Interfacing Motor Controller and Temperature DAQ in Simulink

Introduction

In this tutorial show development of motor controller and temperature data acquisition in Simulink. The thermocouple type K with max6675 module used as temperature instrumentation. The brushed dc motor MY1016 was supplied with 24V DC. While for motor driver mdd10 shield used and it can be used for other type of dc brushed motor. Also, the motor shield can support 10A continuous and 30A peak. This model also comes with an encoder model to get real situation speed of dc motor where the rotary encoder attached to the motor.

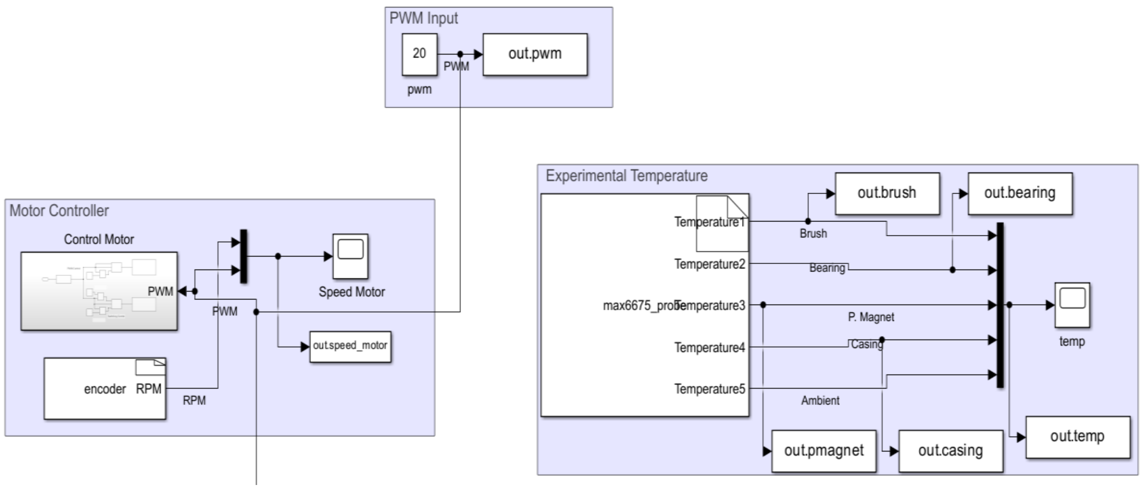


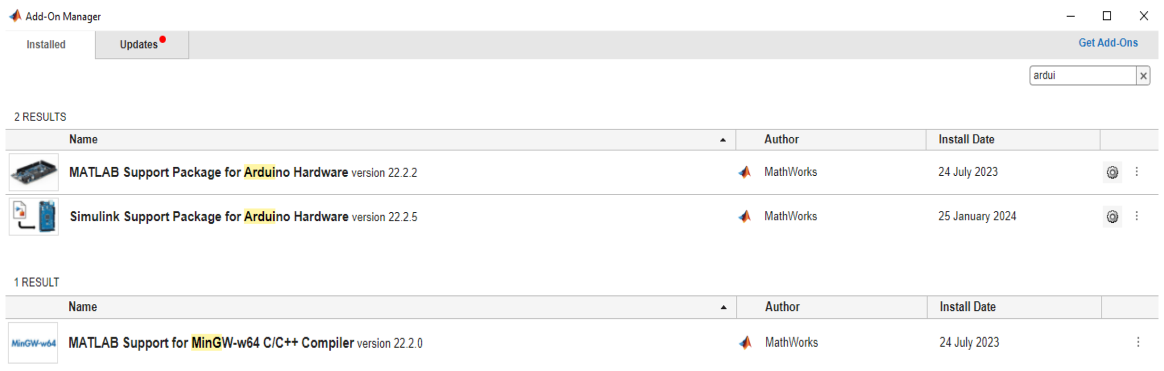
Figure 1.1: Motor controller and temperature daq model.

Development Block

Installation Arduino Library in Simulink:

MATLAB and Simulink developer has library support packages for Arduino hardware where help in program Arduino board using command MATLAB or modeling in Simulink. There are two methods to through MATLAB and manual. Three libraries need to install before building the model:

- MATLAB support package for Arduino hardware.
- Simulink support package for Arduino hardware.
- MATLAB support for MinGW-w64 C/C++ Compiler


Add-On Manager

Installed | Updates | Get Add-Ons

ardui

2 RESULTS

Name	Author	Install Date
MATLAB Support Package for Arduino Hardware version 22.2.2	MathWorks	24 July 2023
Simulink Support Package for Arduino Hardware version 22.2.5	MathWorks	25 January 2024

1 RESULT

Name	Author	Install Date
MATLAB Support for MinGW-w64 C/C++ Compiler version 22.2.0	MathWorks	24 July 2023

Figure 4.1: Library support package for Arduino hardware.

1. Installation through MATLAB by clicking the **Add-Ons** icon and choose **Get Add-Ons** as Figure 4.2 then find the listed library.

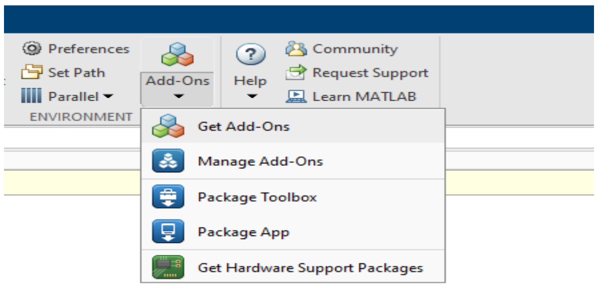
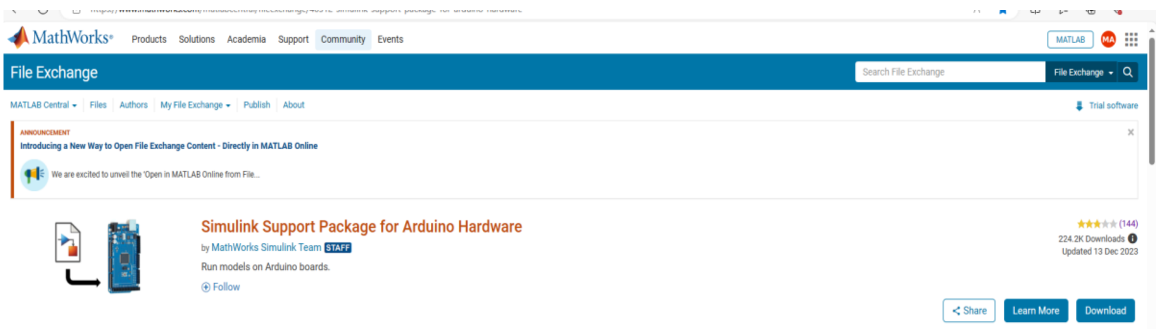


Figure 4.2: Automatic installation.

2. While for manual, need to find the library from the MATLAB official website like Figure 4.3.



ANNOUNCEMENT

Introducing a New Way to Open File Exchange Content - Directly in MATLAB Online

We are excited to unveil the 'Open in MATLAB Online from File...

Simulink Support Package for Arduino Hardware

by MathWorks Simulink Team STAFF

Run models on Arduino boards.

Follow

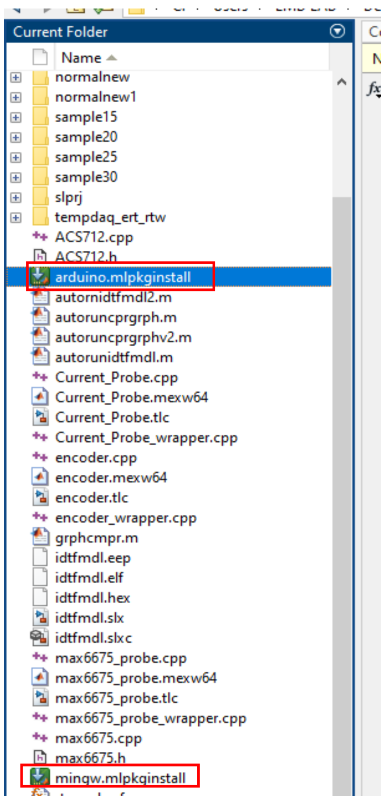
(144)

224.2K Downloads

Updated 13 Dec 2023

Figure 4.3: Download Simulink support package from MATLAB website.

3. The library should be placed as Figure 4.4 in the same model folder. Then, double click the library to start installation. Bear in mind that this process should be done from inside MATLAB.


Current Folder

Name

normalnew
normalnew1
sample15
sample20
sample25
sample30
slprj
tempdaq_ert_rtw
ACS712.cpp
ACS712.h
arduino.mlpkginstall
autornidtfmdl2.m
autoruncprgrph.m
autoruncprgrphv2.m
autorunidtfmdl.m
Current_Probe.cpp
Current_Probe.mexw64
Current_Probe.tlc
Current_Probe_wrapper.cpp
encoder.cpp
encoder.mexw64
encoder.tlc
encoder_wrapper.cpp
grphcmpr.m
idtfmdl.eep
idtfmdl.elf
idtfmdl.hex
idtfmdl.slx
idtfmdl.slxc
max6675_probe.cpp
max6675_probe.mexw64
max6675_probe.tlc
max6675_probe_wrapper.cpp
max6675.cpp
max6675.h
mingw.mlpkginstall

Figure 4.4: Location for support packages.

Development motor controller block:

The motor was controlled using motors shield that has specific pin for pwm and direction. In this case, the available pin for pwm is 3,5,6,9,10,11 while direction pin is 2,4,8,7,12,13. The Arduino pin block can be found in the library.

1. The Arduino pin 5 use to control speed of dc motor through PWM values from 0 – 255 and the values should always in positive. To convert negative input to positive the switching block used, where the principles operation of switching is if the PWM input greater than 0, the input used. Vice versa, the input multiply (-1) then used.
2. The Arduino pin 4 used to control direction rotation of dc motor through +v and -v values from PWM values. The switching block used to control direction of rotation, where principles operation of switching is if the pwm higher than 1, the value was converted to -v for forward rotation. Meanwhile if the pwm lower than 1, the value was converted to +v for reversed rotation.
3. For both pins can be change to other digital pin on Arduino boards based on the motor shield available pin.

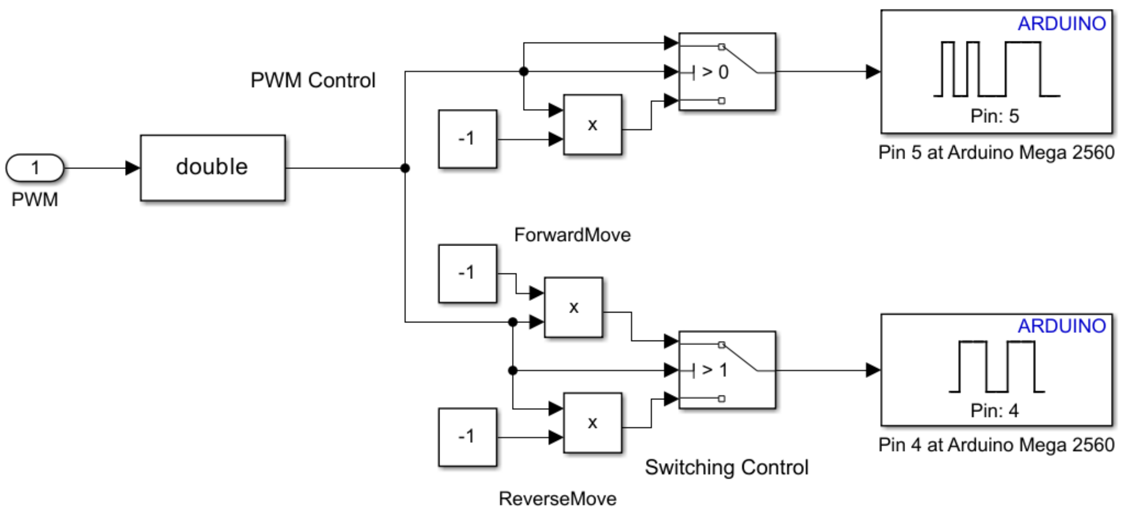
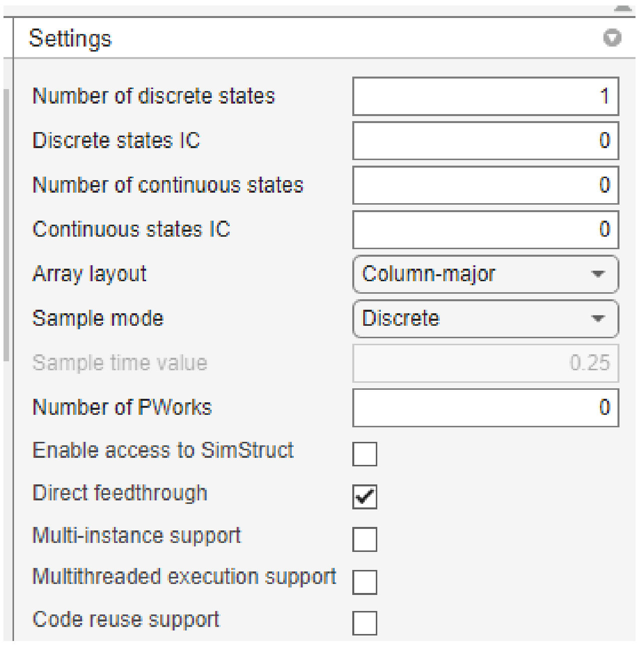


Figure 4.5: Motor controller block.

Development of Encoder DAQ:

The rotary encoder is used to read speed measurement. There is an encoder block in Simulink library, but the library could not read the output from encoder if it rotates counterclockwise. To counter the problem, Arduino code was used by uploading on the s function builder block. The s function builder block can be found in library.

1. In s function builder block can be separated part where Arduino code part, port and parameter part and setting part.
2. Setting part setup as Figure 4.6. The sample mode changes to discrete and sample time values follow as the Max6675 setup for get same time reading data. These sample time values can change based on the sensor and their calculation time used.

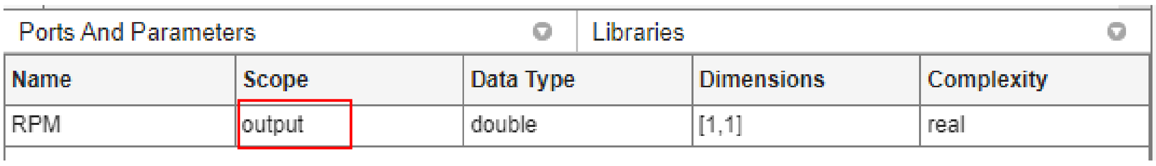


Settings

Number of discrete states	1
Discrete states IC	0
Number of continuous states	0
Continuous states IC	0
Array layout	Column-major
Sample mode	Discrete
Sample time value	0.25
Number of PWorks	0
Enable access to SimStruct	☐
Direct feedthrough	✓
Multi-instance support	☐
Multithreaded execution support	☐
Code reuse support	☐

Figure 4.6: S-function setting part.

3. Ports and Parameters part setup as Figure 4.7, right click at output in the red box to convert output. Change the name based and make sure **data type** in **double.**

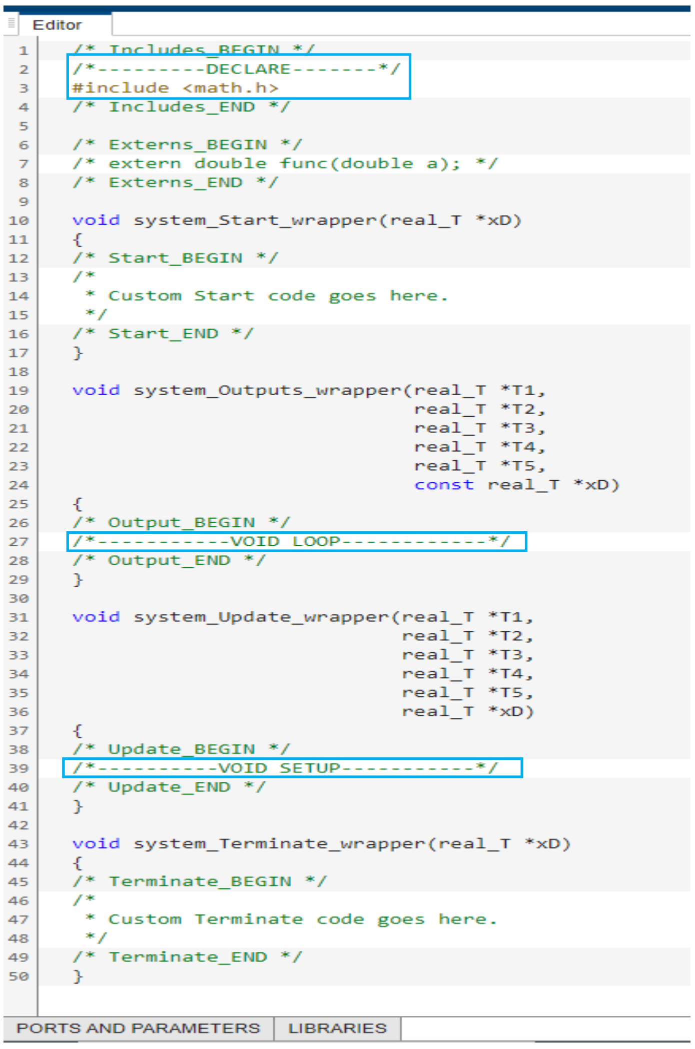


Ports And Parameters | Libraries

Name	Scope	Data Type	Dimensions	Complexity
RPM	output	double	[1,1]	real

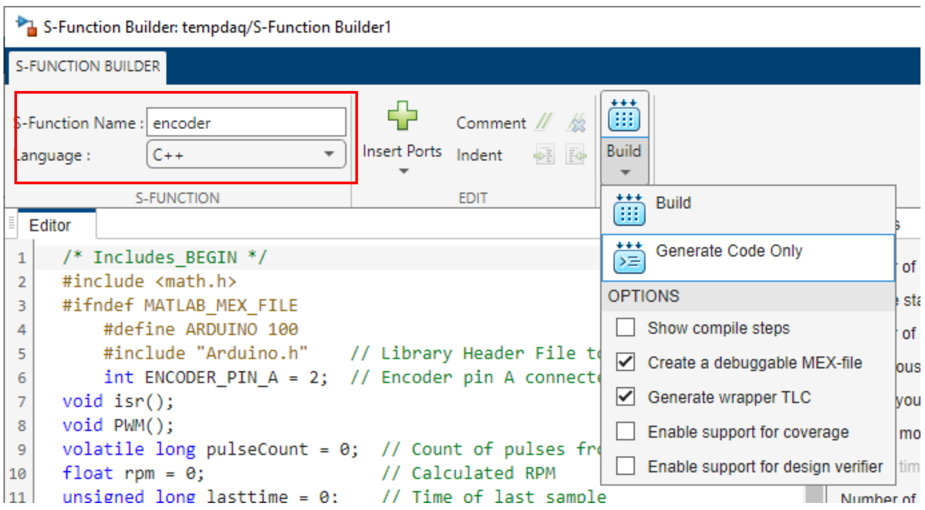
Figure 4.7: S-function port and parameter part.

4. Arduino code part can be separated into declare, void setup and void loop as Figure 4.8 and all the code could see at Encoder code section.
5. At declare part, the code about declare pin used on Arduino, library and function.
6. Void setup part, the code about pin mode used on Arduino.
7. Void loop part, the code about displaying data.
8. After finish write the code and place the library, change the name language as Figure 4.9. The language change to C++ due to Arduino code language.
9. Then click the build icon to generate several files for enabling the encoder operate in Simulink. Notification appears after finish building as Figure 4.10.



```
/* Includes_BEGIN */
/*---------DECLARE-------*/
#include <math.h>
/* Includes_END */

/* Externs_BEGIN */
/* extern double func(double a); */
/* Externs_END */

void system_Start_wrapper(real_T *xD)
{
/* Start_BEGIN */
/*
 * Custom Start code goes here.
 */
/* Start_END */
}

void system_Outputs_wrapper(real_T *T1,
                            real_T *T2,
                            real_T *T3,
                            real_T *T4,
                            real_T *T5,
                            const real_T *xD)
{
/* Output_BEGIN */
/*-----------VOID LOOP------------*/
/* Output_END */
}

void system_Update_wrapper(real_T *T1,
                           real_T *T2,
                           real_T *T3,
                           real_T *T4,
                           real_T *T5,
                           real_T *xD)
{
/* Update_BEGIN */
/*----------VOID SETUP-----------*/
/* Update_END */
}

void system_Terminate_wrapper(real_T *xD)
{
/* Terminate_BEGIN */
/*
 * Custom Terminate code goes here.
 */
/* Terminate_END */
}
```

Figure 4.8: S-function Arduino part.

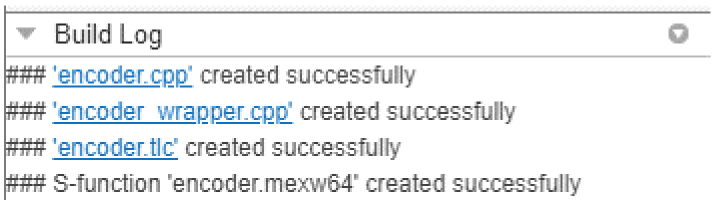


S-Function Name : encoder
Language : C++

☐ Show compile steps
☑ Create a debuggable MEX-file
☑ Generate wrapper TLC
☐ Enable support for coverage
☐ Enable support for design verifier

```
/* Includes_BEGIN */
#include <math.h>
#ifndef MATLAB_MEX_FILE
    #define ARDUINO 100
    #include "Arduino.h"    // Library Header File t
    int ENCODER_PIN_A = 2;  // Encoder pin A connect
void isr();
void PWM();
volatile long pulseCount = 0;  // Count of pulses fr
float rpm = 0;                 // Calculated RPM
unsigned long lasttime = 0;    // Time of last sample
```

Figure 4.9: Building the Max6675 model.



```
### 'encoder.cpp' created successfully
### 'encoder_wrapper.cpp' created successfully
### 'encoder.tlc' created successfully
### S-function 'encoder.mexw64' created successfully
```

Figure 4.10: Notification after building model.

Encoder Code

Declare Code:

```
#include <math.h>
#ifndef MATLAB_MEX_FILE
#define ARDUINO 100
#include "Arduino.h"    // Library Header File to run code on Arduino
int ENCODER_PIN_A = 2;  // Encoder pin A connected to digital pin 2
void isr();
void PWM();
volatile long pulseCount = 0;  // Count of pulses from the encoder
float rpm = 0;                 // Calculated RPM
unsigned long lasttime = 0;    // Time of last sample
int RPM;
void isr() { // function always giving positive values either rotates clockwise or conterclockwise
  pulseCount++;
}
void PWM() { // function calculation to get rpm value
  if (millis() - lasttime >= 250) {
    noInterrupts();
    rpm = ((float)pulseCount / 500) * (60000 / (millis() - lasttime));
    pulseCount = 0;
    lasttime = millis();
    interrupts();
    rpm;
  }
}
#endif
```

Void Setup:

```
if(xD[0] != 1) // void setup()
{
   #ifndef MATLAB_MEX_FILE
  pinMode(ENCODER_PIN_A, INPUT);
  attachInterrupt(digitalPinToInterrupt(ENCODER_PIN_A), isr, RISING);
   #endif
   xD[0] = 1;
}
```

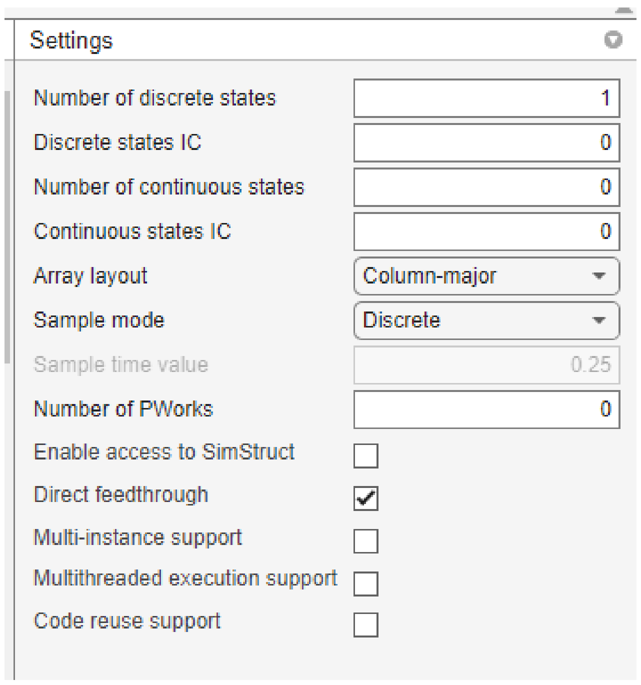
Void Loop:

```
if(xD[0] == 1)  // void loop()
{
   #ifndef MATLAB_MEX_FILE
  PWM();
  RPM[0] = rpm;
    #endif
}
```

Development of Temperature DAQ:

The thermocouple combining with Max6675 module to read temperature measurement. To operate the module using Simulink, Arduino code was used by upload on the s function builder block. The s function builder block can be found in library.

1. In s function builder block can be separated part where Arduino code part, port and parameter part and setting part.
2. Setting part setup as Figure 4.11. The sample mode changes to discrete because Max6675 needs time to calculate data from thermocouple before convert to temperature. So, the sample value time must be 0.25 seconds. These sample time values can change based on the sensor and their calculation time used.

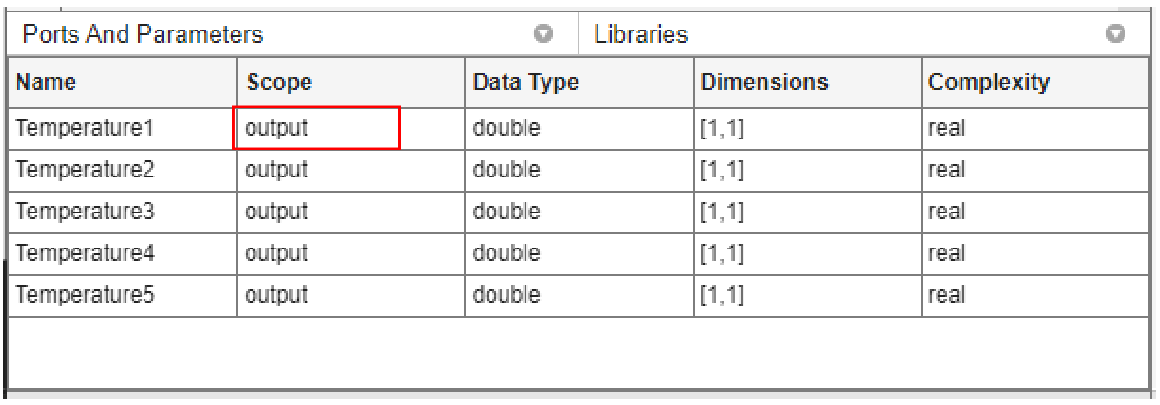


Settings

Number of discrete states	1
Discrete states IC	0
Number of continuous states	0
Continuous states IC	0
Array layout	Column-major
Sample mode	Discrete
Sample time value	0.25
Number of PWorks	0
Enable access to SimStruct	☐
Direct feedthrough	✓
Multi-instance support	☐
Multithreaded execution support	☐
Code reuse support	☐

Figure 4.11: S-function setting parts.

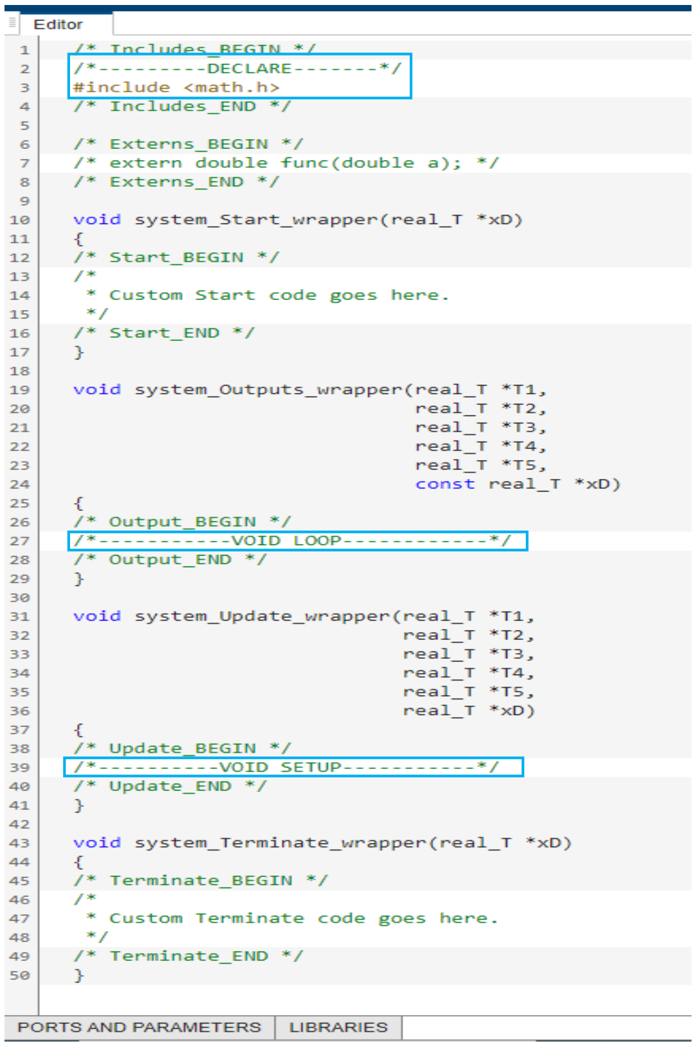
3. Ports and Parameters part setup as Figure 4.12, right click at output in the red box to add more output. Change the name based on thermocouple position and make sure **data type** in **double.**



Name	Scope	Data Type	Dimensions	Complexity
Temperature1	output	double	[1,1]	real
Temperature2	output	double	[1,1]	real
Temperature3	output	double	[1,1]	real
Temperature4	output	double	[1,1]	real
Temperature5	output	double	[1,1]	real

Figure 4.12: Port and parameter setup.

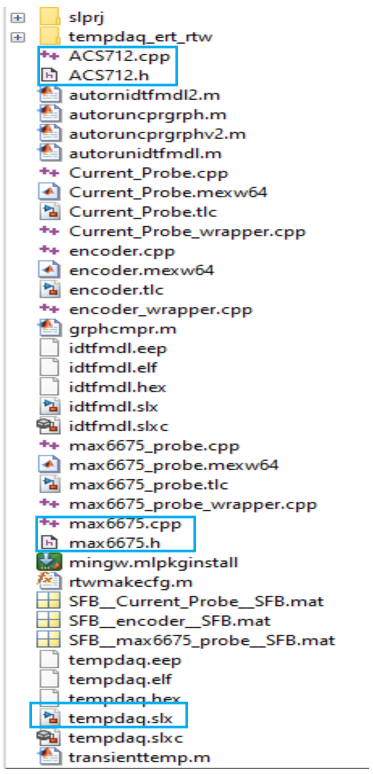
4. Arduino code part can be separated into declare, void setup and void loop as Figure 4.13 and all the code could seen at Max6675 code section.
5. At declare part, the code about declare pin used on Arduino, library and function.
6. Void setup part, the code about pin mode used on Arduino.
7. Void loop part, the code about displaying data.



```
/* Includes_BEGIN */
/*---------DECLARE-------*/
#include <math.h>
/* Includes_END */

/* Externs_BEGIN */
/* extern double func(double a); */
/* Externs_END */

void system_Start_wrapper(real_T *xD)
{
/* Start_BEGIN */
/*
 * Custom Start code goes here.
 */
/* Start_END */
}

void system_Outputs_wrapper(real_T *T1,
                            real_T *T2,
                            real_T *T3,
                            real_T *T4,
                            real_T *T5,
                            const real_T *xD)
{
/* Output_BEGIN */
/*-----------VOID LOOP------------*/
/* Output_END */
}

void system_Update_wrapper(real_T *T1,
                           real_T *T2,
                           real_T *T3,
                           real_T *T4,
                           real_T *T5,
                           real_T *xD)
{
/* Update_BEGIN */
/*----------VOID SETUP-----------*/
/* Update_END */
}

void system_Terminate_wrapper(real_T *xD)
{
/* Terminate_BEGIN */
/*
 * Custom Terminate code goes here.
 */
/* Terminate_END */
}
```

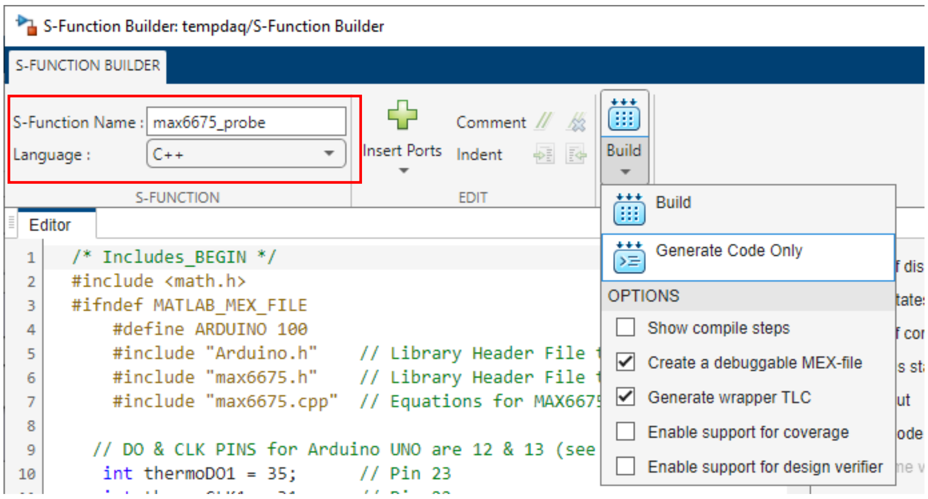
Figure 4.13: Arduino code part.

8. Max6675 was run in Arduino ide using library, so the same library should include into folder that used to create the model as Figure 4.14 where library is highlighted in blue box. Because s function will need the library as same as Arduino ide to run the code.
9. After finish write the code and place the library, change the name language as Figure 4.15. The language change to C++ due to Arduino code language.
10. Then click the build icon to generate several files for enabling the max6675 operate in Simulink. Notification appears after finish building as Figure 4.16.



```
slprj
tempdaq_ert_rtw
ACS712.cpp
ACS712.h
autornidtfmdl2.m
autoruncprgrph.m
autoruncprgrphv2.m
autorunidtfmdl.m
Current_Probe.cpp
Current_Probe.mexw64
Current_Probe.tlc
Current_Probe_wrapper.cpp
encoder.cpp
encoder.mexw64
encoder.tlc
encoder_wrapper.cpp
grphcmpr.m
idtfmdl.eep
idtfmdl.elf
idtfmdl.hex
idtfmdl.slx
idtfmdl.slxc
max6675_probe.cpp
max6675_probe.mexw64
max6675_probe.tlc
max6675_probe_wrapper.cpp
max6675.cpp
max6675.h
mingw.mlpkginstall
rtwmakecfg.m
SFB__Current_Probe__SFB.mat
SFB__encoder__SFB.mat
SFB__max6675_probe__SFB.mat
tempdaq.eep
tempdaq.elf
tempdaq.hex
tempdaq.slx
tempdaq.slxc
transienttemp.m
```

Figure 4.14: Max6675 library placement.



S-Function Builder: tempdaq/S-Function Builder

S-Function Name : max6675_probe

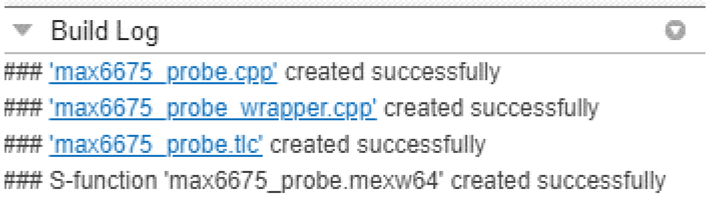
Language : C++

Build: Generate Code Only

OPTIONS
- [ ] Show compile steps
- [x] Create a debuggable MEX-file
- [x] Generate wrapper TLC
- [ ] Enable support for coverage
- [ ] Enable support for design verifier

```
/* Includes_BEGIN */
#include <math.h>
#ifndef MATLAB_MEX_FILE
    #define ARDUINO 100
    #include "Arduino.h"    // Library Header File
    #include "max6675.h"    // Library Header File
    #include "max6675.cpp"  // Equations for MAX667

  // DO & CLK PINS for Arduino UNO are 12 & 13 (see
   int thermoDO1 = 35;      // Pin 23
```

Figure 4.15: Generate process Max6675 block.



Build Log

```
### 'max6675_probe.cpp' created successfully
### 'max6675_probe_wrapper.cpp' created successfully
### 'max6675_probe.tlc' created successfully
### S-function 'max6675_probe.mexw64' created successfully
```

Figure 4.16: Notification of build log.

Max6675 Code

Declare Code:

```
#include <math.h>
#ifndef MATLAB_MEX_FILE
   #define ARDUINO 100
   #include "Arduino.h"    // Library Header File to run code on Arduino
   #include "max6675.h"    // Library Header File to run code for MAX6675 Type K Thermocouple
   #include "max6675.cpp"  // Equations for MAX6675 Type K Thermocouple
  // DO & CLK PINS for Arduino UNO are 12 & 13 (see pin-out-map for other PLC models)
   int thermoDO1 = 35;      // Pin 35
   int thermoCLK1 = 31;     // Pin 31
   int thermoDO2 = 45;      // Pin 45
   int thermoCLK2 = 41;     // Pin 41
   int thermoDO3 = 47;      // Pin 47
   int thermoCLK3 = 51;     // Pin 51
   //Thermocouple 1        // Setup Thermocouple
   int thermoCS1 = 29;     // Pin 29
   MAX6675 thermocouple1(thermoCLK1, thermoCS1, thermoDO1);  // Equation called from max6675.h
   //Thermocouple 2        // Setup Thermocouple
   int thermoCS2 = 33;      // Pin 33
   MAX6675 thermocouple2(thermoCLK1, thermoCS2, thermoDO1);  // Equation called from max6675.h
//Thermocouple 3        // Setup Thermocouple
   int thermoCS3 = 39;     // Pin 39
   MAX6675 thermocouple3(thermoCLK2, thermoCS3, thermoDO2);  // Equation called from max6675.h
   //Thermocouple 4        // Setup Thermocouple
   int thermoCS4 = 43;      // Pin 43
   MAX6675 thermocouple4(thermoCLK2, thermoCS4, thermoDO2);  // Equation called from max6675.h
//Thermocouple 5        // Setup Thermocouple
   int thermoCS5 = 49;     // Pin 49
   MAX6675 thermocouple5(thermoCLK3, thermoCS5, thermoDO3);  // Equation called from max6675.h
#endif
```

Void Setup Code:

```
if(xD[0] != 1) // void setup()
{
   #ifndef MATLAB_MEX_FILE
      //Thermocouple 1 (Pull Voltage up to 5 Volts)
      pinMode(thermoCS1,OUTPUT);digitalWrite(thermoCS1,HIGH);
      //Thermocouple 2 (Pull Voltage up to 5 Volts)
      pinMode(thermoCS2,OUTPUT);digitalWrite(thermoCS2,HIGH);
      //Thermocouple 3 (Pull Voltage up to 5 Volts)
      pinMode(thermoCS3,OUTPUT);digitalWrite(thermoCS3,HIGH);
      //Thermocouple 4 (Pull Voltage up to 5 Volts)
     pinMode(thermoCS4,OUTPUT);digitalWrite(thermoCS4,HIGH);
      //Thermocouple 5 (Pull Voltage up to 5 Volts)
      pinMode(thermoCS5,OUTPUT);digitalWrite(thermoCS5,HIGH);
```

```
    #endif
    //done with initialization
    xD[0] = 1;
}
```

Void Loop Code:

```
if(xD[0] == 1)  // void loop() {
    #ifndef MATLAB_MEX_FILE
        //Thermocouple 1    Readout
        Temperature1[0]=thermocouple1.readCelsius();
        //Thermocouple 2    Readout
        Temperature2[0]=thermocouple2.readCelsius();
         //Thermocouple 3    Readout
        Temperature3[0]=thermocouple3.readCelsius();
        //Thermocouple 4    Readout
        Temperature4[0]=thermocouple4.readCelsius();
        //Thermocouple 5    Readout
        Temperature5[0]=thermocouple5.readCelsius();
    #endif
}
```