TinyML 的 STM32 实现流程

& 手势识别项目 实现

———— 陈志扬 V1.1 2024/8/14

准备:

软件：STM32CubeMX最新(生成项目框架代码，导入AI模型)，Keil uVision5(编译，模拟环境中debug), STM32CubeProgrammer(连板子，烧录)，Matlab(雷达原始数据处理、特征提取，封装函数转C语言)，Google Colab网页 或其他 Python3+tensorflow2.0环境(训练ai分类模型)

模型：sine\_model.tflite, gesture\_model.tflite

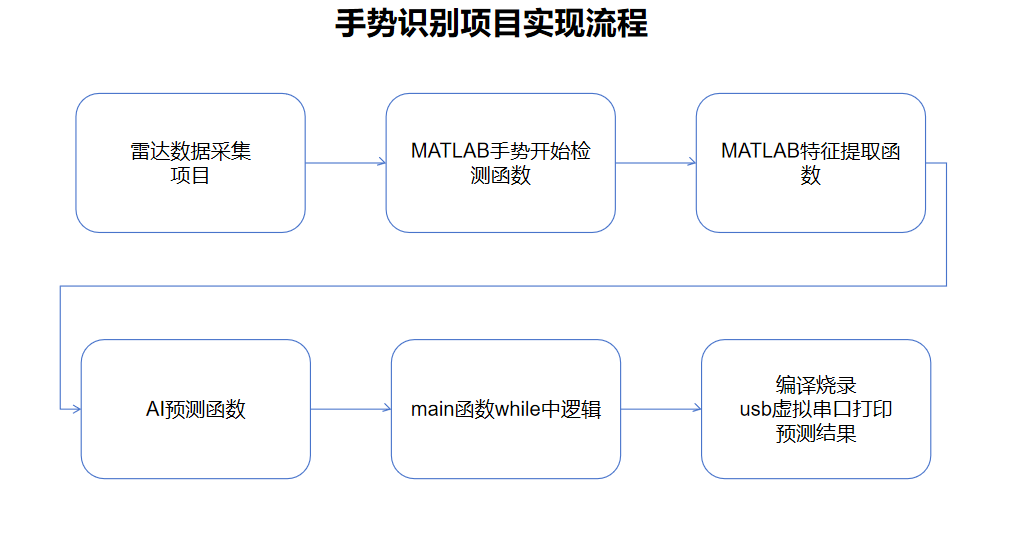
数据集：手势识别特征数据，6000x91，

一共6个手势每个手势，每个手势1000个样本，每个样本6特征x15帧=90个特征，1个对应标签

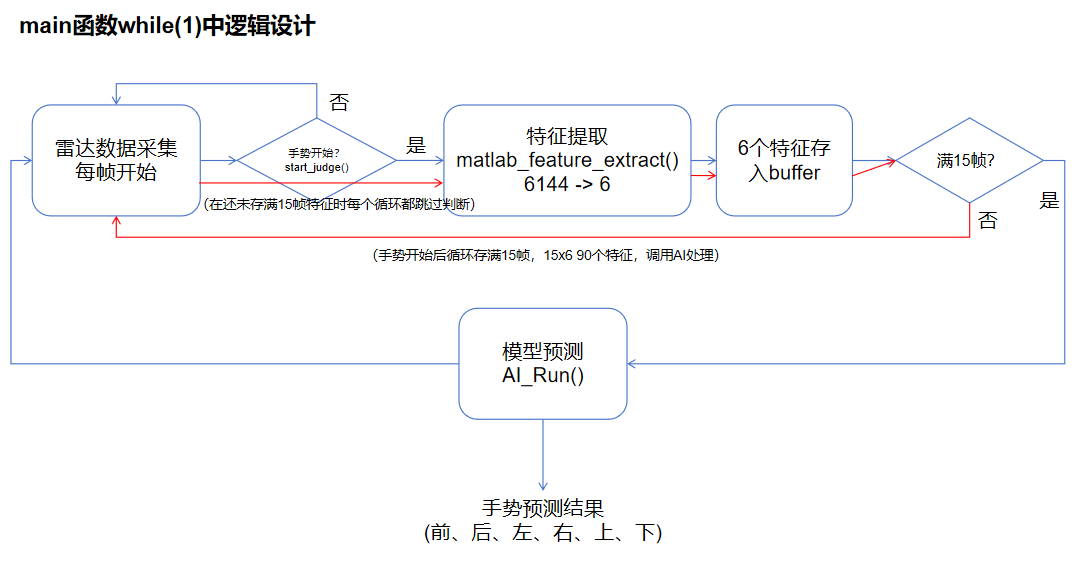
train数据集：x\_train(4800x90), y\_train(4800x1)

test数据集：test\_data.txt(1200x90), test\_label.txt(1200x1)

流程图：



主循环逻辑：



目录：

[一、使用STM32cubeMX 创建ARM项目，引入X-CUBE-AI包，导入深度学习模型网络，自动生成项目代码 2](#_Toc22117)

[二、使用keil模拟，‘use simulator’，实现没有板子软件内模拟debug，实现最简单sine曲线例子。 8](#_Toc32373)

[三、使用keil编译项目，生成.hex文件，使用stm32cubeprogrammer将文件烧录进板子，实现板子跑模型预测，将结果通过usb打印，pc端通过串口助手看usb虚拟串口打印结果。 21](#_Toc493)

[四、雷达数据采集及BIT12转BIT16处理 29](#_Toc8699)

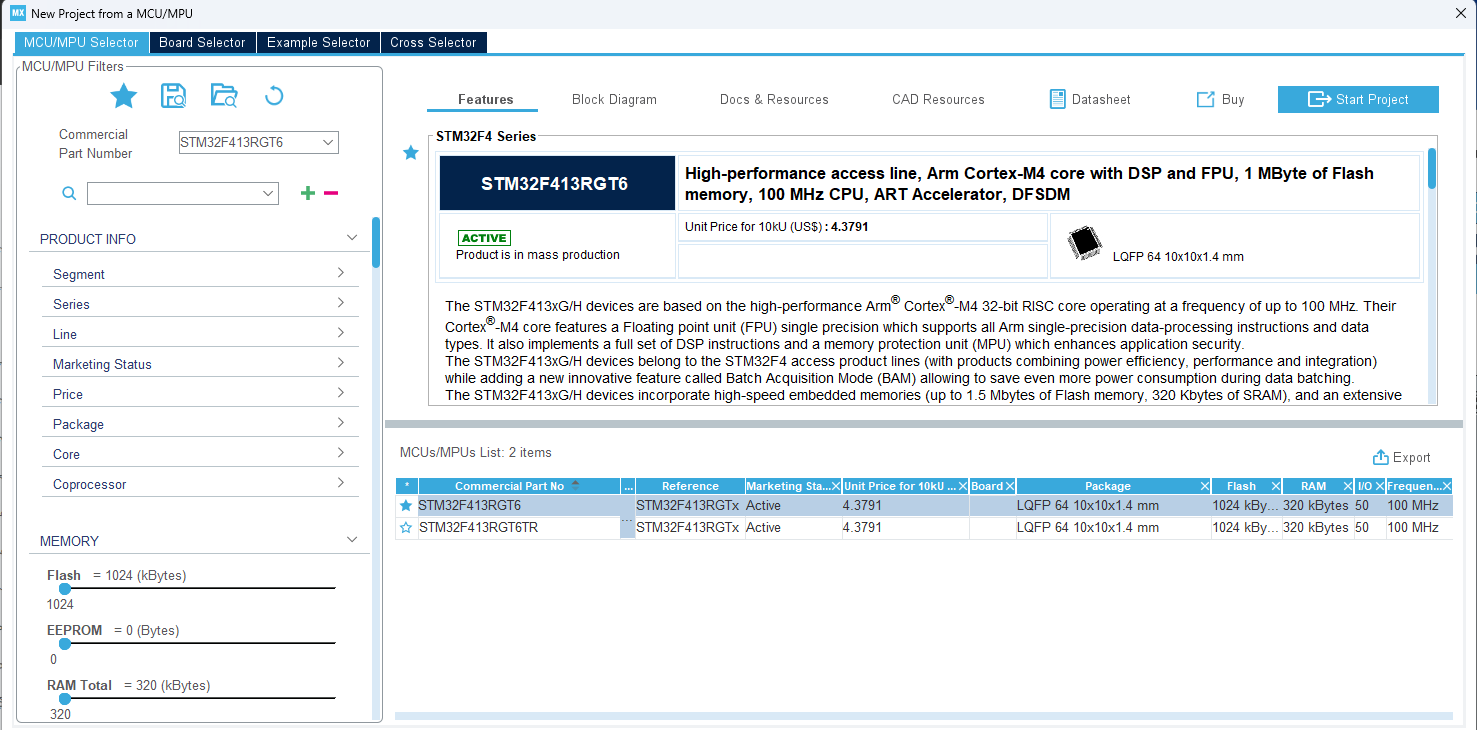
[五、Matlab实现手势开始检测，封装函数，转C；在Keil工程中加入所封装的Matlab函数lib，在main函数中加头文件并使用 31](#_Toc10782)

[六、Matlab实现雷达信号特征提取算法，优化变量类型，封装函数，转C；在Keil工程中加入所封装的Matlab函数lib，在main函数中加头文件并使用 36](#_Toc25022)

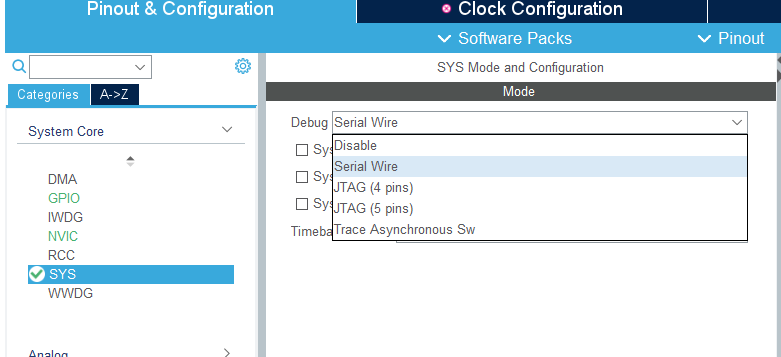
[七、在main函数while循环中设计循环逻辑，实现循环读取雷达每一帧数据，检测手势开始、开始后存取15帧matlab处理后的手势特征，送入手势识别模型中进行预测，通过虚拟串口实时打印结果。 36](#_Toc28662)

# 一、使用STM32cubeMX 创建ARM项目，引入X-CUBE-AI包，导入深度学习模型网络，自动生成项目代码

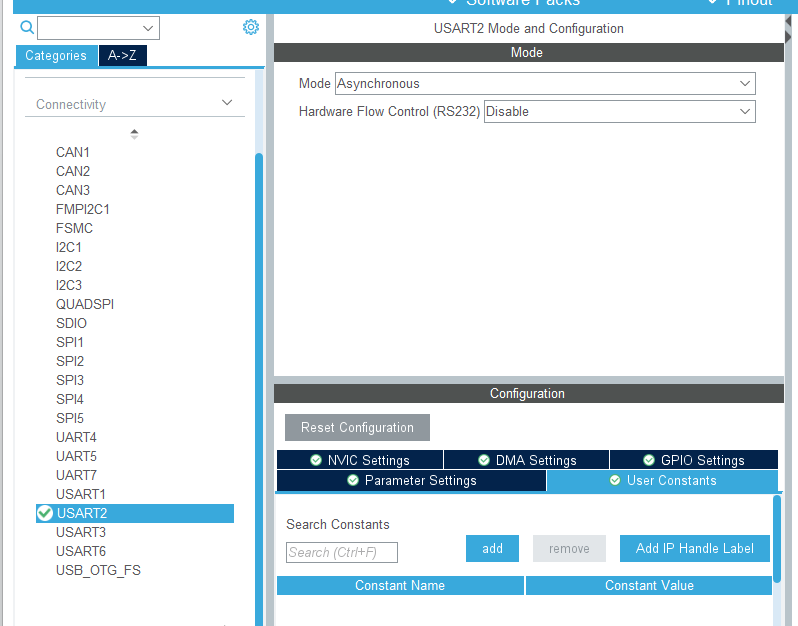
首先选择MCU，选择板子对应型号，start project



之后设置参数

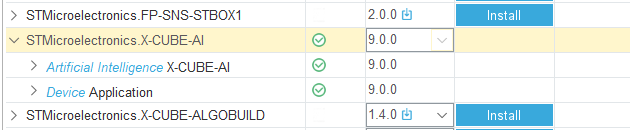


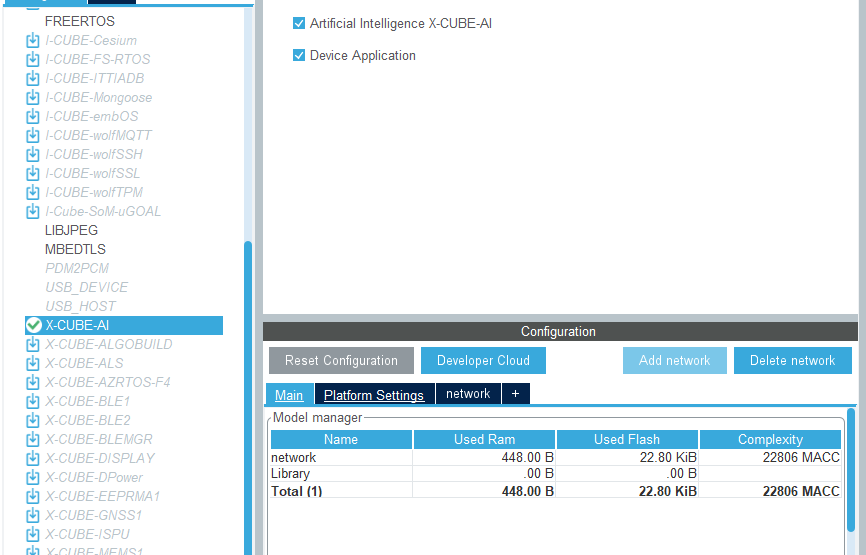
SYS 选择Serial Wire



connectivity中 USART设置Asynchronous

之后跳出时钟设置，默认即可

之后看Middleware中有X-CUBE-AI，若没有点击上面 Software Packs->Select Packs，选择X-Cube-AI



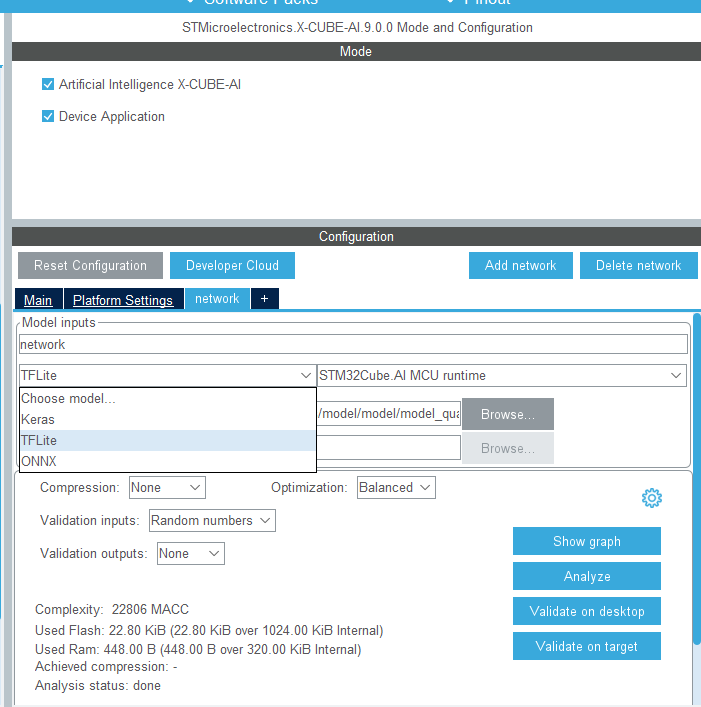
之后导入模型网络，详细说明一下参数设置。

当我们想要添加模型时，点击Add network，会创建一个参数设置界面

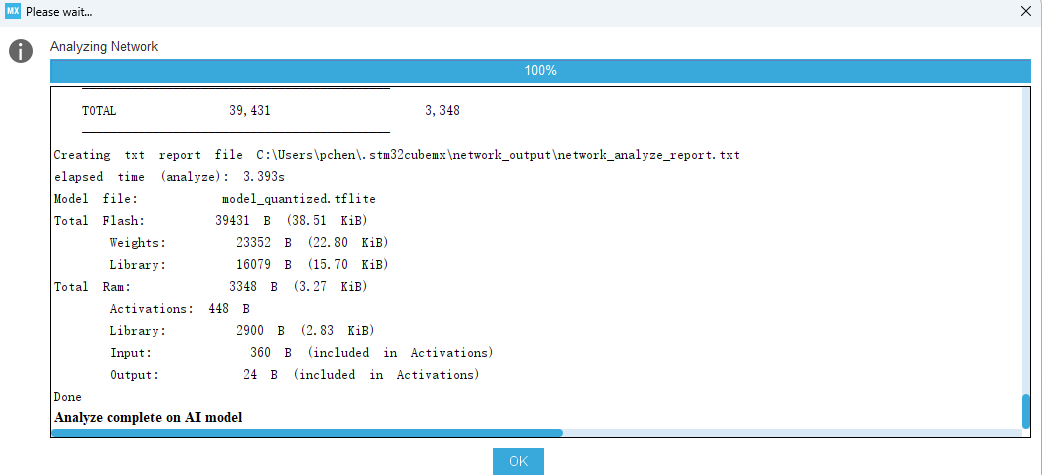
第一行导入网络的命名

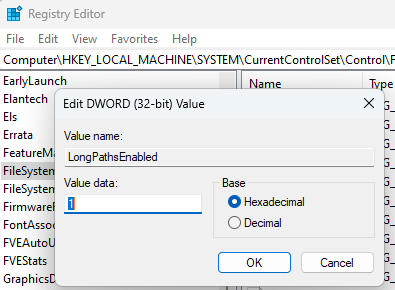
第二行选择是Keras对应.h5模型，选择TFLite对应.tflite模型。（注意：此处若选择.h5模型可以在Compression和Optinization中选择模型优化。）

导入之后通过Show graph可以看模型网络结构

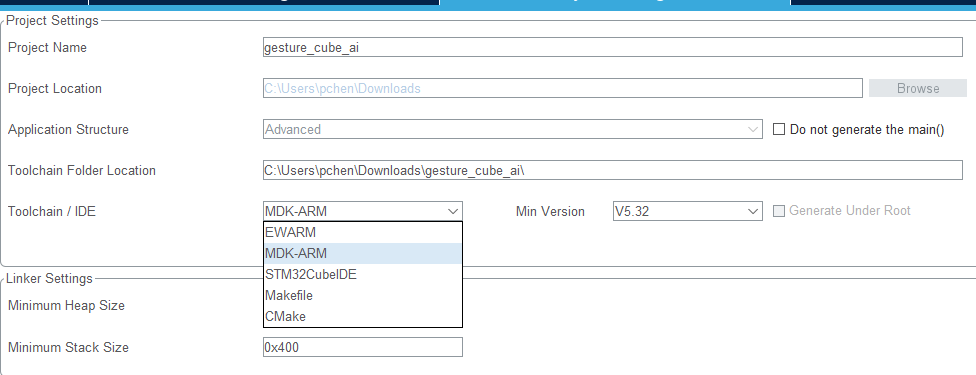


之后点击Analyze，会自动分析模型所需：计算复杂度、占用闪存，内存大小。（Windows跑遇到不接受长语句问题？? 解决：注册表FileSystem中LongPathsEnabled设为1）





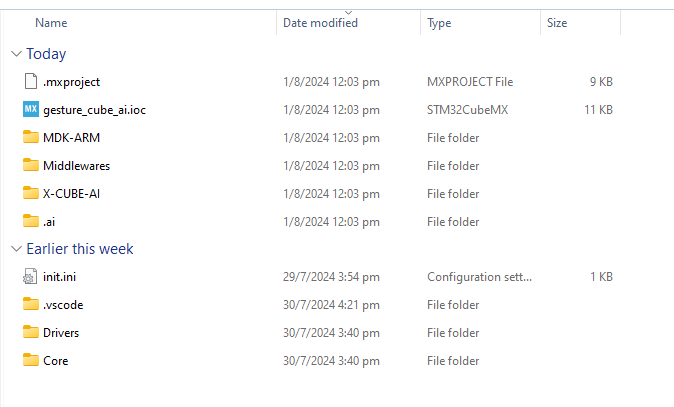
完成设置后



在project manager中选择生成MDK-ARM，修改项目命名及路径。

点击Generate Code

生成目录如下图：



之后若想添加或更改设置，点击配置文件.ioc即可

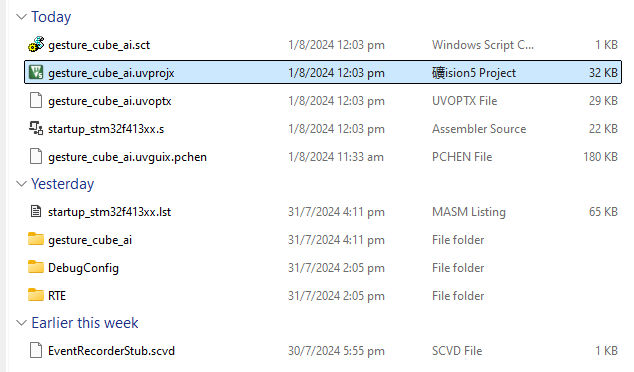
Core为main.c所在位置，主程序入口

Drivers为板子型号对应的依赖文件

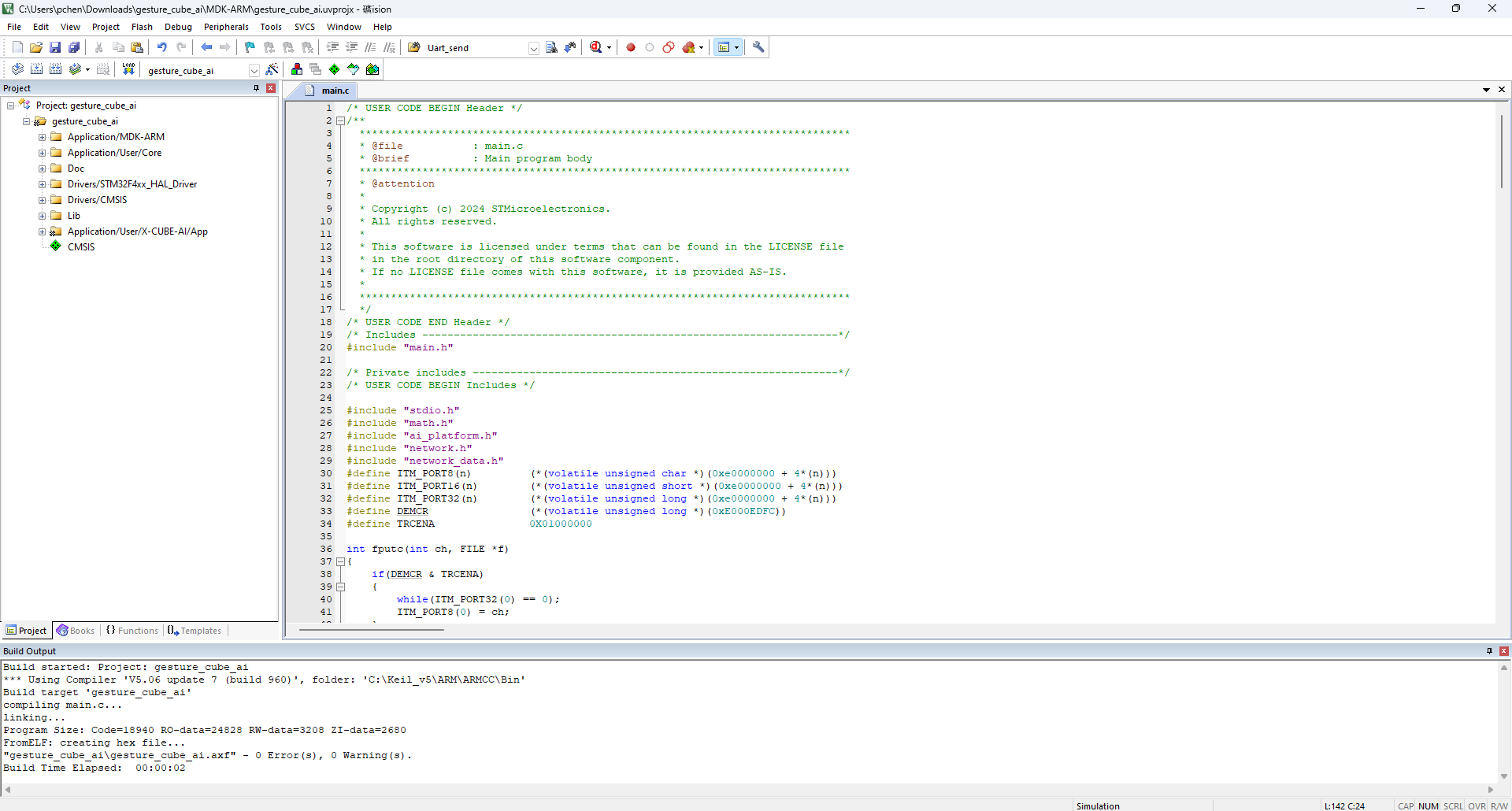
Middlewares包含ai依赖文件

X-CUBE-AI内包含主要的AI相关方法，主要由<network name>.c和<network name>\_data.c定义

打开MDK-ARM

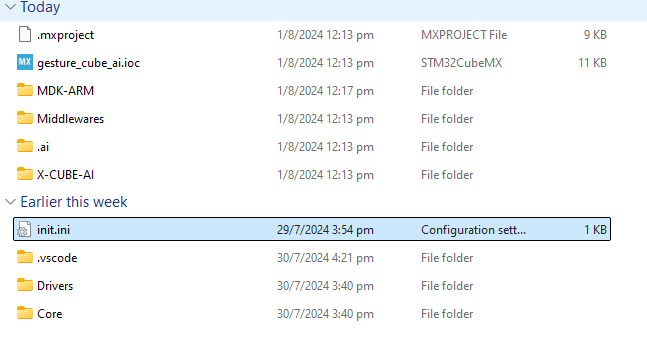


用keil打开工程文件，进入界面，ok~



# 二、使用keil模拟，‘use simulator’，实现没有板子软件内模拟debug，实现最简单sine曲线例子。

1. **首先设置Simulator环境，用keil模拟debug。**

首先在项目文件夹下创建init.ini

复制如下内容：

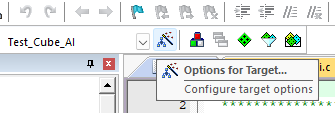
map 0x40000000,0x4000ffff read write

map 0x40010000,0x4001ffff read write

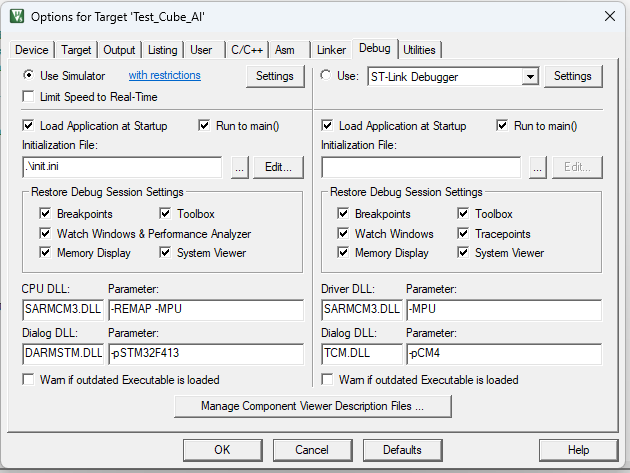
map 0x40020000,0x4002ffff read write

map 0x48000000,0x4800ffff read write

之后点击这个魔术棒



点击debug标签，按如下依次设置



选择Use Simulator

Initialization File选择刚刚创建的init.ini

下面 Dialog DLL 改成 DARMSTM.DLL，参数改成-pSTM32F413项目对应的板子型号

保存

环境设置Ok~

1. **下面开始修改代码，主要修改main.c**

修改头文件，删掉“app\_x-cube-ai.h”这是他X-CUBE-AI库给我们封装的API，我们直接用AI库原始的函数，实际上“app\_x-cube-ai.h”也是调用这些函数。

加入

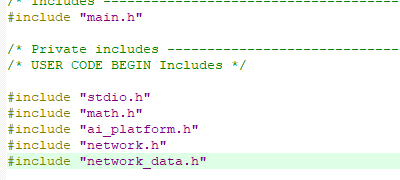
#include "stdio.h"

#include "math.h"

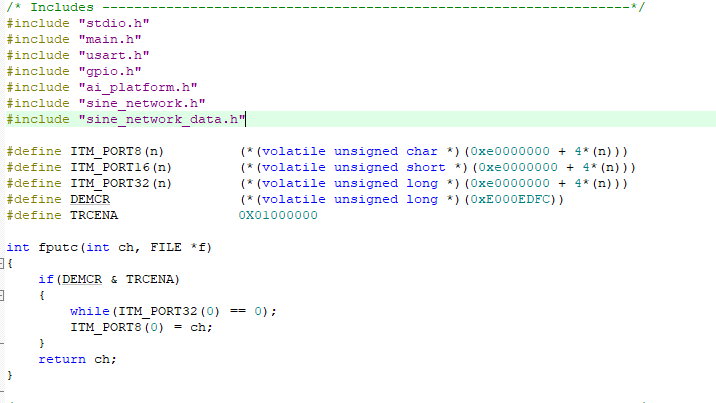
#include "ai\_platform.h"

#include "network.h"// AI\_process

#include "network\_data.h"// AI\_Init



首先在main.c最开始加入如下代码



<https://gitcode.csdn.net/662b46979ab37021bfb1a1b8.html?dp_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6NjUxNDg0NSwiZXhwIjoxNzIyNTAwNDE1LCJpYXQiOjE3MjE4OTU2MTUsInVzZXJuYW1lIjoid2VpeGluXzQ1OTI5NTU4In0.WvFeJBu7A8kkDB6Srd4s4xRxOP1vCnRML09Xz_vBLo4>

实现printf打印

#define ITM\_PORT8(n) (\*(volatile unsigned char \*)(0xe0000000 + 4\*(n)))

#define ITM\_PORT16(n) (\*(volatile unsigned short \*)(0xe0000000 + 4\*(n)))

#define ITM\_PORT32(n) (\*(volatile unsigned long \*)(0xe0000000 + 4\*(n)))

#define DEMCR (\*(volatile unsigned long \*)(0xE000EDFC))

#define TRCENA 0X01000000

int fputc(int ch, FILE \*f)

{

if(DEMCR & TRCENA)

{

while(ITM\_PORT32(0) == 0);

ITM\_PORT8(0) = ch;

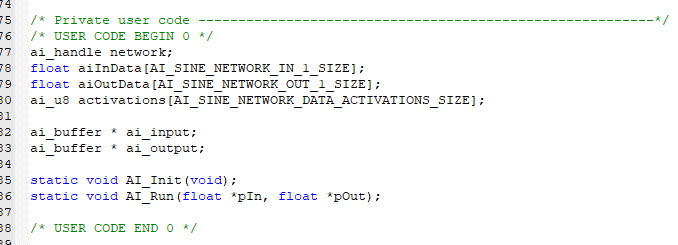
}

return ch;

}

主要功能是修改printf打印底层函数，让我们能够在Debug窗口看到。若报错函数其他文件重复定义，把别的文件的函数定义注释掉。

之后在main函数前定义如下变量及声明函数



主要作用包括定义网络变量ai\_handle，定义输入输出数组，初始化时所需的激活数组，大小均已根据模型analyse时结果自动生成了，直接用。然后是定义初始化和run两个函数。

之后，删掉

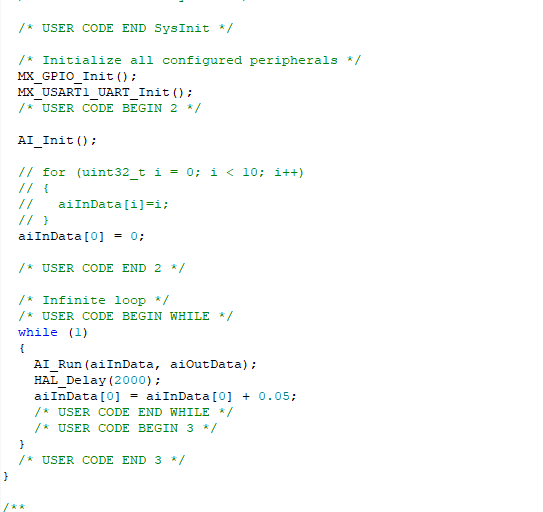
void MX\_X\_CUBE\_AI\_Init(void)

和

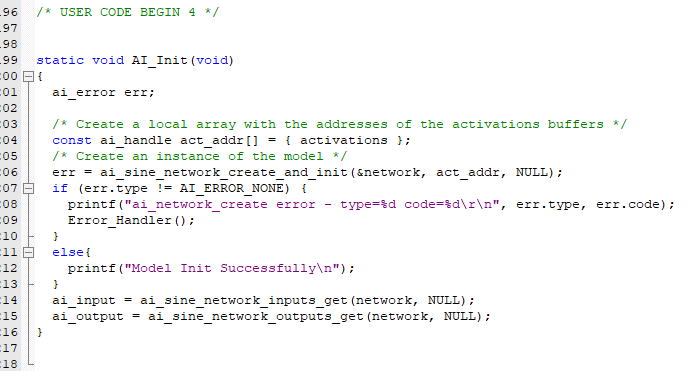
void MX\_X\_CUBE\_AI\_Process(void)

我们不用这个，用自己写的

main函数中和while(1)中代码如下：



接着在USER CODE 4定义我们自己的函数 AI\_Init 和AI\_Run



需要关注 ai\_network\_create\_and\_init(),他本质上是调用了ai\_network\_create()创建网络，用

ai\_network\_data\_weights\_get(void)以及ai\_network\_data\_params\_get()这俩给网络参数

static void AI\_Init(void)

{

ai\_error err;

/\* Create a local array with the addresses of the activations buffers \*/

const ai\_handle act\_addr[] = { activations };

/\* Create an instance of the model \*/

err = ai\_sine\_network\_create\_and\_init(&network, act\_addr, NULL);

if (err.type != AI\_ERROR\_NONE) {

printf("ai\_network\_create error - type=%d code=%d\r\n", err.type, err.code);

Error\_Handler();

}

else{

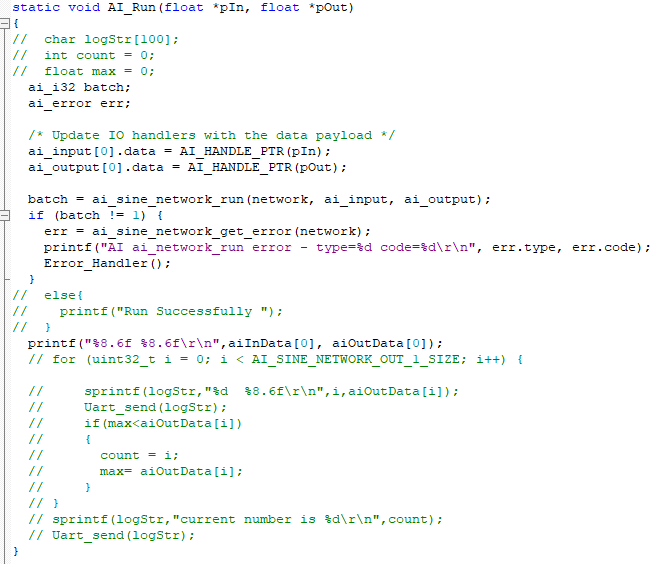
printf("Model Init Successfully\n");

}

ai\_input = ai\_sine\_network\_inputs\_get(network, NULL);

ai\_output = ai\_sine\_network\_outputs\_get(network, NULL);

}



关注ai\_network\_run()，然后输出ai\_output怎么写，怎么打印就行。

static void AI\_Run(float \*pIn, float \*pOut)

{

// char logStr[100];

// int count = 0;

// float max = 0;

ai\_i32 batch;

ai\_error err;

/\* Update IO handlers with the data payload \*/

ai\_input[0].data = AI\_HANDLE\_PTR(pIn);

ai\_output[0].data = AI\_HANDLE\_PTR(pOut);

batch = ai\_sine\_network\_run(network, ai\_input, ai\_output);

if (batch != 1) {

err = ai\_sine\_network\_get\_error(network);

printf("AI ai\_network\_run error - type=%d code=%d\r\n", err.type, err.code);

Error\_Handler();

}

// else{

// printf("Run Successfully ");

// }

printf("%8.6f %8.6f\r\n",aiInData[0], aiOutData[0]);

// for (uint32\_t i = 0; i < AI\_SINE\_NETWORK\_OUT\_1\_SIZE; i++) {

// sprintf(logStr,"%d %8.6f\r\n",i,aiOutData[i]);

// Uart\_send(logStr);

// if(max<aiOutData[i])

// {

// count = i;

// max= aiOutData[i];

// }

// }

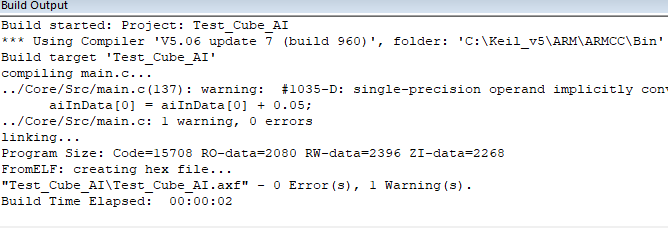
// sprintf(logStr,"current number is %d\r\n",count);

// Uart\_send(logStr);

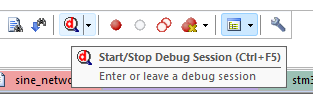
}

别的都不用管，完成

之后编译Build

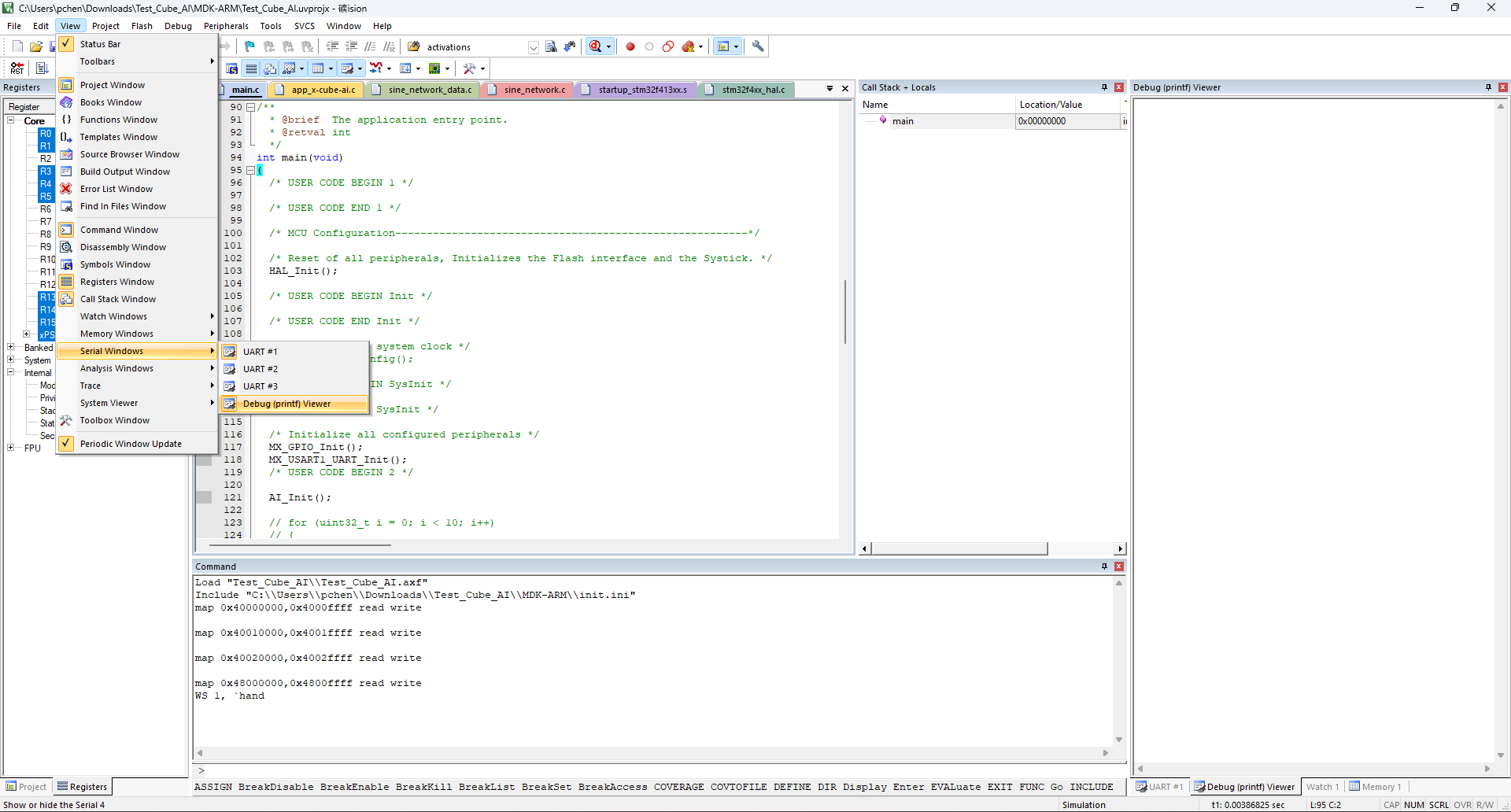


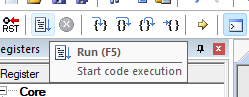
编译完后点击上方debug按钮



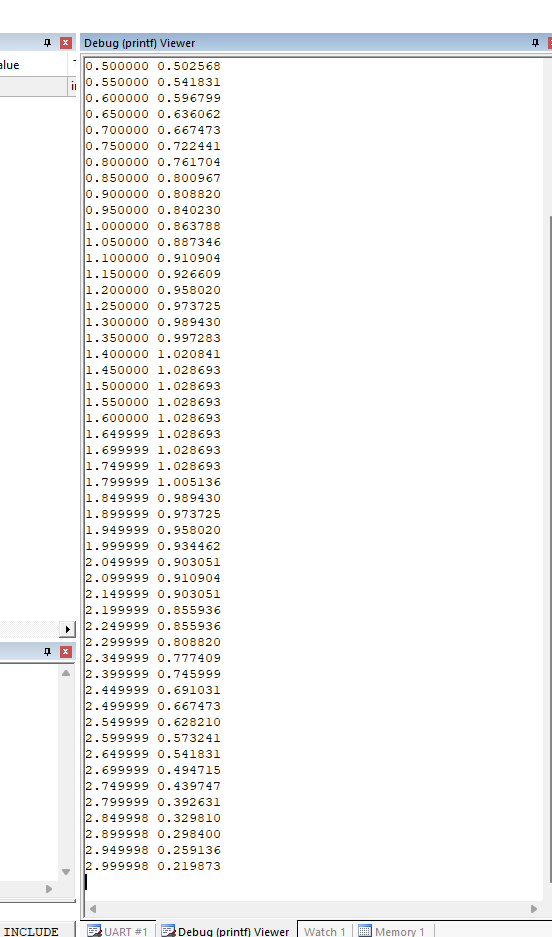
进入debug界面后在view，serial window中打开Debug(printf) Viewer，之后可进行单步调试看哪里卡了。

(若进行没板子的仅软件模拟需要注释掉时钟初始化那一句，要不然debug不动，如果你卡在SystemClock\_Config();了，退出debug，注释了编译再进吧)





之后可直接点击Run，让我们的sine函数预测模型跑数据



（跑到6.300000 -0.062821就没了，因为之后的模型预测不了，训练只用了2pi内的数据）

右键copy，新建txt文件存数据，记得删掉第一行的 ‘successfully’

将模型打印数据存入txt文件中，使用python脚本/matlab读数据，画图效果如下0：

python代码如下：

import matplotlib.pyplot as plt

import numpy as np

f = open("data.txt", "r")

X, Y = [], []

lines = f.readlines()

for line in lines:

value = [float(s) for s in line.split()]

X.append(value[0])

Y.append(value[1])

print(X)

print(Y)

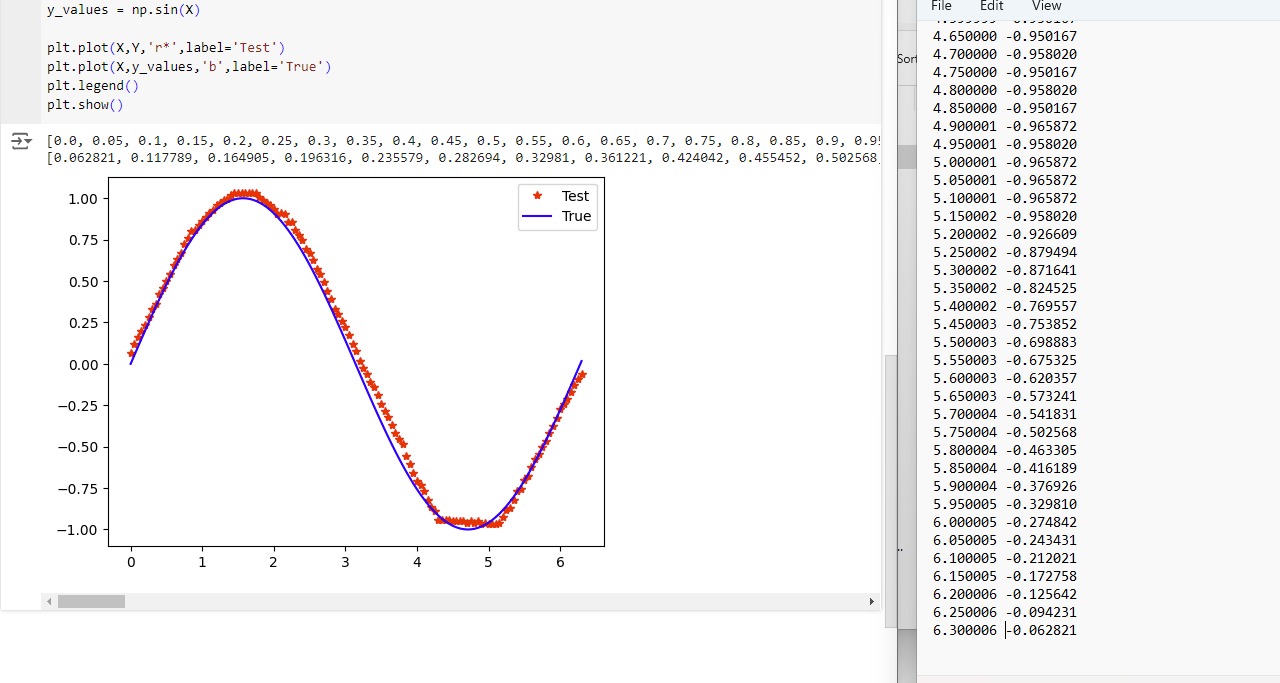
y\_values = np.sin(X)

plt.plot(X,Y,'r\*',label='Test')

plt.plot(X,y\_values,'b',label='True')

plt.legend()

plt.show()



main.c 功能实现参考例子：

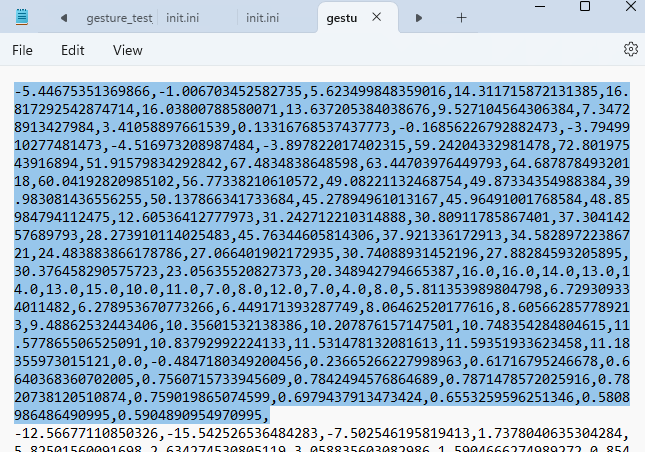
<https://github.com/colin2135/STM32G070_AI_TEST/blob/main/Core/Src/main.c>

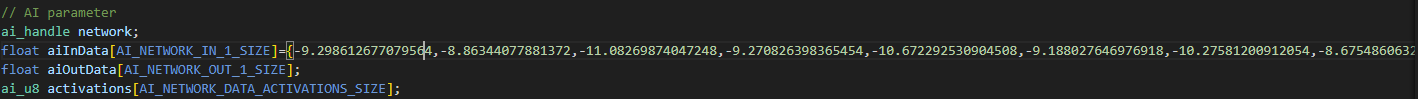
# 三、使用keil编译项目，生成.hex文件，使用stm32cubeprogrammer将文件烧录进板子，实现板子跑模型预测，将结果通过usb打印，pc端通过串口助手看usb虚拟串口打印结果。

首先通过上面的步骤，我们可以实现手势预测结果通过simulator看到，过程是一样的

不过现在一次只能预测一个，手动复制90个数据作为输入，在定义aiInData的时候定义

（直接大括号复制在里面）



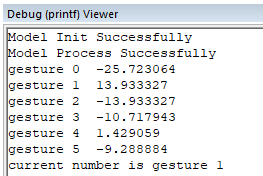


测试数据 gesture\_test\_data.txt

[1200, 90]

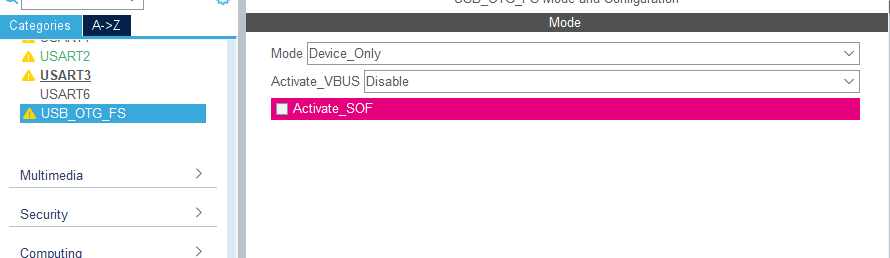
测试标签 gesture\_test\_label.txt

[1200,1]

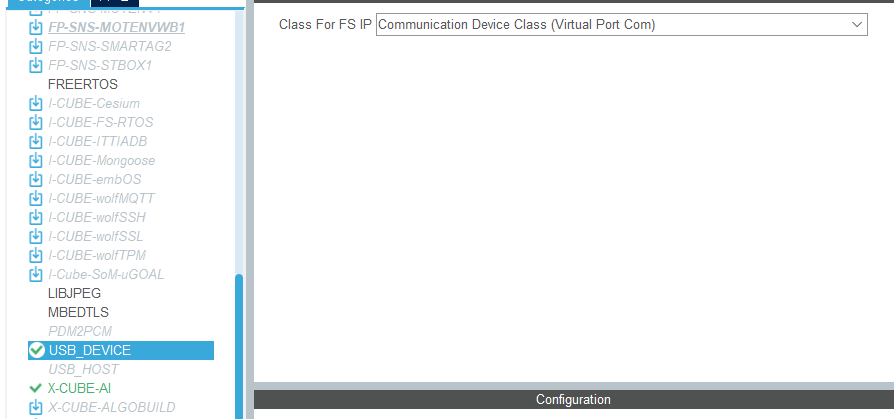


这一块主要说一下怎么烧录到板子并实现USB打印：

connectivity里加这个

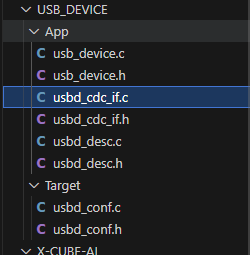


Middleware里加这个

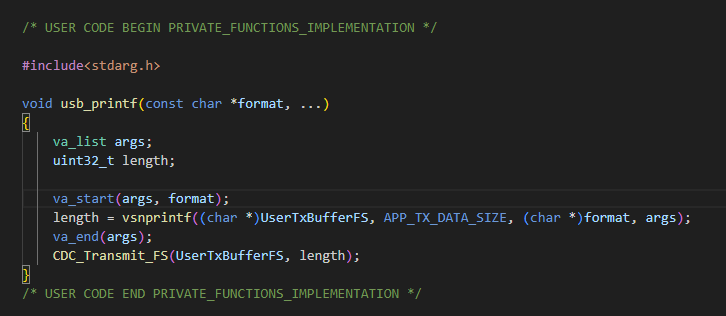


主要是需要我们加入USB\_DEVICE，并设置成虚拟串口

生成代码后，会出现如下，此时我们需要实现usb\_printf函数



在usbd\_cdc\_if.c最后加如下代码



#include<stdarg.h>

void usb\_printf(const char \*format, ...)

{

va\_list args;

uint32\_t length;

va\_start(args, format);

length = vsnprintf((char \*)UserTxBufferFS, APP\_TX\_DATA\_SIZE, (char \*)format, args);

va\_end(args);

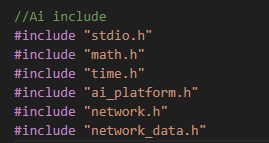
CDC\_Transmit\_FS(UserTxBufferFS, length);

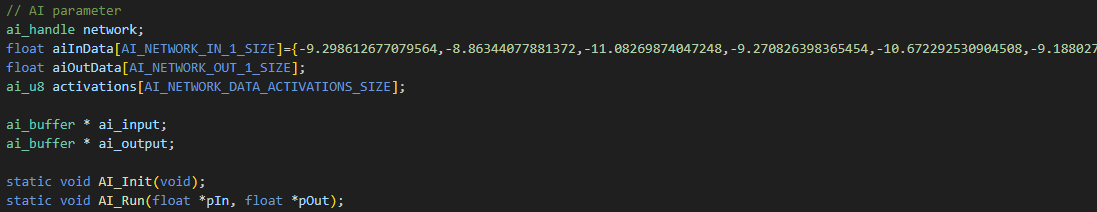
}

主要usb打印函数由uint8\_t CDC\_Transmit\_FS(uint8\_t\* Buf, uint16\_t Len)这个函数来实现

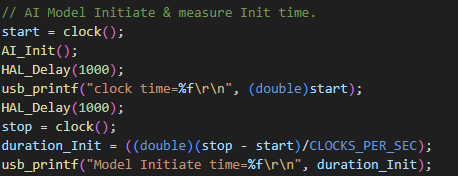
这里推荐可以直接把前面AI的东西加入到现成的雷达数据采集项目上，这样比较轻车熟路。（我想只加usb模块没跑通，估计是因为时钟之类的关系，很复杂）

主要是以下几块





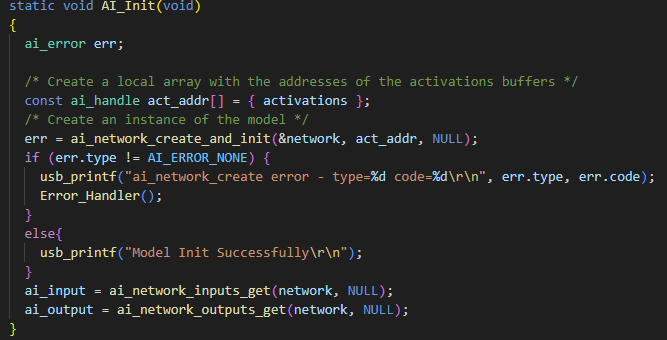
main函数里

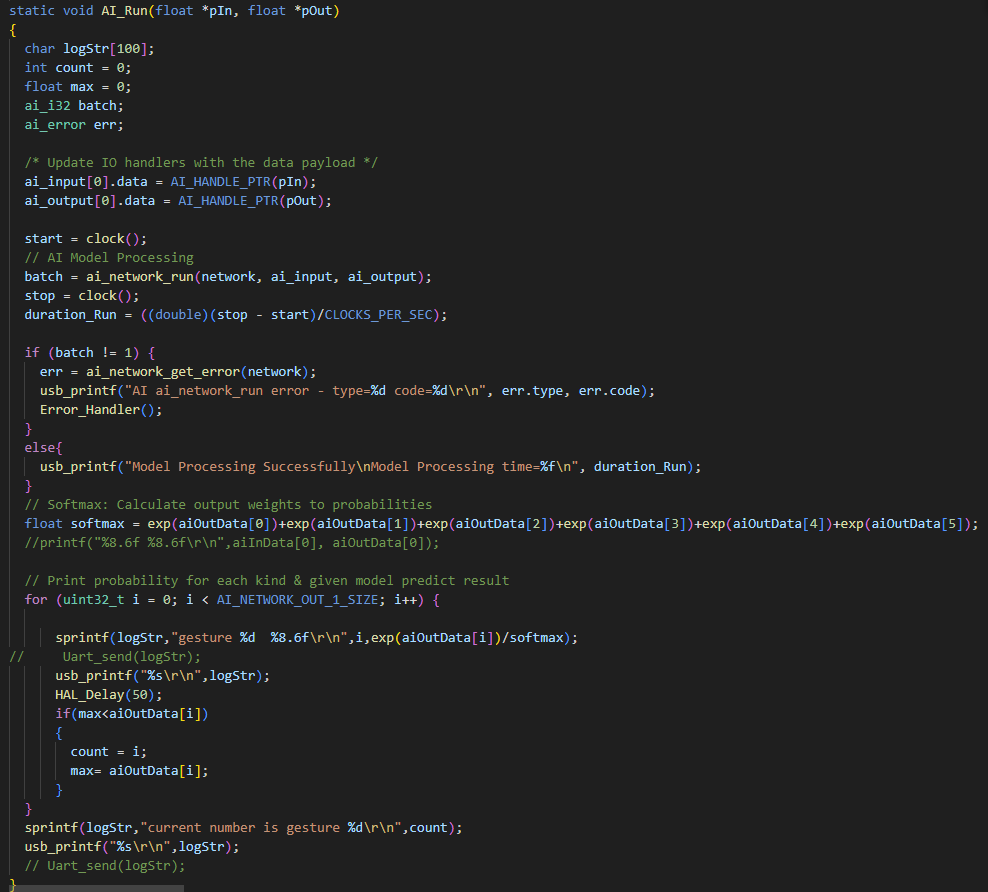




后面AI\_Init和AI\_Run的定义部分

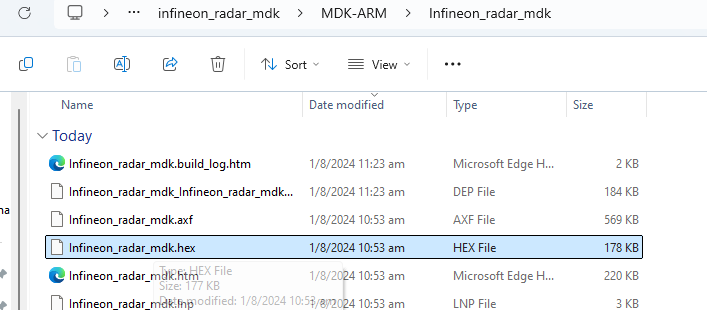
注意修改之前的printf->usb\_printf





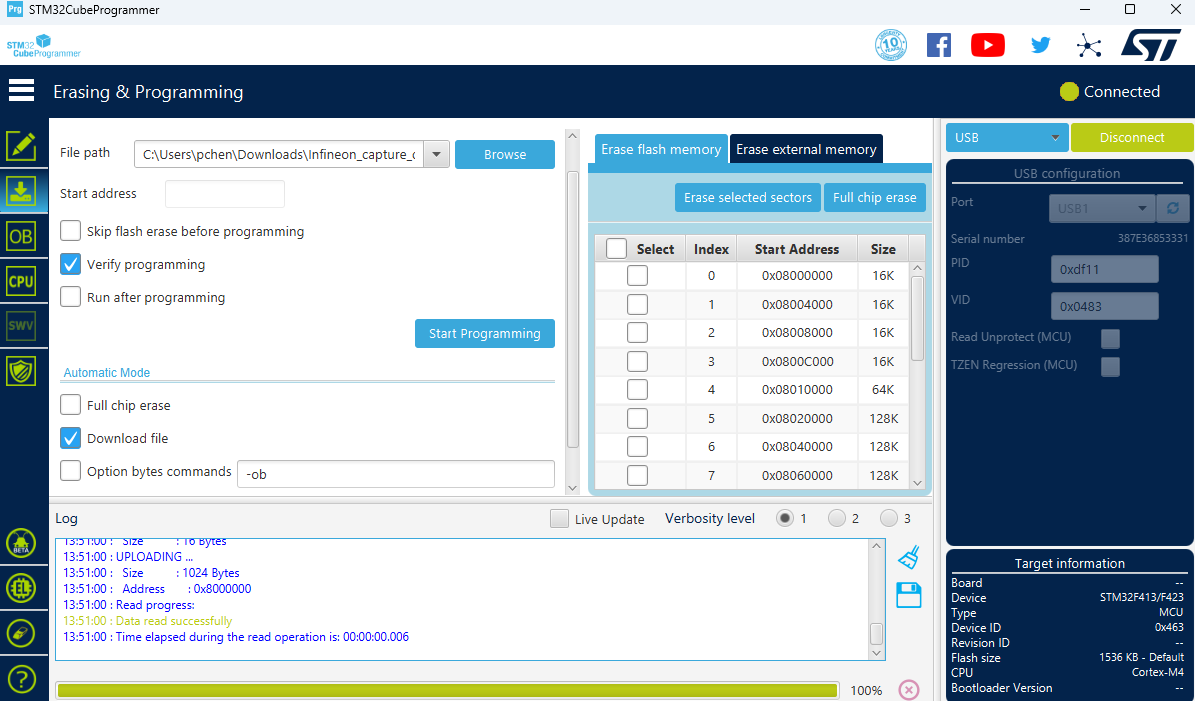
完成之后编译，准备烧录

首先keil编译完后会生成hex文件



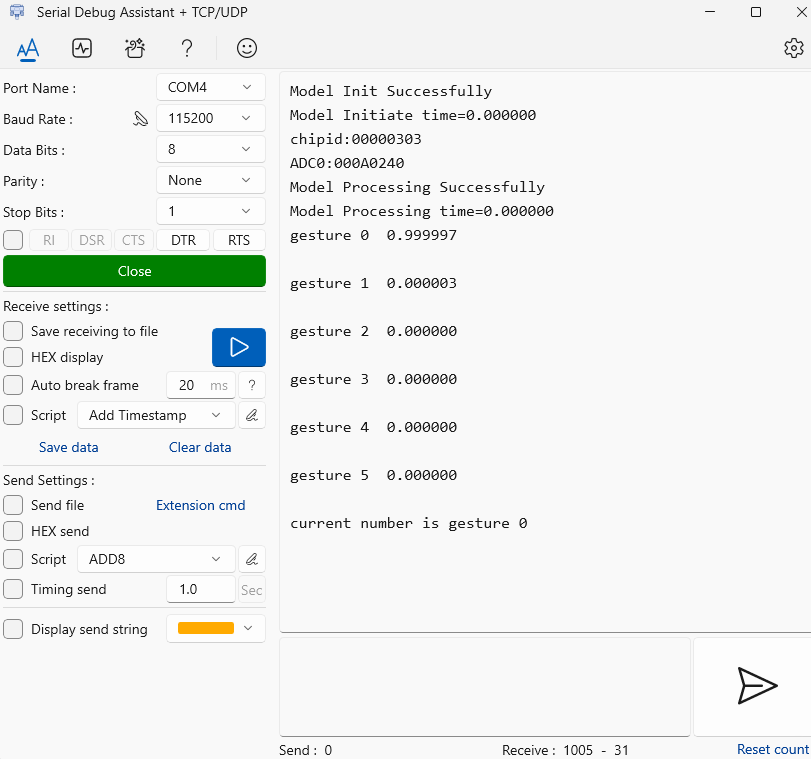
我们需要把这个hex文件烧录到板子

打开stm32cubeprogrammer，选择USB，在短路状态下板子插usb连接电脑，点connect。连接成功之后烧录hex文件。



选择hex点击start programming，成功后disconnect

让板子恢复正常工作模式，打开串口助手，即可成功看到输出

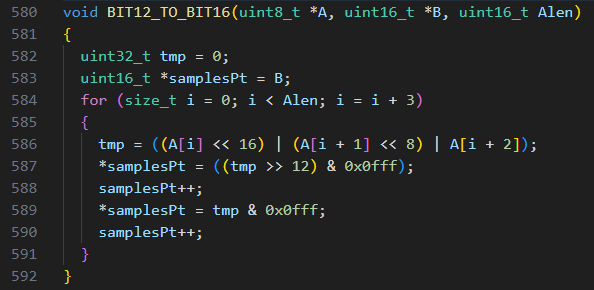


USB CDC Virtual COM printf

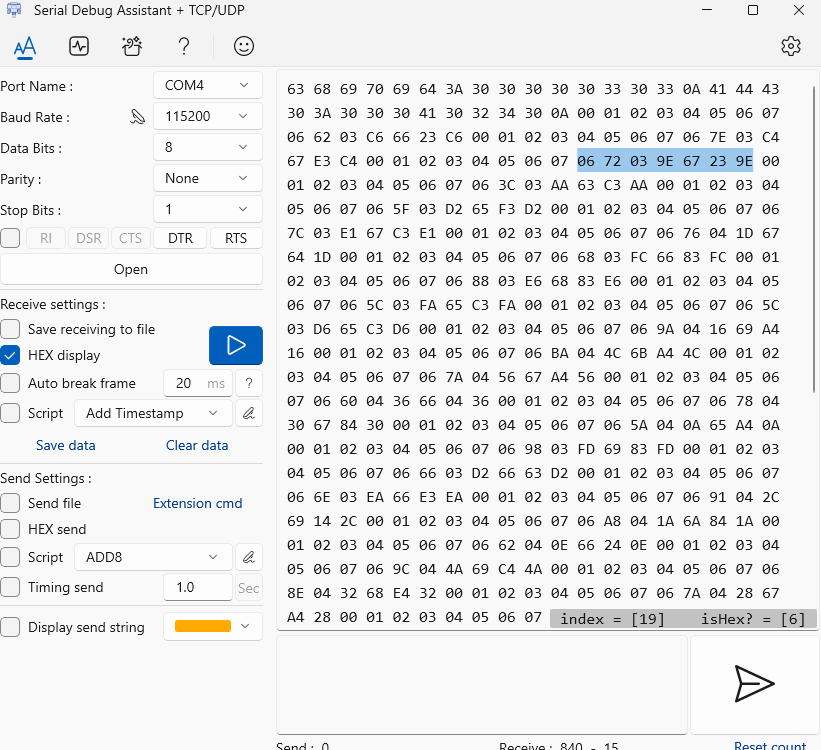
<https://blog.csdn.net/sudaroot/article/details/86627853>

# 四、雷达数据采集及BIT12转BIT16处理

雷达原始数据为BIT12，实际上每1.5个数据代表一个真实数据，即雷达数据的每三个数实际上是两个可以用的数，要把中间的数据从中间拆开，设计一个BIT12 转 BIT16的函数，涉及位运算。



Checking BIT12\_to\_BIT16 function is correctly working.



验证bit12 转换bit16：

思路主要为打印雷达原始数据的3个数和转bit16之后的4个数，进行对比

// Check bit12 to bit16 is right

testArray[0] = (resultArray[(frame\_idx % NUM\_FRAMES\_BUFFER) \* NUM\_SAMPLES\_PER\_FRAME] >> 8) & 0xff;

testArray[1] = resultArray[(frame\_idx % NUM\_FRAMES\_BUFFER) \* NUM\_SAMPLES\_PER\_FRAME] & 0xff;

testArray[2] = (resultArray[(frame\_idx % NUM\_FRAMES\_BUFFER) \* NUM\_SAMPLES\_PER\_FRAME+1] >> 8) & 0xff;

testArray[3] = resultArray[(frame\_idx % NUM\_FRAMES\_BUFFER) \* NUM\_SAMPLES\_PER\_FRAME+1] & 0xff;

CDC\_Transmit\_FS(headflag, 8);

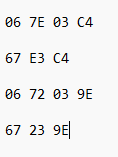
HAL\_Delay(1);

CDC\_Transmit\_FS(testArray, 4); //传输雷达数据

HAL\_Delay(1);

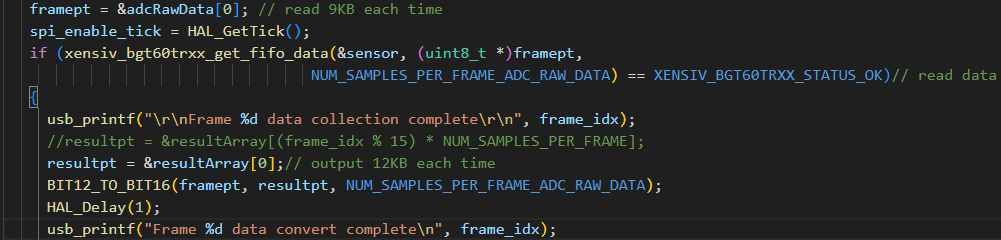
CDC\_Transmit\_FS(framept, 3); //传输雷达数据

例子如下：



可以看出正确

在main函数中while下写雷达原始数据读取和处理读取并转换每一帧adcRawData (64x32x3x1.5) resultArray (64x32x3)



# 五、Matlab实现手势开始检测，封装函数，转C；在Keil工程中加入所封装的Matlab函数lib，在main函数中加头文件并使用

start\_judge.m 文件

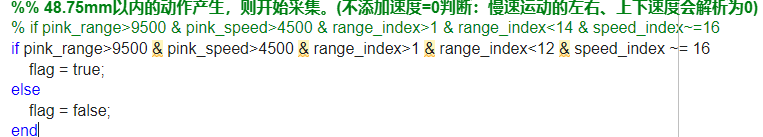
函数头：

function [flag] = start\_judge(Data)

%% 输入: Data 64x32 雷达原始数据取第三个通道Rx3，输出：flag：true/false

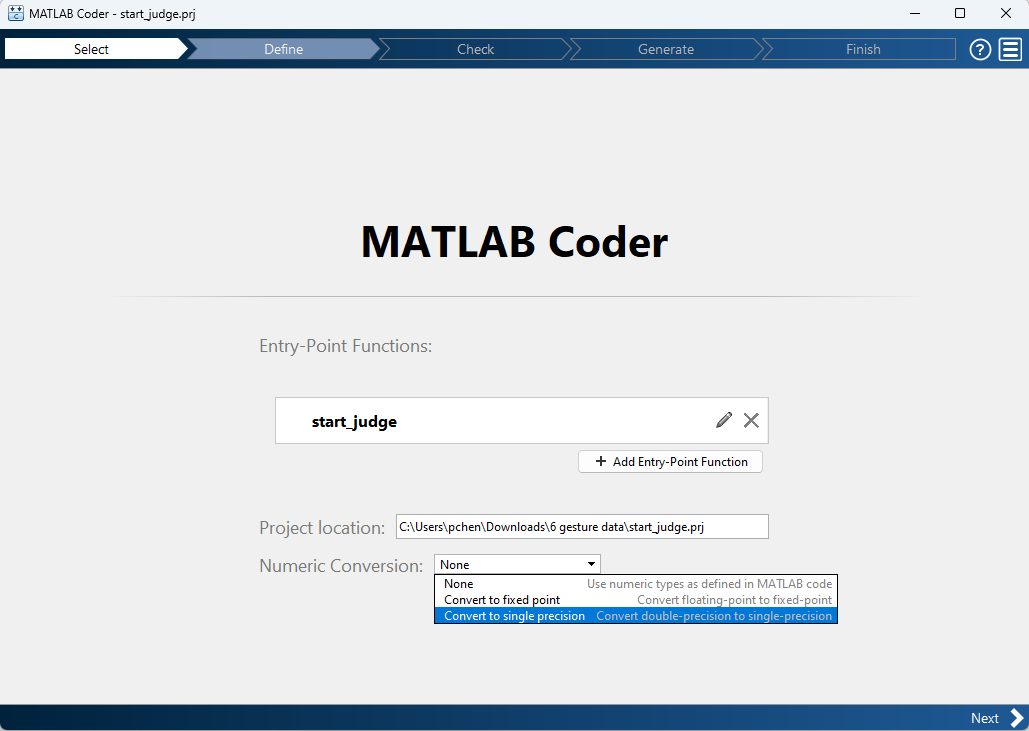
设计思路：

1. 对原始数据做距离FFT和多普勒FFT
2. 对speed\_profile按chirp维非相参积累得到 距离-时间图（32\*frame）
3. 对speed\_profile按range维非相参积累得到 速度-时间图（32\*frame）
4. 最后设计阈值判断是否有手势开始，返回flag值

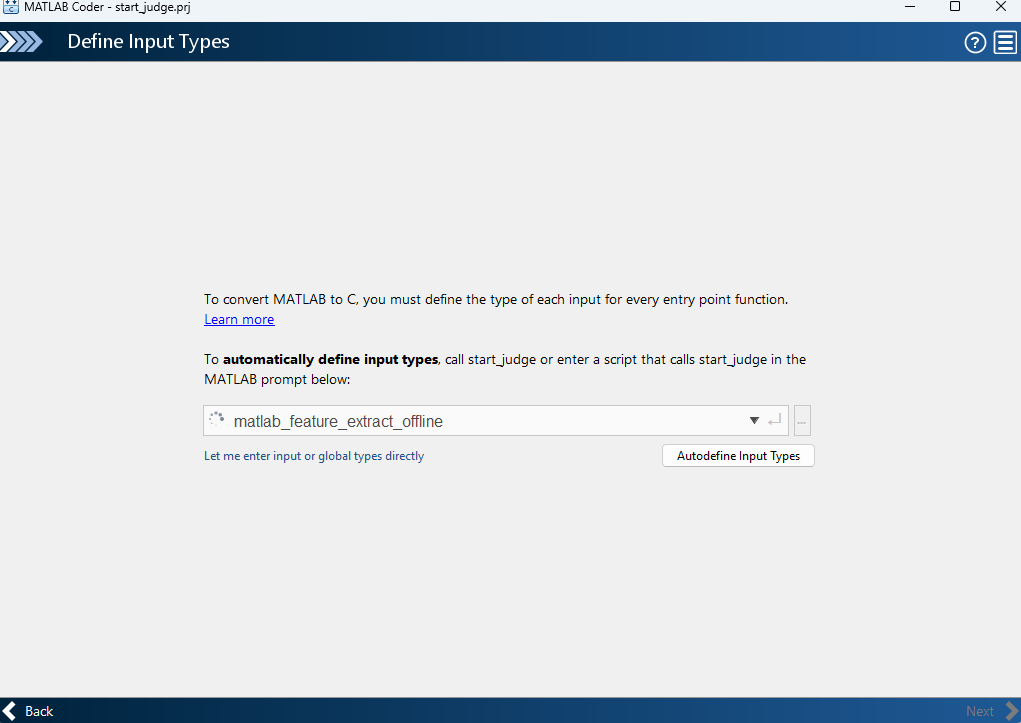


准备好后打开Matlab Coder

选择要转的函数，选择单精度



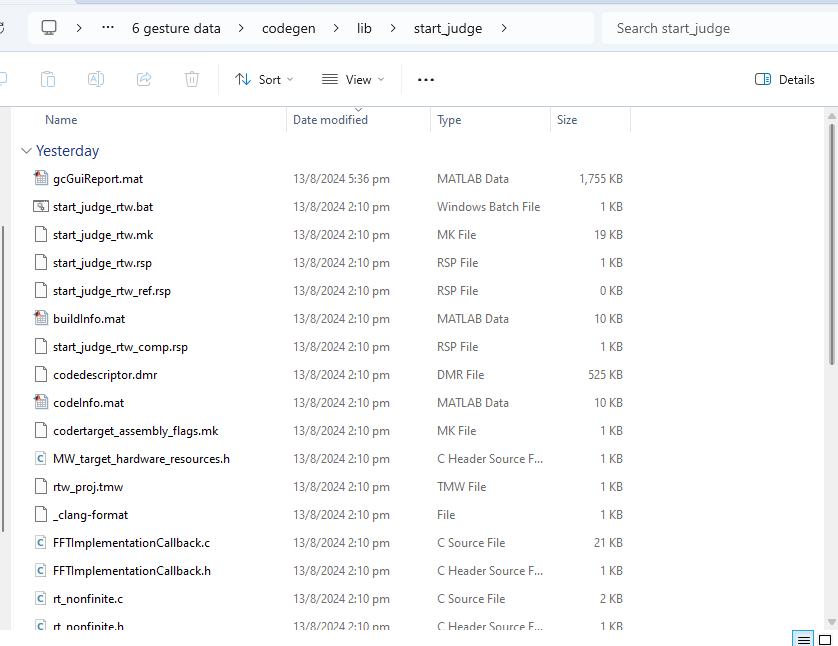
之后的页面选择入口函数，点击auto define input types



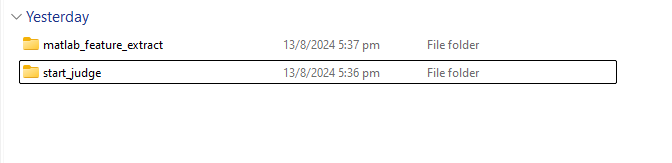
此处入口函数要单独再写一个m，简单调用我们想转的函数，只要输入数据格式给的是对的就OK，只是为了让matlab明白应该如何去调用咱们想转的函数。（此处步骤与下一步重合较多，因下一步比较重要，会写的详细，可以参考下一章步骤）

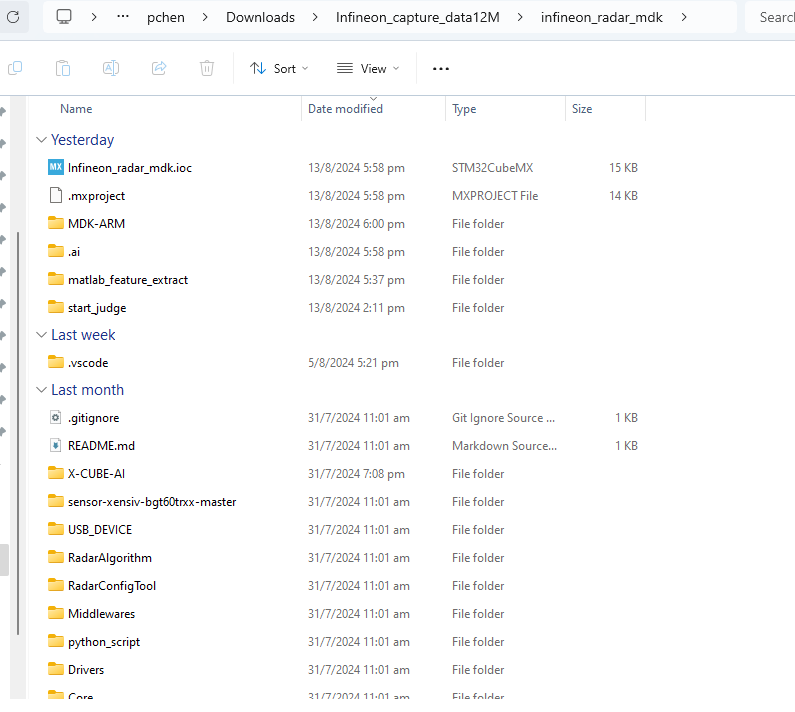
之后check是否有issue，没有就可以转了。

转完之后会生成包含该函数的lib

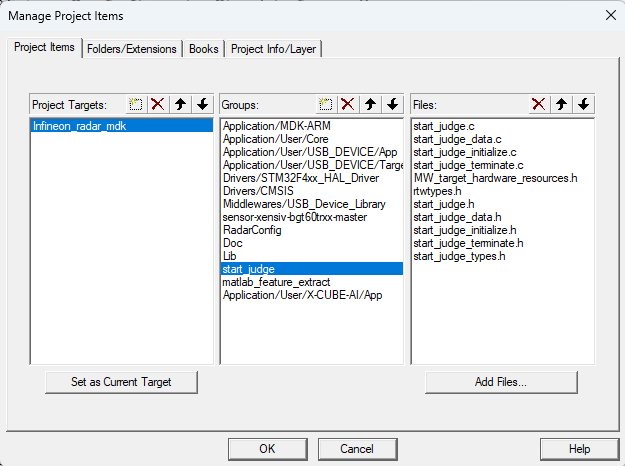


此时需要复制整个文件start\_judge到项目文件路径下

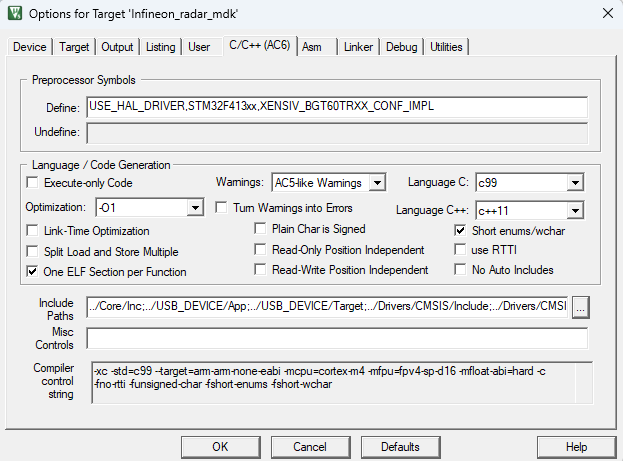




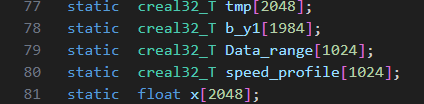
在keil中 Manage Project Items中加入该文件夹下的所有.c和.h文件



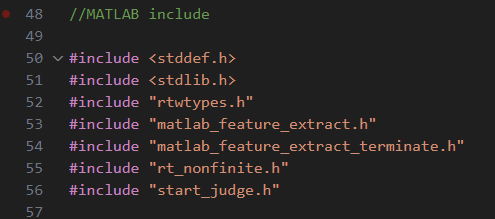
在魔棒工具—>C/C++—>Include Paths 中将start\_judge文件夹加入目标编译环境



在start\_judge.c的函数中将占用内存大的数组加static，避免放入栈中。



在main.c中加入start\_judge.h头文件



在main函数while中调用start\_judge()即可

本块中省略大多细节步骤，因与下一部分重合较多，细节请见下一部分。

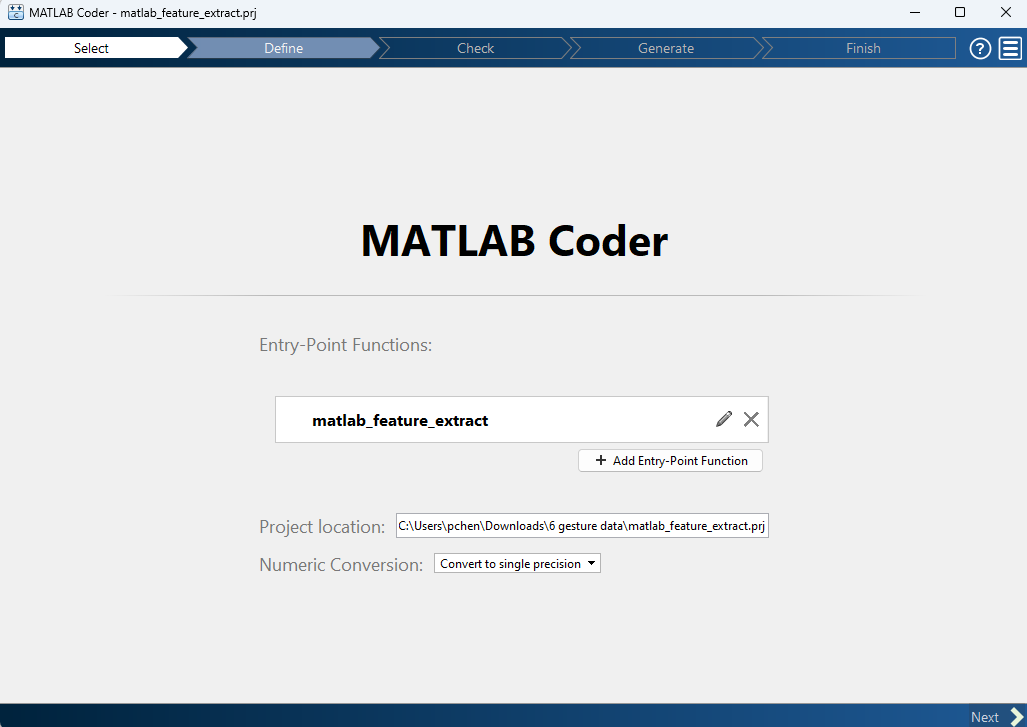
# 六、Matlab实现雷达信号特征提取算法，优化变量类型，封装函数，转C；在Keil工程中加入所封装的Matlab函数lib，在main函数中加头文件并使用

matlab\_feature\_extract.m

函数头为function [featuresMatrix] = matlab\_feature\_extract(Data, frame\_idx)

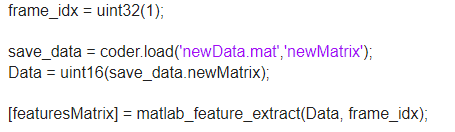
%% Data 64x32x3 当前帧数据 frame\_idx 1x1当前输入帧编号；输出featuresMatrix 1x6 输出特征

在coder中选择matlab\_feature\_extract函数，点转成单精度选项

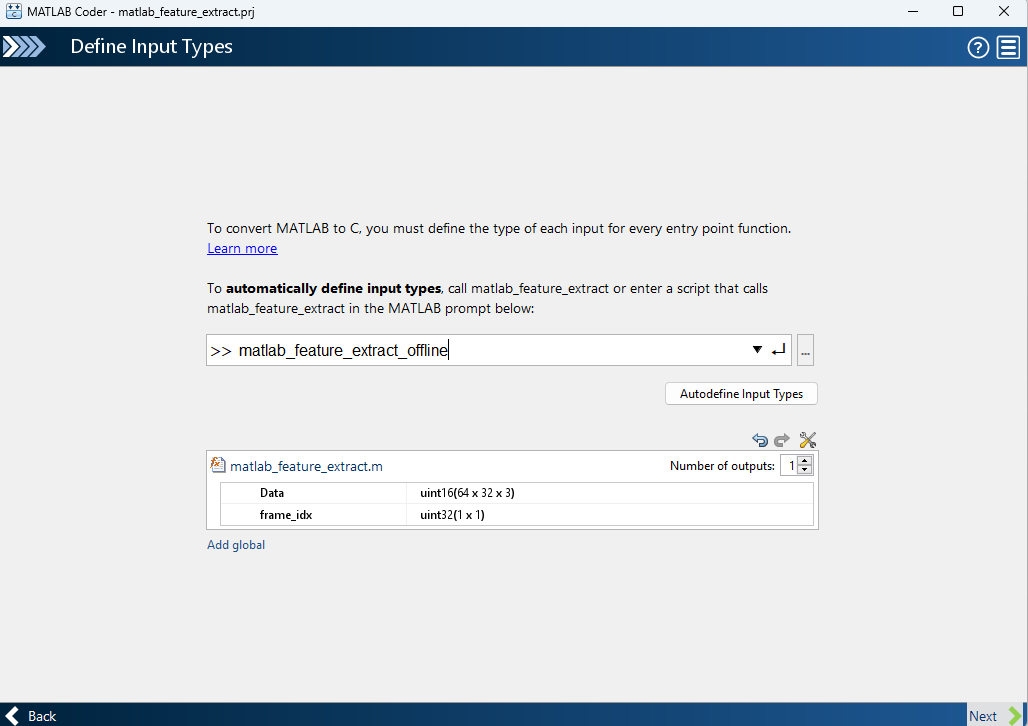


设计一个调用该函数的m文件，当作函数入口

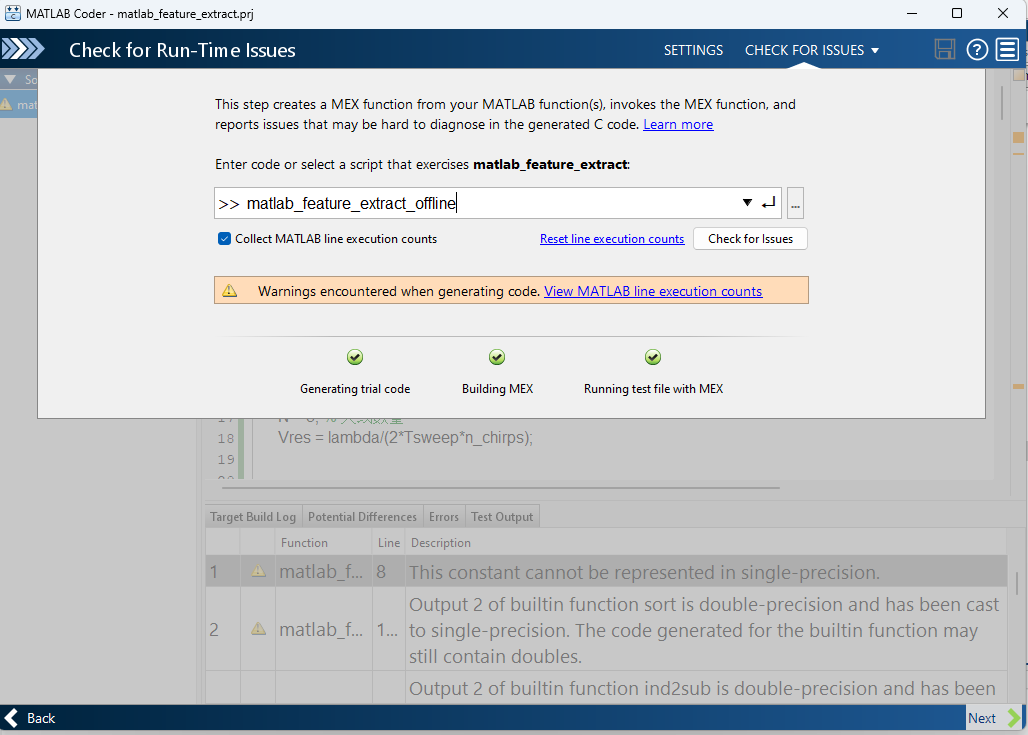
包含以下语句即可， 其中newData.mat中仅存有64x32x3的数据



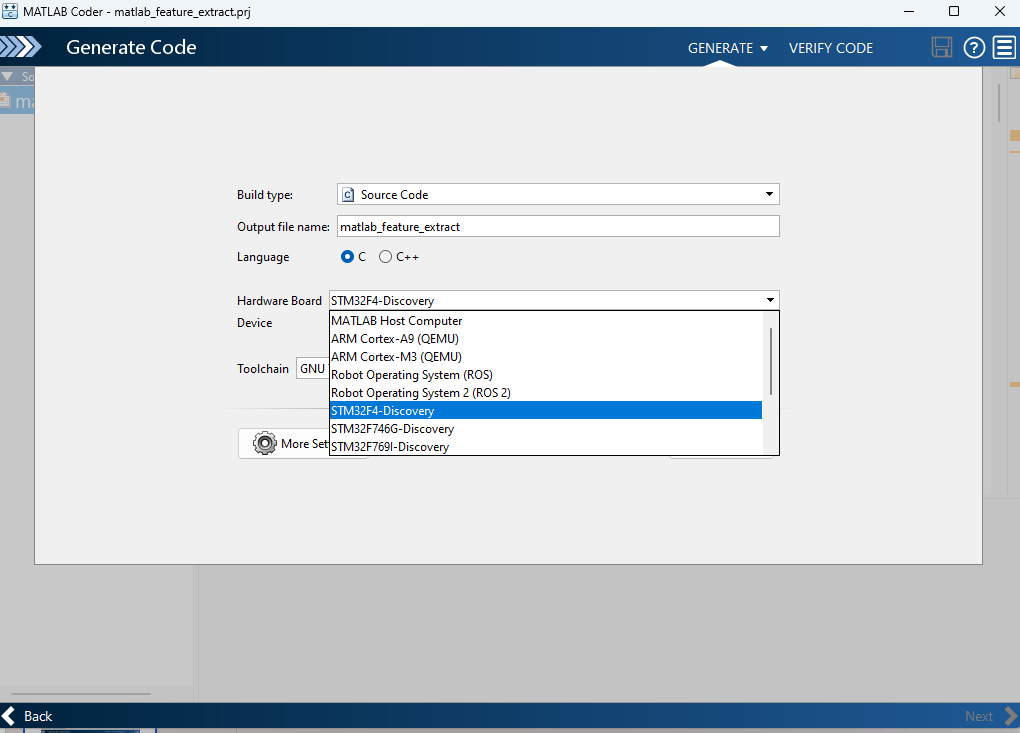
在下一步选择该文件，点击autodefine input types，matlab会自动给出输入类型、大小，正确的

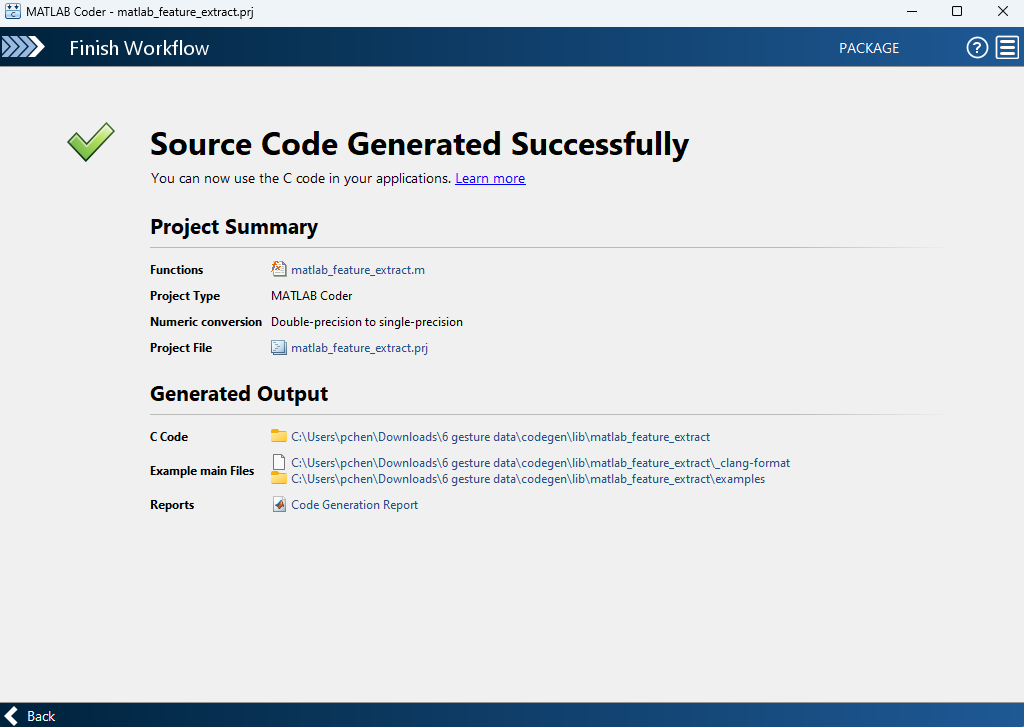


之后check for issues，没有报错即可生成c代码



下一个页面记得选择硬件平台，STM32F4

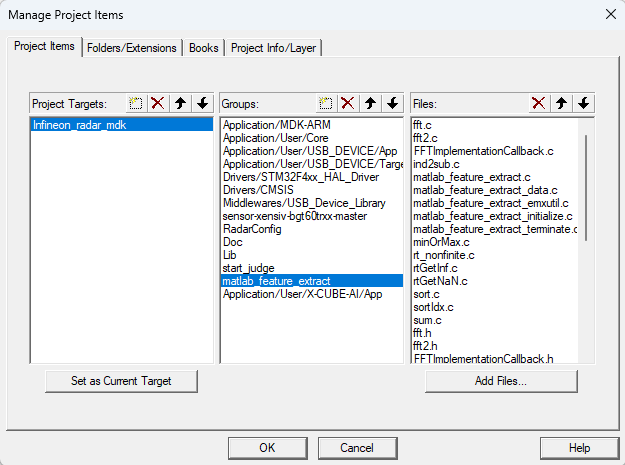




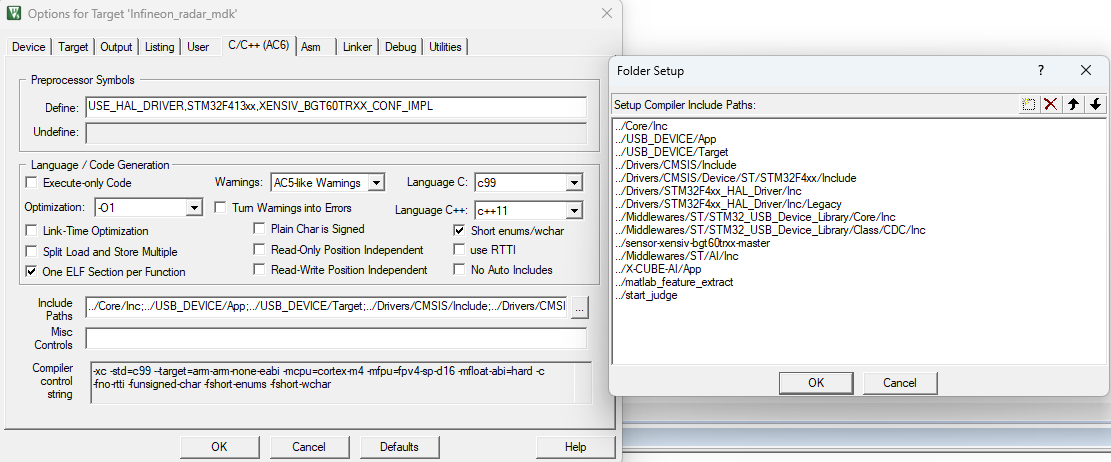
生成后将matlab\_feature\_extract文件夹复制到项目文件夹下，

并在keil中加入进编译资源和编译目标路径

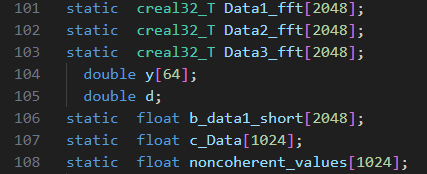
手动将matlab\_feature\_extract文件夹下所有.c .h加进来，（此处可能有些文件是和前面判断开始函数的是重合的，只能保留一个，不然编译的时候会报错说某些函数有不止一个定义）

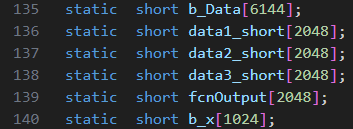


魔棒工具、C/C++、Include Paths，加matlab\_feature\_extract文件夹

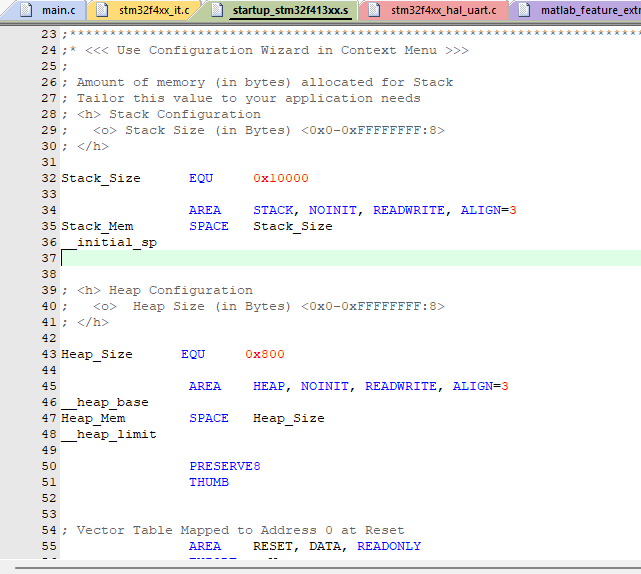


完成之后需要进入matlab\_feature\_extract.c中，将开始声明的大数组全部加上static，不然我们的内存不够，matlab\_feature\_extract()函数会卡死

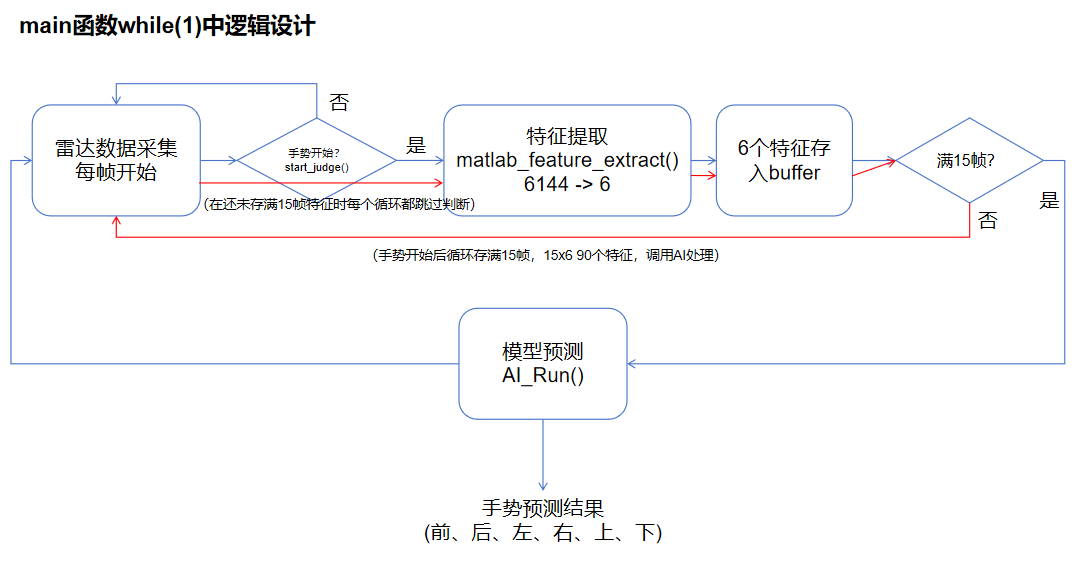




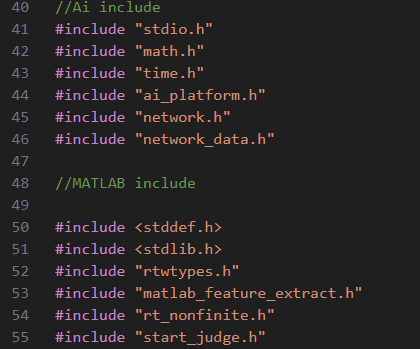
之后进入项目里stm32系统的.s文件中，修改栈的大小，stack\_size至 0x10000



# 七、在main函数while循环中设计循环逻辑，实现循环读取雷达每一帧数据，检测手势开始、开始后存取15帧matlab处理后的手势特征，送入手势识别模型中进行预测，通过虚拟串口实时打印结果。



至此，我们已经有了所有的部分，判断开始、特征提取、Ai预测，可以在原有的雷达采数据项目上实现我们的手势识别项目。只需要在main函数的while中实现我们需要的逻辑。



加入头文件，确保调用的方法能找到

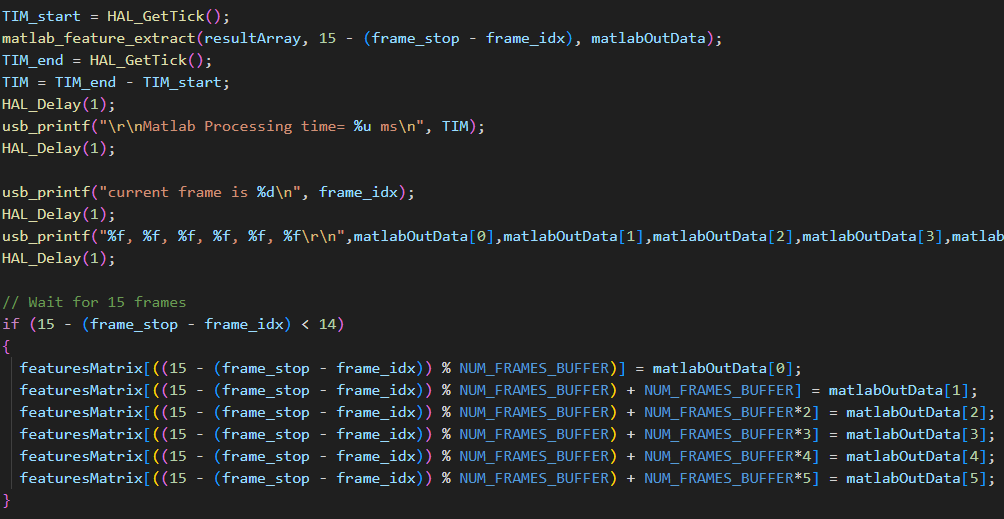
While循环中，每次开始即是每帧调用雷达fifo读取数据存到buffer中，我们前面已经实现bit12转bit16，拿到64x32x3每帧的原始数据，我们首先要用这个东西给start\_judge()看是否该帧手势开始，开始后则要跳过这个start\_judge()，连续存15帧的数据，处理完输出之后，才能回到原来状态。所以不能每次循环都调用start\_judge()。

这里我们设新变量frame\_stop,因原有变量frame\_idx每次while即每帧都会+1，当我们检测到手势开始的时候，就让frame\_stop比frame\_idx大16（需要考虑该帧，该帧结束时frame\_idx也会++），以收集之后的15帧内容。

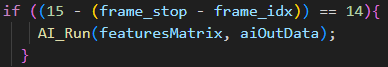
所以思路就是，在雷达fifo之后，我们先判断



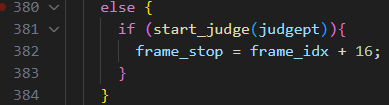
若该项成立，则进入每一帧的特征提取和存储



之后若满15帧，我们调用AI进行处理



若frame\_stop不比frame\_idx大，则进入else，在else里我们调用start\_judge去判断该帧是否为手势开始帧，输入为每帧第三个天线的64x32数据。如果返回false，则无事发生，没有手势开始进入下一个while/读下一帧；若返回true，则此时我们将frame\_stop设成比frame\_idx大16，此后循环存15帧数据并处理一次。



在每一次ai处理输出完后，下一帧将不满足frame\_stop比frame\_idx大的条件，也就是立刻返回初始状态，进else，进入是否有手势的判断。

实际测试可以完成连续的帧采集，初始为手势开始判断状态，若手势开始时进入15帧采集和预测结果输出，之后再次进入手势开始判断状态，如此循环。

模型训练时遇到了一个loss全变成nan的问题，原因：输入中包含了nan，解决：使用numpy的方法

<https://chatgpt.com/share/f7adcc06-c875-47d8-a662-4d042afc42e5>