini**: make px4_fmu-v3_default
- **Pixhawk 2**: make px4_fmu-v3_default
- **mRo Pixhawk**: make px4_fmu-v3_
- **HKPilot32**: make px4_fmu-v2_default
- **Pixfalcon**: make px4_fmu-v2_default
- **Dropix**: make px4_fmu-v2_default
- **MindPX/MindRacer**: make airmind_mindpx-v2_default
- **mRo X-2.1**: make auav_x21_default
- **Crazyflie 2.0**: make bitcraze_crazyflie_default
- Intel® Aero: make intel_aerofc-v1_default

Pixhawk 1 (FMUv2)
2M flash Version (FMUv3)

mRo Pixhawk (FMUv3)

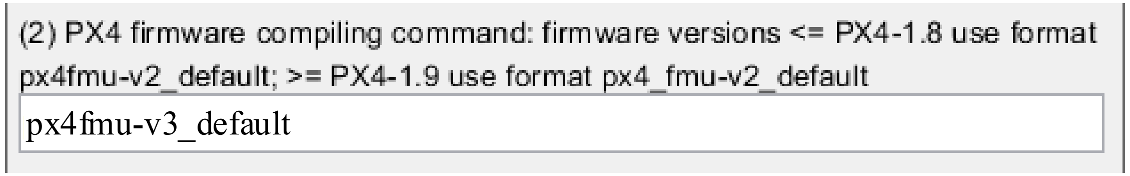
Cube (Pixhawk 2, FMUv3)

Pixhawk 3 Pro (FMUv4)

Pixhawk 4 (FMUv5)

Pixhawk 4 Mini (FMUv5)

1. Course Learning

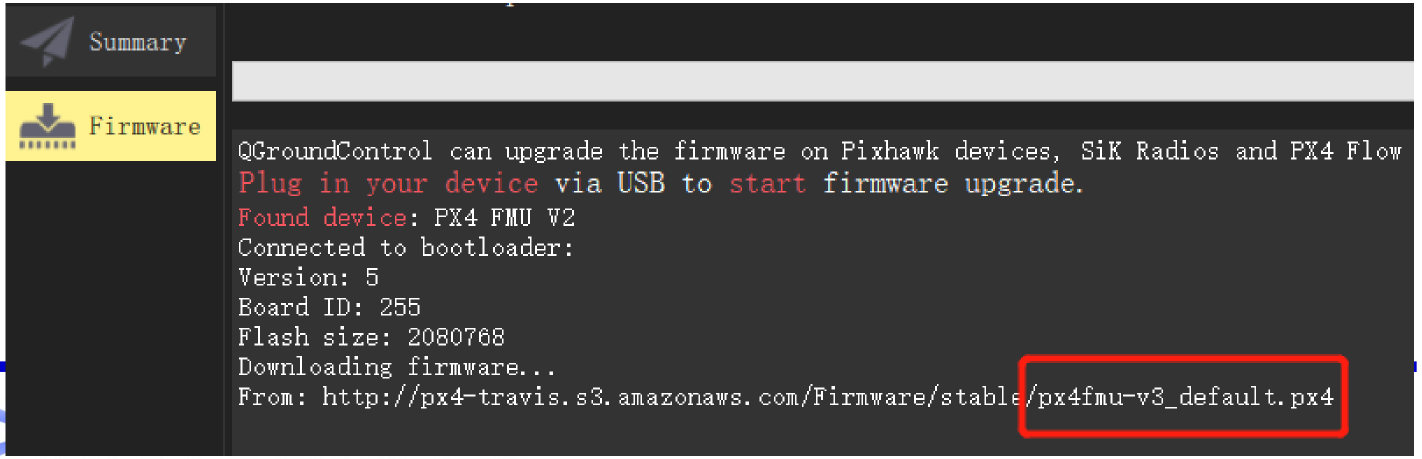


Method to find out the desired compiling command for your Pixhawk:

1) Open QGroundControl (QGC) and enter the “Setting” (Gear icon) – “Firmware” Page;

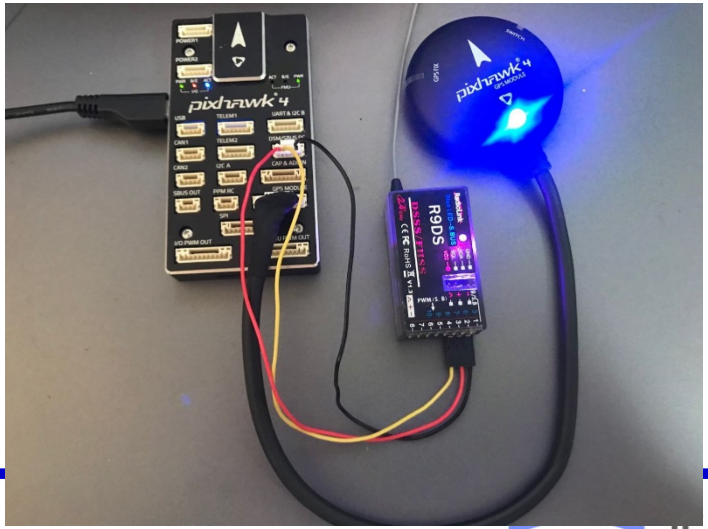
2) Connect Pixhawk with a USB cable, and the QGC will turn to the state in the right figure, then click “OK” to update;

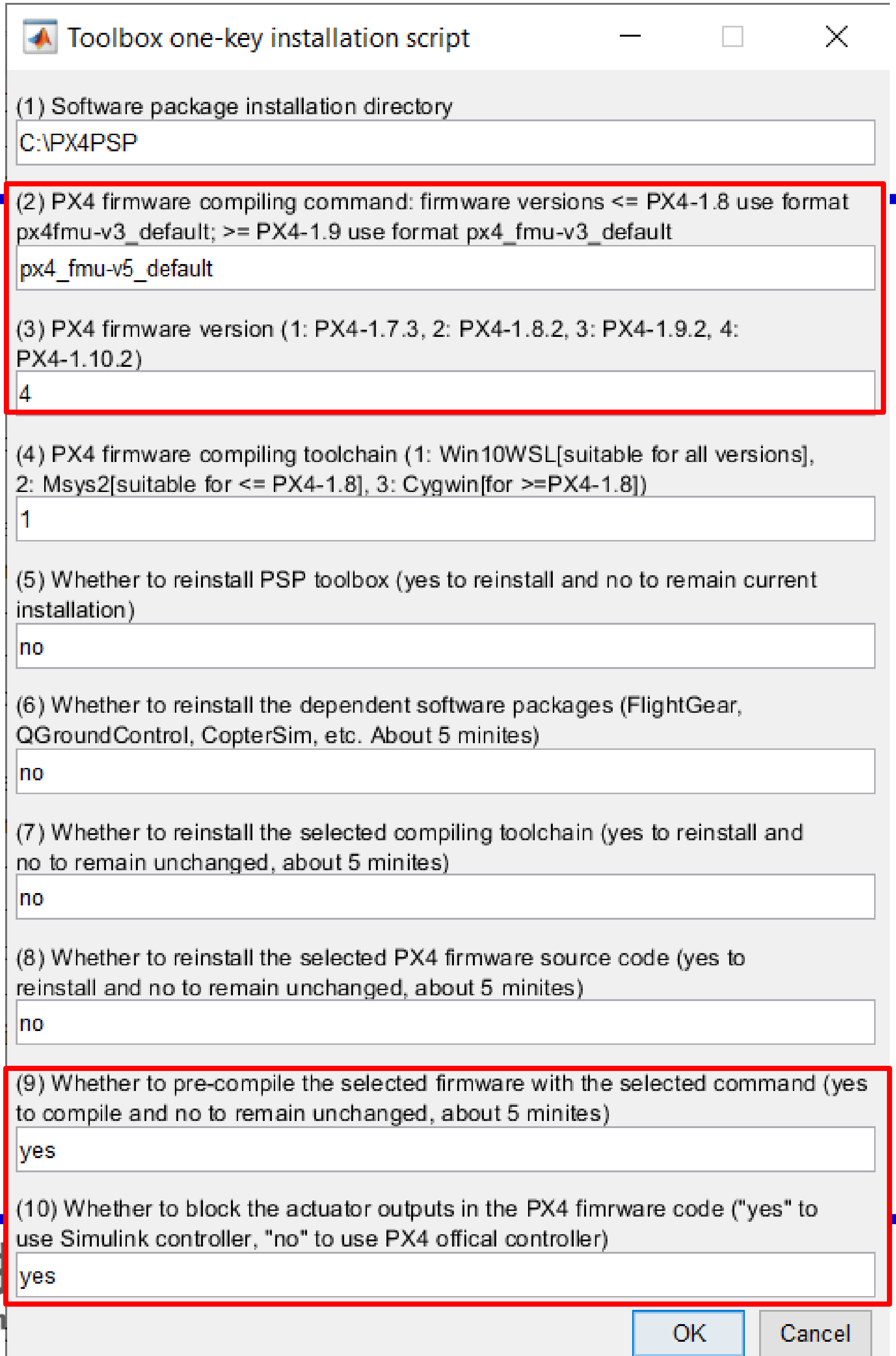
3) QGC will auto download the desired .px4 firmware, so the desired compiling command can be found in the download link. For example, px4fmu-v3_default is obtained for Pixhawk 1 (2Mb Flash).



1. Course Learning

- If using latest Pichawk4 autopilot (shows as follow), the compile order is “**px4_fmu-v5_default**” Recommend to
- use the latest PX4 firmware version “**4**” — PX4-1.10.2
- Enter “**yes**” for the (10) option to block PX4 output
- All the rest configuration shows on the right

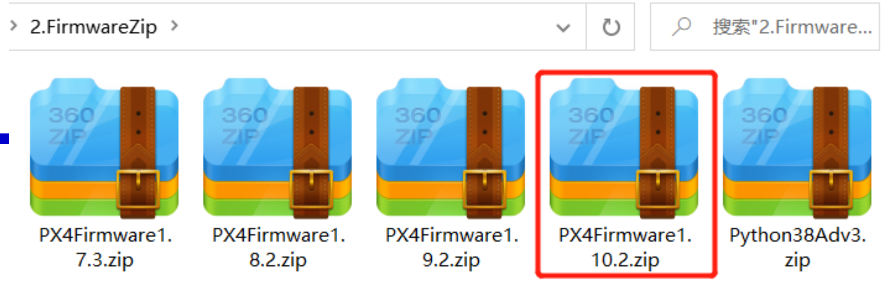


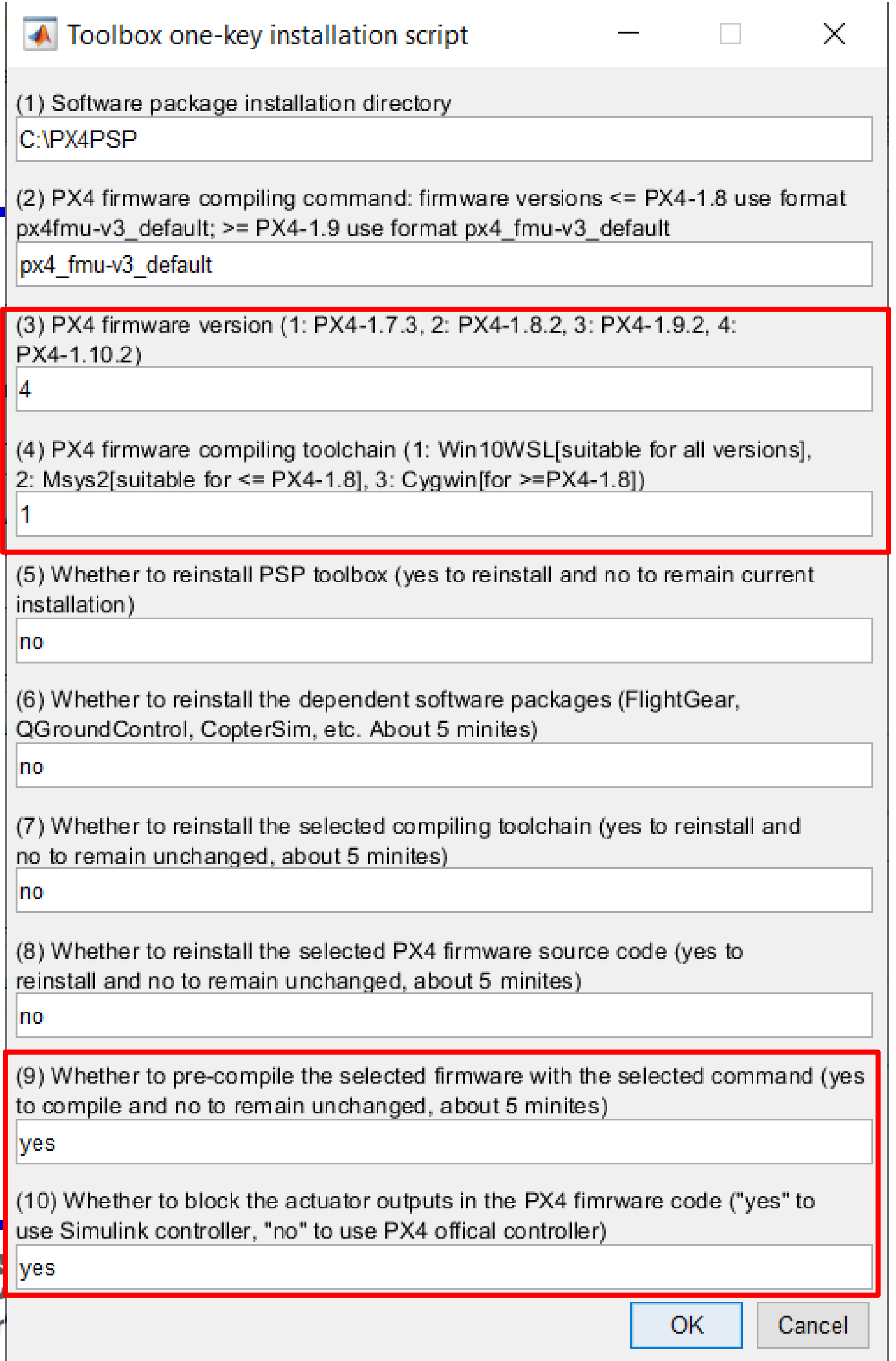


1. Course Learning

1.6 Use your own PX4 source code

- If you need to use your own PX4 firmware code, please compress your "**Firmware**" folder into a "**Firmware.zip**" file, rename it according to the version (for example, if your code is based on PX4 1.10, name it "**PX4Firmware1.10.2.zip**") and copy it to the "**2.FirmwareZip**" folder of the installation package to override, select "**4**" in the firmware version in the installation option on the right.
- It is recommended to use Win10WSL compiler, so choose "**1**" for the compiler.
- Whether to block PX4 output option (10), select "**yes**", the script will automatically complete all required firmware modifications to adapt to this platform



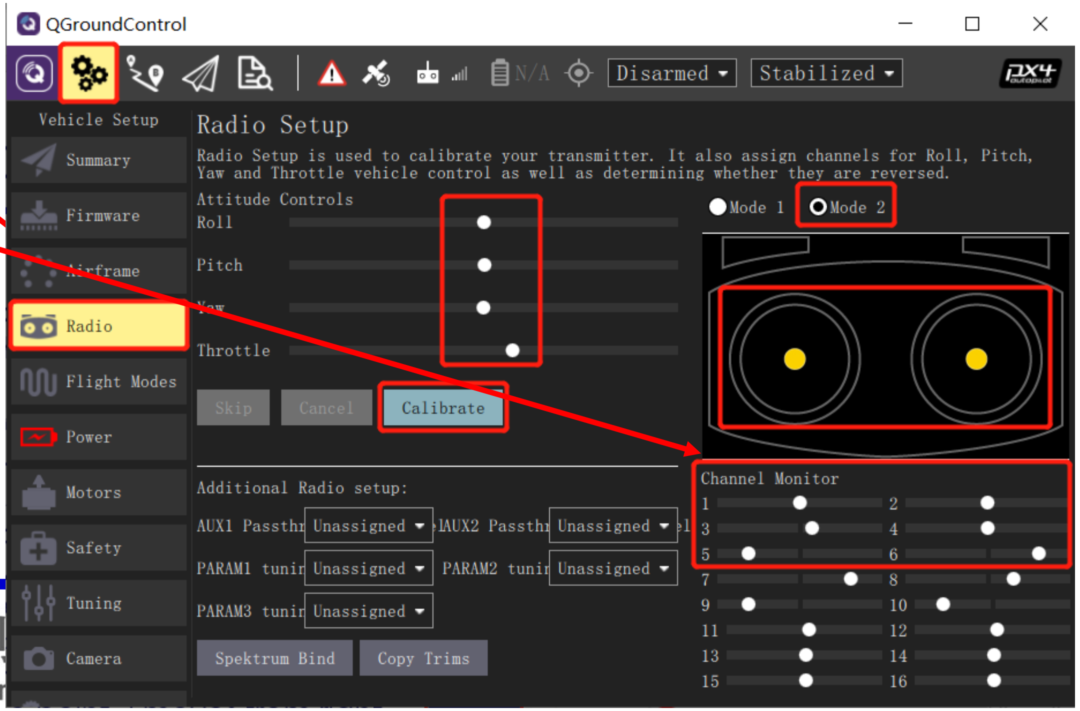


1. Course Learning

1.7 RC Transmitter configuration and calibration

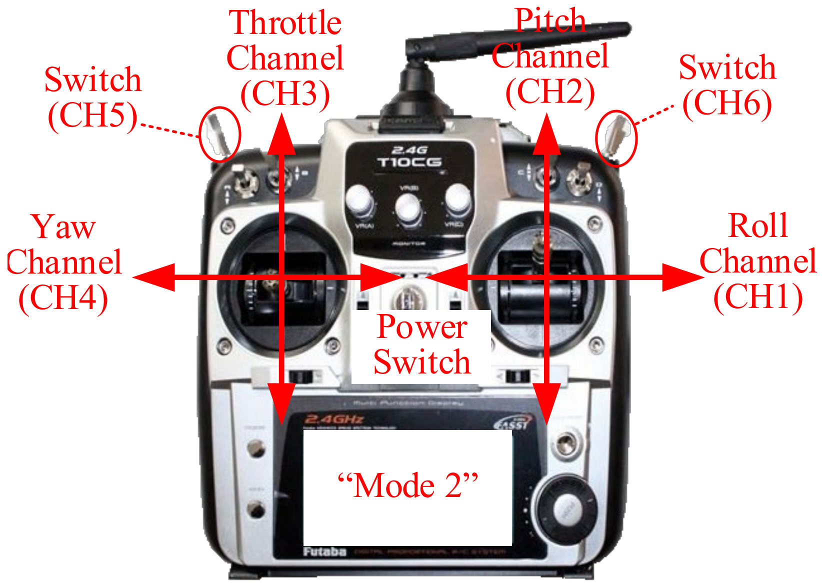
1. Connect the Pixhawk with the RC receiver correctly, and then connect the Pixhawk and the computer via a USB cable, power on the RC Transmitter, open the QGroundControl, and click the "**Radio**" tab.
2. Turn the **CH1 to CH5** channels of the RC Transmitter from left to right (or from top to bottom) (see the upper right picture), and observe the small points of each channel in the red box area on the right side of the QGC in the lower right picture. If you observe: the small point **1, 2, 4, 5**, and 6 move from left to right (PWM from **1100 to 1900**); the small point 3 moves from right to left, indicating that the RC Transmitter is set correctly. Otherwise, you need to reconfigure the RC Transmitter.
3. Click the "**Calibrate**" button in the lower right picture and follow the prompts to calibrate the RC Transmitter.

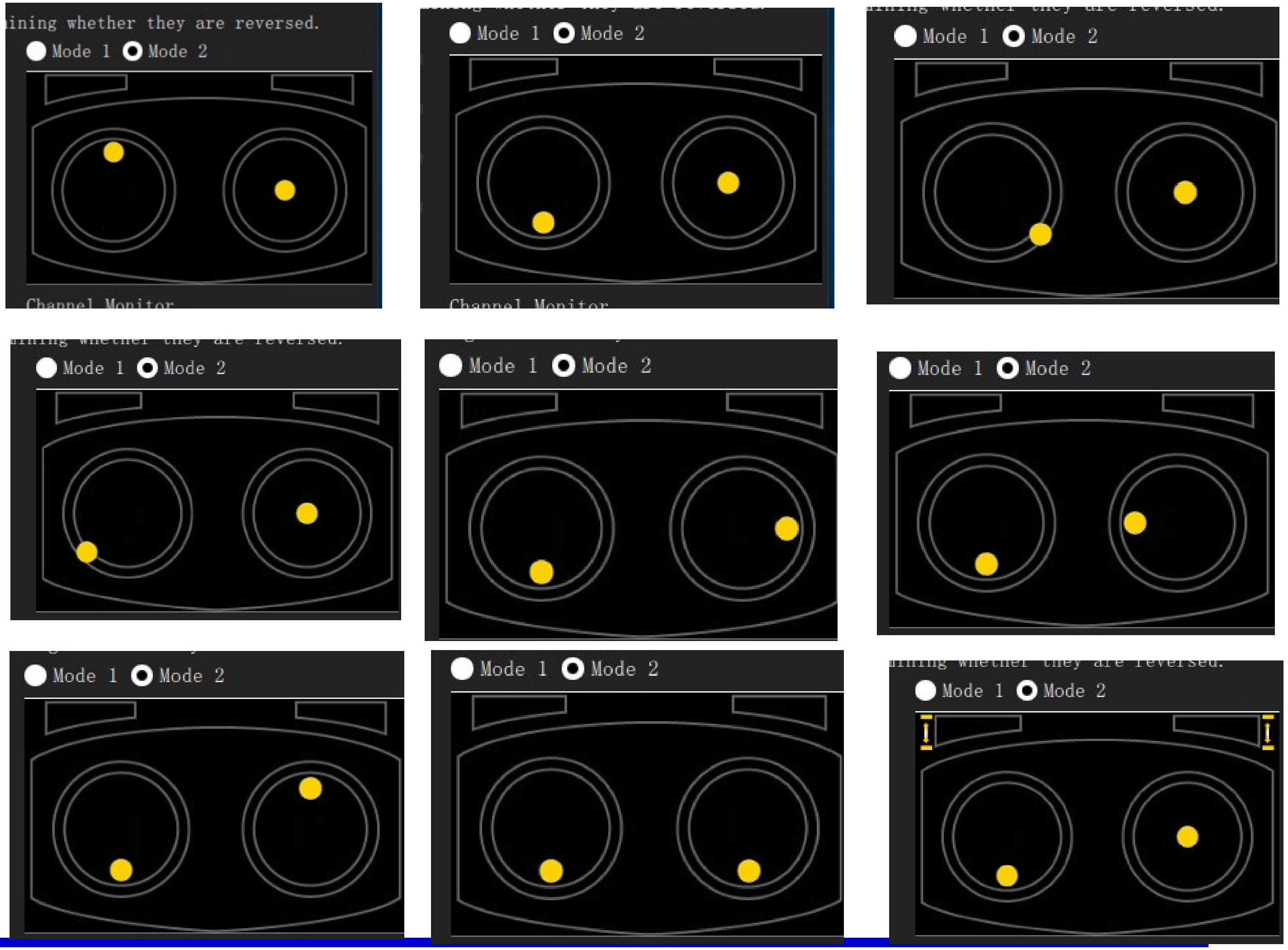
Note: the arrows point to the increasing direction of PWM



1. Course Learning

4. Click ‘**Calibrate**’-‘**Next**’ in QGC ground station, then place stick as the right picture shows (follow the QGC real-time instructions) to finish the calibration.



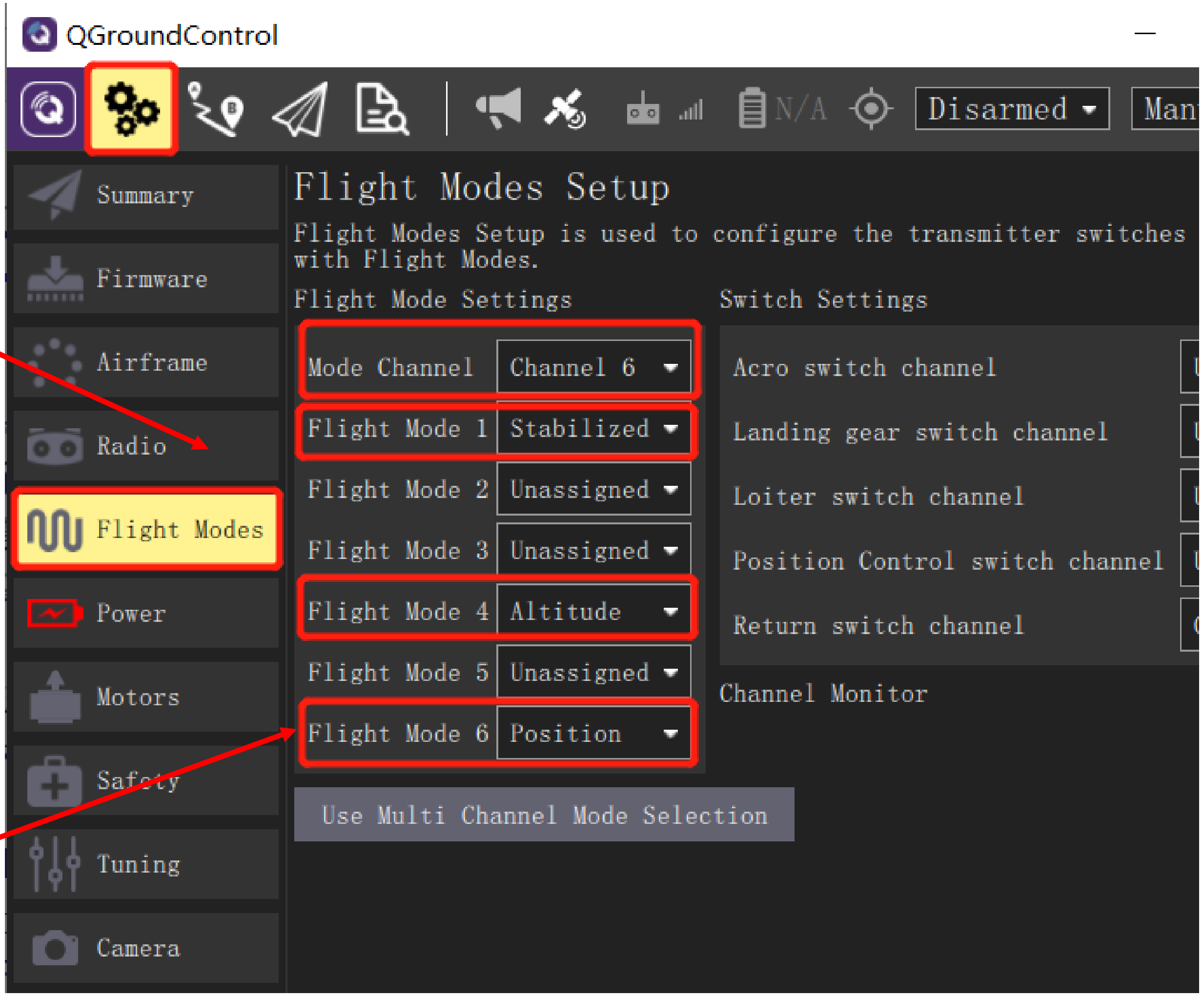


Throttle : control up-down movement
Pitch : control forward-backward
Yaw : control vehicle head direction
Roll : control left-right movement

1. Course Learning

1.8 flight mode setting

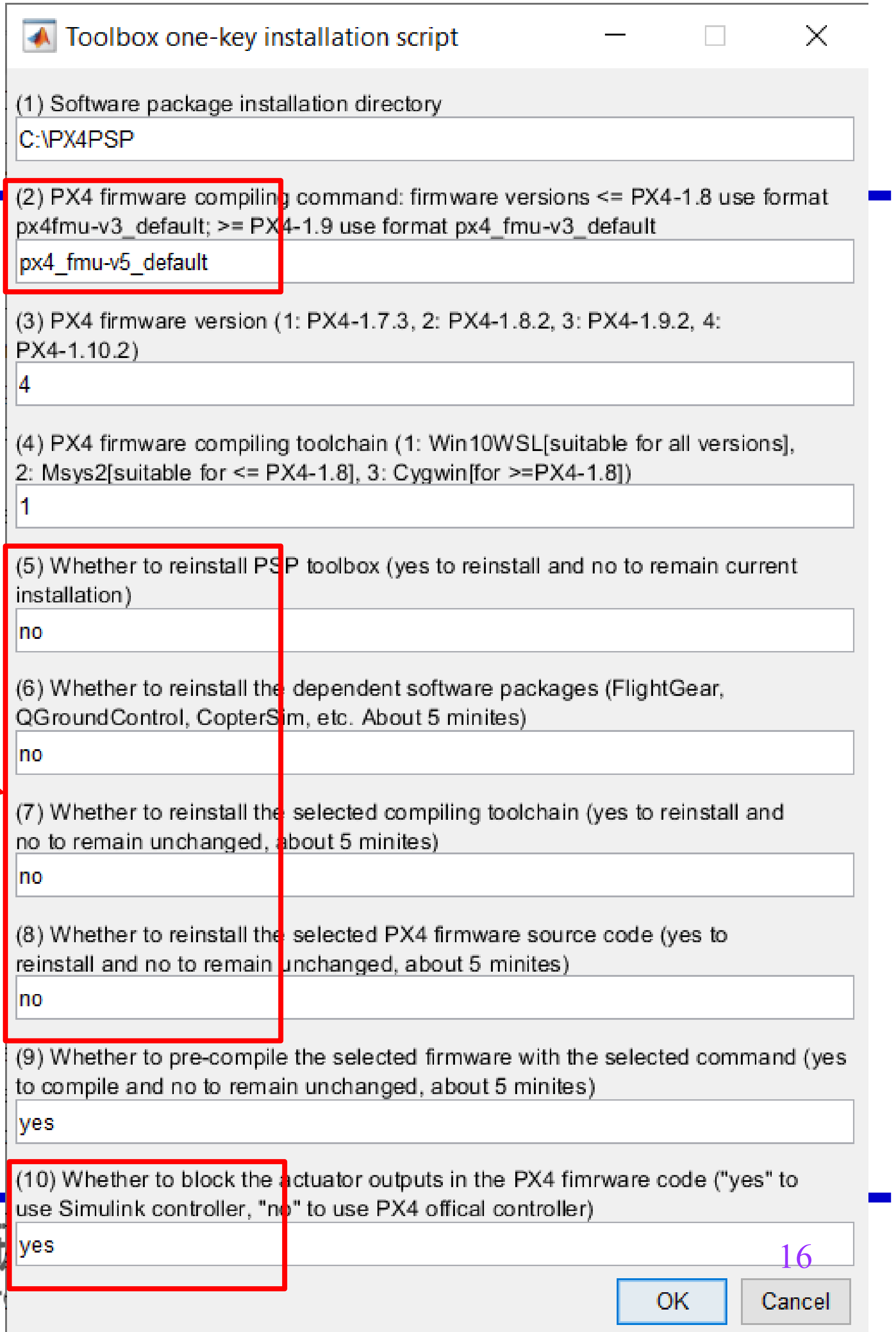
- After the above RC transmitter calibration steps, click on the QGC ground station to enter the "**Flight Modes**" setting page, and select "**Mode Channel**" as the previously tested CH6 channel. Since the CH6 channel is a three-position switch, the top, middle, and lower positions of the switch correspond to the three labels "**Flight Mode 1, 4, and 6**" respectively.
- As shown in the figure on the right, set these three labels to "**Stabilized**" (self-stabilization mode, only attitude control), "**Altitude**" (fixed height mode, with attitude and height control) and "**Position**" (fixed-point mode, with attitude, fixed height and horizontal position control). In the subsequent HIL simulation, you can experience different control effects by switching different modes.



1. Course Learning

1.9 Switch development mode configuration

- If you want to switch from **vision/swarm** mode to **low-level flight control development** mode, you only need to re-run the ‘**OnekeyScript.p**’ script, choose required compiling command for Pixhawk hardware, and block the PX4 outputs.
- For saving installation time, some installed components already installed can be skip, choose ‘**no**’ is ok. (shown on the right)
- If you only need to change firmware compile command in **flight control development** mode, for example, from ‘**px4_fmu-v3_default**’ to ‘**px4_fmu-v5_default**’, you only need to enter ‘**PX4CMD px4_fmu-v5_default**’ in MATLAB command window, no need to rerun the installation script.



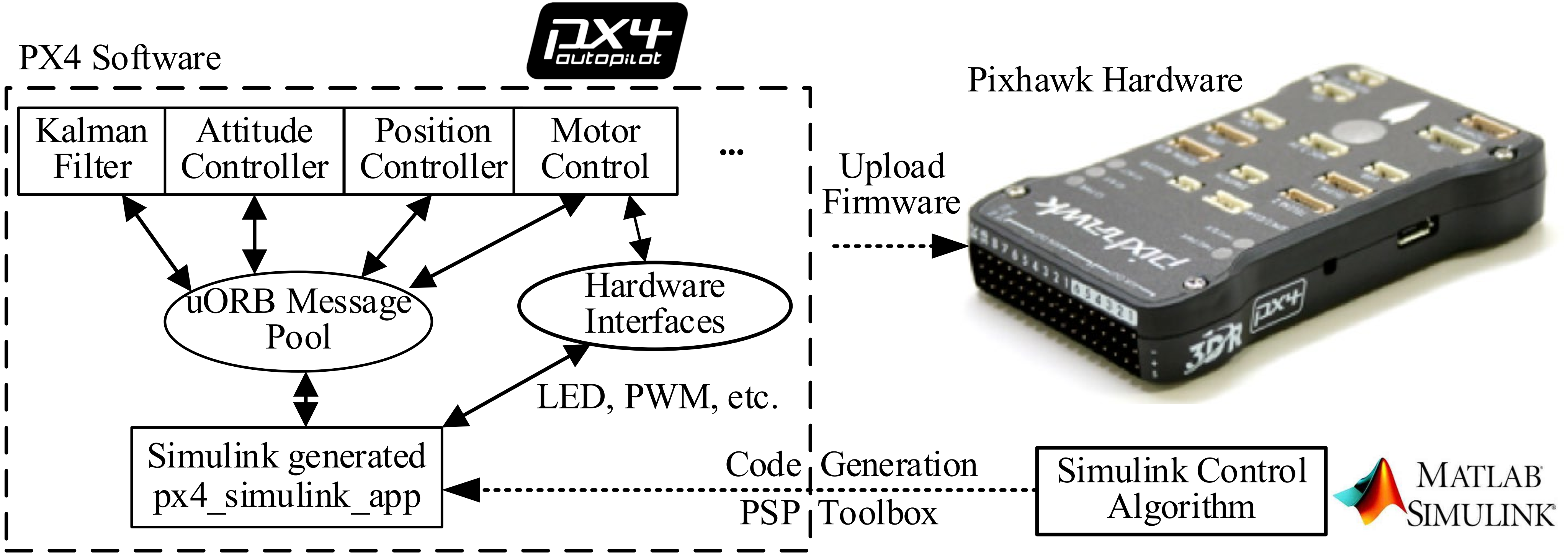
Content

1. Course Learning
2. Platform Framework
3. Advanced Examples
4. Summary

2. Platform Framework

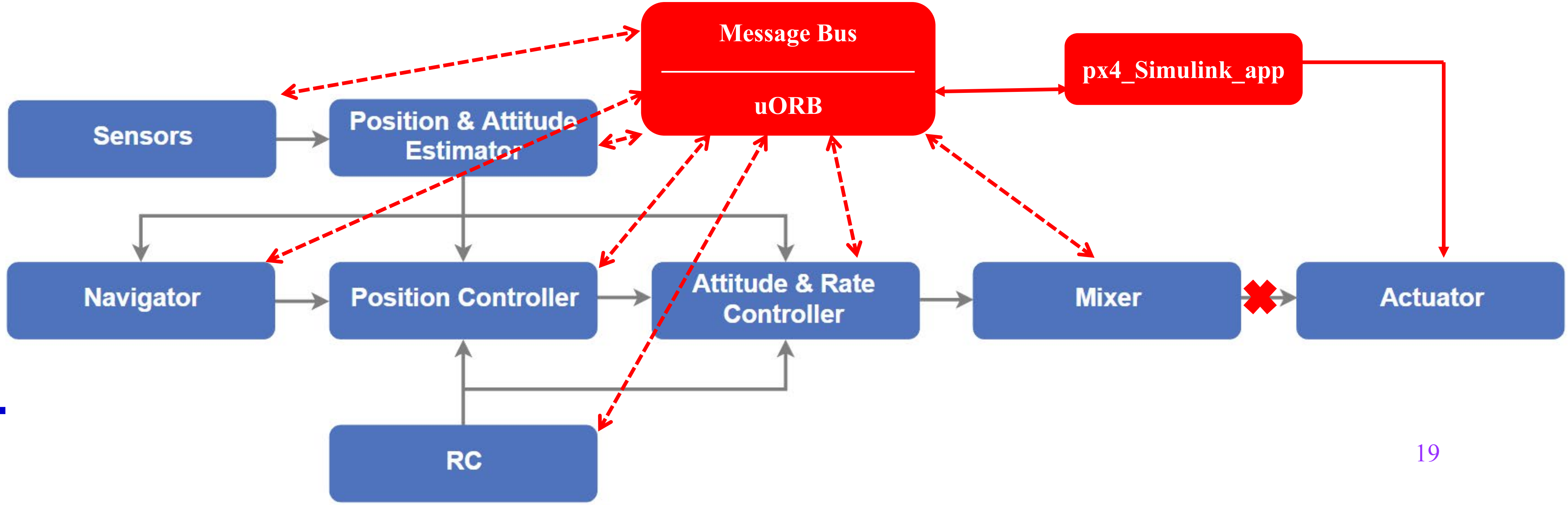
2.1 Pixhawk/PX4/Simulink code generate platform structure

- Pixhawk is the hardware (equivalent to a mainframe computer), PX4 is the flight control software (equivalent to the Windows OS), the Simulink controller generate the code and compile it into firmware (equivalent to the system iso image), and uploads it to the Pixhawk hardware (equivalent to reinstalling the system) , Simulink controller runs in parallel with a new thread (equivalent to a third-party APP on the computer) independent of the official PX4 controller (equivalent to system pre-installed software)



2.2 Why block PX4 output

- PX4 adopts uORB publish and subscribe message mechanism, any APP can obtain and publish data from uORB message pool
- Simulink code is generated to Pixhawk to generate an APP named **px4_simulink_app**, which can communicate with other APPs in PX4 through the uORB message pool
- **px4_simulink_App** cannot access the motor at the same time as the PX4 controller, otherwise there will be conflicts, so the PX4 official output needs to be blocked



Note: Another feasible method is to modify the PX4 module startup script file "Firmware\ROMFS\px4fmu_common\init.d\rcS" and comment out the module you want to block

2.3 How to replace PX4 official filter, mixer and other APPs with Simulink controller

- The generated Simulink code can also be used to replace some native modules (sensors, filters, attitude controllers, etc.) of the PX4 control software as shown on the right, but the PX4 firmware code needs to be manually modified to block the output interface of the original module. For example, if you want to use Simulink to implement a filter module (input sensor data, output state filter data) to replace the original PX4 filter, you need to manually block the "**Position & Attitude Estimator**" filter module in the picture, and then publish the filtered attitude data (corresponding to the uORB message named **vehicle_attitude**) to the uORB message pool. The specific procedure is as follows:
- Open the "**Firmware\src\modules\ekf2\ekf2_main.cpp**" file (or **ekf2.cpp** file in PX4-1.11, corresponding code for the extended Kalman filter module);
- Block out the sending code related to the "**ORB_ID(vehicle_attitude)**" message. For example, search for the code line with the keyword "**_att_pub**" and find the sending code line with "**publish**" and "**att**" in it, and replace it with "**UNUSED(att);**". Here **UNUSED** is used to prevent the compiler from warning about unused variables.
- Write the attitude filter in Simulink, and use the uORB Write module to send the **vehicle_attitude** message to replace the attitude filter function.

```
//_att_pub.publish(att);
UNUSED(att);
```

2. Platform Framework

2.4 Simulink automatic code generation cor

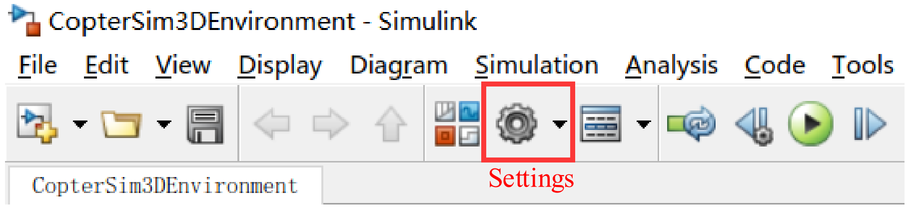
Open any **.slx** demo file

1. Entering Simulink **setting** page (R2019b and above should go to **MODELING** tab)

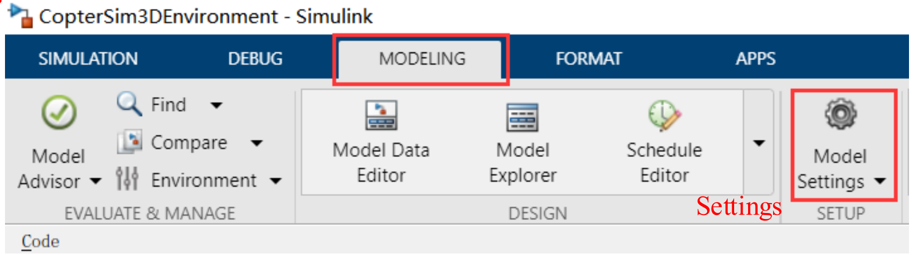
2. After select “**Hardware board**” setting to “**Pixhawk PX4**”, it will automatically finish all code generation configuration

3. Allows customize “**task priority**”

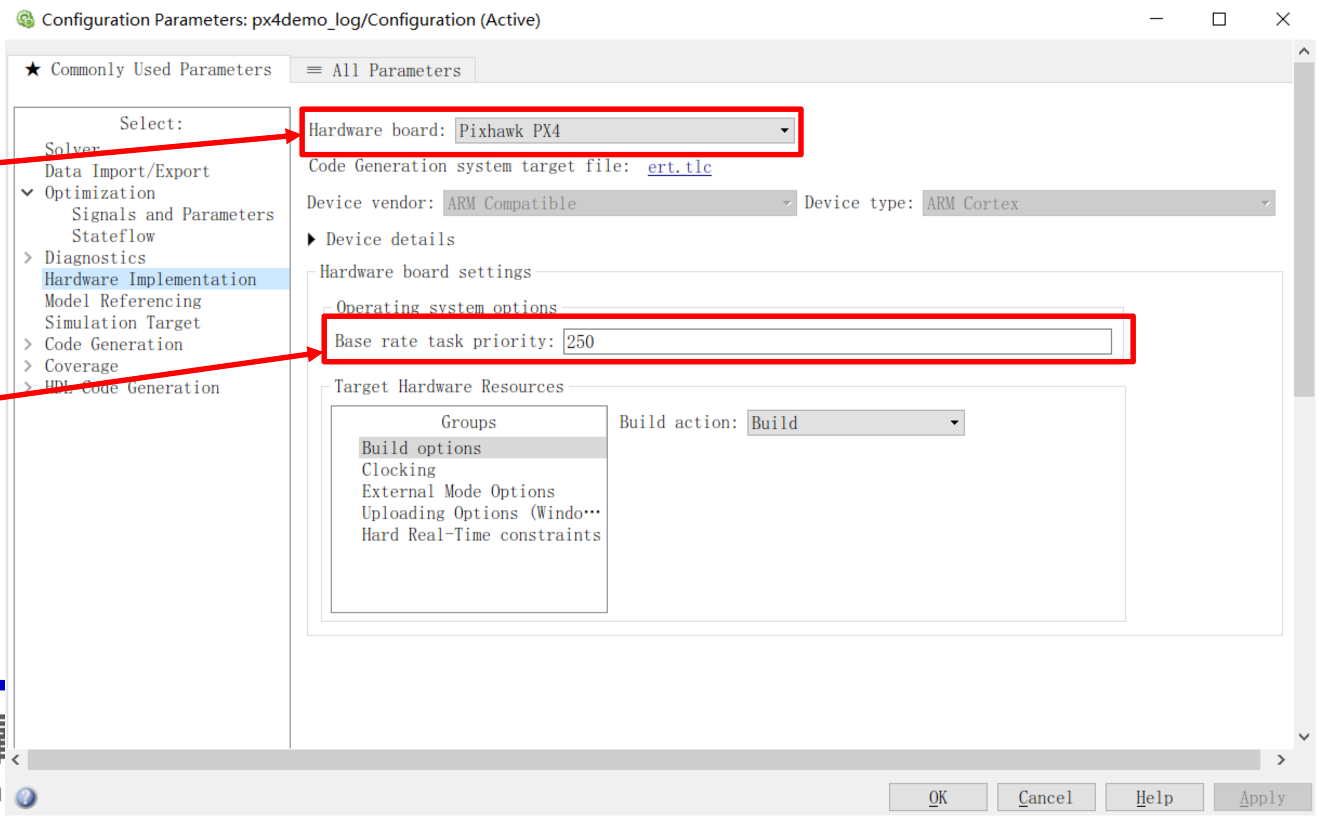
4. Setting compile options



(a) Simulink “Settings” button on MATLAB 2017b~2019a



(b) Simulink “Settings” button on MATLAB 2019b and above



2. Platform Framework

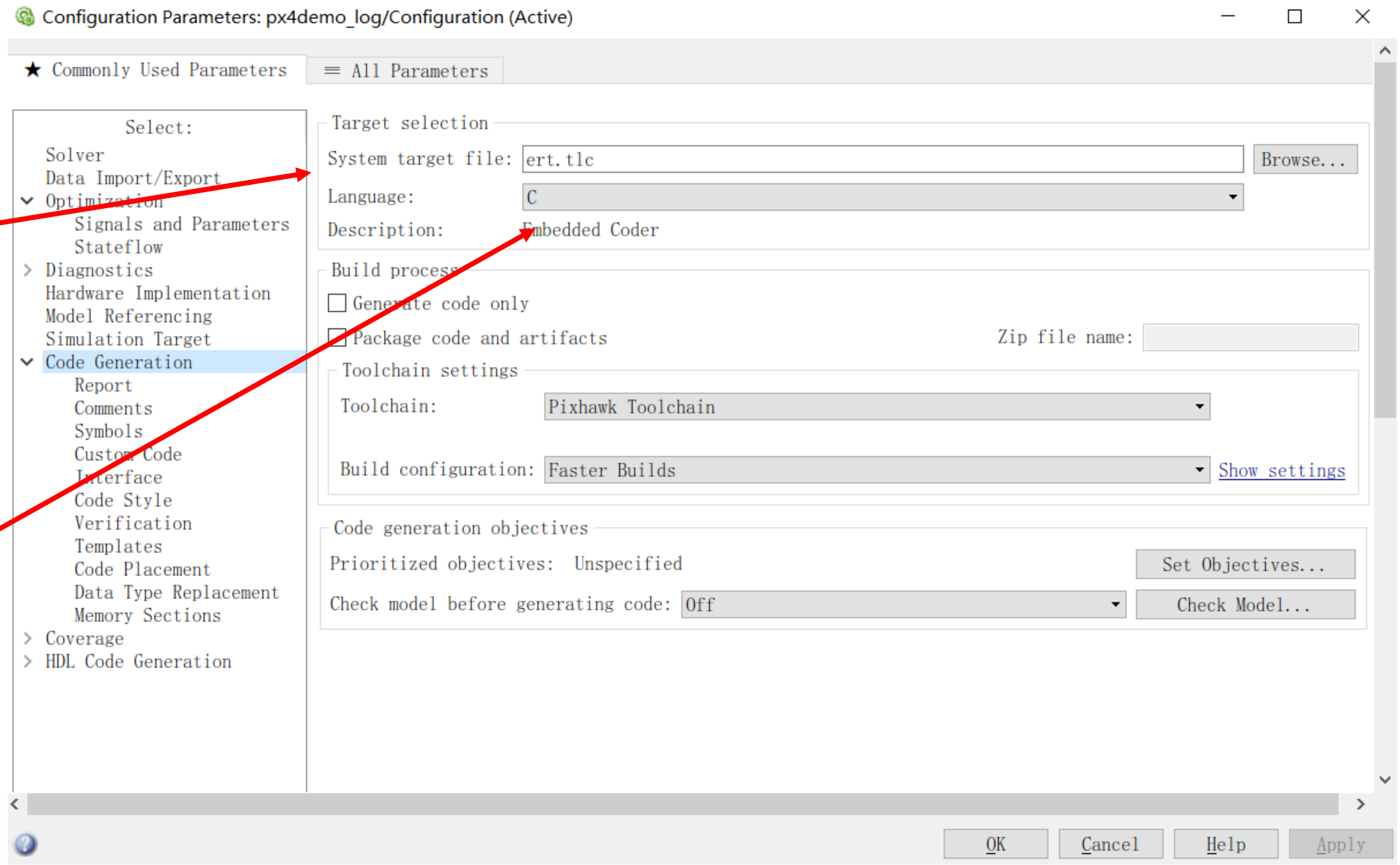
2.4 Simulink automatic code generation setting

The configuration of code generation is mainly on the Code Generation page

1. "**System Target File**" corresponds to the operating platform of the generated code, which is the code template

2. "**Language**" corresponding to the generated language, C or C++ can be selected

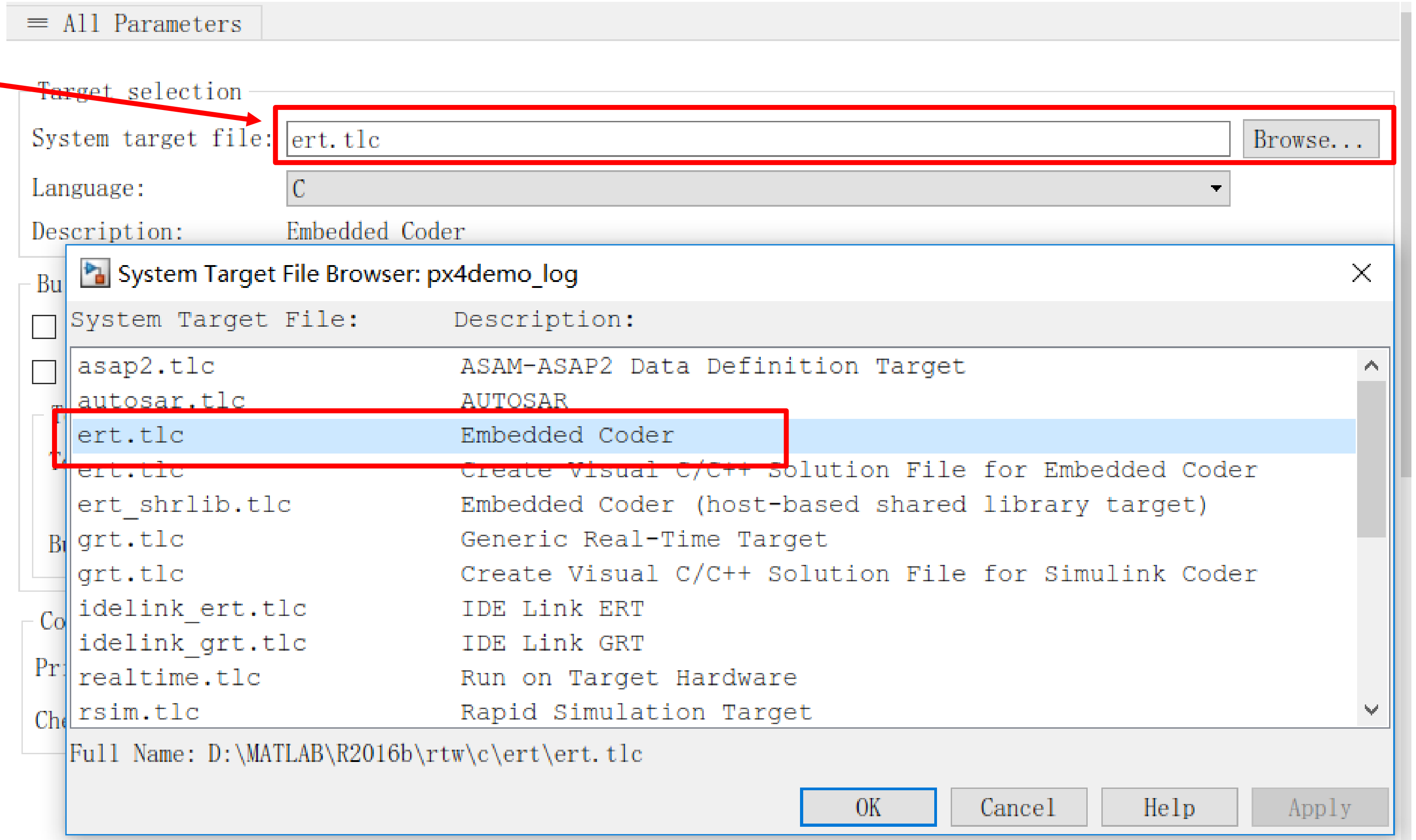
3. There are some compiler setting options



2. Platform Framework

2.4 Simulink automatic code generation setting

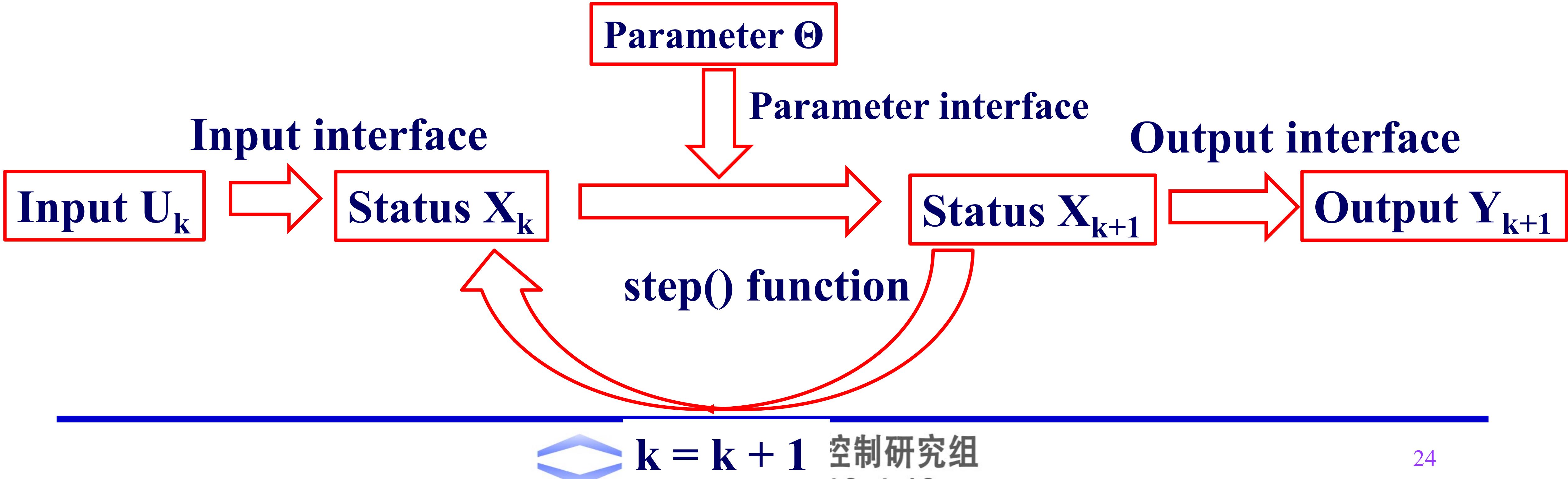
- **ert.tlc** is the most commonly used method of code generation
- The main program is finally a **step()** function
- You need to use interrupts or timers in the embedded system by yourself to call according to the set step
- **For example**: the simulation step is 0.001s, the embedded interrupt is the same



2. Platform Framework

2.4 Simulink automatic code generation setting

Schematic diagram of embedded system operation generated by **ert.tlc**. The **step()** function can choose approximate integration methods such as **Runge-Kutta** method and Euler method; the parameter interface allows real-time change of model parameters; the input and output interface allows other programs to call.

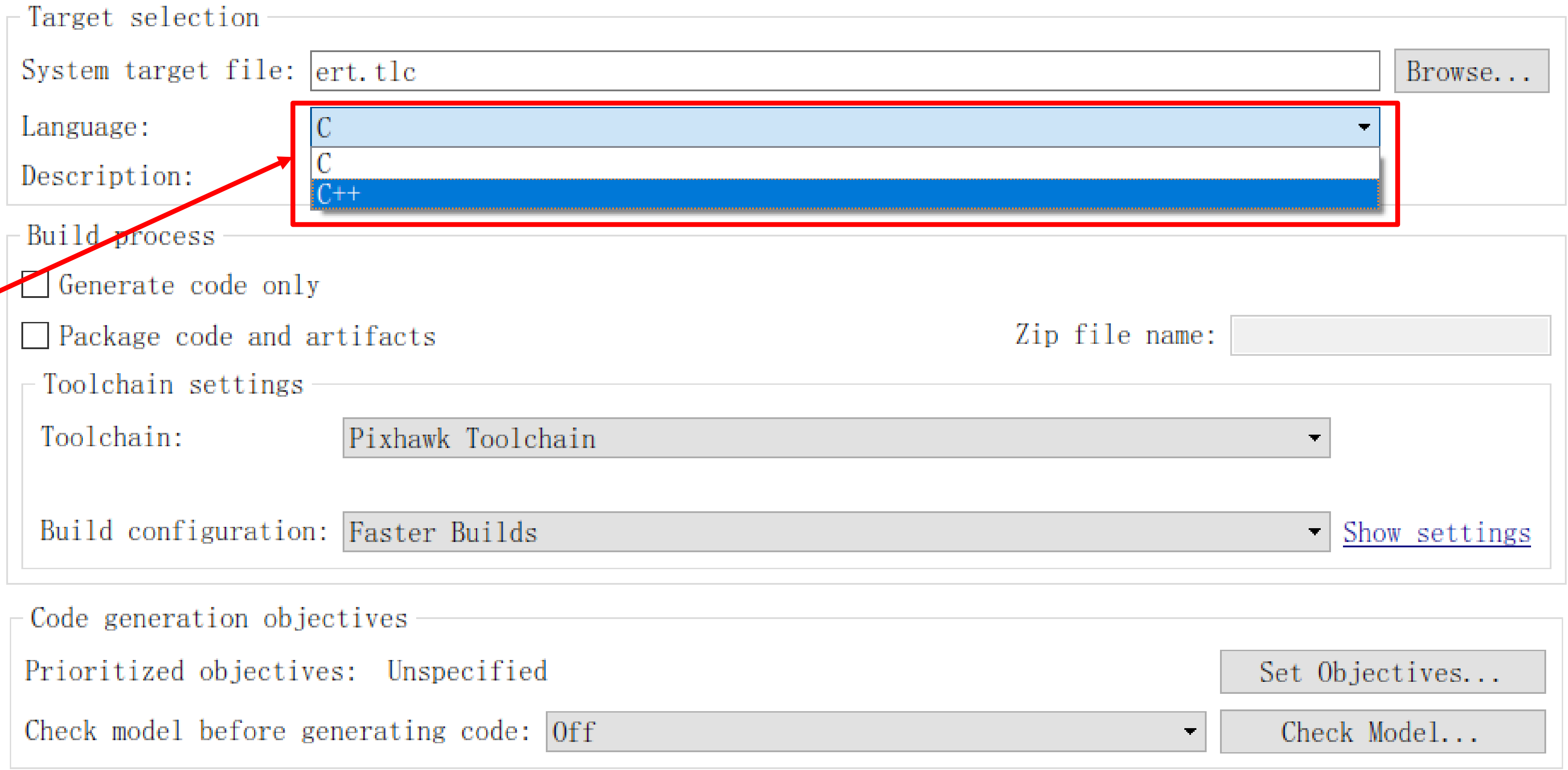


2. Platform Framework

2.4 Simulink automatic code generation setting

Difference between C/C++

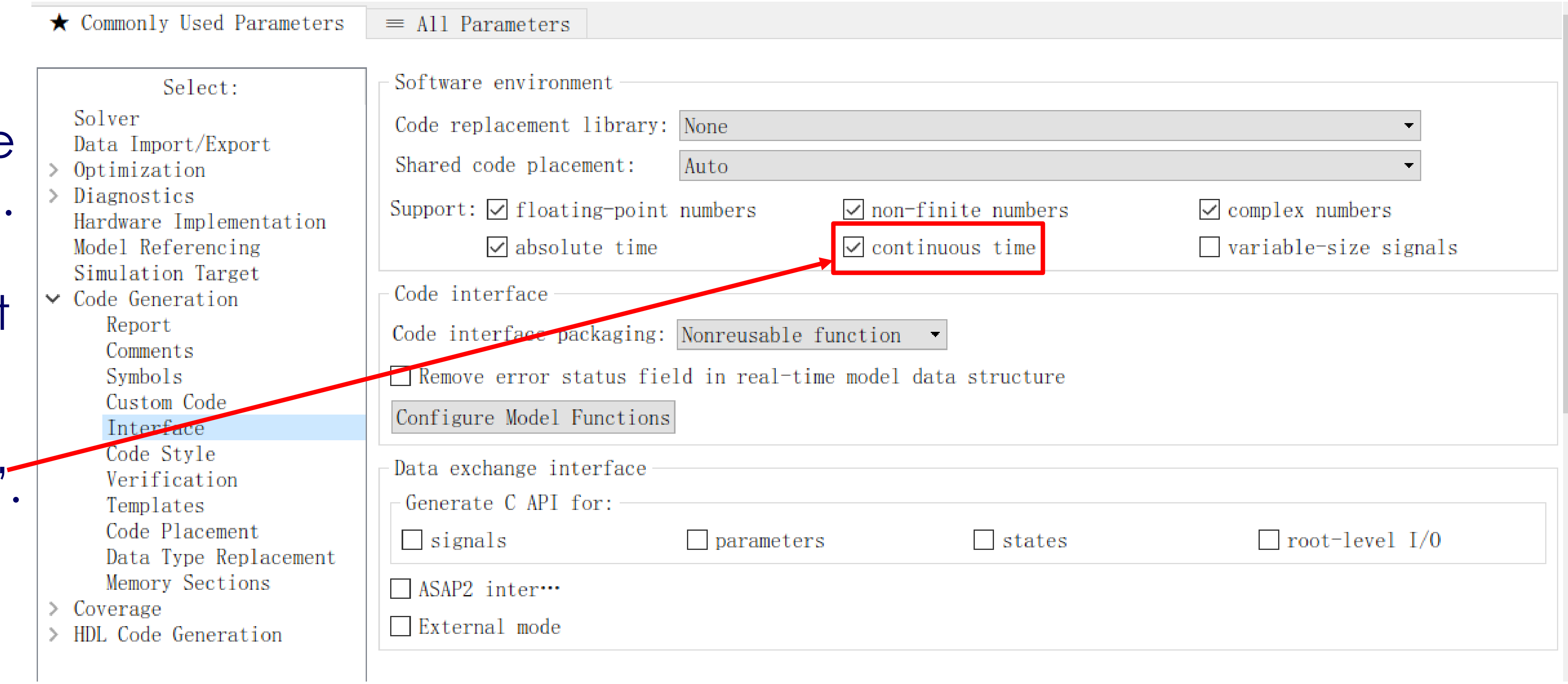
- C generated code easier, but weaker scalability
- C++ can encapsulate the entire program as a class
- Facilitate later inheritance and expansion



2. Platform Framework

2.4 Simulink automatic code generation setting

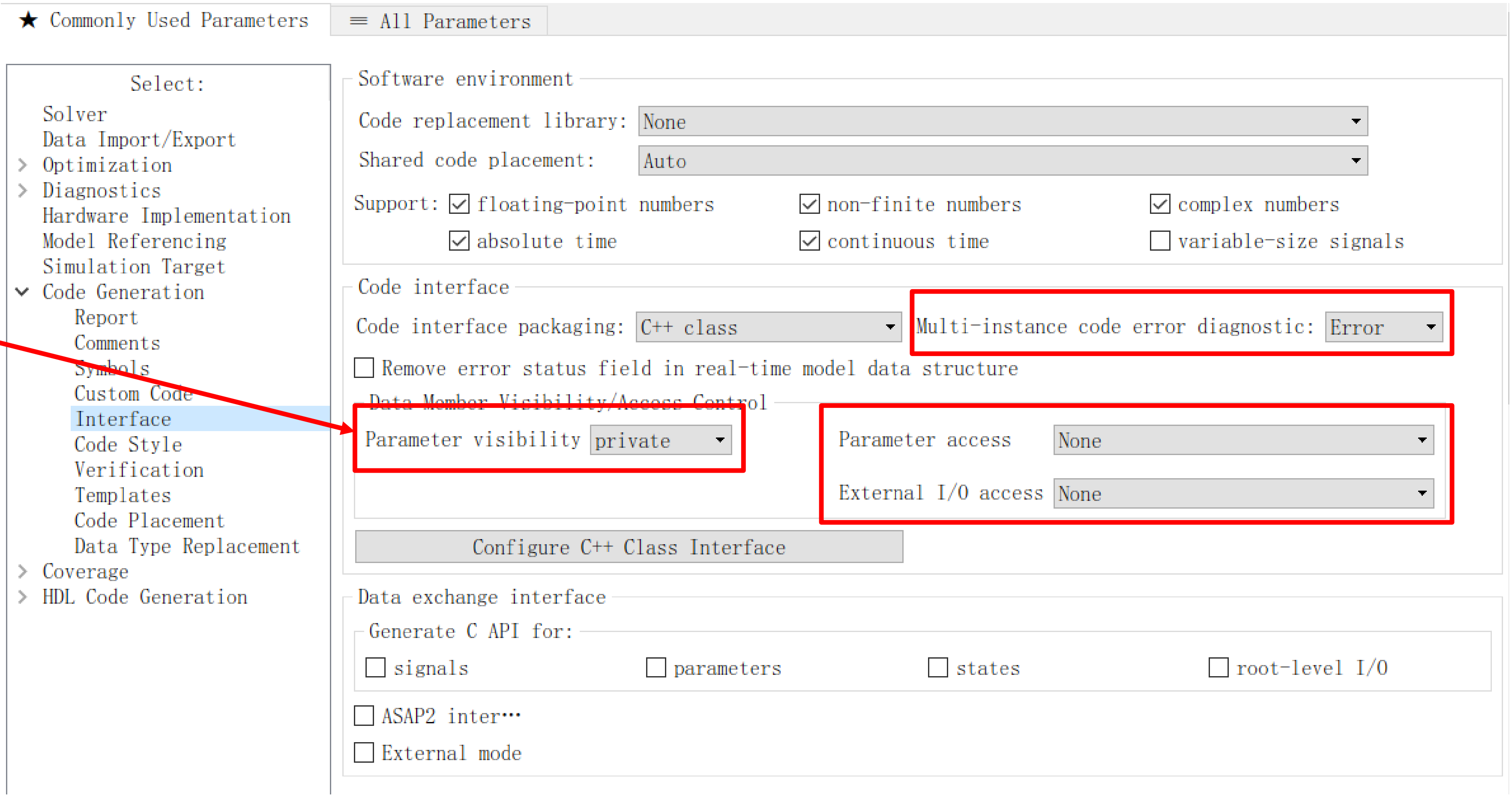
- Interface corresponds to C
- Allows to set some simulate methods. For example, whether it support plural, whether it support "**continuous time**".
- Also allows to define output interface



2. Platform Framework

2.4 Simulink automatic code generation setting

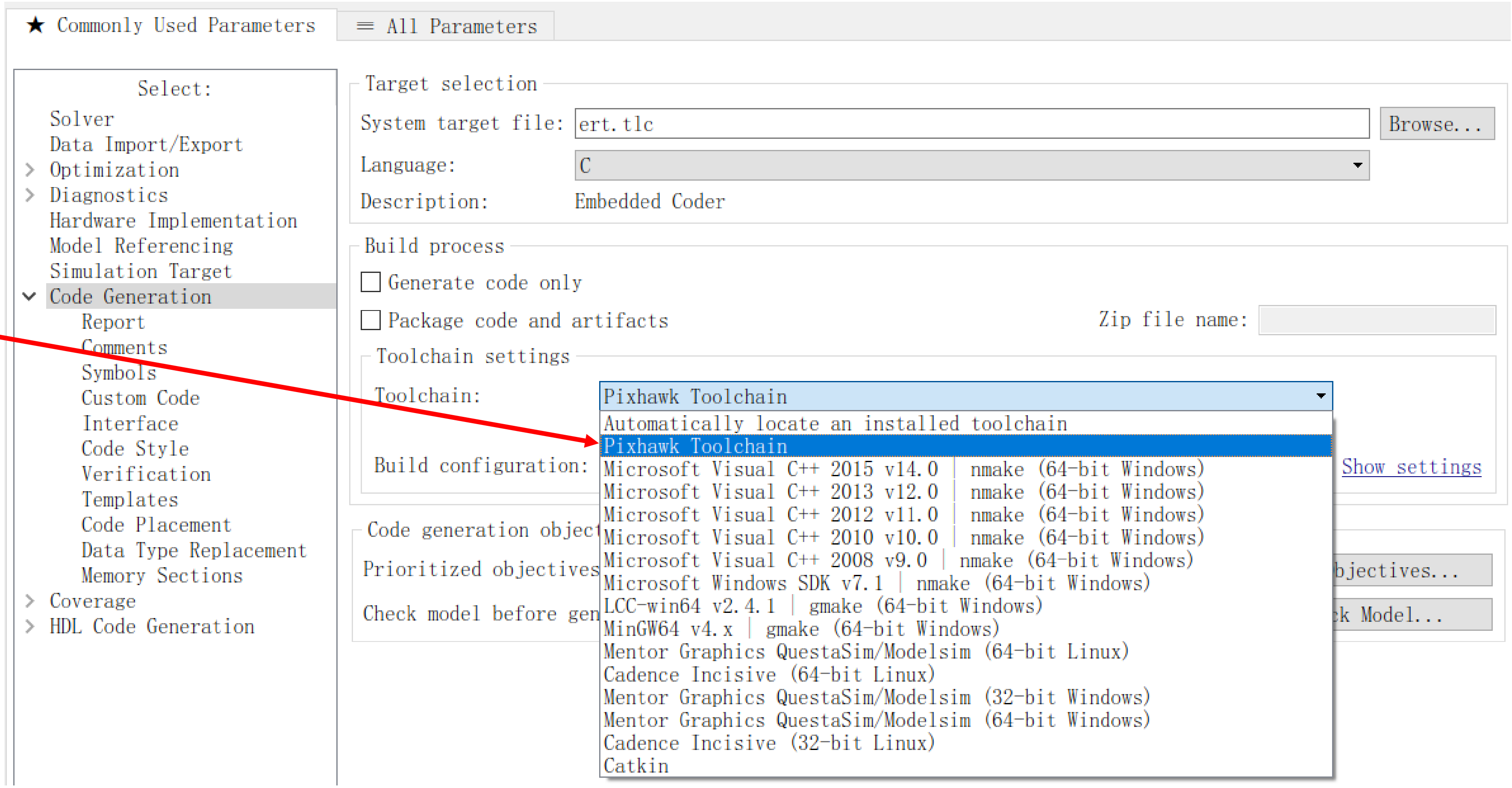
- Interface correspond to C++
- You can also set whether the "**Parameter visibility**" is "**Public**"
- Whether to support **multi-instance**
- And whether to generate various external interfaces



2. Platform Framework

2.4 Simulink automatic code generation setting

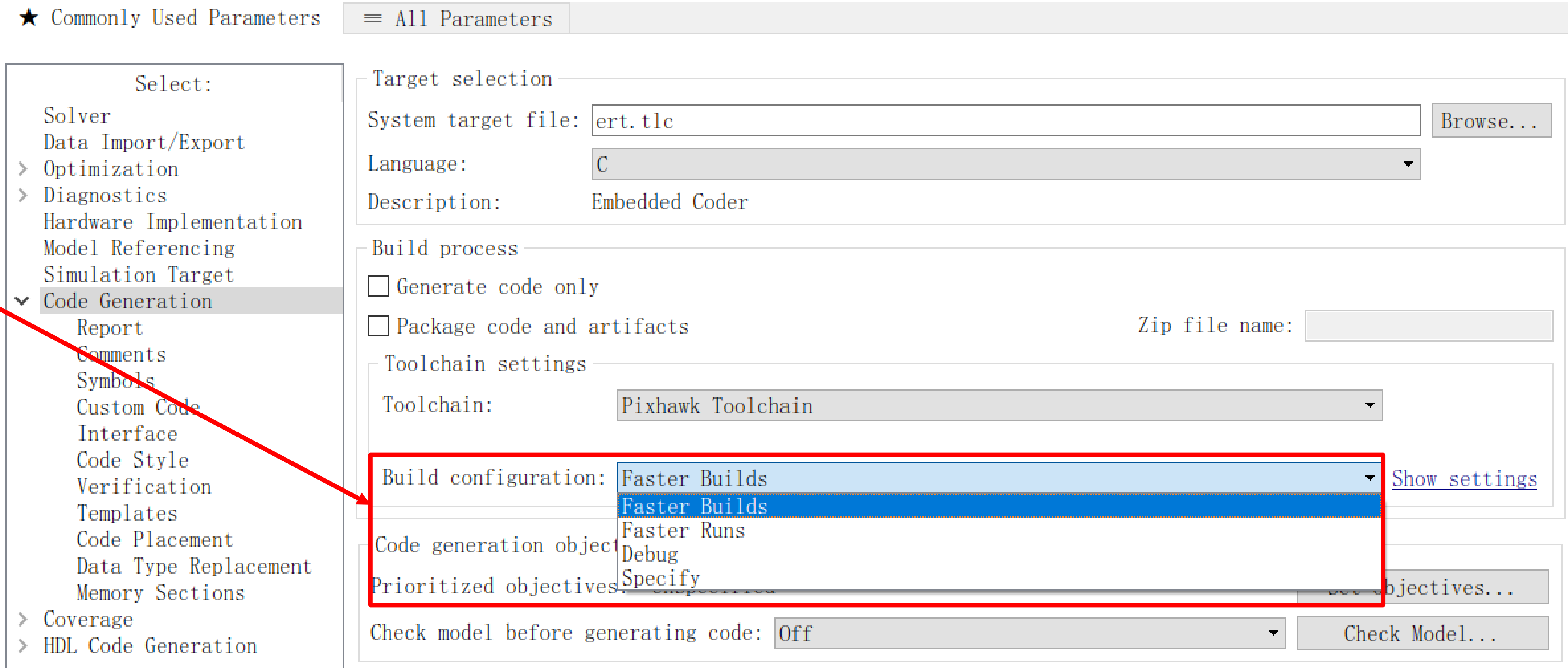
- Select target compilation toolchain
- Choose "**Pixhawk Toolchain**" here
- also allows choosing Visual Studio C++ or other compiler



2. Platform Framework

2.4 Simulink automatic code generation setting

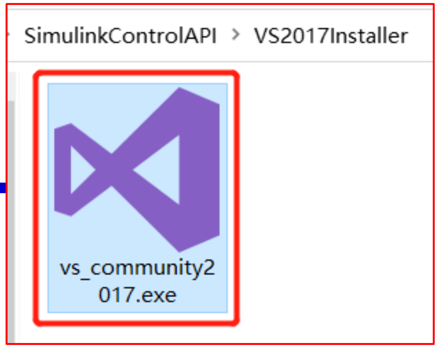
- Choose build configuration
- **Faster Builds** get a smaller amount of code, which makes compilation faster
- **Faster Runs** will optimize the code and compile to ensure faster running efficiency



Content

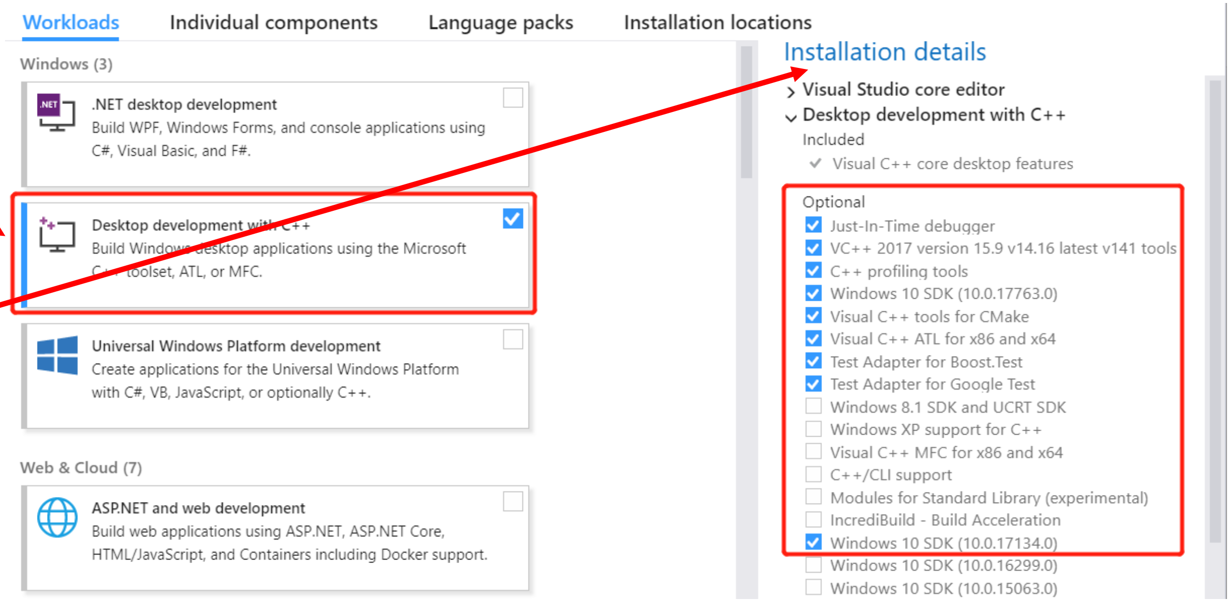
3. Advanced Example

Note: Visual Studio compiler is needed in many examples in this section, please install in advance



3.0 Install Visual Studio 2017 (other versions can also be used, only if MATLAB can recognize it)

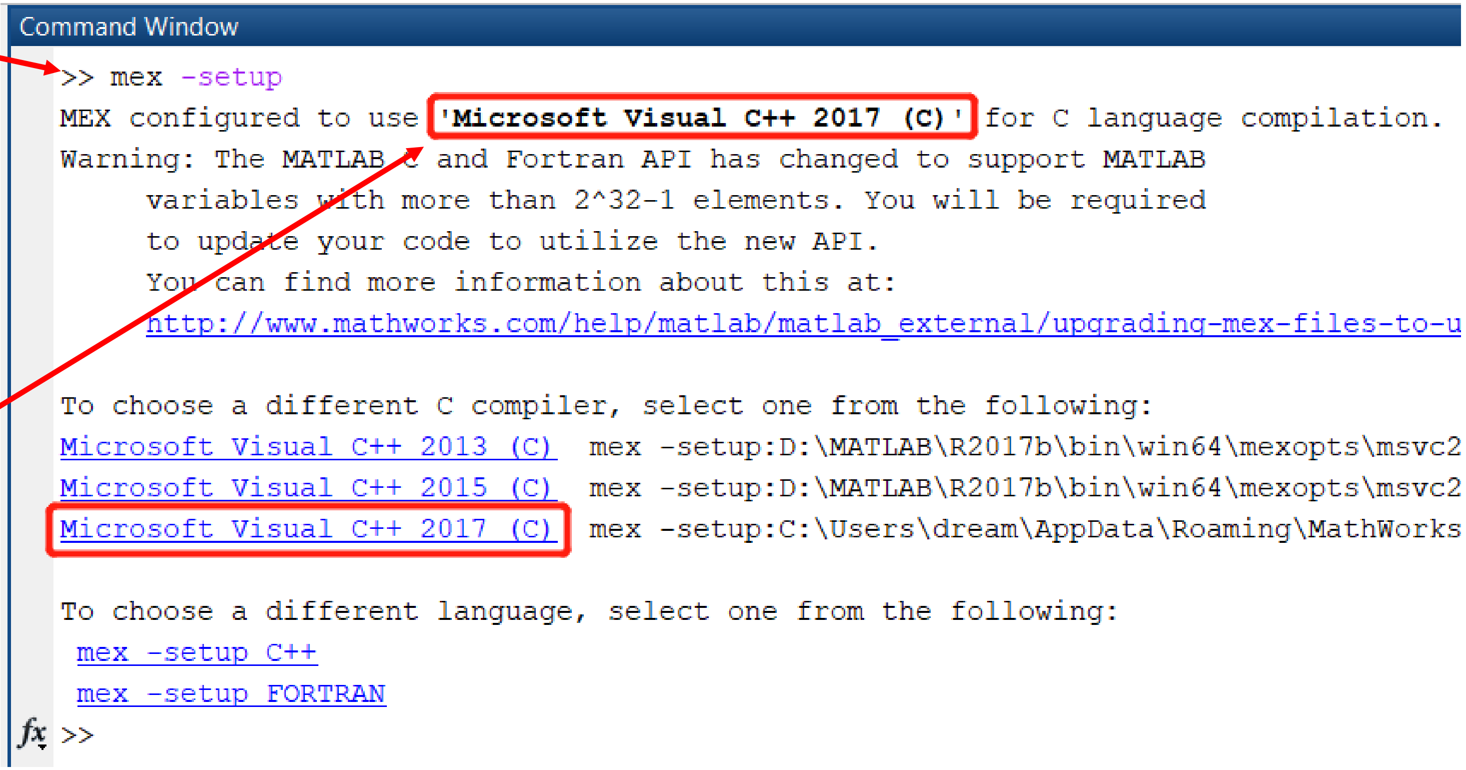
- The Visual Studio (VS) compiler is needed in many places in subsequent courses, such as MATLAB
- The use of S-Function Builder module, Simulink automatically generates C/C++ model code, etc.
- It is recommended to install Visual Studio 2017. The online installation steps (internet required) are as follows:
- Double-click "**RflySimAPIs\SimulinkControlAPI\VS2017Installer\vs_community2017.exe**"
- This course content only needs to check the "**Desktop development with C++**" on the right.
- Note: If you want to use Unreal Engine 4 (UE4)'s C++ plugin development in the future, you can also check the latest Window 10 SDK in the "**Installation details**" on the right; then click the "**Individual components**" tab and check **.NET 4.7.2** (or the latest version) and the corresponding pack package. Click install again.



3. Advanced Examples

3.0 Configure C++ Compiler for MATLAB

- Enter the command "**mex -setup**" in the MATLAB command line window
- Generally speaking, the VS 2017 compiler will be automatically recognized and installed. As shown in the right figure, "MEX configured to use 'Microsoft Visual C++ 2017' for", indicating that the installation is correct
- This page can also switch to other compilers such as Visual Studio 2015

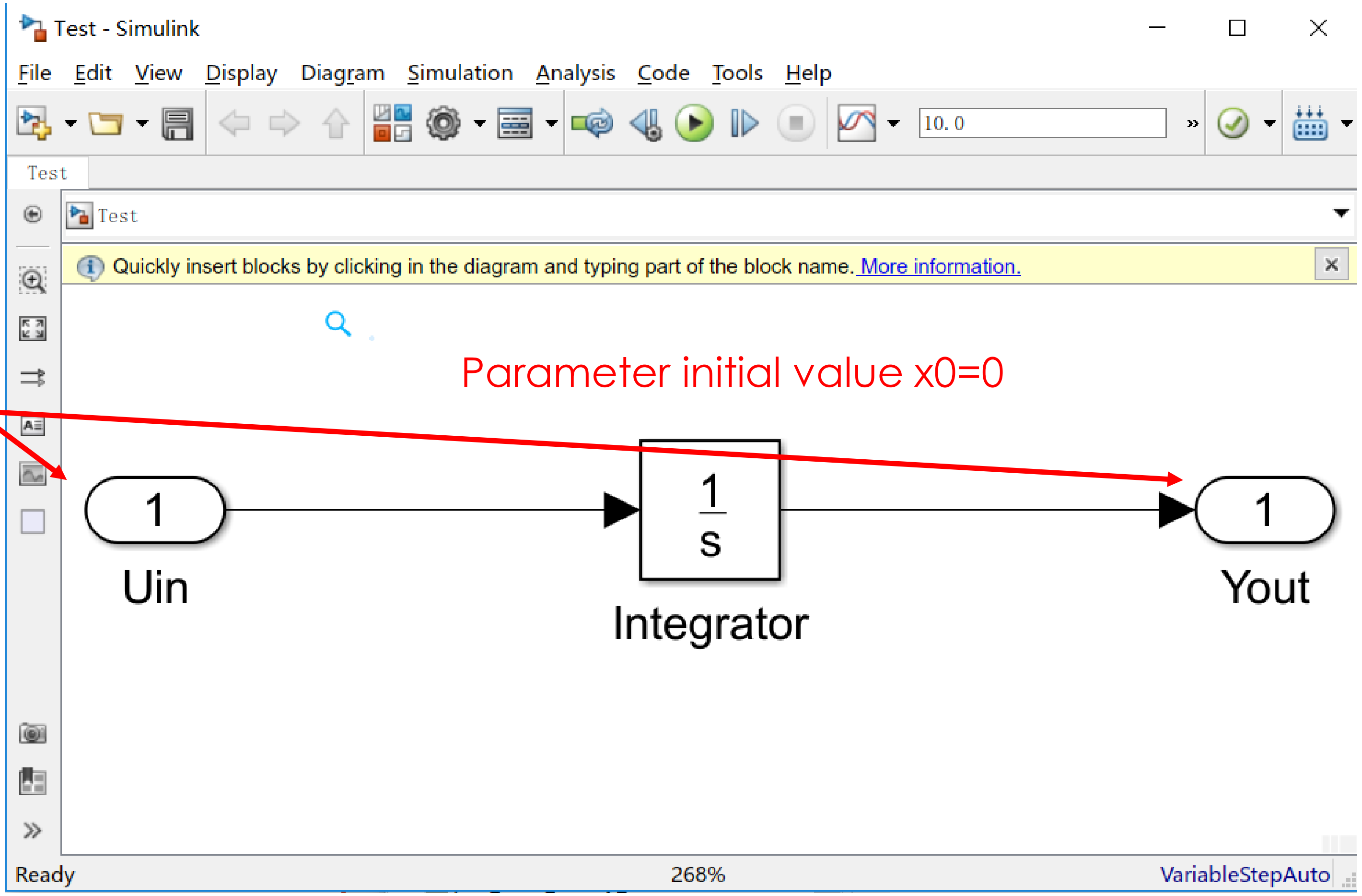


3. Advanced Examp

*Note: This example requires the VS compiler, please follow the steps in **Section 1.8** of "**RflySim_Lesson_01_Introduction.pdf**" to install and configure*

3.1 Self-generated C/C++ code examples

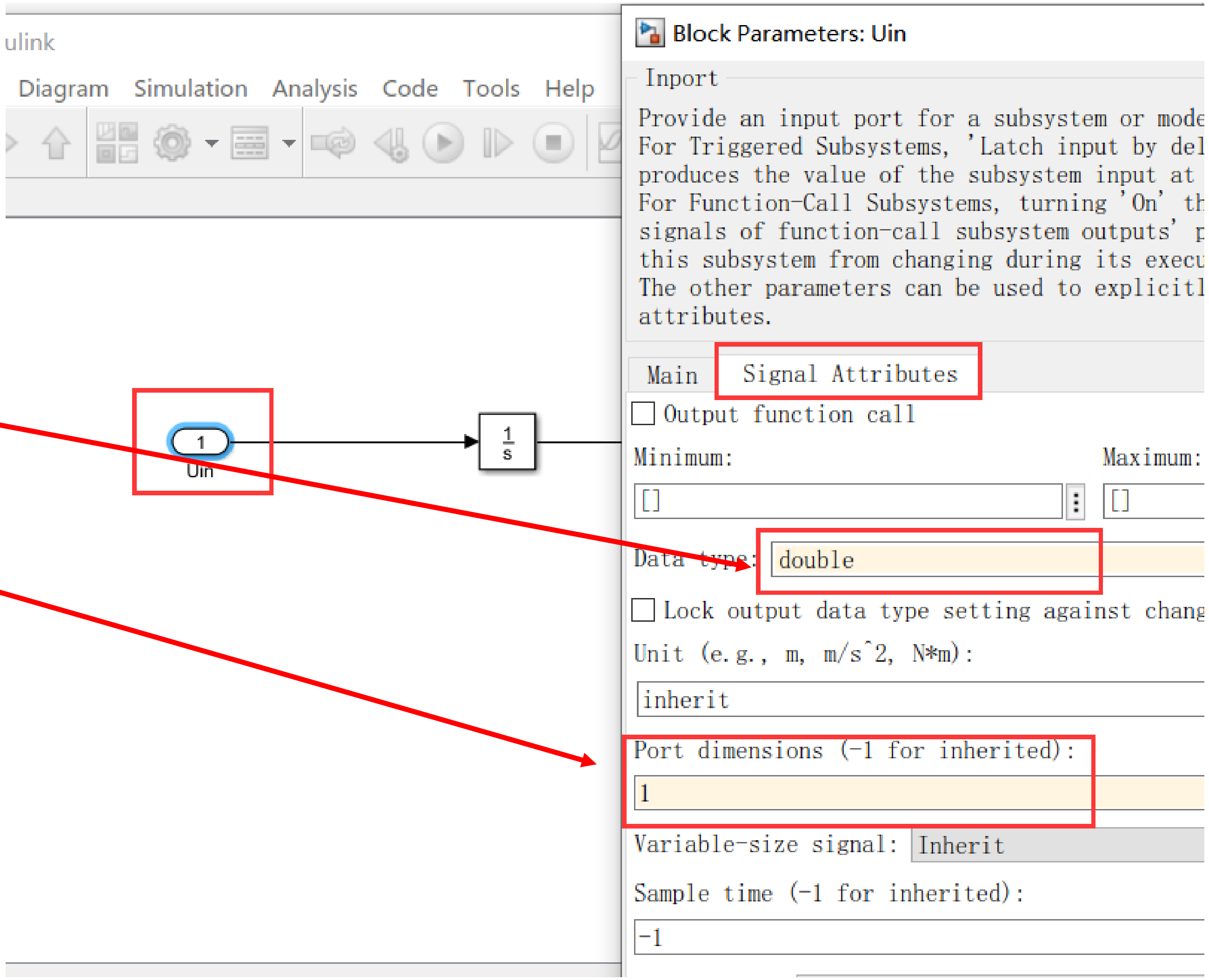
- Create a Simulink model according to the right picture (save it to name CodeGenExample.slx)
- Name the input as "**Uin**"
- Name the output as "**Yout**"
- The initial value of the integral is defined as "**X0**"
- The names of the above variables need to be remembered, they correspond to the variable names of the generated C++ code



3. Advanced Examples

3.1 Self-generated C/C++ code examples

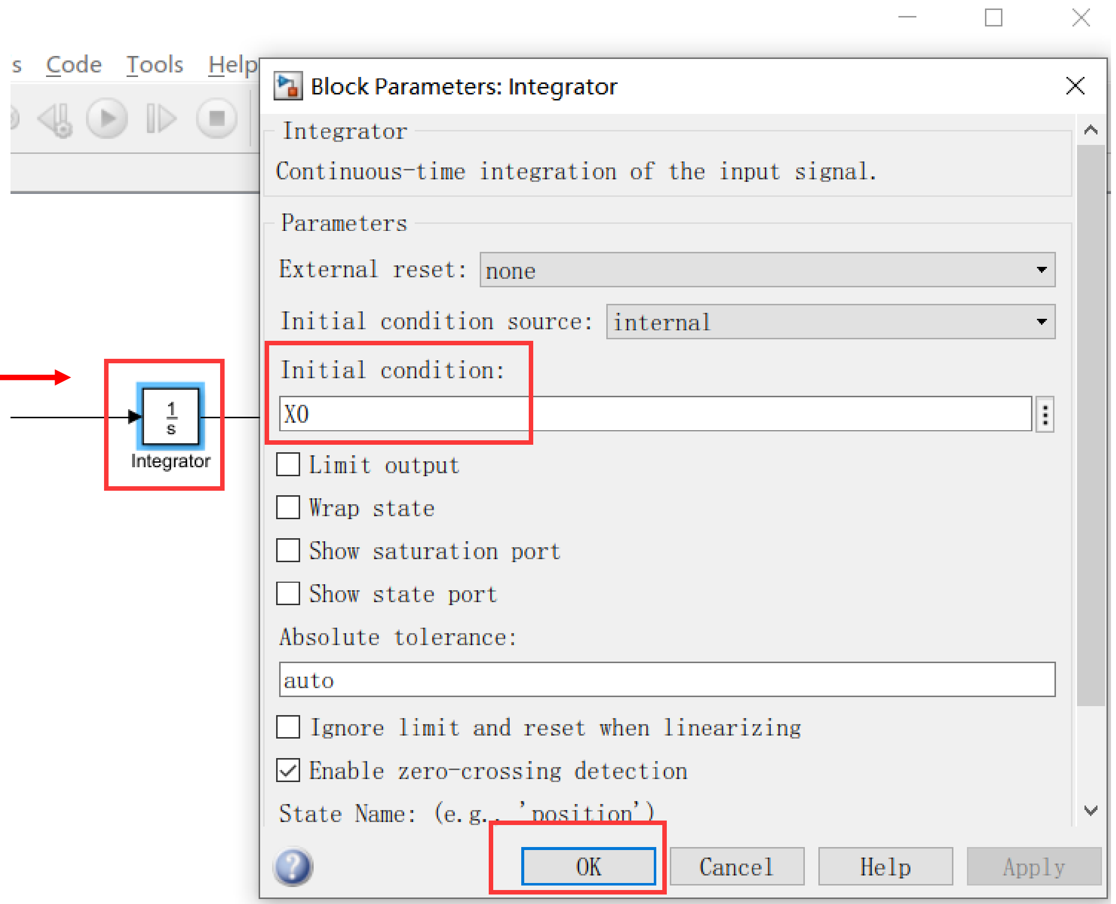
- Double-click the **Uin** icon to enter the parameter setting page
- Enter the “**Signal Attributes**” page
- Set the data type "**Data Type**" to "**double**"
- Set the data dimension "**Port dimensions**" to "**1**"
- In this way, we define the data format of the input interface after the code is generated.
- Similarly, set the "**Uout**" output interface



3. Advanced Examples

3.1 Self-generated C/C++ code examples

- Double-click the integrator module to enter the “**Block Parameter**” page.
- Set a named parameter “**X0**”
- This will be a demo to show how the Simulink variable is shown in the generated C/C++ code
- Then, we can access this variable in our project

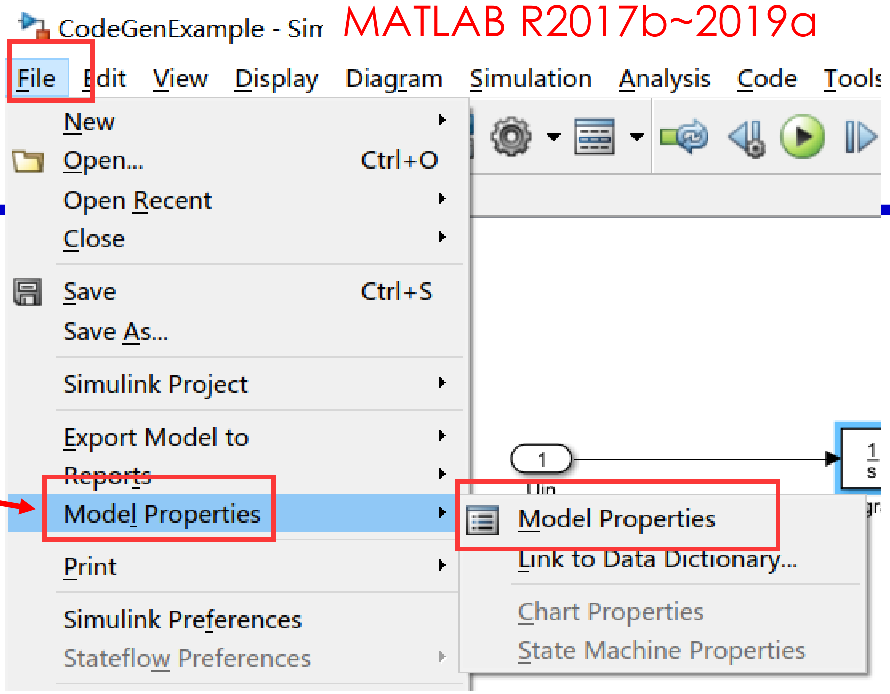


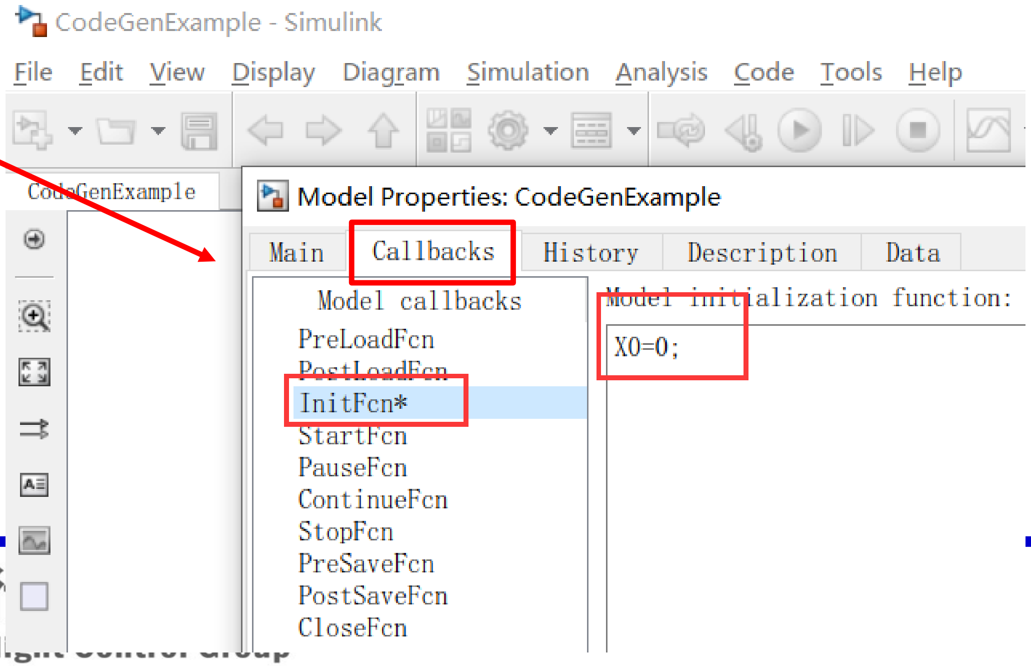
3. Advanced Examples

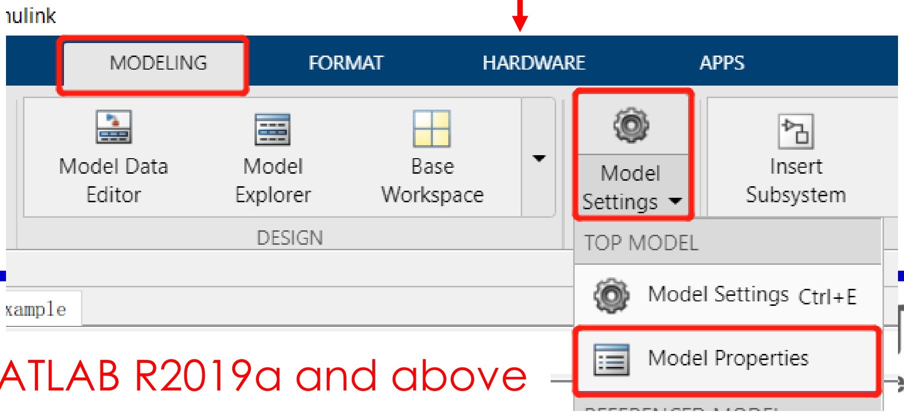
3.1 Self-generated C/C++ code examples

- Open the Simulink menu bar – "**File – Model Property – Model Property**" page for MATLAB 2017b~2019a, and "**MODELING – Model Settings – Model Properties**" for MATLAB 2019b and above.
- Add the initialization script "**X0=0**" in the "**Callbacks - InitFcn**" tab
- Click the Simulink "**Run**" button to see if it can run correctly.

MATLAB R2017b~2019a





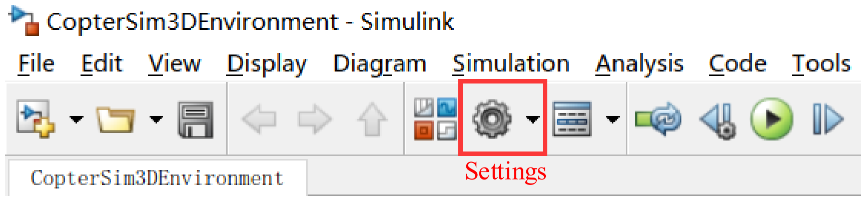


MATLAB R2019a and above

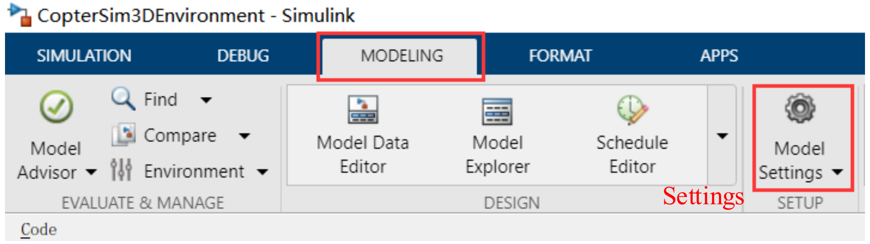
3. Advanced Examples

3.1 Self-generated C/C++ code examples

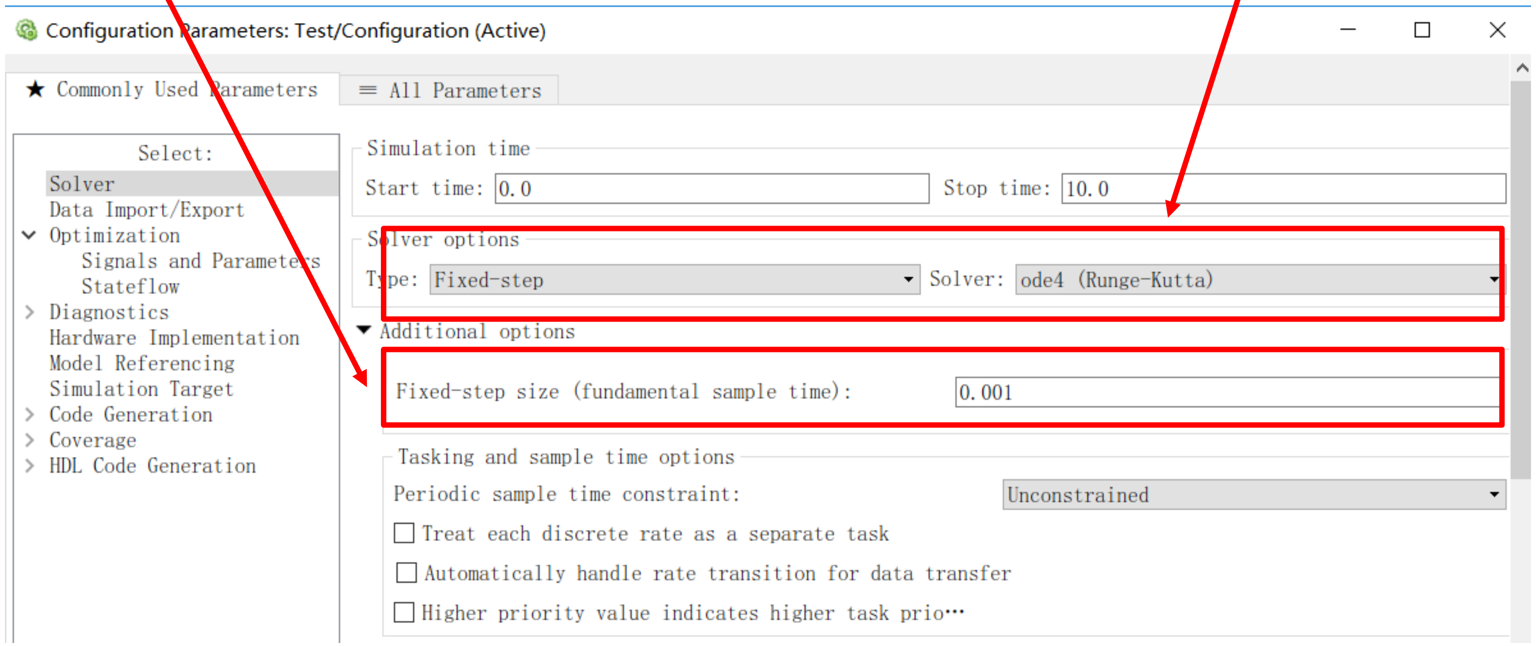
- Open the settings page, set the simulation to "**fixed-step**" size, "**ode4 (Runge-Kutta)**" method solver, step size is "**0.001**"s (or other value based on the actual situation)



(a) Simulink "Settings" button on MATLAB 2017b~2019a



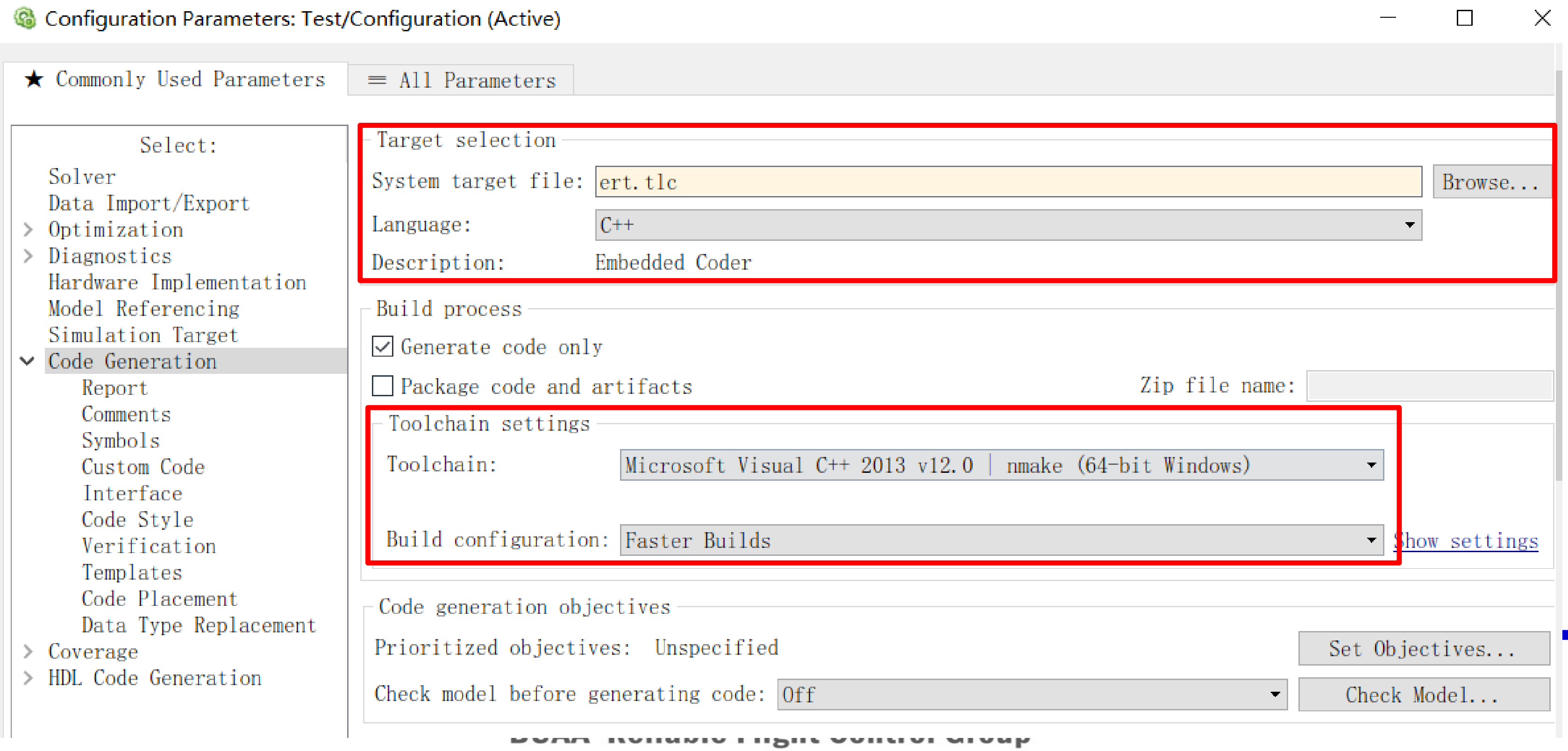
(b) Simulink "Settings" button on MATLAB 2019b and above



3. Advanced Examples

3.1 Self-generated C/C++ code examples

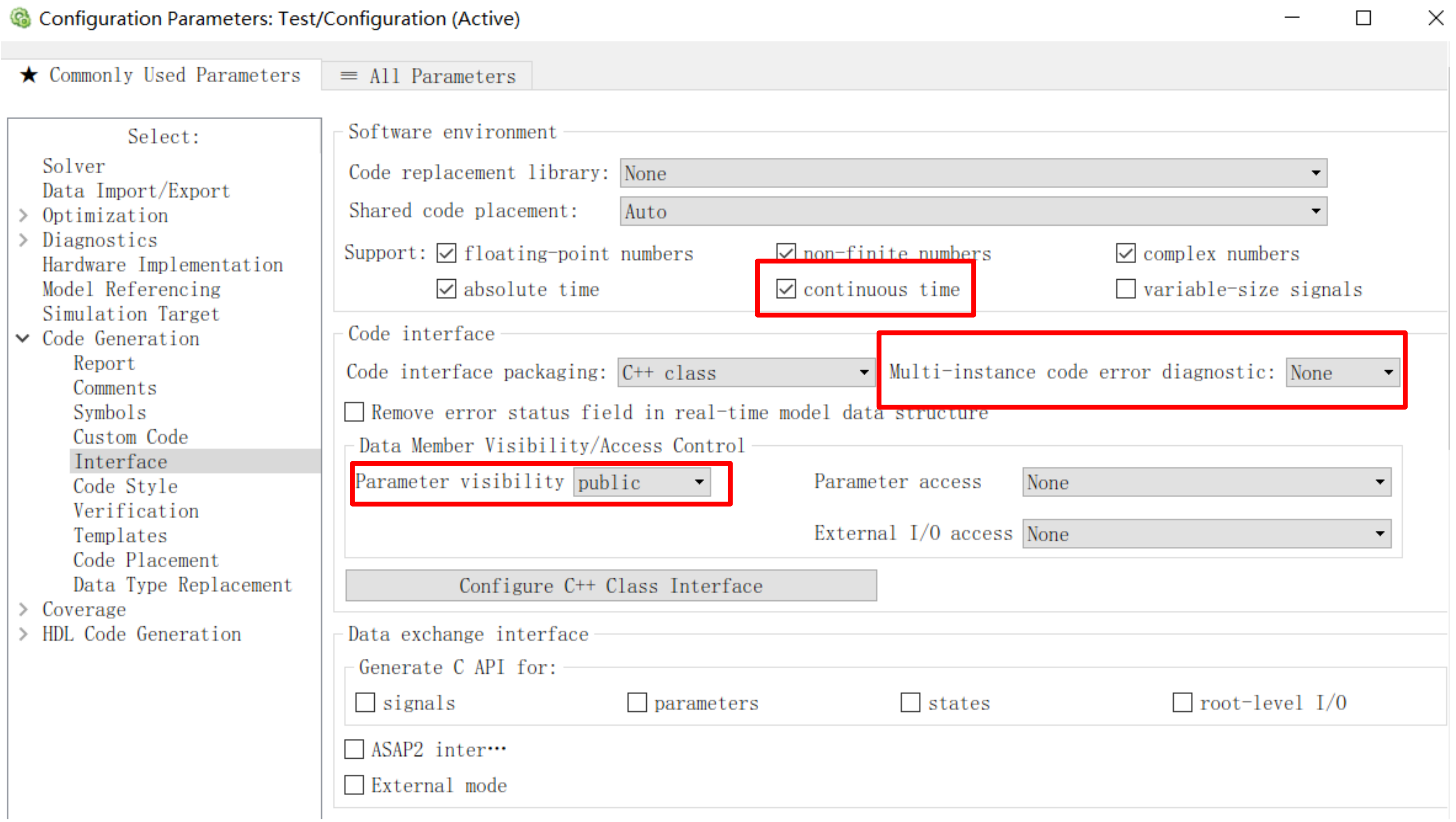
- Choose "**ert.tlc**" as the code generation method, which can be used for Windows, Linux and various embedded platforms; choose "**C++**" as the language, which is convenient to call the generated code through inheritance; choose "**Visual Studio C++ ****" as the Toolchain



3. Advanced Examples

3.1 Self-generated C/C++ code examples

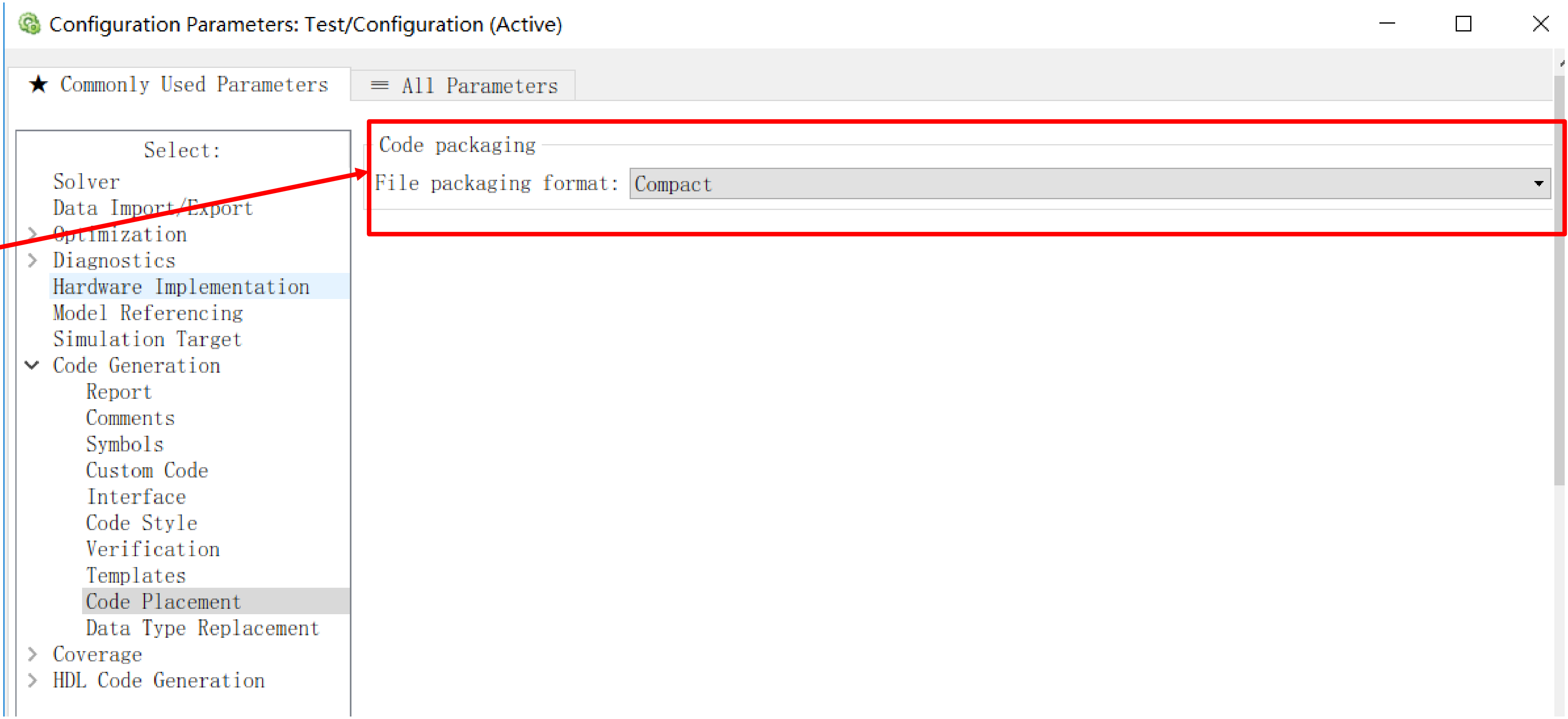
- Because it contains a continuous module (integration module), you need to check "**continuous time**", otherwise the compilation will report an error.
- In addition, the "**parameter visibility**" is set to "**public**", and the parameter structure is a public variable for easy access in a class



3. Advanced Examples

3.1 Self-generated C/C++ code examples

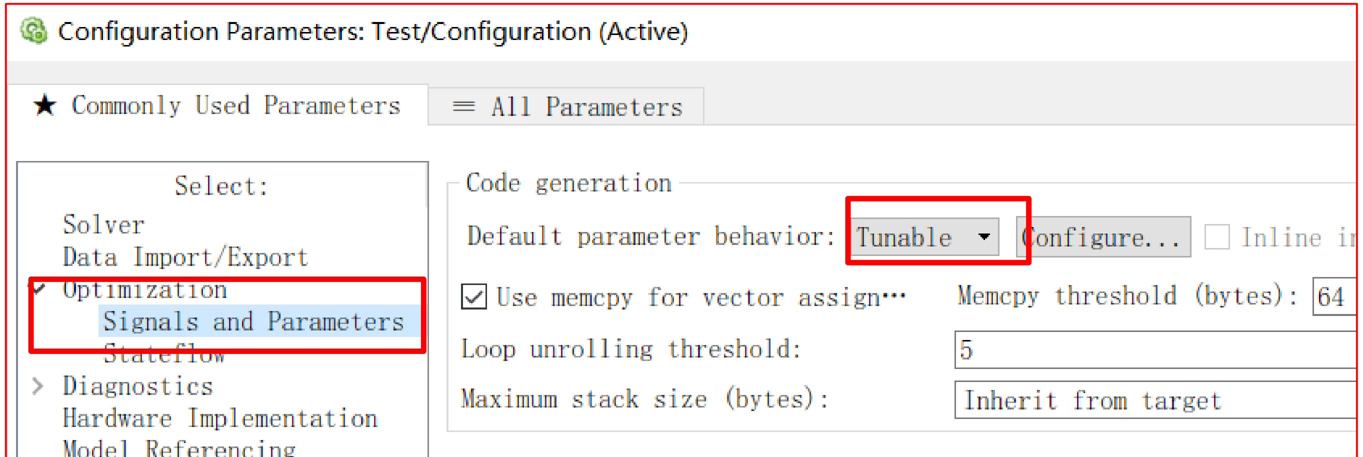
Set the "**File packaging format**" to "**compact**" on the "**Code packaging**" page, and try to avoid generating redundant files to make the code the most readable



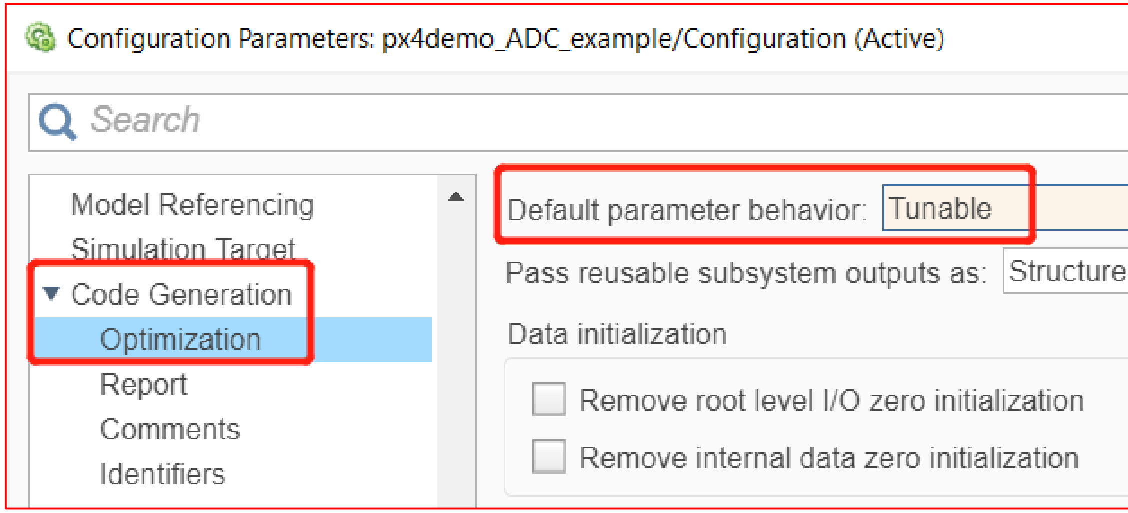
3. Advanced Examples

3.1 Self-generated C/C++ code examples

- Setting the parameter to "**Tunable**" allows us to modify the parameter at runtime. **Note**: The inline form saves more memory, but it is inconvenient to access parameters, and it is not convenient to implement real-time parameter modification or model fault injection.



Parameter behavior option on MATLAB R2017b~2019a

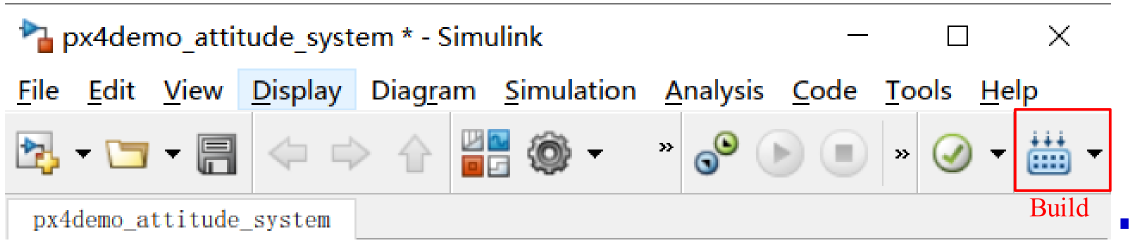


Parameter behavior option on MATLAB R2019b and above

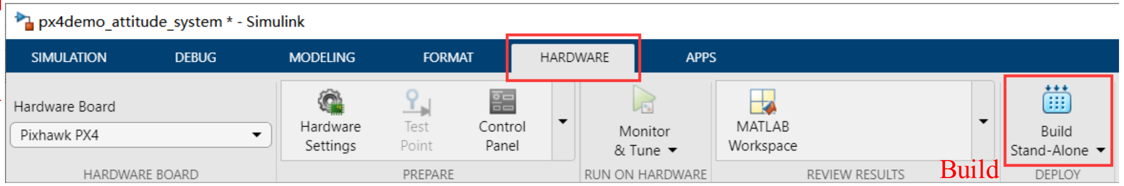
3. Advanced Examples

3.1 Self-generated C/C++ code examp

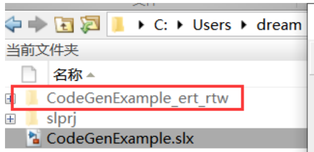
- Click the “**Build**” (compile) button to generate code
- Generate three files:
- The file “**ert_main.cpp**” contains an example of calling the generated code
- The two files ******.cpp** and ******.h** contain a C++ class generated by the Simulink project just now
- Note: for **MATLAB R2019b** and above versions, you should click “**APPS - CODE GENERATION - Simulink Coder**” to observe the “**HARDWARE**” tab

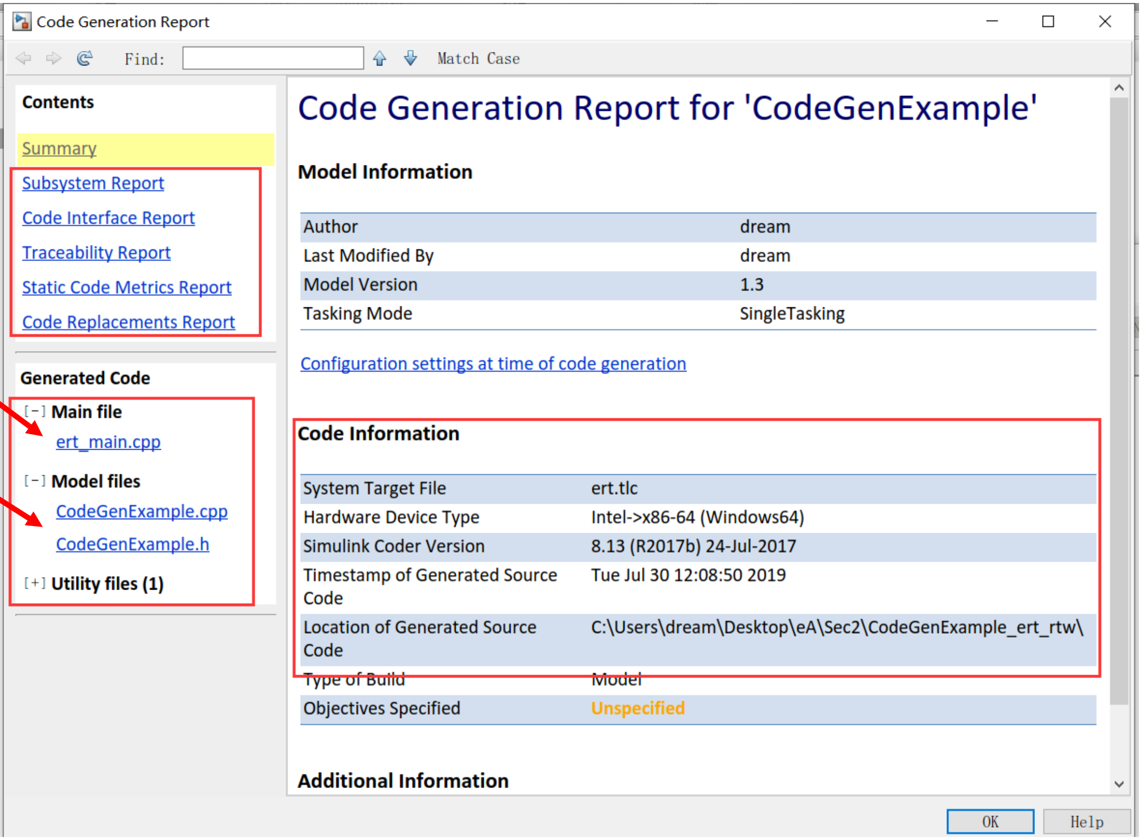


(a) Simulink “Build” button on MATLAB 2017b~2019a



(b) Simulink “Build” button on MATLAB 2019b and above





3. Advanced Examples

3.1 Self-generated C/C++ code exampl

- The picture on the right shows the generated C++ class (in the ******.h** file): ******ModelClass**
- ******_P** is the parameter structure
- ******_U** is the input structure
- ******_Y** is the output structure
- **step()** is a single step update function
- **initializie()** is the initialization function
- **terminate()** is the termination function

```
// Parameters (auto storage)
struct P_CodeGenExample_T_ {
  real_T X0;                          // Variable: X0
                                      //  Referenced by: '<Root>/Integrator'

};

// Parameters (auto storage)
typedef struct P_CodeGenExample_T_ P_CodeGenExample_T;
```

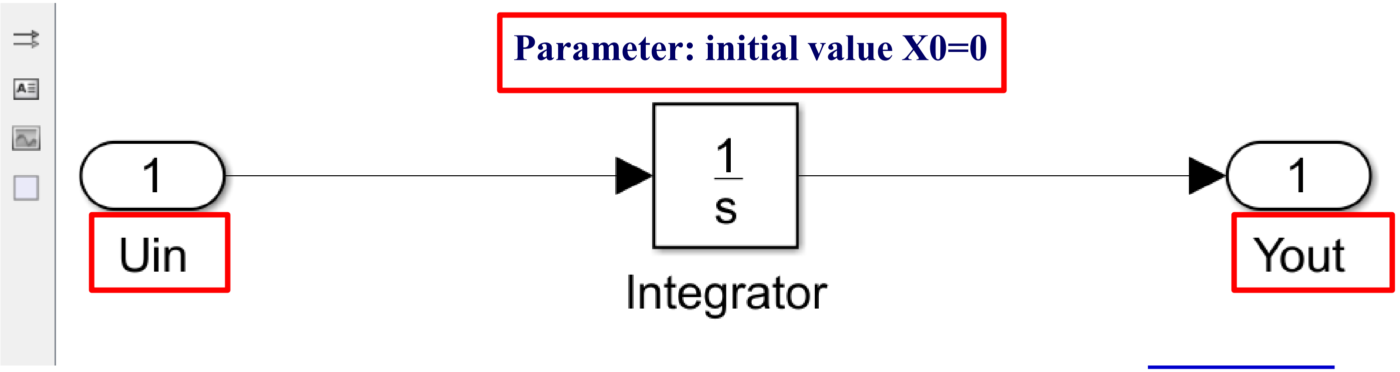
```
// Class declaration for model CodeGenExample
class CodeGenExampleModelClass {
  // public data and function members
 public:
  // Tunable parameters
  P_CodeGenExample_T CodeGenExample_P;

  // External inputs
  ExtU_CodeGenExample_T CodeGenExample_U;

  // External outputs
  ExtY_CodeGenExample_T CodeGenExample_Y;

  // model initialize function
  void initialize();

  // model step function
  void step();

  // model terminate function
  void terminate();

  // Constructor
  CodeGenExampleModelClass();

  // Destructor
  ~CodeGenExampleModelClass();

```

3. Advanced Examples

```
// External inputs (root inport signals with auto storage)
typedef struct {
  real_T Uin;                          // '<Root>/Uin'
} ExtU_CodeGenExample_T;

// External outputs (root outports fed by signals with auto st
typedef struct {
  real_T Uout;                         // '<Root>/Uout'
} ExtY_CodeGenExample_T;

```

3.1 Self-generated C/C++ code examples

- Running framework of **ert_main.cpp** file
- The file needs to be written by the user
- Before the program runs, create a new instance of **TestModelClass** and initialize it
- For example: **TestModelClass m_testClass**; **m_testClass.initializie();**
- Generate an interrupt or timer, and call the callback function every 0.001s, in which the following operations are performed: 1. Update input information; 2. Update parameter information; 3. Call the **step()** function; 4. Update the output information.
- **m_testClass. Test_U. Uin=***;**
- **m_testClass. Test_P.X0=***;**
- **m_testClass.step();**
- *****= m_testClass. Test_Y.Yout;**
- When quit call **m_testClass. Terminate()**;



Note: After configuration in this page, input make ***** command will call cmake to compile the code generated by Simulink to a px4_Simulink_app, and set it to start on boot.

3. Advanced Examples

3.2 Pixhawk code generation toolbox result analysis

Summary of the changes to the original firmware of the PX4 code of this platform:

1. For PX4-1.8 and less, add '**modules/px4_simulink_app' in 'Firmware\cmake\configs****.cmake**' file; for PX4-1.9 and above, add "**px4_simulink_app**" in the "**MODULES**" region of "**Firmware\boards\px4\fmu-v*\default.cmake**"
2. Setup '**px4_simulink_app**' folder and empty_file.c + CMakeLists.txt under '**Firmware\src\modules**'
3. Add startup commands : '**px4_simulink_app start**' in '**Firmware\ROMFS\px4fmu_common\init.d\rcS**'

nuttx_px4fmu-v3_default.cmake — C: > PX4PSP > Firmware > cmake > configs > nuttx_px4fmu-v3_default.cmake

```
set(config_module_list
    #
    # Board support modules
    #
    modules/px4_simulink_app
    drivers/rgbled_ncp5623c
    drivers/adis16448
    drivers/airspeed
    drivers/blinkm
    drivers/bmi160
    drivers/bmp280
    drivers/boards
```

CMakeLists.txt — C: > PX4PSP > Firmware > src > modules > px4_simulink_app > CMakeLists.txt

```
## This is a place-holder cmakelist.txt file
## It will get replaced by the Pixhawk PSP

px4_add_module(
MODULE modules__px4_simulink_app
MAIN px4_simulink_app
STACK_MAIN 2000
SRCS
    empty_file.c
DEPENDS
    platforms__common
)
```

rcS — C: > PX4PSP > Firmware > ROMFS > px4fmu_common > init.d > rcS

```
# Boot is complete, inform MAVLink app(s) that
mavlink boot_complete
rgbled_ncp5623c start
px4_simulink_app start
```

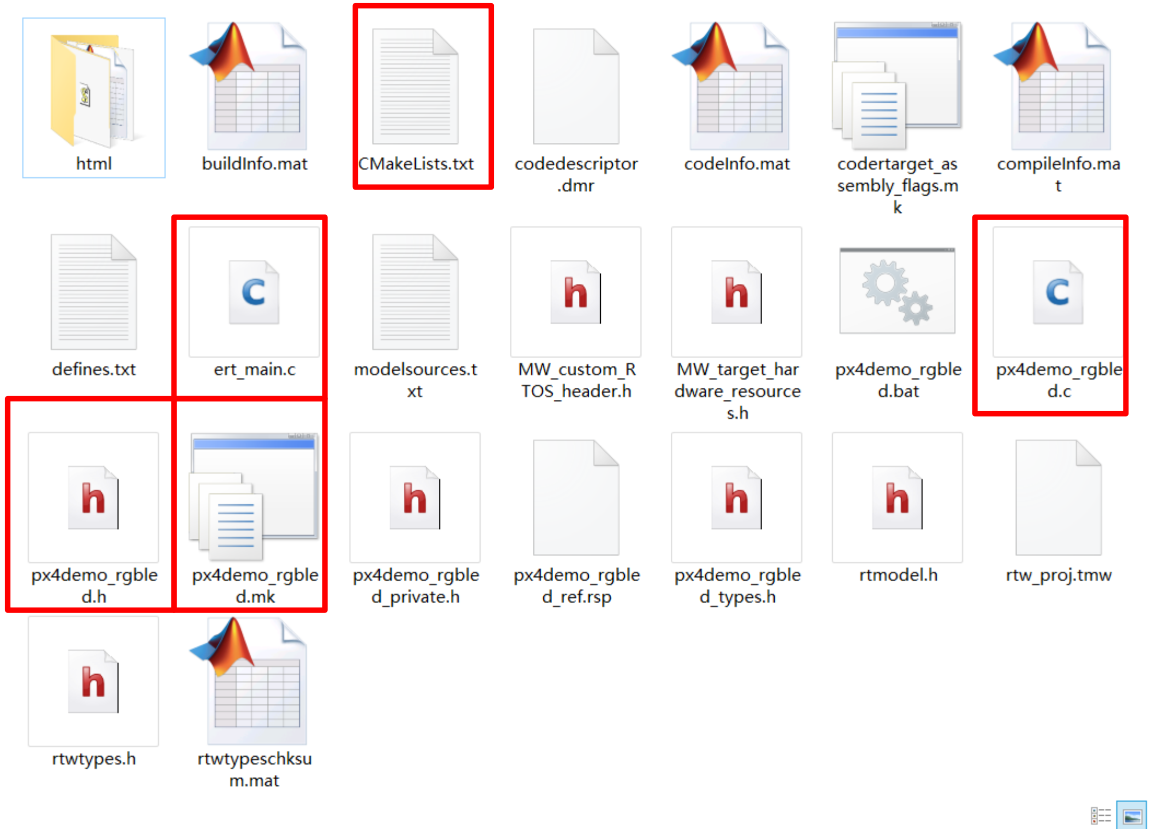
empty_file.c — C: > PX4PSP > Firmware > src > modules > px4_simulink_app > empty_file.c > ...

```
// this is a place-holder file to complete the build process

__EXPORT int px4_simulink_app_main(int argc, char *argv[]);

int px4_simulink_app_main(int argc, char *argv[]) //Px4 App
{

  return 0;
}
```

3. Advanced Examples

3.2 Pixhawk code generation toolbox result analysis

- Open any generated **ert_rtw** folder after **.slx** demo file compiled (e.g. LED demo), the main files generated includes:
- **CMakeLists.txt**
- **ert_main.c**
- *****.h**
- *****.c**
- *****.mk** This file is used to copy the code to a suitable location (px4_simulink_app folder) after MATLAB completes the code generation, and call the PX4 compilation command (e.g., "**make px4_fmu-v3_default**") to compile the firmware

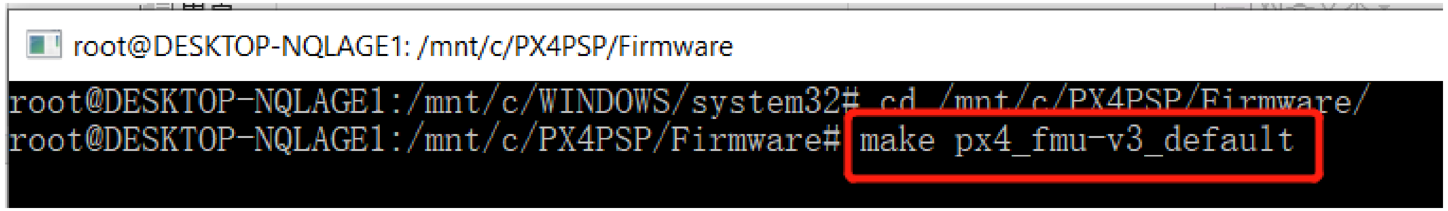


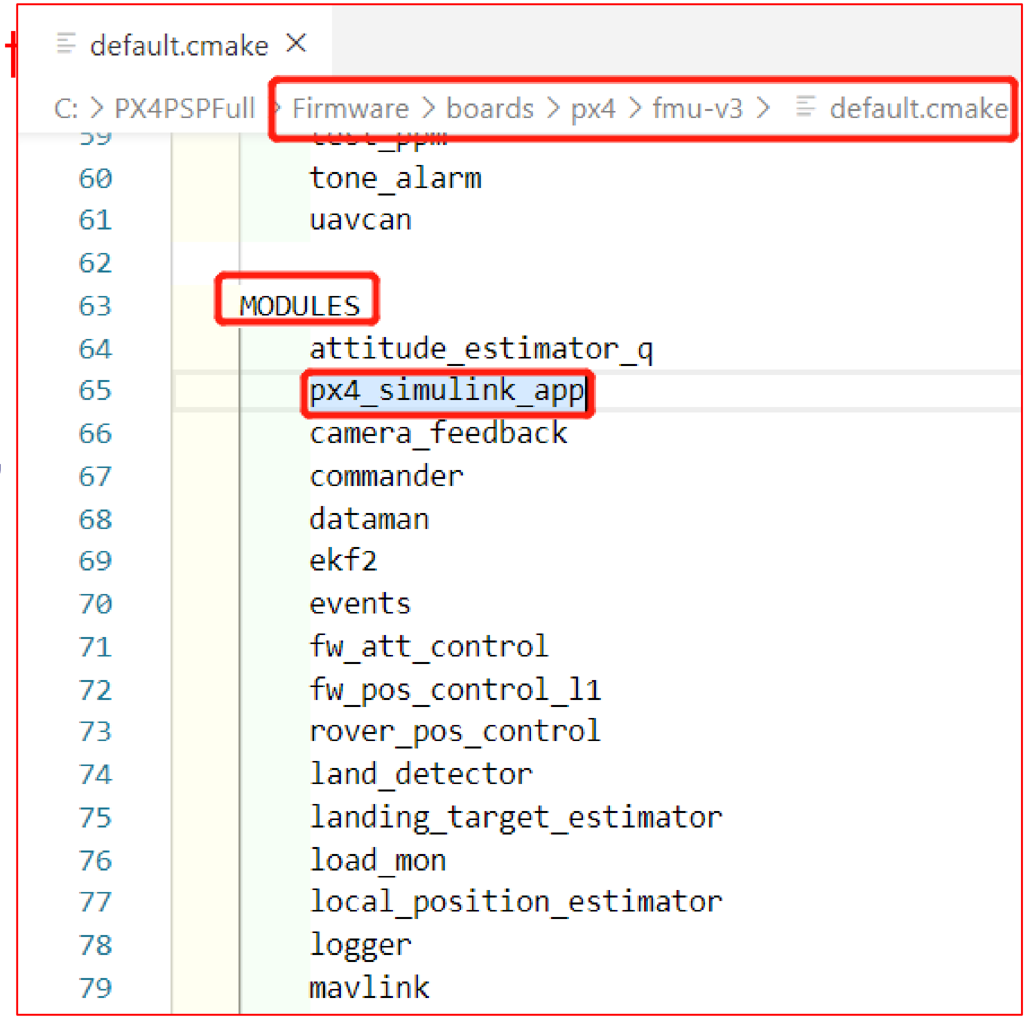
3. Advanced Examples

3.2 Pixhawk code generation toolbox result

- PX4 firmware compilation principle (take PX4 1.7 firmware fmu-v2 as an example):

1. Open the compiler Win10WSL/Cygwin/Msys2
2. Enter "**make px4_fmu-v3_default**"
   This command will call cmake to open the "**Firmware\boards\px4\fmu-v3\default.cmake**" file (PX4 1.8 and above versions will call "**cmake\configs\nuttx_px4fmu-v3_default.cmake**")
3. Compile code in the **px4_simulink_app** folder

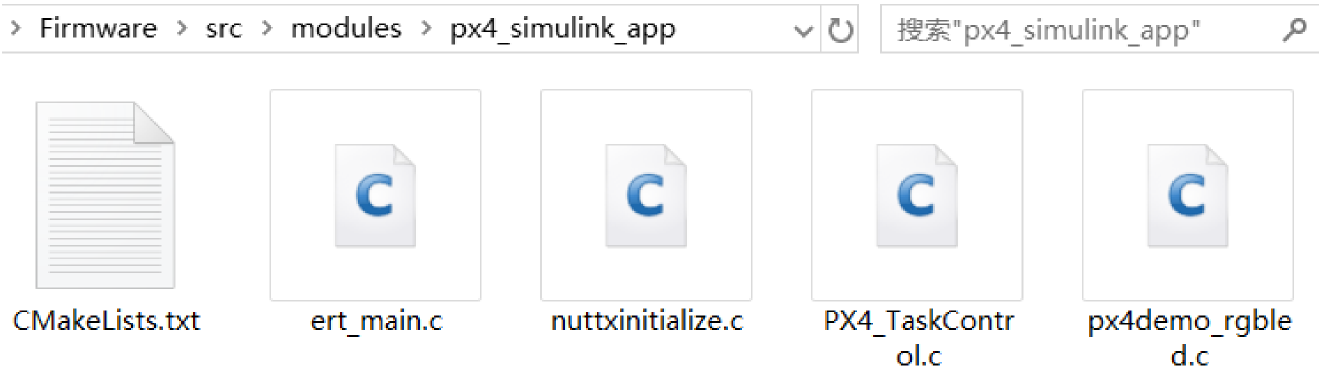


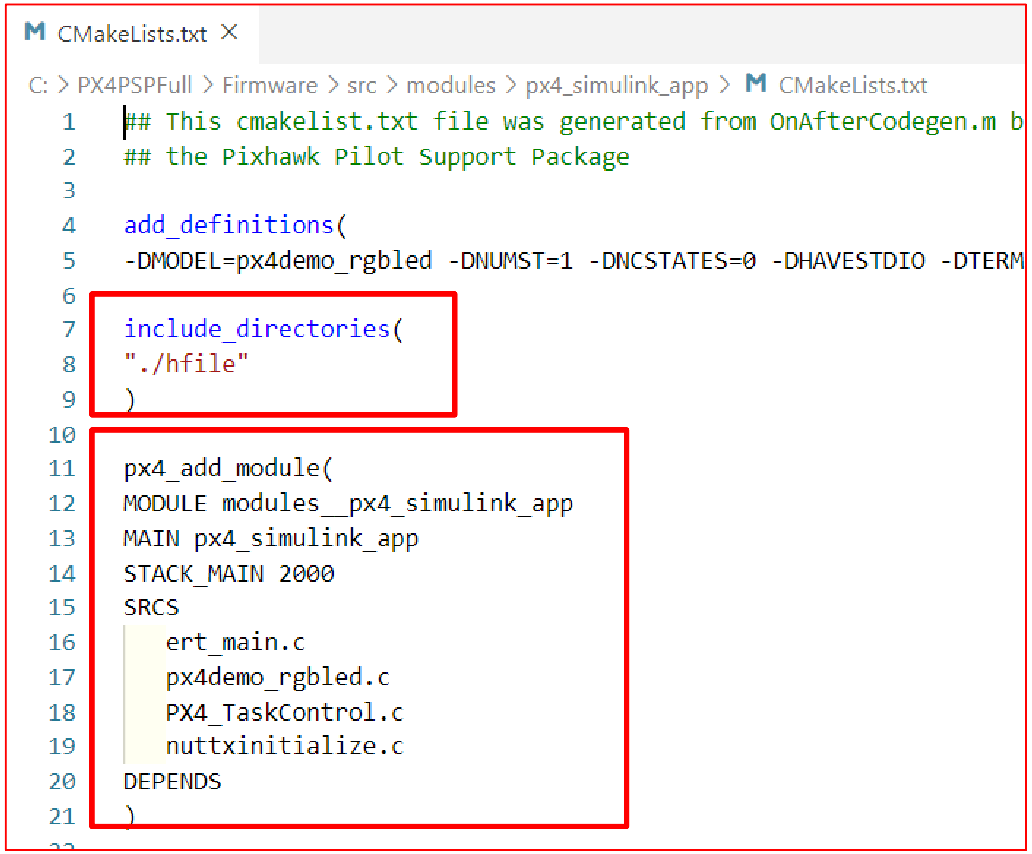


3. Advanced Examples

3.2 Pixhawk code generation toolbox result analysis

- PX4 firmware compilation principle:
- 4. Find the **CmakeLists.txt** file in the **px4_simulink_app** folder
- This file defines the way to compile the app thread
- The first is the path containing the source file
- The second is the main dependency of the app and thread priority






```
## This cmakelist.txt file was generated from OnAfterCodegen.m b
## the Pixhawk Pilot Support Package

add_definitions(
-DMODEL=px4demo_rgbled -DNUMST=1 -DNCSTATES=0 -DHAVESTDIO -DTERM

include_directories(
"./hfile"
)

px4_add_module(
MODULE modules__px4_simulink_app
MAIN px4_simulink_app
STACK_MAIN 2000
SRCS
    ert_main.c
    px4demo_rgbled.c
    PX4_TaskControl.c
    nuttxinitialize.c
DEPENDS
)
```

3. Advanced Examples

3.3 Pixhawk code generation toolbox modu

- These modules are composed of S functions plus **tlc (Target Language Compiler)** files
- Among them, the tlc file is a code generation template, which defines how the module generates code to access the driver interface of PX4 to exchange information with the underlying hardware
- The format of the tlc file can refer to MATLAB related tutorials

https: //www.mathworks.com/help/pdf_doc/rtw/index.html

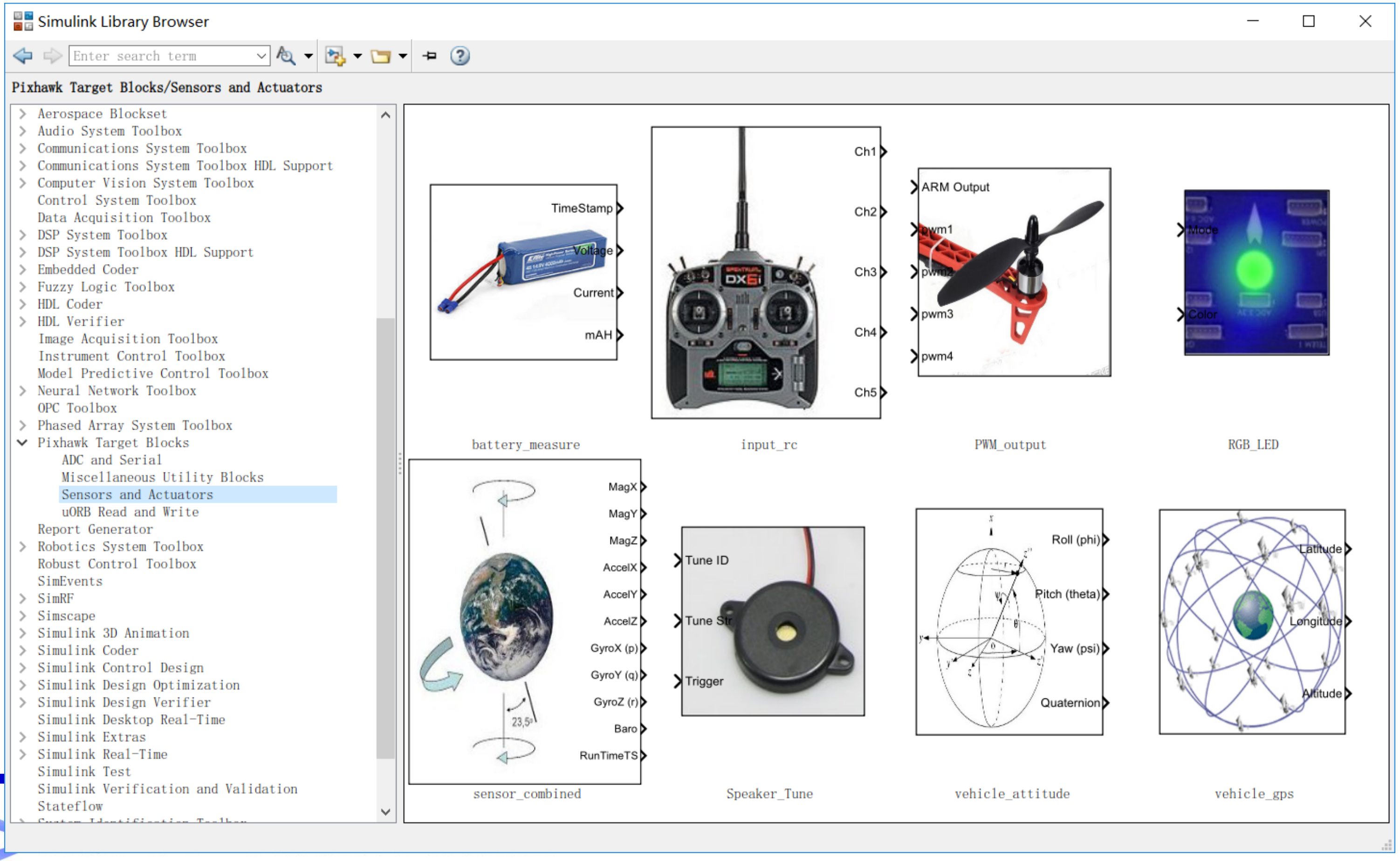
Simulink Coder Getting Started Guide

Simulink Coder User's Guide

Simulink Coder Target Language Compiler

Simulink Coder Reference

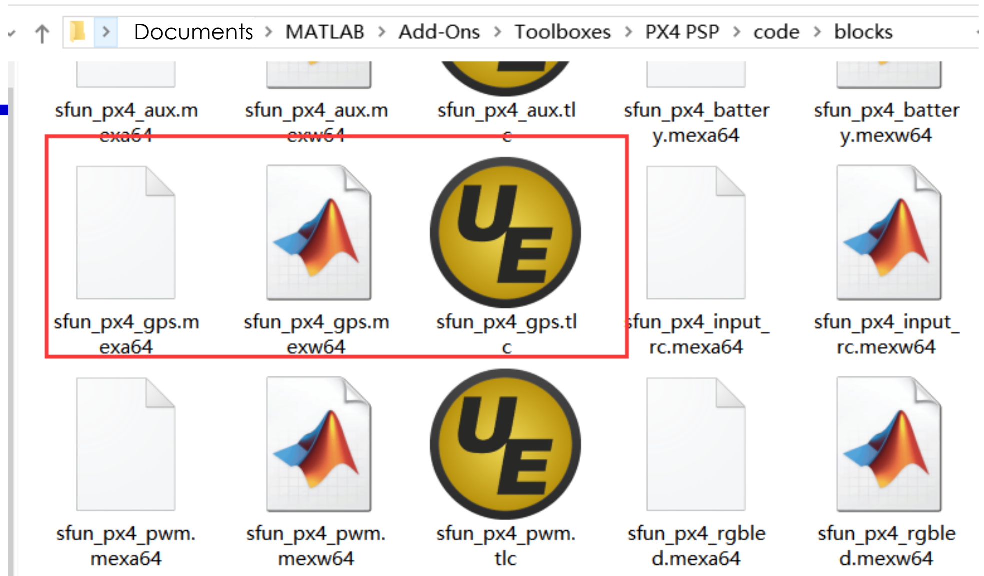
Simulink Coder Release Notes

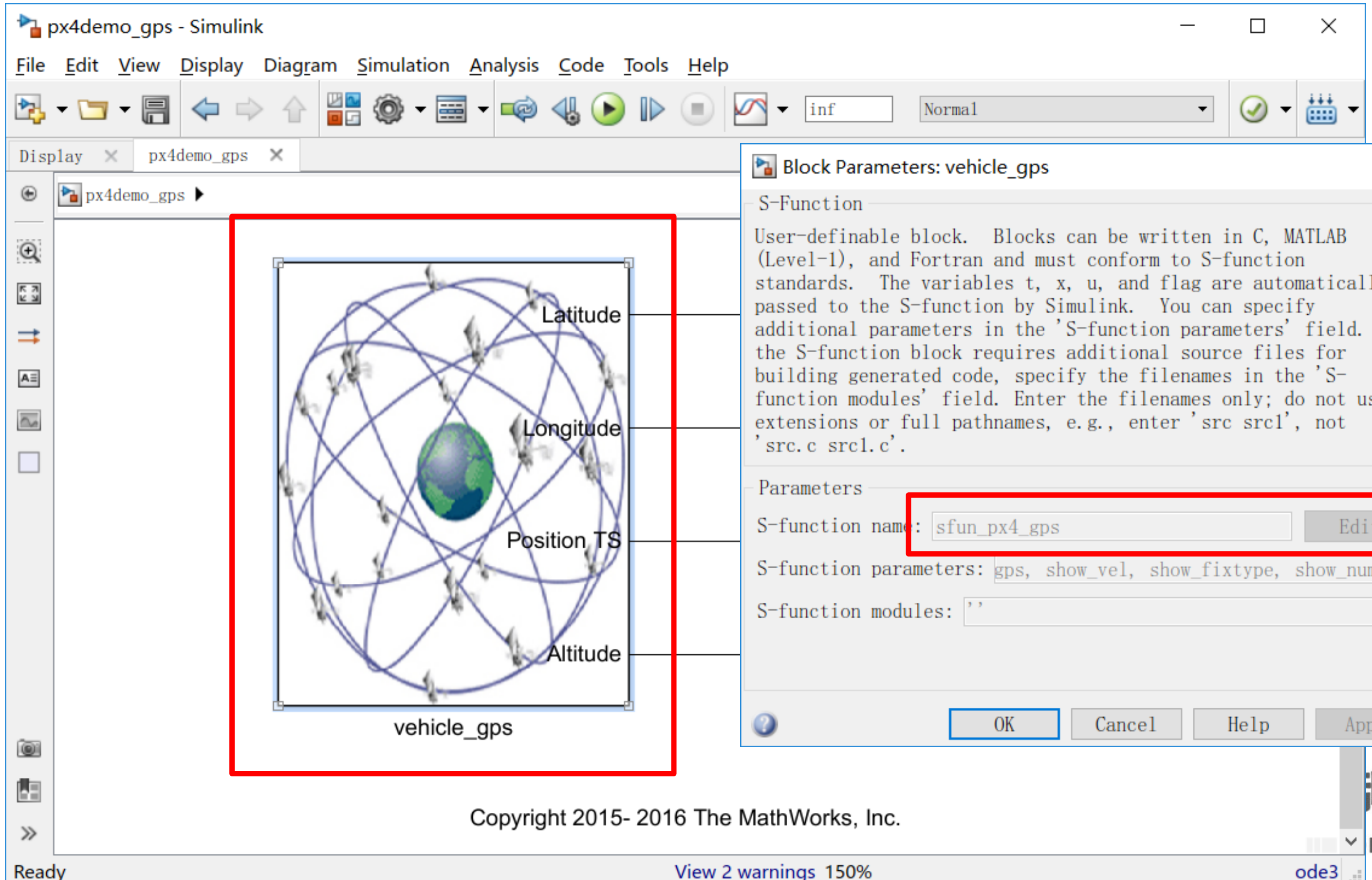


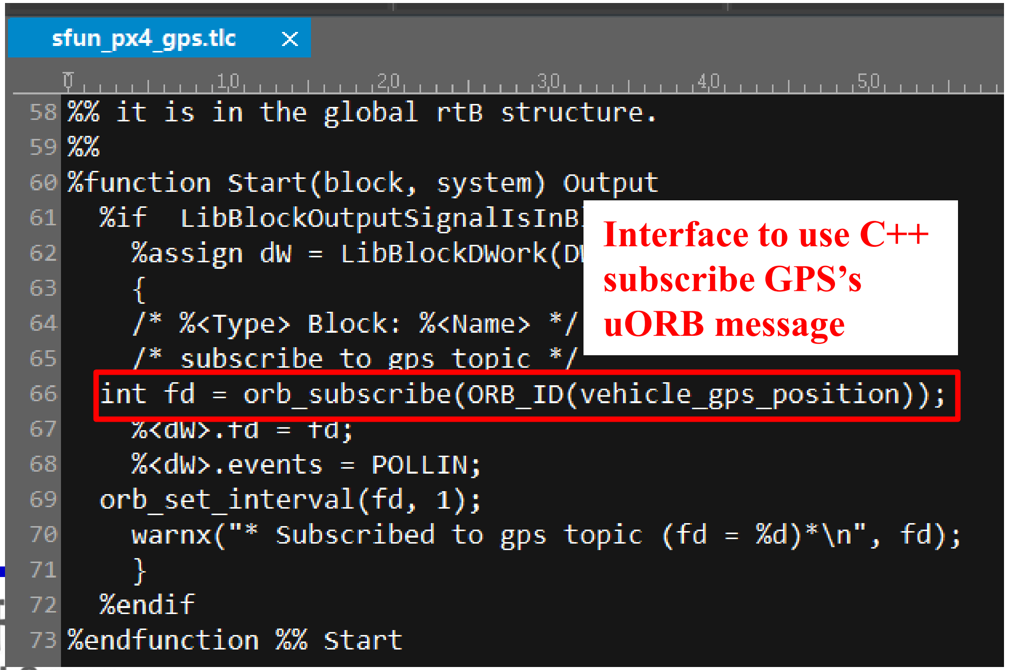
3. Advanced Examples

3.3 Pixhawk code generation toolbox module program

- Get the S-function (tlc) position from the Simulink module properties







Thanks