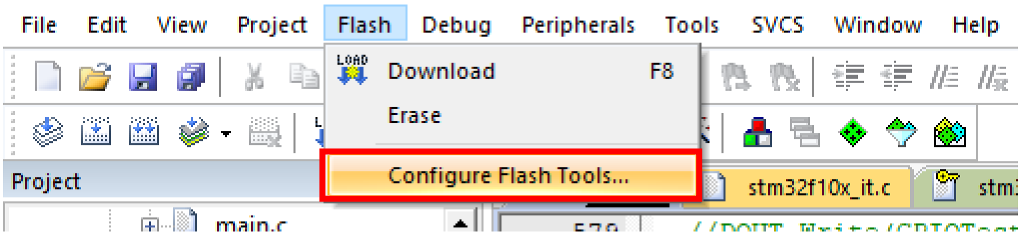
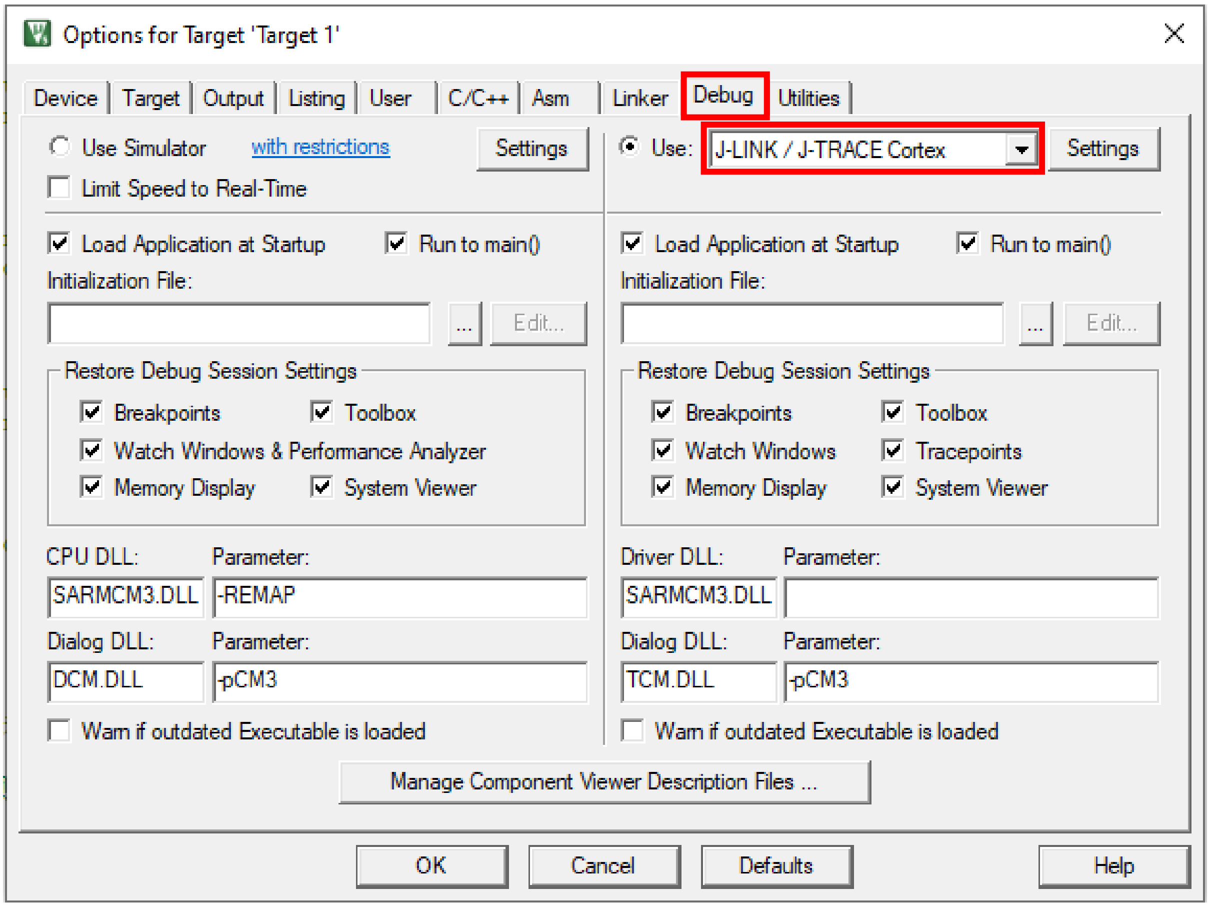
User manual for EMB8618I

1. Basic setup (J-Link)

Step 1: To configure the debugger setting, click the **"Flash"** tag and **"Configure Flash Tools…"** to open the configuration menu.



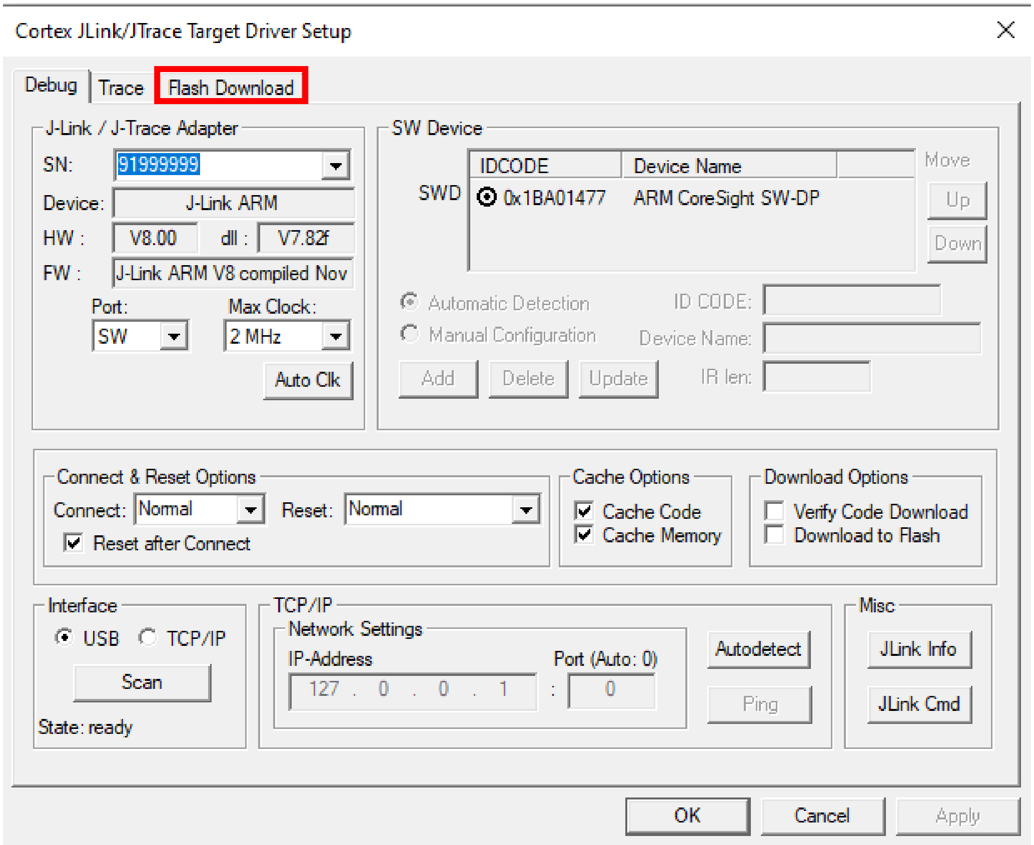
Step 2: Once the configuration menu appears, switch to the **"Debug"** tag and change the debugger to **"J-LINK/J-TRACE Cortex".** After that, click the **"Settings"** button for configuring debugger settings.



○ Use Simulator
☐ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter: -REMAP
Dialog DLL: DCM.DLL Parameter: -pCM3
☐ Warn if outdated Executable is loaded

● Use: J-LINK / J-TRACE Cortex
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows ☑ Tracepoints
☑ Memory Display ☑ System Viewer
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TCM.DLL Parameter: -pCM3
☐ Warn if outdated Executable is loaded

Step 3: For the Cortex JLink/JTrace Target Driver Setup, switch to the **"Flash Download"** tag.



Cortex JLink/JTrace Target Driver Setup

J-Link / J-Trace Adapter

SN: 91999999

Device: J-Link ARM

HW : V8.00 dll : V7.82f

FW : J-Link ARM V8 compiled Nov

Port: SW Max Clock: 2 MHz

SW Device

IDCODE	Device Name
SWD ● 0x1BA01477	ARM CoreSight SW-DP

● Automatic Detection ID CODE:

○ Manual Configuration Device Name:

IR len:

Connect & Reset Options

Connect: Normal Reset: Normal

☑ Reset after Connect

Cache Options

☑ Cache Code

☑ Cache Memory

Download Options

□ Verify Code Download

□ Download to Flash

Interface

● USB ○ TCP/IP

State: ready

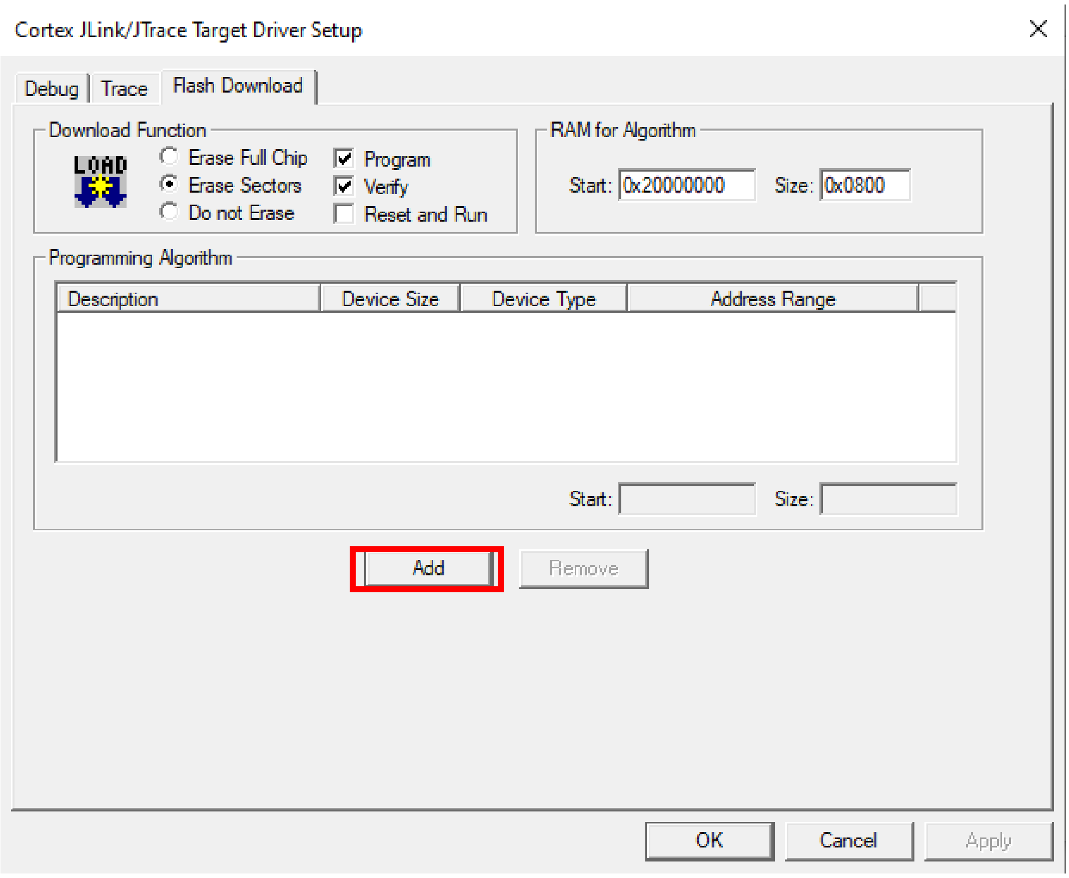
TCP/IP

Network Settings

IP-Address: 127 . 0 . 0 . 1 Port (Auto: 0): 0

Misc

Step 4: Clicking the **"Add"** button for adding the programming Algorithm.



Cortex JLink/JTrace Target Driver Setup

Download Function

○ Erase Full Chip ☑ Program

● Erase Sectors ☑ Verify

○ Do not Erase □ Reset and Run

RAM for Algorithm

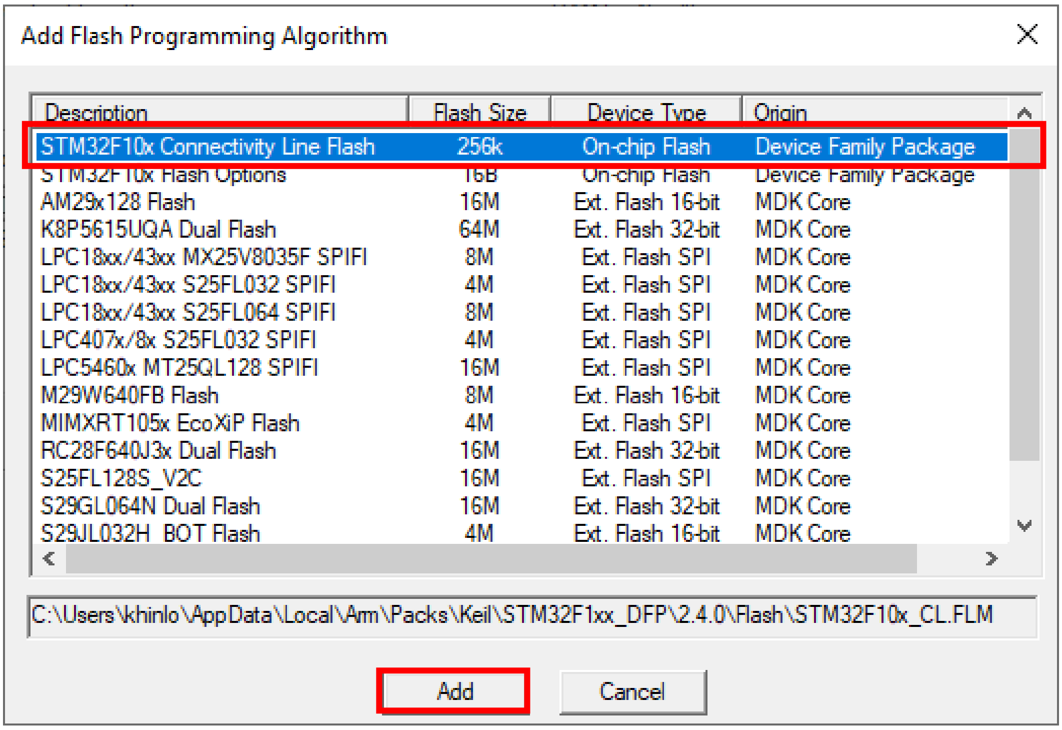
Start: 0x20000000 Size: 0x0800

Programming Algorithm

Description	Device Size	Device Type	Address Range

Start: Size:

Step 5: Once opened the Flash Programming Algorithm menu, you should add the "STM32F10x Connectivity Line Flash".

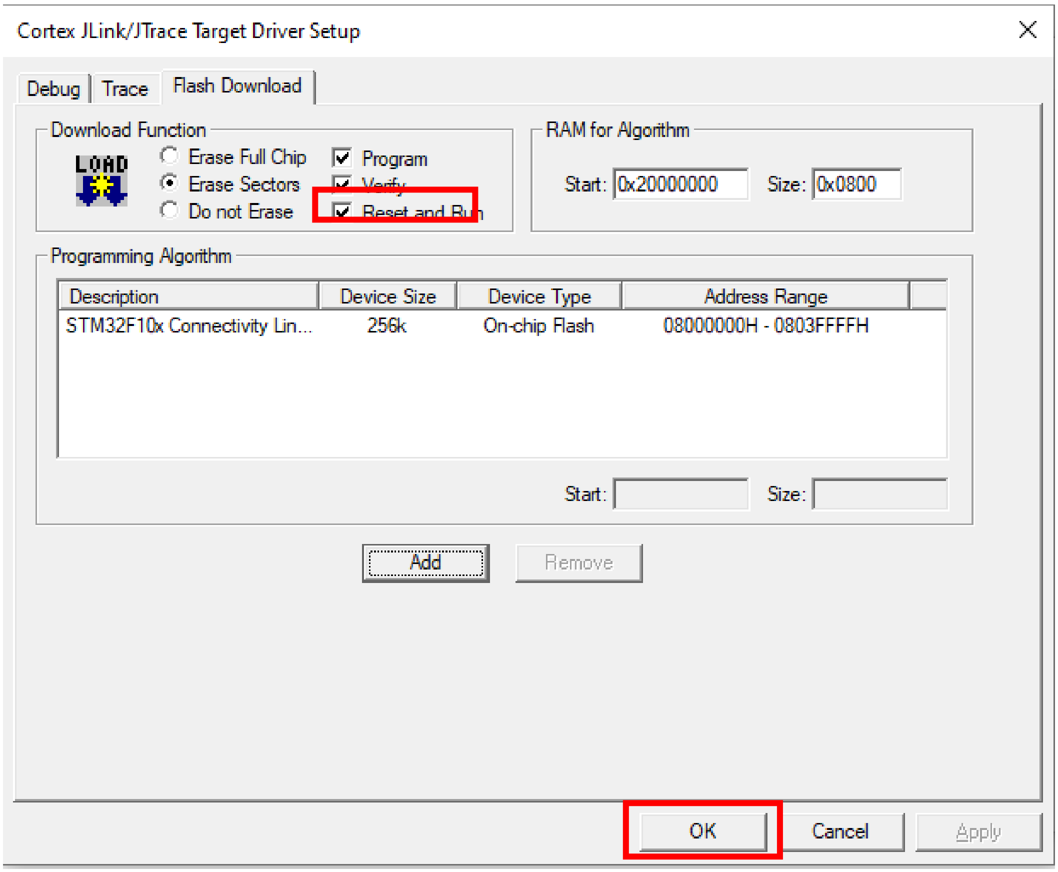


Add Flash Programming Algorithm

Description	Flash Size	Device Type	Origin
STM32F10x Connectivity Line Flash	256k	On-chip Flash	Device Family Package
STM32F10x Flash Options	16B	On-chip Flash	Device Family Package
AM29x128 Flash	16M	Ext. Flash 16-bit	MDK Core
K8P5615UQA Dual Flash	64M	Ext. Flash 32-bit	MDK Core
LPC18xx/43xx MX25V8035F SPIFI	8M	Ext. Flash SPI	MDK Core
LPC18xx/43xx S25FL032 SPIFI	4M	Ext. Flash SPI	MDK Core
LPC18xx/43xx S25FL064 SPIFI	8M	Ext. Flash SPI	MDK Core
LPC407x/8x S25FL032 SPIFI	4M	Ext. Flash SPI	MDK Core
LPC5460x MT25QL128 SPIFI	16M	Ext. Flash SPI	MDK Core
M29W640FB Flash	8M	Ext. Flash 16-bit	MDK Core
MIMXRT105x EcoXiP Flash	4M	Ext. Flash SPI	MDK Core
RC28F640J3x Dual Flash	16M	Ext. Flash 32-bit	MDK Core
S25FL128S_V2C	16M	Ext. Flash SPI	MDK Core
S29GL064N Dual Flash	16M	Ext. Flash 32-bit	MDK Core
S29JL032H_BOT Flash	4M	Ext. Flash 16-bit	MDK Core

C:\Users\khinlo\AppData\Local\Arm\Packs\Keil\STM32F1xx_DFP\2.4.0\Flash\STM32F10x_CL.FLM

Step 6: Remember to check the **"Reset and Run"** and click OK for confirming the changes.



Cortex JLink/JTrace Target Driver Setup

Download Function

○ Erase Full Chip
● Erase Sectors
○ Do not Erase

☑ Program
☑ Verify
☑ Reset and Run

RAM for Algorithm

Start: 0x20000000 Size: 0x0800

Programming Algorithm

Description	Device Size	Device Type	Address Range
STM32F10x Connectivity Lin...	256k	On-chip Flash	08000000H - 0803FFFFH

Start: Size:

2. Pin configuration:

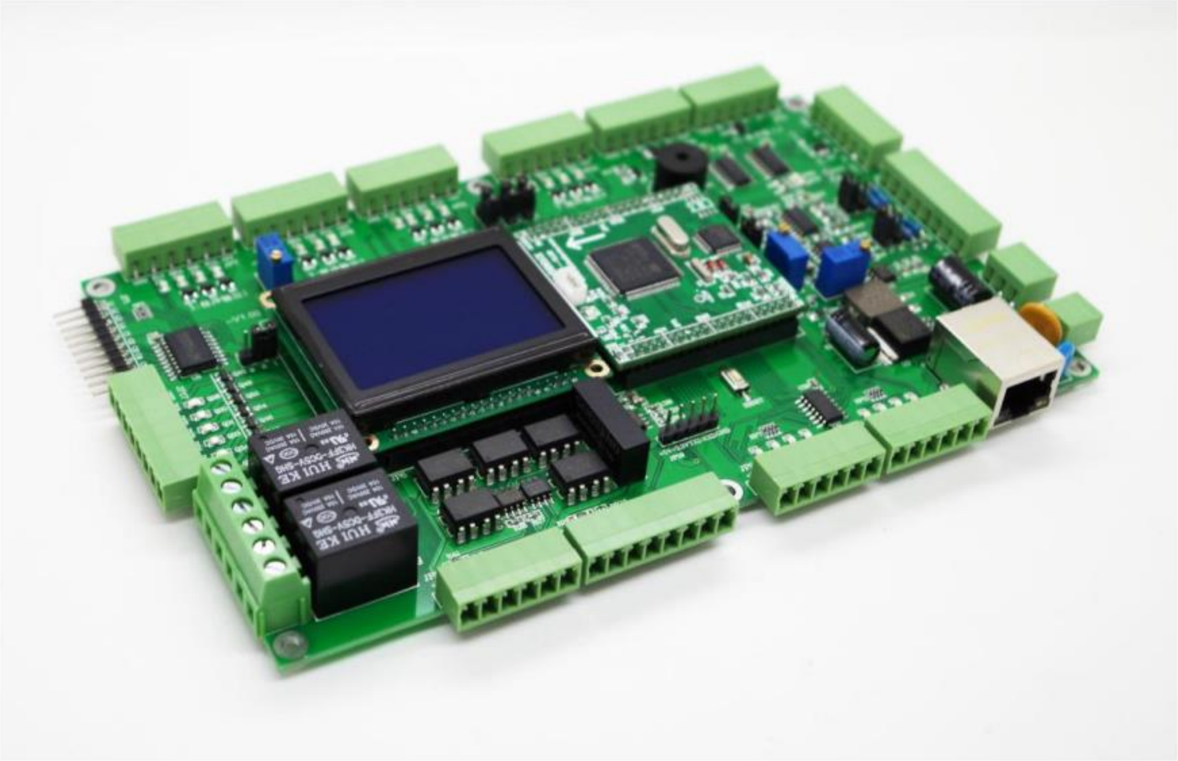


Figure 1: Full image of EMB8618I

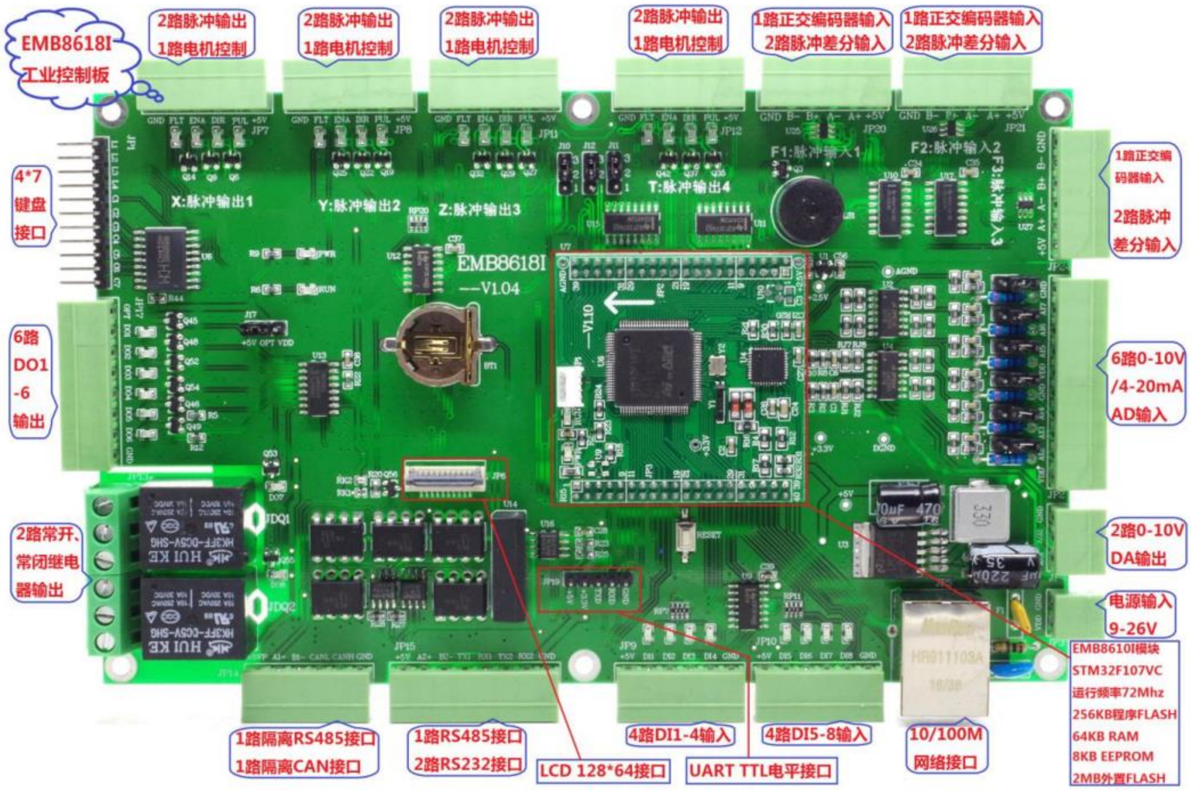
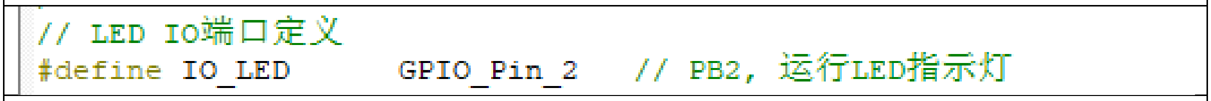


Figure 2: Pin description of EMB8618I

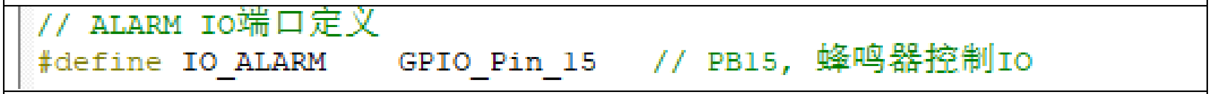
EMB8618I IO Configuration

Core LED:



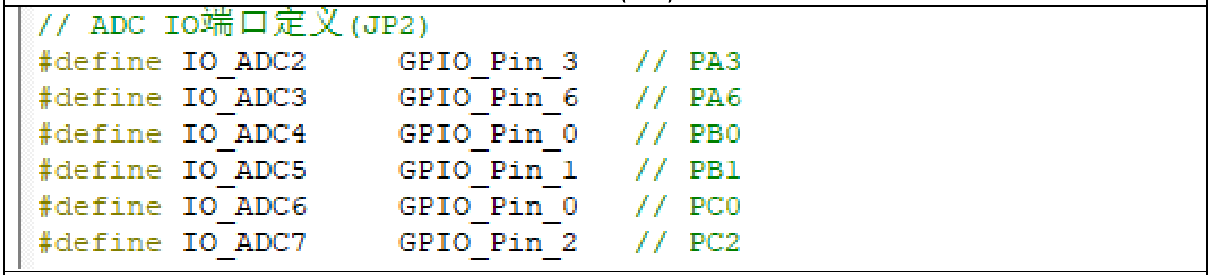
```
// LED IO端口定义
#define IO_LED       GPIO_Pin_2    // PB2, 运行LED指示灯
```

ALARM IO:



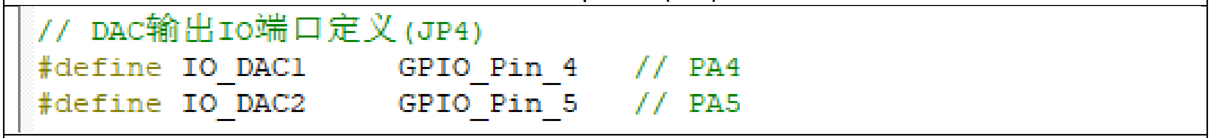
```
// ALARM IO端口定义
#define IO_ALARM     GPIO_Pin_15   // PB15, 蜂鸣器控制IO
```

ADC IO (JP2):



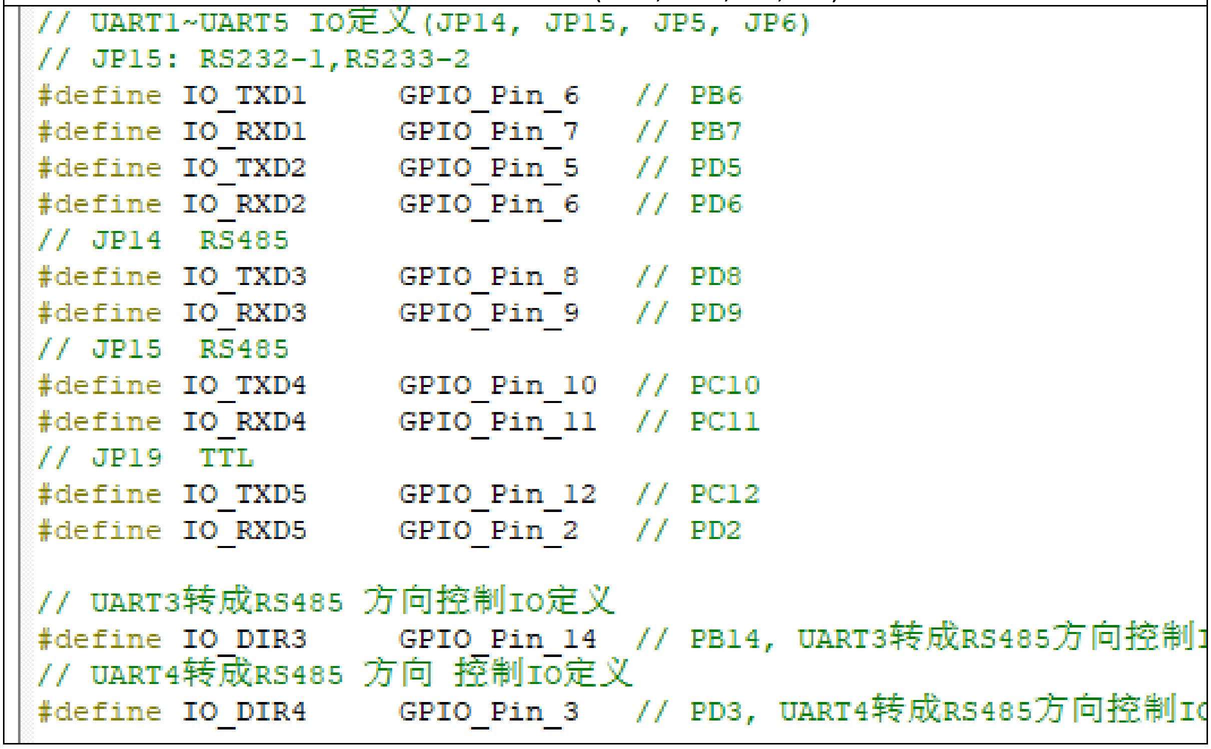
```
// ADC IO端口定义(JP2)
#define IO_ADC2      GPIO_Pin_3    // PA3
#define IO_ADC3      GPIO_Pin_6    // PA6
#define IO_ADC4      GPIO_Pin_0    // PB0
#define IO_ADC5      GPIO_Pin_1    // PB1
#define IO_ADC6      GPIO_Pin_0    // PC0
#define IO_ADC7      GPIO_Pin_2    // PC2
```

DAC output IO (JP4):



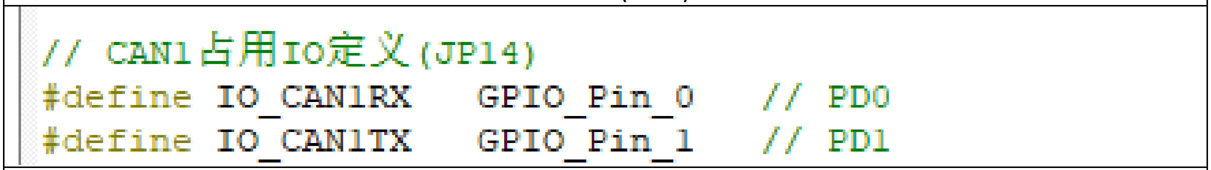
```
// DAC输出IO端口定义(JP4)
#define IO_DAC1      GPIO_Pin_4    // PA4
#define IO_DAC2      GPIO_Pin_5    // PA5
```

UART1 ~ UART5 IO (JP14, JP15, JP5, JP6):



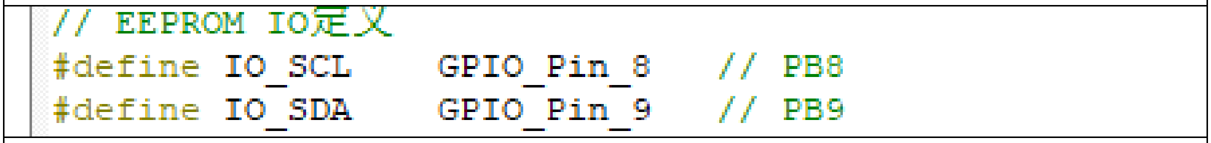
```
// UART1~UART5 IO定义(JP14, JP15, JP5, JP6)
// JP15: RS232-1,RS233-2
#define IO_TXD1      GPIO_Pin_6    // PB6
#define IO_RXD1      GPIO_Pin_7    // PB7
#define IO_TXD2      GPIO_Pin_5    // PD5
#define IO_RXD2      GPIO_Pin_6    // PD6
// JP14  RS485
#define IO_TXD3      GPIO_Pin_8    // PD8
#define IO_RXD3      GPIO_Pin_9    // PD9
// JP15  RS485
#define IO_TXD4      GPIO_Pin_10   // PC10
#define IO_RXD4      GPIO_Pin_11   // PC11
// JP19  TTL
#define IO_TXD5      GPIO_Pin_12   // PC12
#define IO_RXD5      GPIO_Pin_2    // PD2

// UART3转成RS485 方向控制IO定义
#define IO_DIR3      GPIO_Pin_14   // PB14, UART3转成RS485方向控制I
// UART4转成RS485 方向 控制IO定义
#define IO_DIR4      GPIO_Pin_3    // PD3, UART4转成RS485方向控制IO
```

CAN1 IO (JP14):



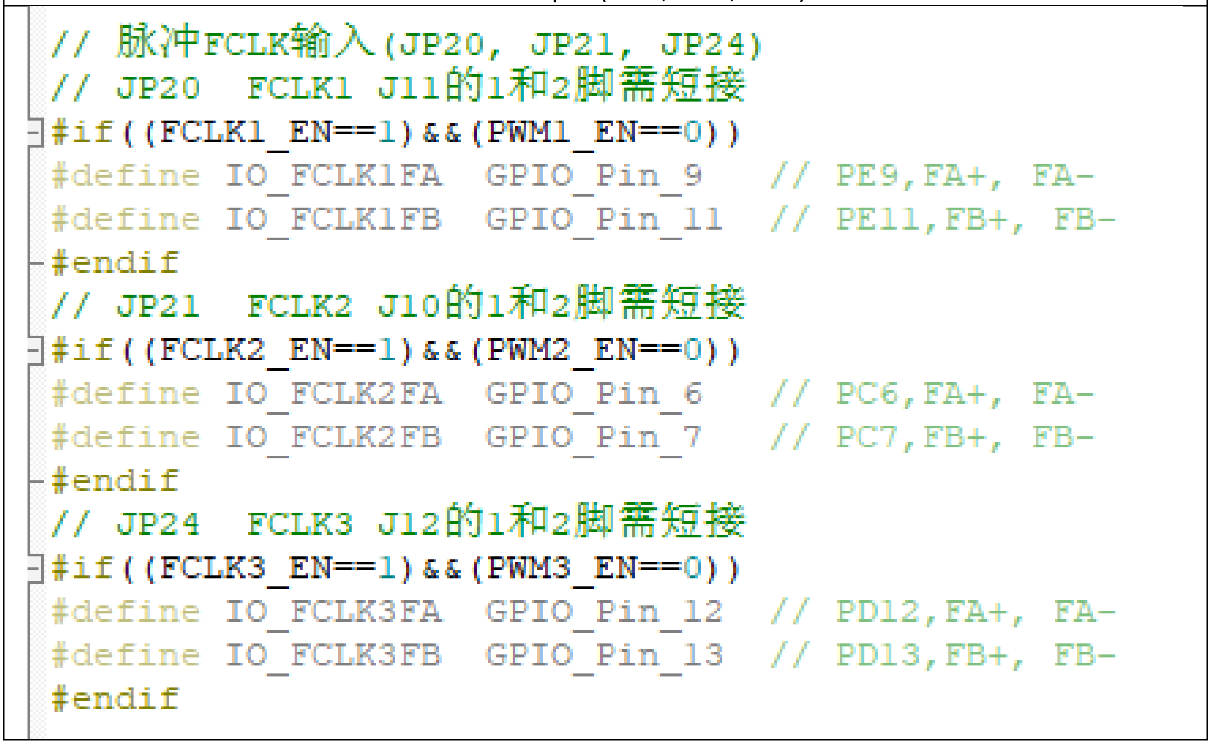
```
// CAN1占用IO定义(JP14)
#define IO_CAN1RX   GPIO_Pin_0   // PD0
#define IO_CAN1TX   GPIO_Pin_1   // PD1
```

EEPROM IO:



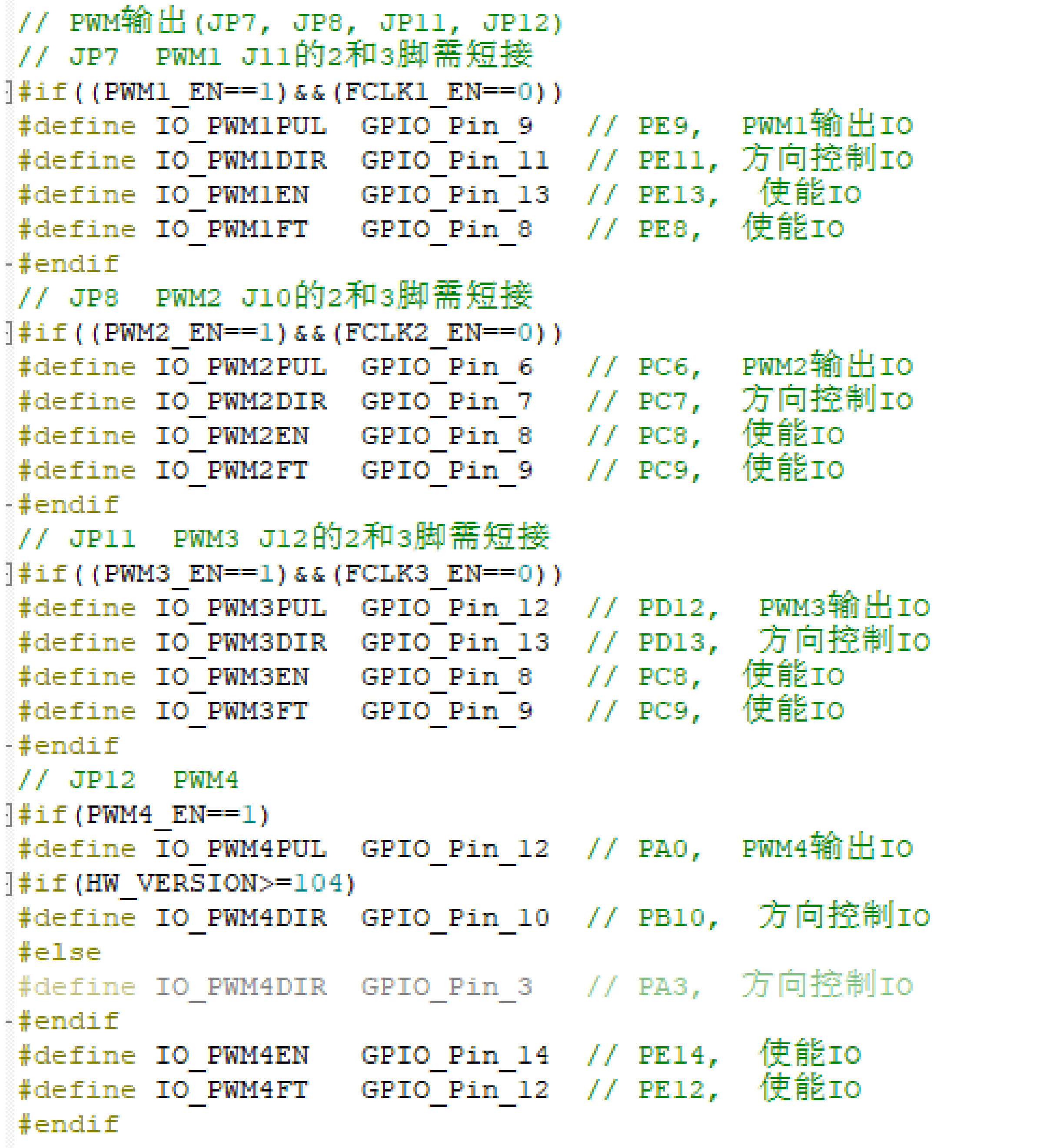
```
// EEPROM IO定义
#define IO_SCL     GPIO_Pin_8   // PB8
#define IO_SDA     GPIO_Pin_9   // PB9
```

FCLK Pulse input (JP20, JP21, JP24):



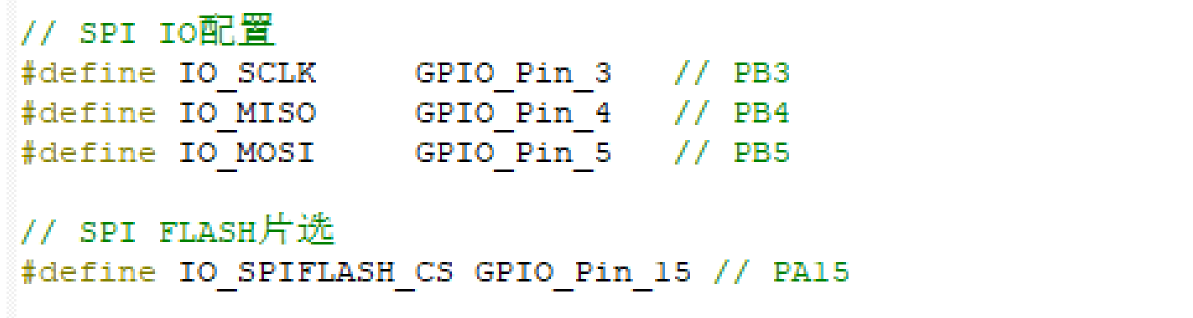
```
// 脉冲FCLK输入(JP20, JP21, JP24)
// JP20  FCLK1 J11的1和2脚需短接
#if((FCLK1_EN==1)&&(PWM1_EN==0))
#define IO_FCLK1FA  GPIO_Pin_9   // PE9,FA+, FA-
#define IO_FCLK1FB  GPIO_Pin_11  // PE11,FB+, FB-
#endif
// JP21  FCLK2 J10的1和2脚需短接
#if((FCLK2_EN==1)&&(PWM2_EN==0))
#define IO_FCLK2FA  GPIO_Pin_6   // PC6,FA+, FA-
#define IO_FCLK2FB  GPIO_Pin_7   // PC7,FB+, FB-
#endif
// JP24  FCLK3 J12的1和2脚需短接
#if((FCLK3_EN==1)&&(PWM3_EN==0))
#define IO_FCLK3FA  GPIO_Pin_12  // PD12,FA+, FA-
#define IO_FCLK3FB  GPIO_Pin_13  // PD13,FB+, FB-
#endif
```

PWM output (JP7, JP8, JP11, JP12):



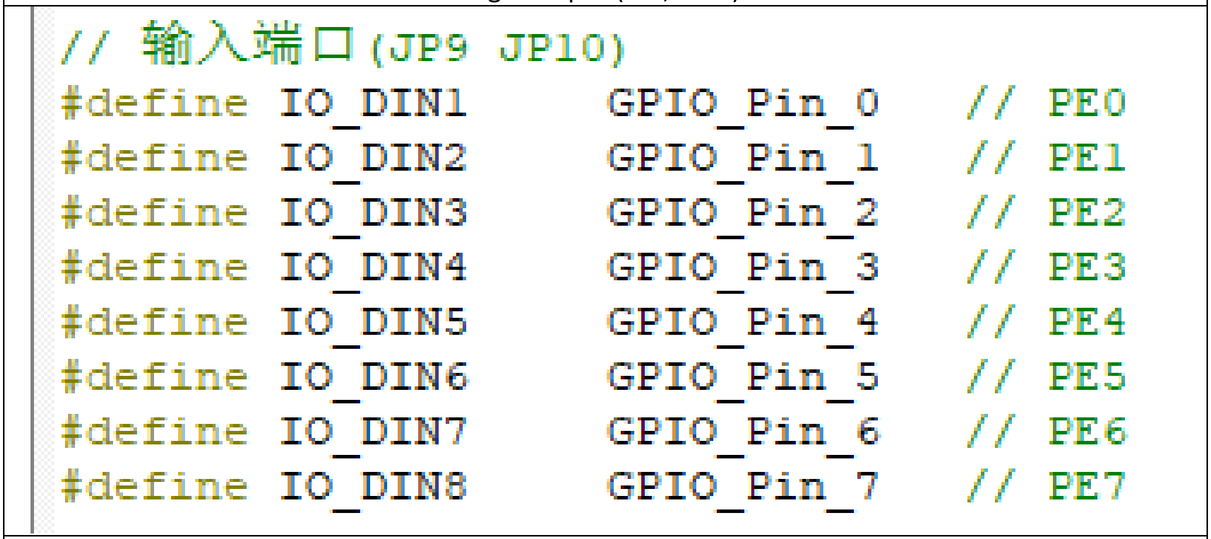
```
 // PWM输出(JP7, JP8, JP11, JP12)
 // JP7  PWM1 J11的2和3脚需短接
]#if((PWM1_EN==1)&&(FCLK1_EN==0))
 #define IO_PWM1PUL  GPIO_Pin_9   // PE9,  PWM1输出IO
 #define IO_PWM1DIR  GPIO_Pin_11  // PE11, 方向控制IO
 #define IO_PWM1EN   GPIO_Pin_13  // PE13,  使能IO
 #define IO_PWM1FT   GPIO_Pin_8   // PE8,  使能IO
-#endif
 // JP8  PWM2 J10的2和3脚需短接
]#if((PWM2_EN==1)&&(FCLK2_EN==0))
 #define IO_PWM2PUL  GPIO_Pin_6   // PC6,  PWM2输出IO
 #define IO_PWM2DIR  GPIO_Pin_7   // PC7,  方向控制IO
 #define IO_PWM2EN   GPIO_Pin_8   // PC8,  使能IO
 #define IO_PWM2FT   GPIO_Pin_9   // PC9,  使能IO
-#endif
 // JP11  PWM3 J12的2和3脚需短接
]#if((PWM3_EN==1)&&(FCLK3_EN==0))
 #define IO_PWM3PUL  GPIO_Pin_12  // PD12,  PWM3输出IO
 #define IO_PWM3DIR  GPIO_Pin_13  // PD13,  方向控制IO
 #define IO_PWM3EN   GPIO_Pin_8   // PC8,  使能IO
 #define IO_PWM3FT   GPIO_Pin_9   // PC9,  使能IO
-#endif
 // JP12  PWM4
]#if(PWM4_EN==1)
 #define IO_PWM4PUL  GPIO_Pin_12  // PA0,  PWM4输出IO
]#if(HW_VERSION>=104)
 #define IO_PWM4DIR  GPIO_Pin_10  // PB10,  方向控制IO
 #else
 #define IO_PWM4DIR  GPIO_Pin_3   // PA3,  方向控制IO
-#endif
 #define IO_PWM4EN   GPIO_Pin_14  // PE14,  使能IO
 #define IO_PWM4FT   GPIO_Pin_12  // PE12,  使能IO
 #endif
```

SPI IO:



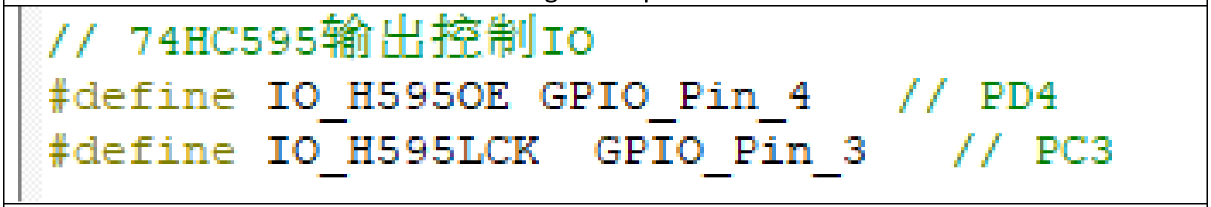
```
// SPI IO配置
#define IO_SCLK     GPIO_Pin_3   // PB3
#define IO_MISO     GPIO_Pin_4   // PB4
#define IO_MOSI     GPIO_Pin_5   // PB5

// SPI FLASH片选
#define IO_SPIFLASH_CS GPIO_Pin_15 // PA15
```

Digital input (JP9, JP10):



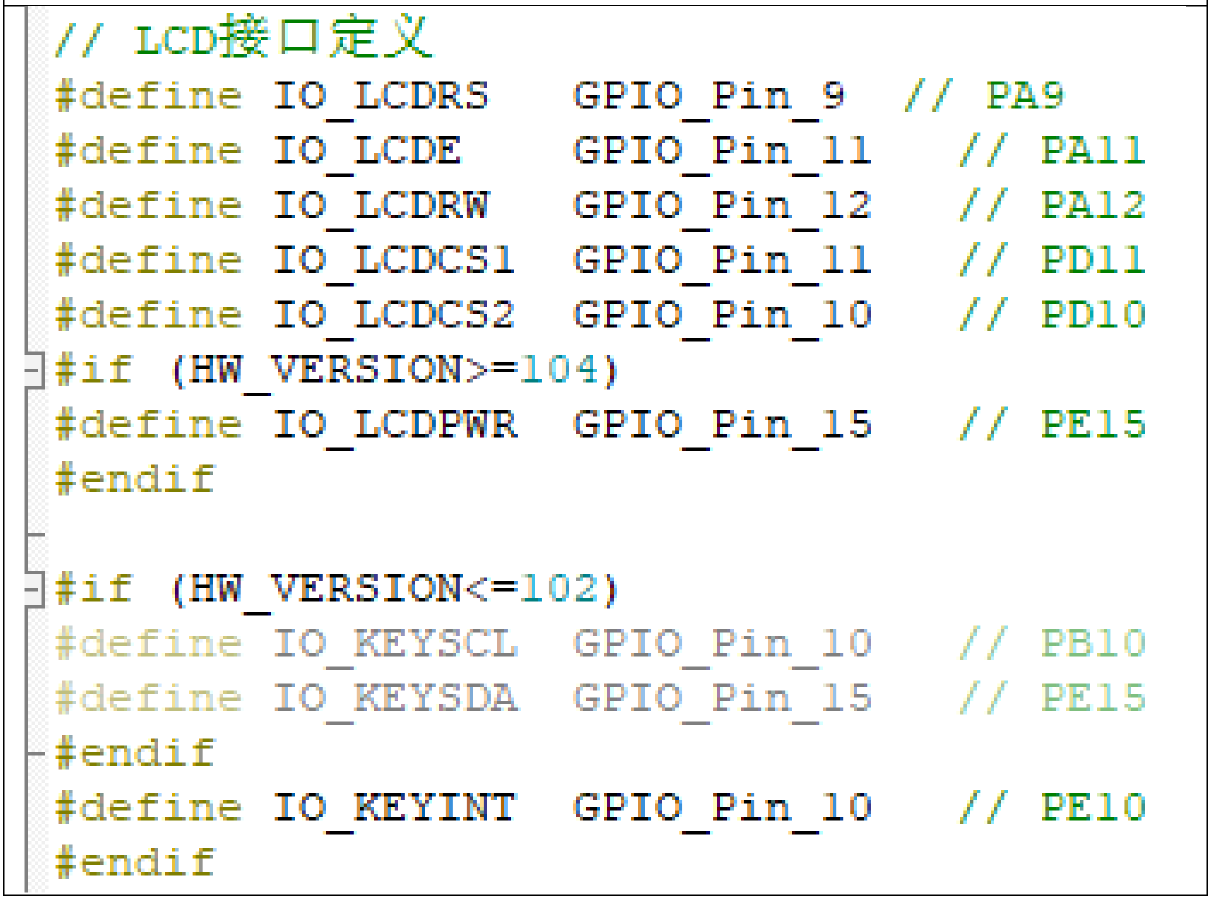
```
// 输入端口(JP9 JP10)
#define IO_DIN1     GPIO_Pin_0   // PE0
#define IO_DIN2     GPIO_Pin_1   // PE1
#define IO_DIN3     GPIO_Pin_2   // PE2
#define IO_DIN4     GPIO_Pin_3   // PE3
#define IO_DIN5     GPIO_Pin_4   // PE4
#define IO_DIN6     GPIO_Pin_5   // PE5
#define IO_DIN7     GPIO_Pin_6   // PE6
#define IO_DIN8     GPIO_Pin_7   // PE7
```

Digital output:



```
// 74HC595输出控制IO
#define IO_H595OE GPIO_Pin_4   // PD4
#define IO_H595LCK  GPIO_Pin_3   // PC3
```

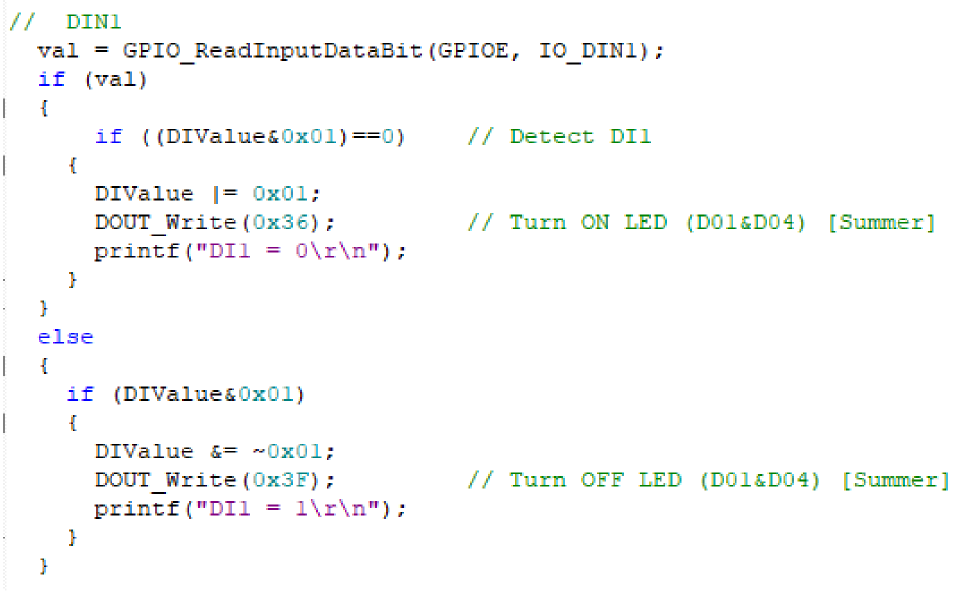
LAC:



```
// LCD接口定义
#define IO_LCDRS   GPIO_Pin_9  // PA9
#define IO_LCDE    GPIO_Pin_11   // PA11
#define IO_LCDRW   GPIO_Pin_12   // PA12
#define IO_LCDCS1  GPIO_Pin_11   // PD11
#define IO_LCDCS2  GPIO_Pin_10   // PD10
#if (HW_VERSION>=104)
#define IO_LCDPWR  GPIO_Pin_15   // PE15
#endif

#if (HW_VERSION<=102)
#define IO_KEYSCL  GPIO_Pin_10   // PB10
#define IO_KEYSDA  GPIO_Pin_15   // PE15
#endif
#define IO_KEYINT  GPIO_Pin_10   // PE10
#endif
```

3. GPIO

A binary address is used to control the Digital input pin.



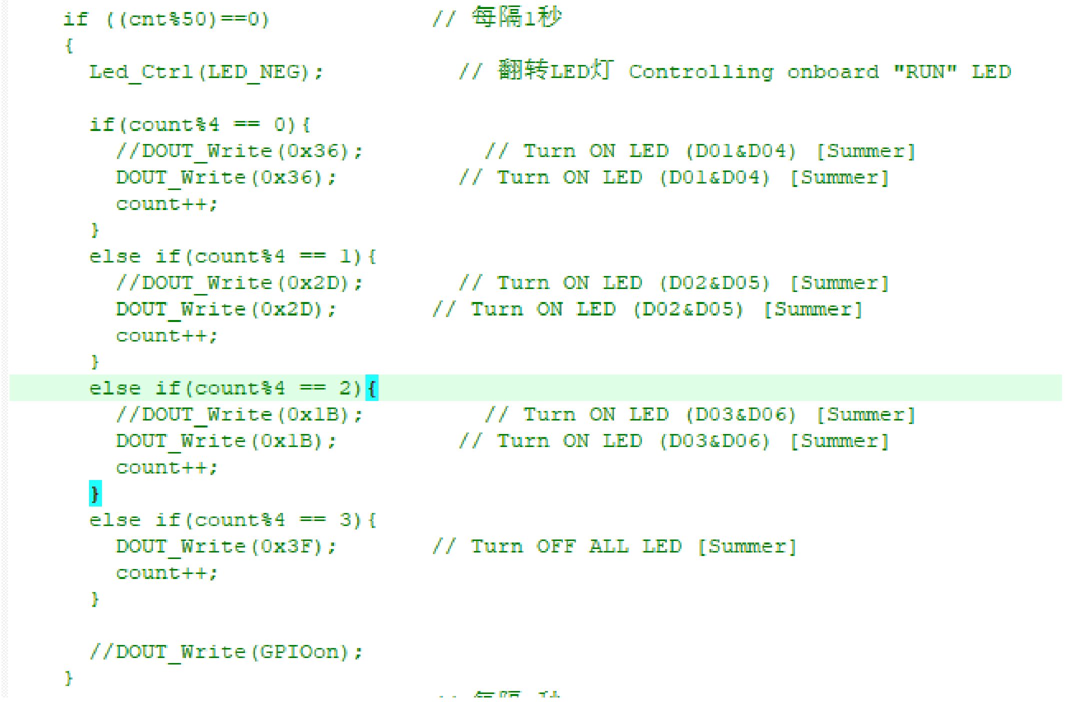
```
//  DIN1
  val = GPIO_ReadInputDataBit(GPIOE, IO_DIN1);
  if (val)
  {
      if ((DIValue&0x01)==0)     // Detect DI1
    {
      DIValue |= 0x01;
      DOUT_Write(0x36);          // Turn ON LED (DO1&DO4) [Summer]
      printf("DI1 = 0\r\n");
    }
  }
  else
  {
    if (DIValue&0x01)
    {
      DIValue &= ~0x01;
      DOUT_Write(0x3F);          // Turn OFF LED (DO1&DO4) [Summer]
      printf("DI1 = 1\r\n");
    }
  }
```

GPIO is using the binary address for controlling and configuring. The logic of DO1-DO6 and DO7-DO8 is reversed.

Operation	Address	Address (binary)
Turn on DO1 and DO4	0x36	0011 0110
Turn on DO2 and DO5	0x2D	0010 1101
Turn on DO3 and DO6	0x1B	0001 1011
Turn Off DO1-DO6	0x3F	0011 1111

For DO1-DO6, “1” represents OFF, and “0” represents ON

For DO7-DO8, “0” represents ON, and “1” represents OFF



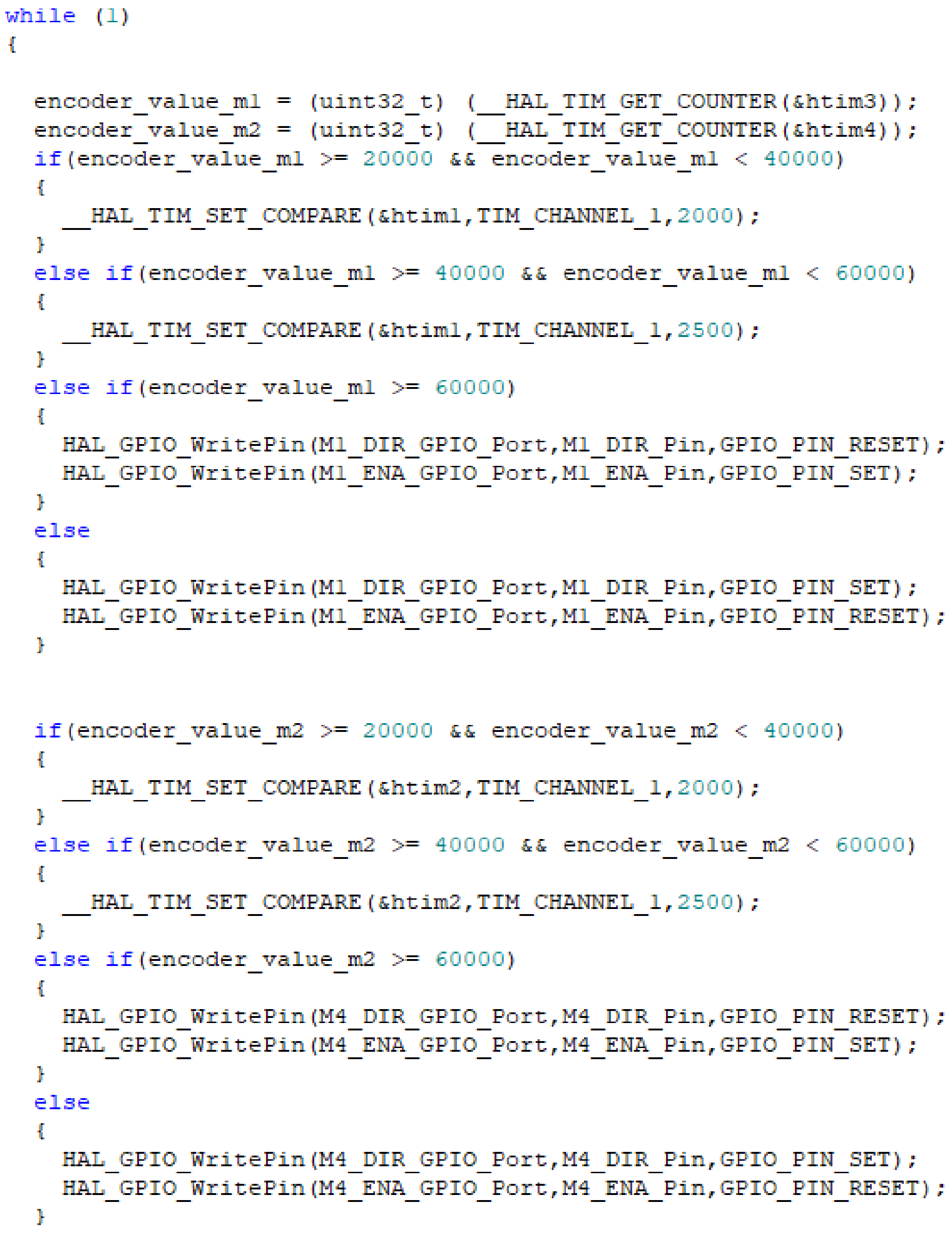
```
    if ((cnt%50)==0)             // 每隔1秒
    {
      Led_Ctrl(LED_NEG);           // 翻转LED灯 Controlling onboard "RUN" LED

      if(count%4 == 0){
        //DOUT_Write(0x36);          // Turn ON LED (DO1&DO4) [Summer]
        DOUT_Write(0x36);          // Turn ON LED (DO1&DO4) [Summer]
        count++;
      }
      else if(count%4 == 1){
        //DOUT_Write(0x2D);        // Turn ON LED (DO2&DO5) [Summer]
        DOUT_Write(0x2D);        // Turn ON LED (DO2&DO5) [Summer]
        count++;
      }
      else if(count%4 == 2){
        //DOUT_Write(0x1B);          // Turn ON LED (DO3&DO6) [Summer]
        DOUT_Write(0x1B);          // Turn ON LED (DO3&DO6) [Summer]
        count++;
      }
      else if(count%4 == 3){
        DOUT_Write(0x3F);        // Turn OFF ALL LED [Summer]
        count++;
      }

      //DOUT_Write(GPIOon);
    }
```

4. DC Motor Setup

For EMB8618I, the pulse input and pulse output are sharing the same channel. To adjust the pulse mode, the jumper (JP10, JP12, JP11) has to modify.

Pulse channel	Jumper Position	Pulse Input mode	Pulse output mode
PWM/FCLK1	J11	Connecting Pin 1 and 2	Connecting Pin 2 and 3
PWM/FCLK2	J10		
PWM/FCLK3	J12		
PWM4			Default

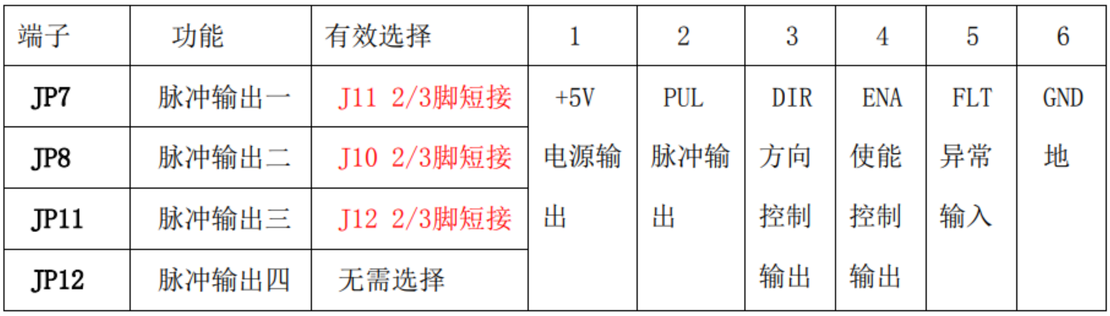
To control PWM, the HAL library can be applied.



```
while (1)
{

  encoder_value_m1 = (uint32_t) (__HAL_TIM_GET_COUNTER(&htim3));
  encoder_value_m2 = (uint32_t) (__HAL_TIM_GET_COUNTER(&htim4));
  if(encoder_value_m1 >= 20000 && encoder_value_m1 < 40000)
  {
    __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,2000);
  }
  else if(encoder_value_m1 >= 40000 && encoder_value_m1 < 60000)
  {
    __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,2500);
  }
  else if(encoder_value_m1 >= 60000)
  {
    HAL_GPIO_WritePin(M1_DIR_GPIO_Port,M1_DIR_Pin,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(M1_ENA_GPIO_Port,M1_ENA_Pin,GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(M1_DIR_GPIO_Port,M1_DIR_Pin,GPIO_PIN_SET);
    HAL_GPIO_WritePin(M1_ENA_GPIO_Port,M1_ENA_Pin,GPIO_PIN_RESET);
  }


  if(encoder_value_m2 >= 20000 && encoder_value_m2 < 40000)
  {
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,2000);
  }
  else if(encoder_value_m2 >= 40000 && encoder_value_m2 < 60000)
  {
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,2500);
  }
  else if(encoder_value_m2 >= 60000)
  {
    HAL_GPIO_WritePin(M4_DIR_GPIO_Port,M4_DIR_Pin,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(M4_ENA_GPIO_Port,M4_ENA_Pin,GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(M4_DIR_GPIO_Port,M4_DIR_Pin,GPIO_PIN_SET);
    HAL_GPIO_WritePin(M4_ENA_GPIO_Port,M4_ENA_Pin,GPIO_PIN_RESET);
  }
```

5. Stepper Motor Setup

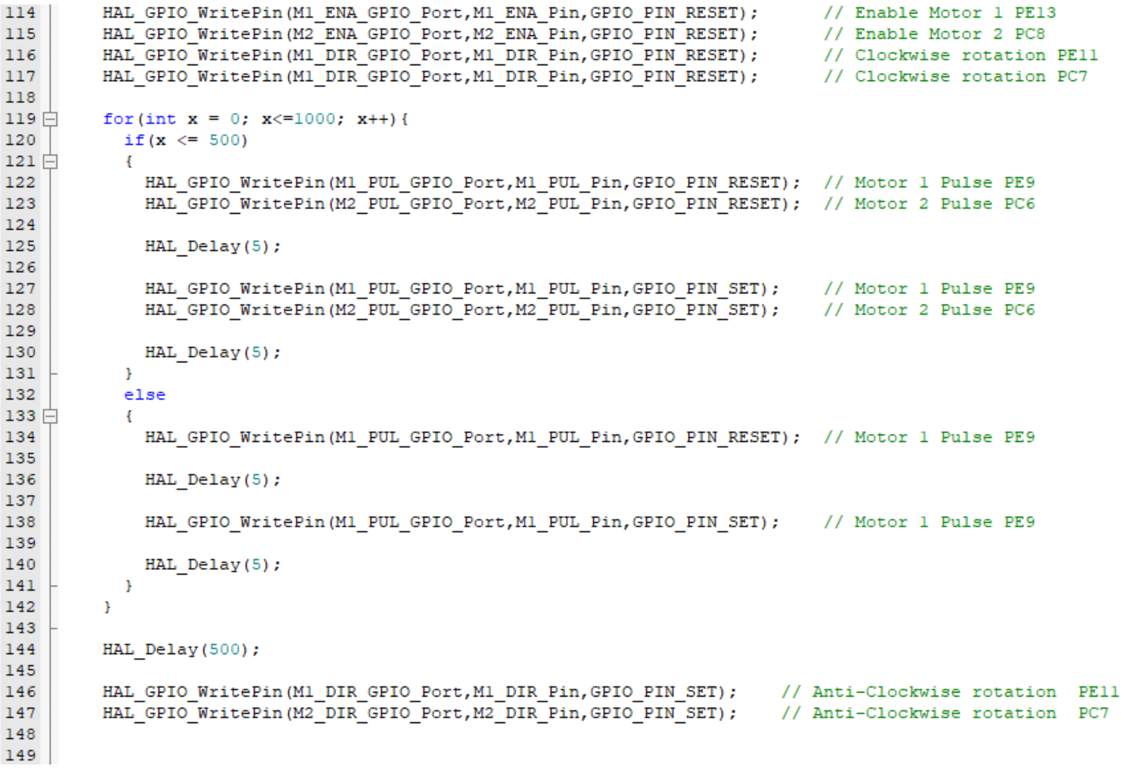
To control the stepper motor, the PUL pin can be configurated as digital output for generating the pulse. To change the velocity of the stepper motor, you need to reconfigure the pulse/revolution of the motor driver and delay for generating the pulse.



端子	功能	有效选择	1	2	3	4	5	6
JP7	脉冲输出一	J11 2/3脚短接	+5V 电源输出	PUL 脉冲输出	DIR 方向控制输出	ENA 使能控制输出	FLT 异常输入	GND 地
JP8	脉冲输出二	J10 2/3脚短接						
JP11	脉冲输出三	J12 2/3脚短接						
JP12	脉冲输出四	无需选择						

Figure 3: Pulse output pin description

Microstep Driver				
Microstep	Pulse/rev	S1	S2	S3
NC	NC	ON	ON	ON
1	200	ON	ON	OFF
2/A	400	ON	OFF	ON
2/B	400	OFF	ON	ON
1	800	ON	OFF	OFF
8	1600	OFF	ON	OFF
16	3200	OFF	OFF	ON
32	6400	OFF	OFF	OFF



```
HAL_GPIO_WritePin(M1_ENA_GPIO_Port,M1_ENA_Pin,GPIO_PIN_RESET);      // Enable Motor 1 PE13
HAL_GPIO_WritePin(M2_ENA_GPIO_Port,M2_ENA_Pin,GPIO_PIN_RESET);      // Enable Motor 2 PC8
HAL_GPIO_WritePin(M1_DIR_GPIO_Port,M1_DIR_Pin,GPIO_PIN_RESET);      // Clockwise rotation PE11
HAL_GPIO_WritePin(M1_DIR_GPIO_Port,M1_DIR_Pin,GPIO_PIN_RESET);      // Clockwise rotation PC7

for(int x = 0; x<=1000; x++){
  if(x <= 500)
  {
    HAL_GPIO_WritePin(M1_PUL_GPIO_Port,M1_PUL_Pin,GPIO_PIN_RESET);  // Motor 1 Pulse PE9
    HAL_GPIO_WritePin(M2_PUL_GPIO_Port,M2_PUL_Pin,GPIO_PIN_RESET);  // Motor 2 Pulse PC6

    HAL_Delay(5);

    HAL_GPIO_WritePin(M1_PUL_GPIO_Port,M1_PUL_Pin,GPIO_PIN_SET);    // Motor 1 Pulse PE9
    HAL_GPIO_WritePin(M2_PUL_GPIO_Port,M2_PUL_Pin,GPIO_PIN_SET);    // Motor 2 Pulse PC6

    HAL_Delay(5);
  }
  else
  {
    HAL_GPIO_WritePin(M1_PUL_GPIO_Port,M1_PUL_Pin,GPIO_PIN_RESET);  // Motor 1 Pulse PE9

    HAL_Delay(5);

    HAL_GPIO_WritePin(M1_PUL_GPIO_Port,M1_PUL_Pin,GPIO_PIN_SET);    // Motor 1 Pulse PE9

    HAL_Delay(5);
  }
}

HAL_Delay(500);

HAL_GPIO_WritePin(M1_DIR_GPIO_Port,M1_DIR_Pin,GPIO_PIN_SET);    // Anti-Clockwise rotation  PE11
HAL_GPIO_WritePin(M2_DIR_GPIO_Port,M2_DIR_Pin,GPIO_PIN_SET);    // Anti-Clockwise rotation  PC7
```