Inter-Integrated Circuit : I²C

Microcontroller Application and Development 2565

Sorayut Glomglome

I2C

- I2C เป็นโปรโตคอลสื่อสารอนุกรม ประกอบด้วยขาสัญญาณ
  - SDA: สัญญาณข้อมูล
  - SCL: สัญญาณนาฬิกา
  - https://www.youtube.com/watch?v=6IAkYpmA1DQ

STM32F767 I2C

- I^2C Master features:
  - Clock generation
  - Start and Stop generation

- I^2C Slave features:
  - Programmable I^2C Address detection
  - Dual Addressing Capability to acknowledge 2 slave addresses
  - Stop bit detection

- Supports different communication speeds:
  - Standard Speed (up to 100 kHz)
  - Fast Speed (up to 400 kHz)
  - The I2C bus frequency can be increased up to 1 MHz. For more details about the complete solution, please contact your local ST sales representative

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

STM32F767 I2C

- I²C Master features:
  - Clock generation
  - Start and Stop generation

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

- I²C Slave features:
  - Programmable I²C Address detection
  - Dual Addressing Capability to acknowledge 2 slave addresses
  - Stop bit detection

I²C (Inter-Integrated Circuit)

- Supports different communication speeds:
  - Standard Speed (up to 100 kHz)
  - Fast Speed (up to 400 kHz)
  - The I2C bus frequency can be increased up to 1 MHz. For more details about the complete solution, please contact your local ST sales representative

STM32F767 I2C

- Slave and master modes, multimaster capability
- Standard-mode (Sm), with a bitrate up to 100 kbit/s
- Fast-mode (Fm), with a bitrate up to 400 kbit/s
- Fast-mode Plus (Fm+), with a bitrate up to 1 Mbit/s and 20 mA output drive I/Os
- 7-bit and 10-bit addressing mode
- multiple 7-bit slave addresses

STM32F767 I2C

Table 7. I2C implementation

I2C features(1)	I2C1	I2C2	I2C3	I2C4
Standard-mode (up to 100 kbit/s)	X	X	X	X
Fast-mode (up to 400 kbit/s)	X	X	X	X
Fast-mode Plus with 20 mA output drive I/Os (up to 1 Mbit/s)	X	X	X	X
Programmable analog and digital noise filters	X	X	X	X
SMBus/PMBus hardware support	X	X	X	X
Independent clock	X	X	X	X

1. X: supported.

Mode selection

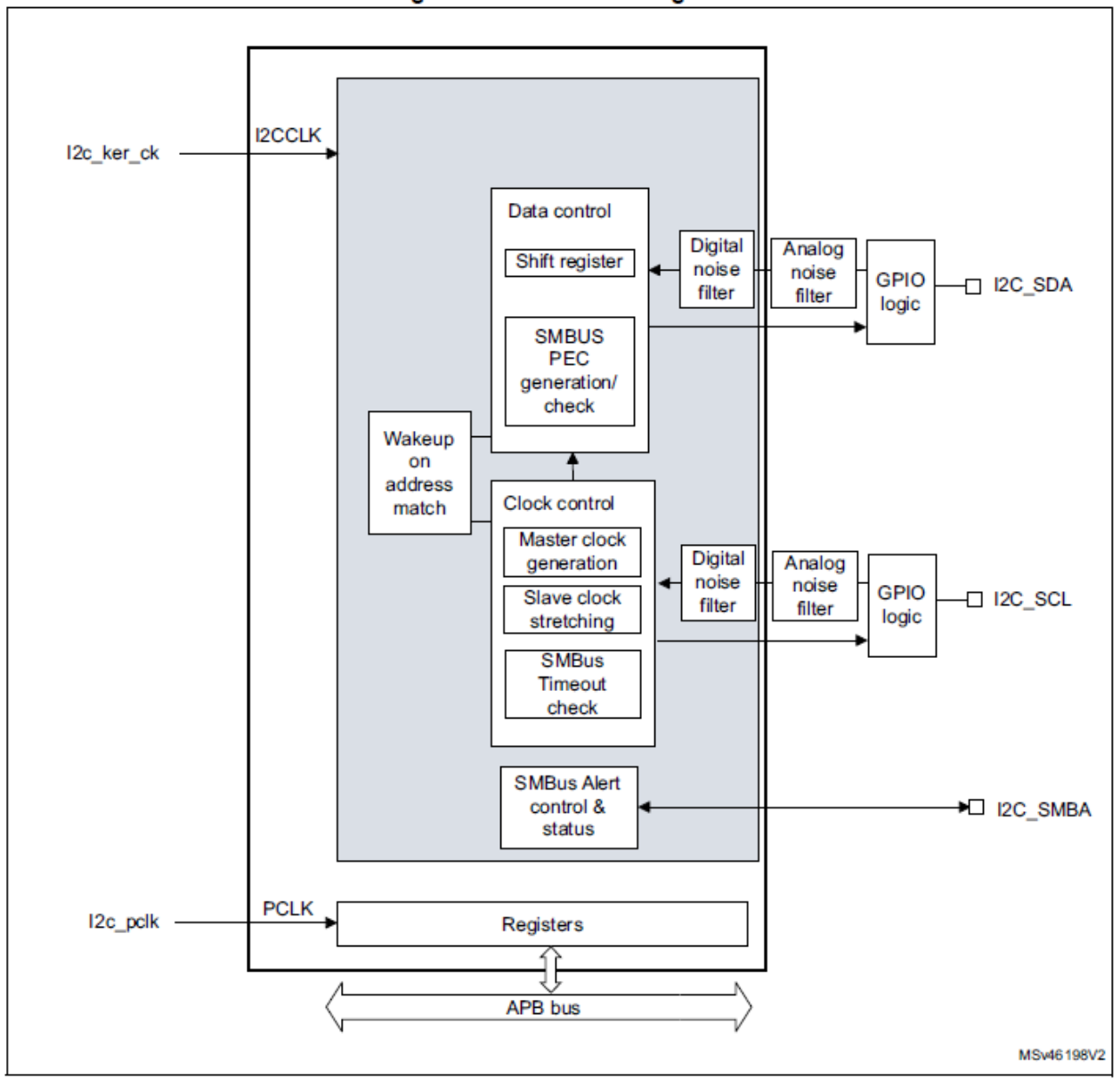
The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

STM32F767 I2C

Figure 349. I2C block diagram



Communication Flow

- In Master mode, the I2C interface initiates a data transfer and generates the clock signal.
- A serial data transfer always begins with a START condition and ends with a STOP condition.
- Both START and STOP conditions are generated in master mode by software.

Communication Flow

- In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address.
- Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the START condition contain the address (one in 7-bit mode, two in 10-bit mode).
- The address is always transmitted in Master mode.

I2C UNO Pins : D14 & D15

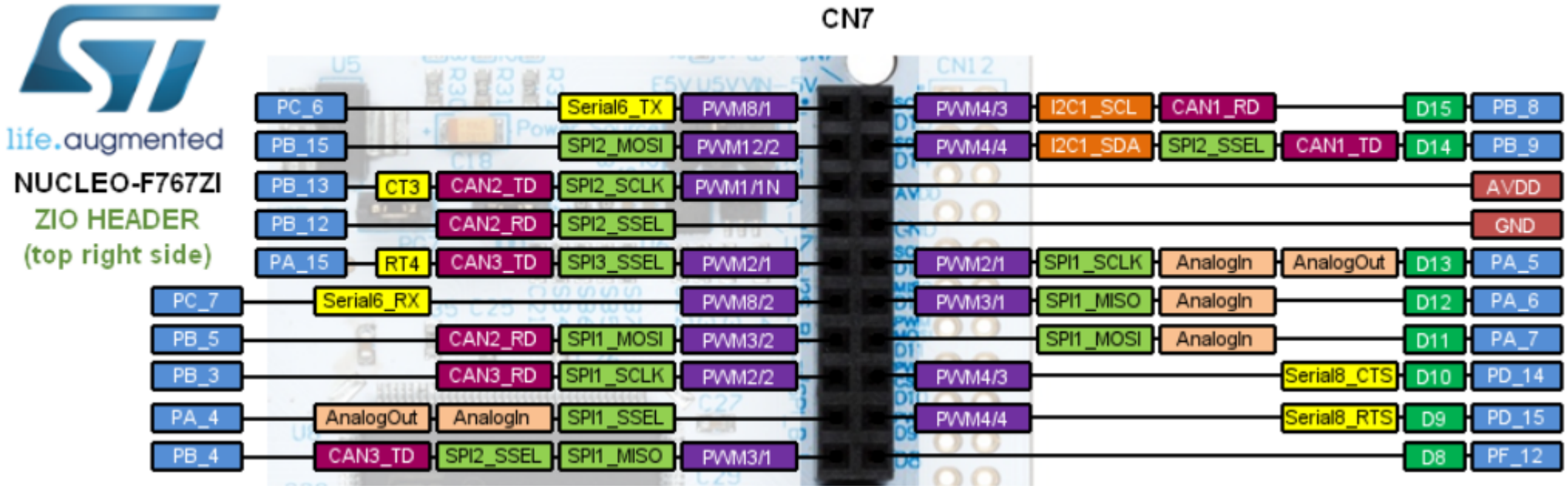
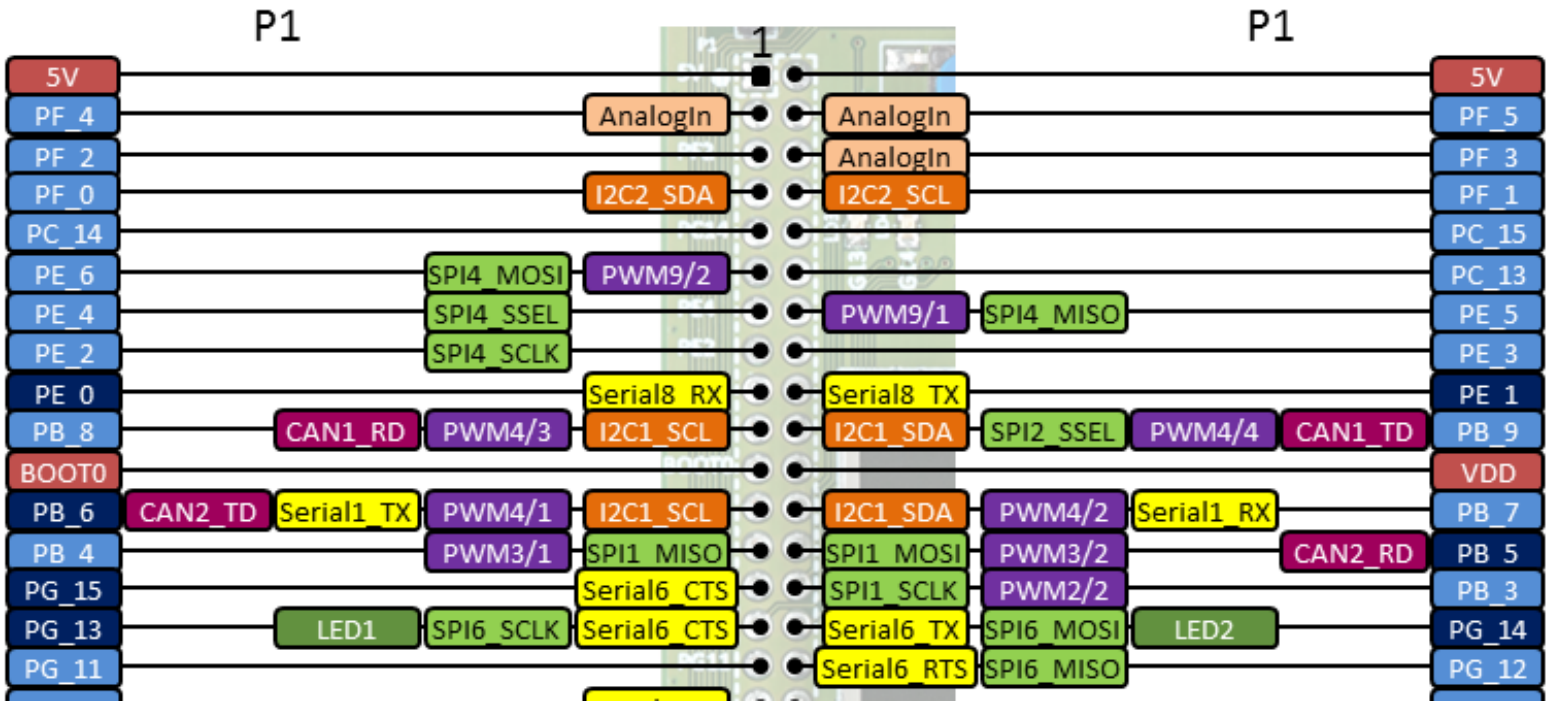


Table 13. STM32F765xx, STM32F767xx, STM32F768Ax and STM32F769xx alternate function mapping (continued)

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	I2C4/UART5/TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/DFSDM1/CEC	I2C1/2/3/4/USART1/CEC	SPI1/I2S1/SPI2/I2S2/SPI3/I2S3/SPI4/5/6	SPI2/I2S2/SPI3/I2S3/SAI1/I2C4/UART4/DFSDM1	SPI2/I2S2/SPI3/I2S3/SPI6/USART1/2/3/UART5/DFSDM1/SPDIF	SPI6/SAI2/USART6/UART4/5/7/8/OTG_FS/SPDIF	CAN1/2/TIM12/13/14/QUADSPI/FMC/LCD	SAI2/QUADSPI/SDMMC2/DFSDM1/OTG2_HS/OTG1_FS/LCD	I2C4/CAN3/SDMMC2/ETH	UART7/FMC/SDMMC1/MDIOS/OTG2_FS	DCMI/LCD/DSI	LCD	SYS
Port B	PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	DFSDM1_CKIN5	USART1_RX	-	-	-	I2S4_SDA	FMC_NL	DCMI_VSYNC	-	EVENTOUT
	PB8	-	I2C4_SCL	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	DFSDM1_CKIN7	UART5_RX	-	CAN1_RX	SDMMC2_D4	ETH_MII_TXD3	SDMMC_D4	DCMI_D6	LCD_B6	EVENTOUT
	PB9	-	I2C4_SDA	TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS/I2S2_WS	DFSDM1_DATIN7	UART5_TX	-	CAN1_TX	SDMMC2_D5	I2C4_SMBA	SDMMC_D5	DCMI_D7	LCD_B7	EVENTOUT
	PB10	-	TIM2_CH3	-	-	I2C2_SCL	SPI2_SCK/I2S2_CK	DFSDM1_DATIN7	USART3_TX	-	- QUADSPI_BK1_NCS	OTG_HS_ULPI_D3	ETH_MII_RX_ER	-	-	LCD_G4	EVENTOUT
	PB11	-	TIM2_CH4	-	-	I2C2_SDA	-	DFSDM1_CKIN7	USART3_RX	-	-	OTG_HS_ULPI_D4	ETH_MII_TX_EN/ETH_RMII_TX_EN	-	DSI_TE	LCD_G5	EVENTOUT
	PB12	-	TIM1_BKIN	-	-	I2C2_SMBA	SPI2_NSS/I2S2_WS	DFSDM1_DATIN1	USART3_CK	UART5_RX	CAN2_RX	OTG_HS_ULPI_D5	ETH_MII_TXD0/ETH_RMII_TXD0	OTG_HS_ID	-	-	EVENTOUT
	PB13	-	TIM1_CH1N	-	-	-	SPI2_SCK/I2S2_CK	DFSDM1_CKIN1	USART3_CTS	UART5_TX	CAN2_TX	OTG_HS_ULPI_D6	ETH_MII_TXD1/ETH_RMII_TXD1	-	-	-	EVENTOUT
	PB14	-	TIM1_CH2N	-	TIM8_CH2N	USART1_TX	SPI2_MISO	DFSDM1_DATIN2	USART3_RTS	UART4_RTS	TIM12_CH1	SDMMC2_D0	-	OTG_HS_DM	-	-	EVENTOUT
	PB15	RTC_REFIN	TIM1_CH3N	-	TIM8_CH3N	USART1_RX	SPI2_MOSI/I2S2_SD	DFSDM1_CKIN2	-	UART4_CTS	TIM12_CH2	SDMMC2_D1	-	OTG_HS_DP	-	-	EVENTOUT

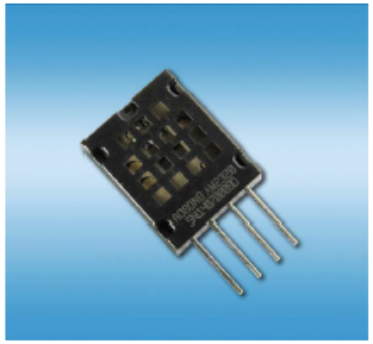
Also PB8 & PB9 on STM32F429-DISC1



AOSONG

Digital Temperature and Humidity Sensor

AM2320 Product Manual



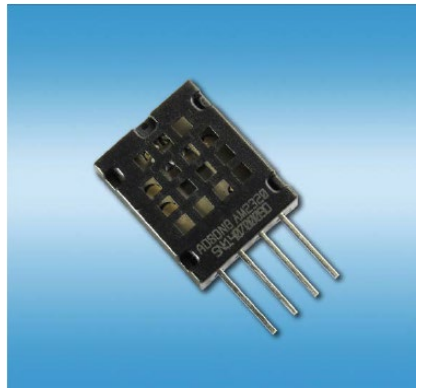
Product Features:

- Ultra-small size
- Super cost-effective
- Ultra-low voltage operation
- Excellent long-term stability
- Standard I2C and single-bus output

For more information, please visit: www. aosong .com

AM2320

- **Digital** Temperature & Humidity Sensor
- Standard I2C and single-bus output (1Wire)
- CRC checksum



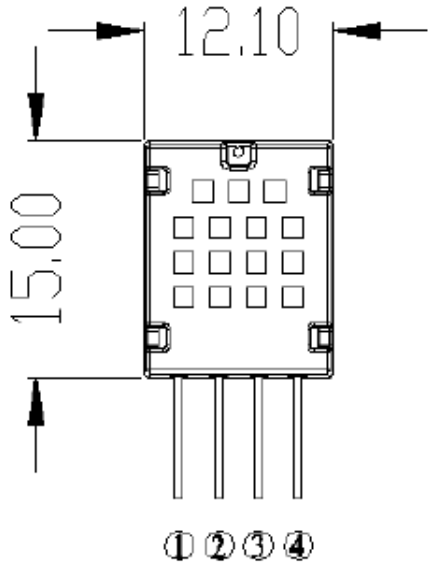


Table 4: AM2320 pin assignment

Pin	Name	Description
1	VDD	Power supply(3.1-5.5V)
2	SDA	Serial data, bidirectional port
3	GND	Ground
4	SCL	Serial clock input port (single bus ground)

Sensor performance : Relative Humidity

Table 1: AM2320 relative humidity performance table

parameter	condition	min	typ	max	unit
resolution			0.1		%RH
Range		0		99.9	%RH
Accuracy	25℃		±3		%RH
Repeatability			±0. 1		%RH
Interchangeability		Completely interchangeable			
Response time	1/e(63%)		<5		S
Sluggish			±0.3		%RH
Drift	Typical values		<0.5		%RH /yr

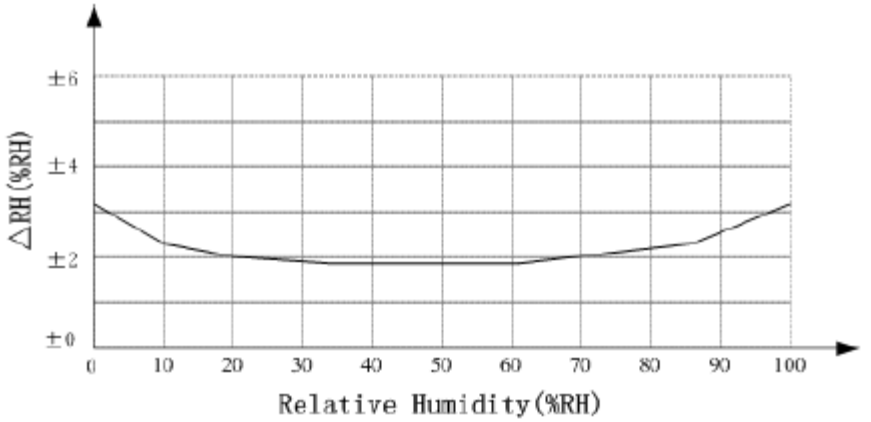


Figure 2: 25 ℃ relative humidity of maximum error AM2320

Sensor performance : Temperature

Table 2: AM2320 relative temperature performance table

parameter	condition	min	typ	max	unit
resolution			0.1		℃
			16		bit
Accuracy			±0.5		℃
Range		-40		80	℃
Repeatability			±0.2		℃
Interchangeability					
Response time	1/e(63%)		<5		S
Drift			±0.1		℃/yr

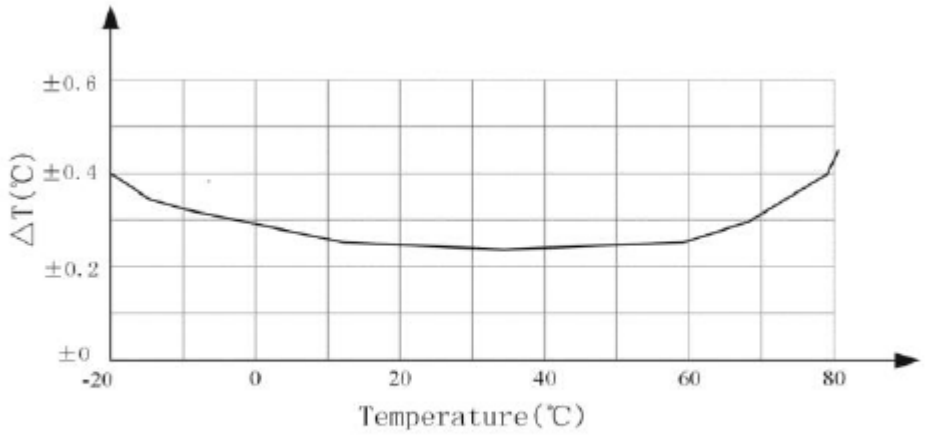


Figure 3: The maximum error of the temperature sensor

AM2320 Wiring

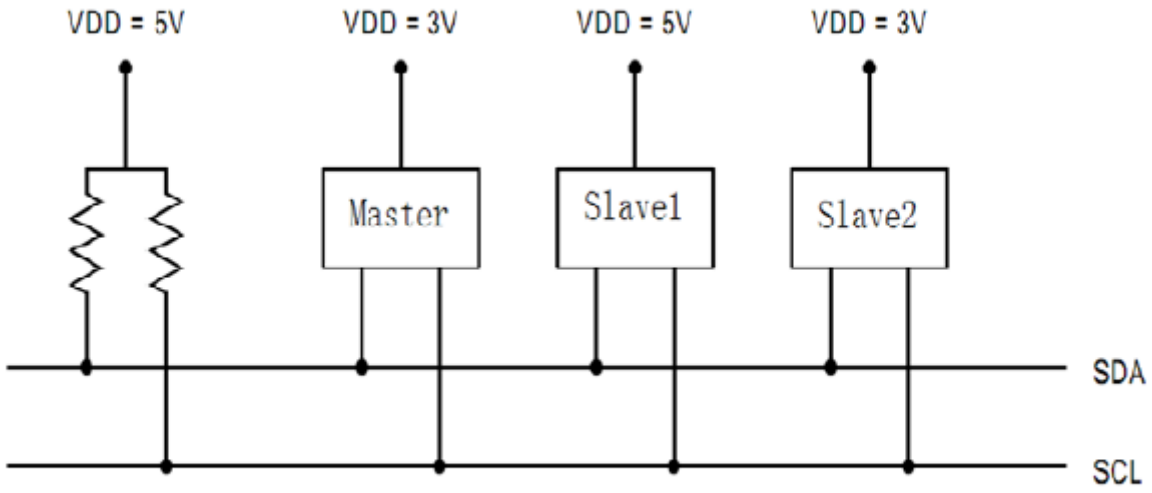
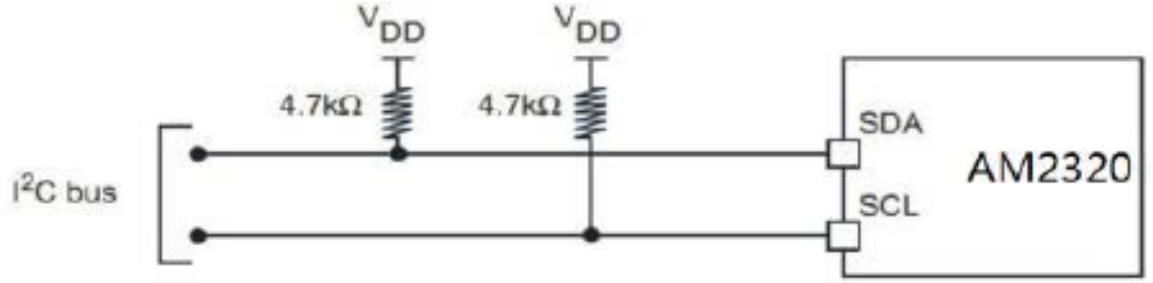


Figure 5: I ^{2} C typical configuration



Data Packet Format

◎**Communication data (information frame) format**

Data formats:

Data length:

◎

I^2C data+W/R	Function Code	Data Area	CRC
1byte	1 byte	N-byte	16-bit CRC (cyclic redundancy code)

Table 7:C Mod Bus part of the function code

Function Code	Definitions	Operation (binary)
0x03	Reading Register Data	Read one or more data registers
0x10	Write Multiple Registers	Multiple sets of binary data to write multiple registers

AM2320 Registers

Table 8: AM2320 Data Register Table

Register information	Address	Register information	Address	Register information	Address	Register information	Address
High humidity	0x00	Model High	0x08	Users register a high	0x10	Retention	0x18
Low humidity	0x01	Model Low	0x09	Users register a low	0x11	Retention	0x19
High temperature	0x02	The version number	0x0A	Users register 2 high	0x12	Retention	0x1A
Low temperature	0x03	Device ID (24-31) Bit	0x0B	Users register 2 low	0x13	Retention	0x1B
Retention	0x04	Device ID (24-31) Bit	0x0C	Retention	0x14	Retention	0x1C
Retention	0x05	Device ID (24-31) Bit	0x0D	Retention	0x15	Retention	0x1D
Retention	0x06	Device ID (24-31) Bit	0x0E	Retention	0x16	Retention	0x1E
Retention	0x07	Status Register	0x0F	Retention	0x17	Retention	0x1F

Sending Command to AM2320 to Read Data

1. Function code "03": Read registers multiplexed sensor

The host sends reading frame format:

START + (I^2C address + W) + function code (0x03) + start address + number of registers + STOP

Host read return data:

START + (I^2C address + R) + sequential read sensor data returned + STOP

Sensor response frame format:

Function code (0x03) + number + data +CRC [1]

For example: Host sequential read sensor data: the starting address for the register data of four sensors 0x00.

Sensor data register address and data:

Register Address	Register data	Data Description	Register Address	Register data	Data Description
0x00	0x01	High humidity	0x02	0x00	High temperature
0x01	0XF4	Low humidity	0x03	0xFA	Low temperature

Host message format sent:

The host sends	Byte count	Transmitting information	Remarks
Sensor address	1	0xB8	Sensor C address (0xB8) + W (0)
Function Code	1	0x03	Read register
Starting address	1	0x00	Register start address is 0x00
Number of registers	1	0x04	Read the number of register

AM2320 Respond with Data

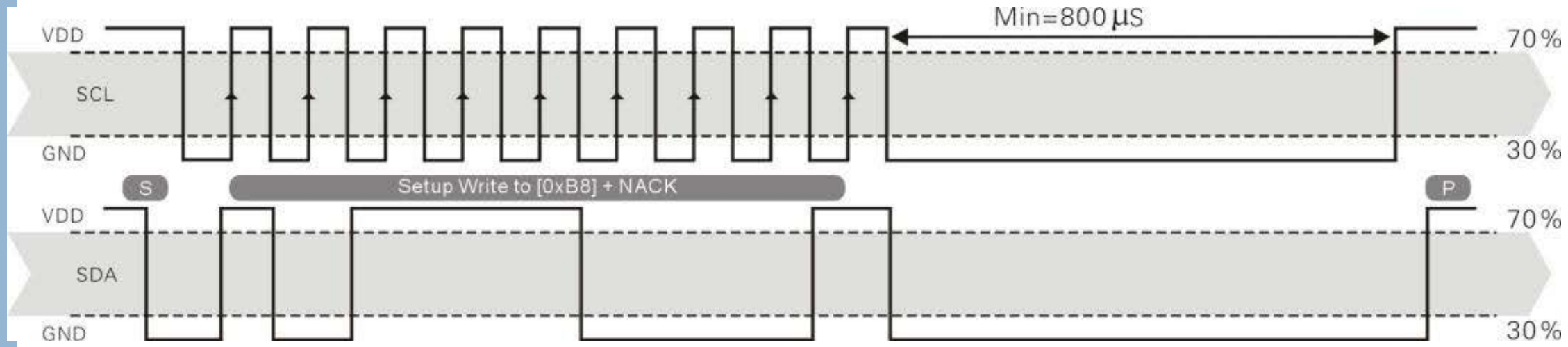
Slave response	Byte count	Transmitting information	Remarks
Function Code	1	0x03	Read register
Returns the number of bytes	1	0x04	Returns 4 of 4 byte register
Register 1	1	0x01	Address for the content of 0x00 (high humidity bytes)
Register 2	1	0XF4	Address for the content of 0x01 (low humidity bytes)
Register 3	1	0x00	Address for the content of 0x01 (low humidity bytes)
Register 4	1	0XFA	Address for the content 0x03 (temperature low byte)
CRC code	2	31A5	Sensors calculate the CRC code returned, low byte first;

Humidity: 01F4 = 1×256+15×16+4 =500 => humidity = 500÷10=50.0%RH;

Temperature: 00FA= 15×16+10 = 250 => temperature = 250÷10 = 25.0℃

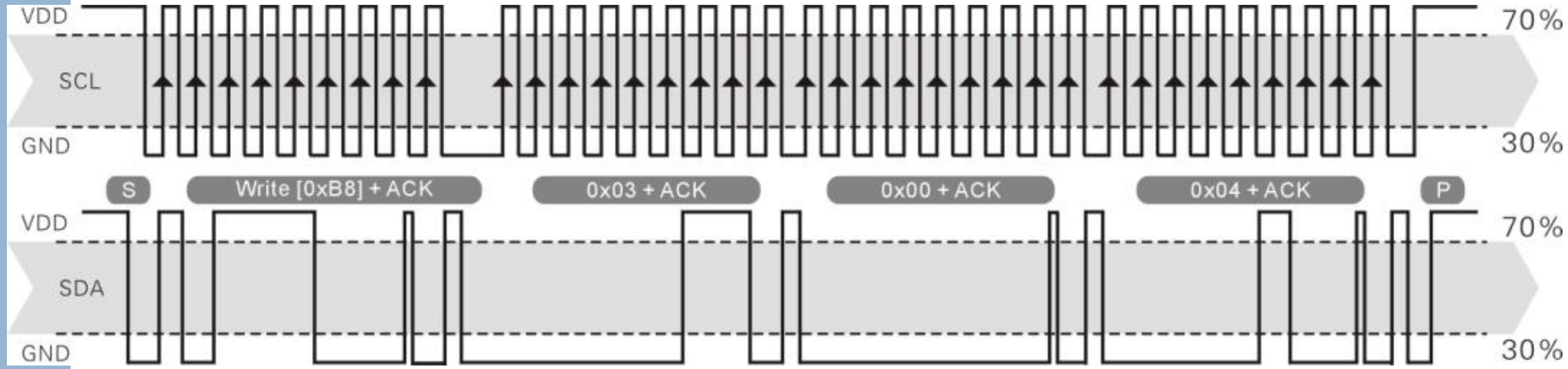
Timing Diagram : Step 1 Wake Sensor

- the starting signal + 0xB8 + wait (> 800us) + stop signal

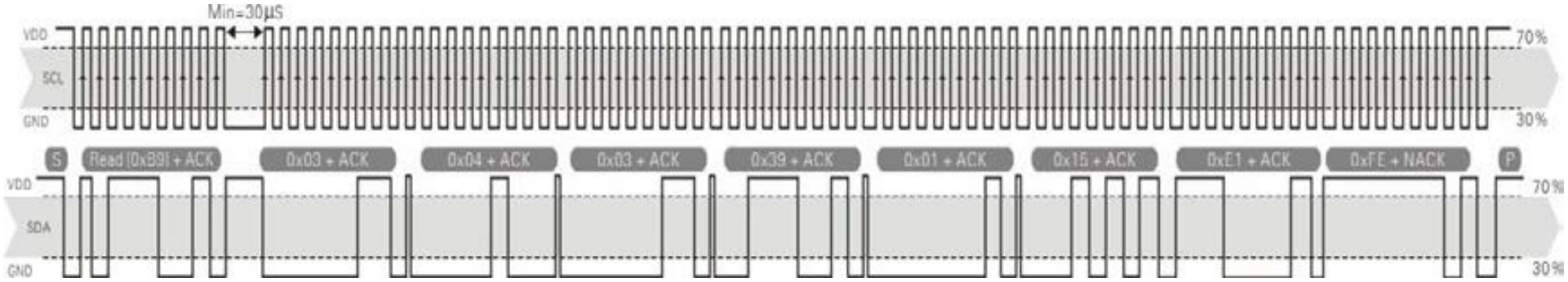


Timing Diagram : Step 2 Send Read Command

- The host sends commands to: START + 0xB8 (SLA) + 0x03 (function code) + 0x00 (starting address) +0x04 (register length) + STOP



Timing Diagram : Step 3 Receive Sensor Data

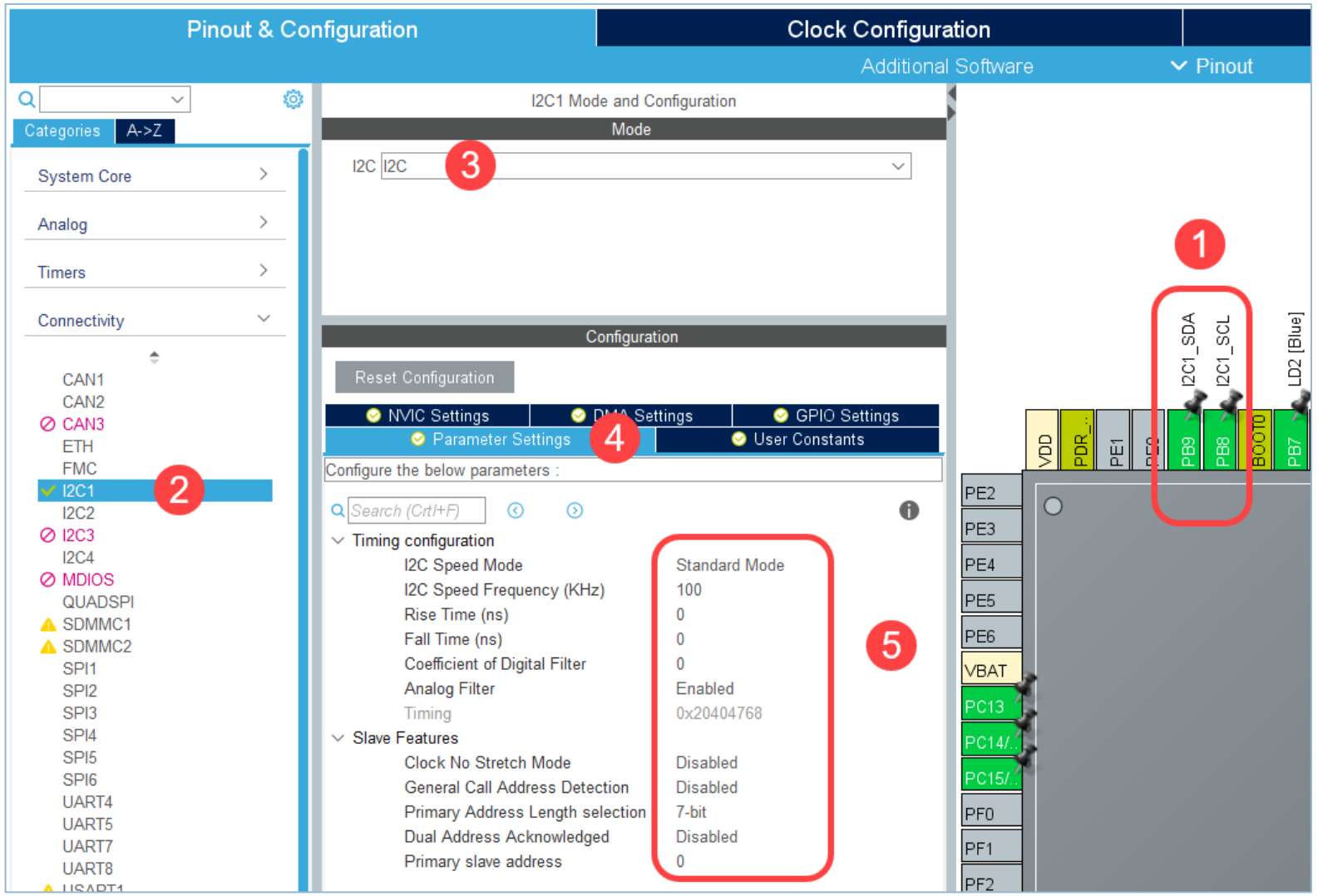


Host read back the data as follows:

- **0x03** (Function Code) + **0x04** (data length) +
  0x03 (high humidity) + **0x39** (low humidity) +
  0x01 (high temperature) + **0x15** (low temperature) +
  0xE1 (CRC checksum low byte) + **0xFE** (CRC checksum high byte)
- Therefore: **0339H** = 825_{10} => humidity = 825÷10 = 82.5% RH

  0115H = 277_{10} => temperature = 277÷10 = 27.7 ℃

Setting STM32Cube for I2C (AM2320)



Additional Setting STM32Cube for I2C (AM2320)

- Config PD8 & PD9 as UART3
- Config PB0 as LED1

I2C HAL Lib

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()

Declaration main.c

```
39  /* Includes ------------------------------------------------------------------*/
40  #include "main.h"
41  #include "stm32f7xx_hal.h"
42
43  /* USER CODE BEGIN Includes */
44  #include "string.h"
45  /* USER CODE END Includes */
46
47  /* Private variables ---------------------------------------------------------*/
48
49  I2C_HandleTypeDef hi2c1;
50
51  UART_HandleTypeDef huart3;
52
53  /* USER CODE BEGIN PV */
54  /* Private variables ---------------------------------------------------------*/
55  float h=30.0,t=40.0;
56  uint8_t step = 0;
57  HAL_StatusTypeDef status;
58  /* USER CODE END PV */
59
60  /* Private function prototypes -----------------------------------------------*/
61  void SystemClock_Config(void);
62  static void MX_GPIO_Init(void);
63  static void MX_USART3_UART_Init(void);
64  static void MX_I2C1_Init(void);
65
66  /* USER CODE BEGIN PFP */
67  /* Private function prototypes -----------------------------------------------*/
68  uint16_t CRC16_2(uint8_t *, uint8_t );
69  /* USER CODE END PFP */
```

Declare local variables

```
80  int main(void)
81  {
82    /* USER CODE BEGIN 1 */
83    char str[50];
84    uint8_t cmdBuffer[3];
85    uint8_t dataBuffer[8];
86    /* USER CODE END 1 */
```

Setting before enter while loop

```
104   /* Initialize all configured peripherals */
105   MX_GPIO_Init();
106   MX_USART3_UART_Init();
107   MX_I2C1_Init();
108   /* USER CODE BEGIN 2 */
109   sprintf(str, "\n\rAM2320 I2C DEMO Starting . . .\n\r");
110
111   HAL_UART_Transmit(&huart3, (uint8_t*) str, strlen(str),200);
112
113   cmdBuffer[0] = 0x03;
114   cmdBuffer[1] = 0x00;
115   cmdBuffer[2] = 0x04;
116   /* USER CODE END 2 */
```

```
118   /* Infinite loop */
119   /* USER CODE BEGIN WHILE */
120   while (1)
121   {
122
123   /* USER CODE END WHILE */
124
125   /* USER CODE BEGIN 3 */
126     //Send Temp & Humid via UART2
127     sprintf(str, "Temperature = %4.1f\tHumidity = %4.1f\n\r", t, h);
128     while(__HAL_UART_GET_FLAG(&huart3,UART_FLAG_TC)==RESET){}
129     HAL_UART_Transmit(&huart3, (uint8_t*) str, strlen(str),200);
130
131     HAL_Delay(5000); //>3000 ms
132     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
133
134     //Wake up sensor
135     HAL_I2C_Master_Transmit(&hi2c1, 0x5c<<1, cmdBuffer, 3, 200);
136     //Send reading command
137     HAL_I2C_Master_Transmit(&hi2c1, 0x5c<<1, cmdBuffer, 3, 200);
138
139     HAL_Delay(1);
140
141     //Receive sensor data
142     HAL_I2C_Master_Receive(&hi2c1, 0x5c<<1, dataBuffer, 8, 200);
143
144     uint16_t Rcrc = dataBuffer[7] << 8;
145     Rcrc += dataBuffer[6];
146     if (Rcrc == CRC16_2(dataBuffer, 6)) {
147       uint16_t temperature = ((dataBuffer[4] & 0x7F) << 8) + dataBuffer[5];
148       t = temperature / 10.0;
149       t = (((dataBuffer[4] & 0x80) >> 7)== 1) ? (t * (-1)) : t ; // the temperature can be negative
150
151       uint16_t humidity = (dataBuffer[2] << 8) + dataBuffer[3];
152       h = humidity / 10.0;
153     }
154   }
155   /* USER CODE END 3 */
```

While loop

HAL_I2C_Master_Transmit

HAL_I2C_Master_Transmit

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmits in master mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Master_Receive

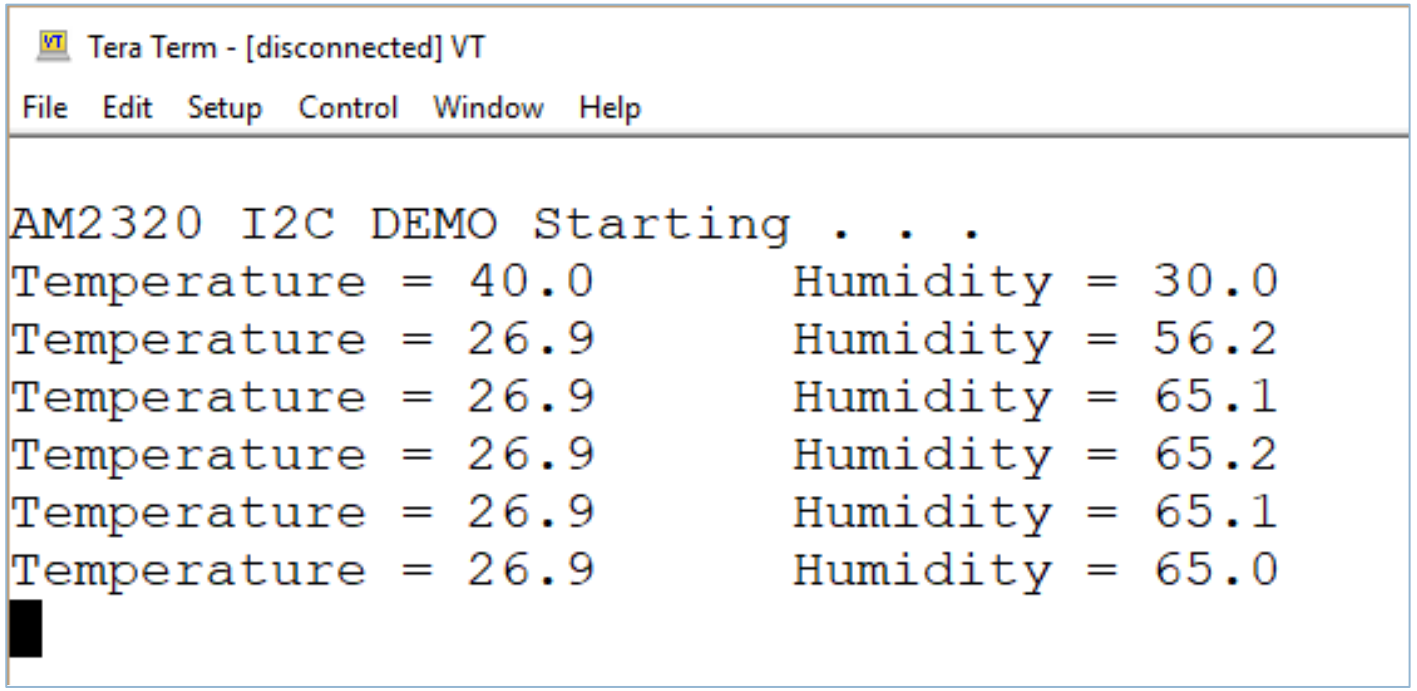
HAL_I2C_Master_Receive

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receives in master mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

CRC Checksum

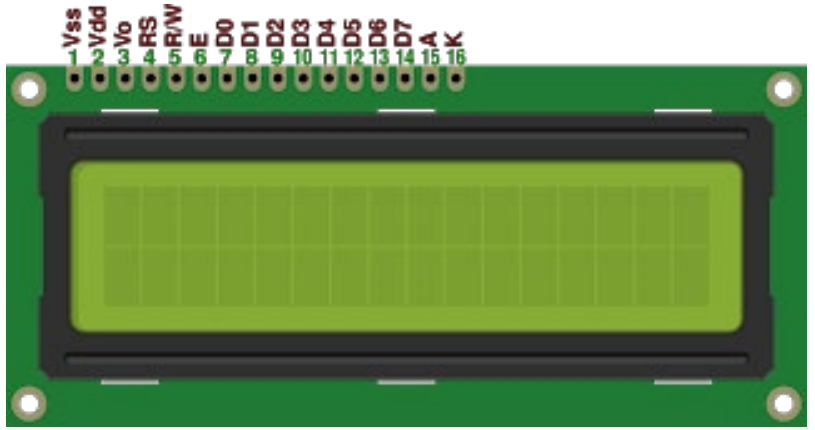
```
316 /* USER CODE BEGIN 4 */
317 uint16_t CRC16_2(uint8_t *ptr, uint8_t length)
318 {
319       uint16_t  crc = 0xFFFF;
320       uint8_t   s   = 0x00;
321
322       while(length--) {
323         crc ^= *ptr++;
324         for(s = 0; s < 8; s++) {
325           if((crc & 0x01) != 0) {
326             crc >>= 1;
327             crc ^= 0xA001;
328           } else crc >>= 1;
329         }
330       }
331       return crc;
332 }
333 /* USER CODE END 4 */
```

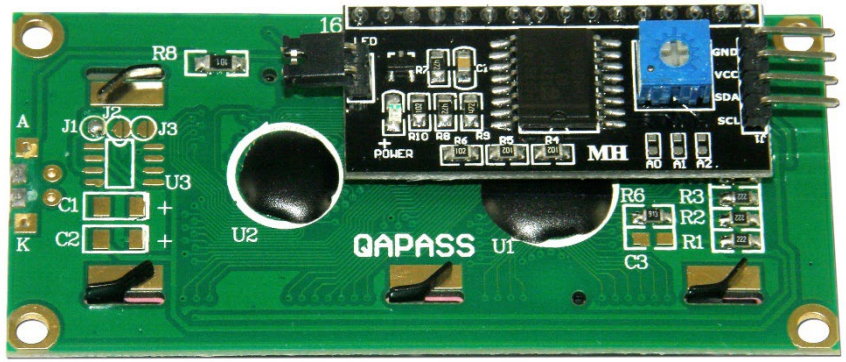
Show result in Tera Term via UART2



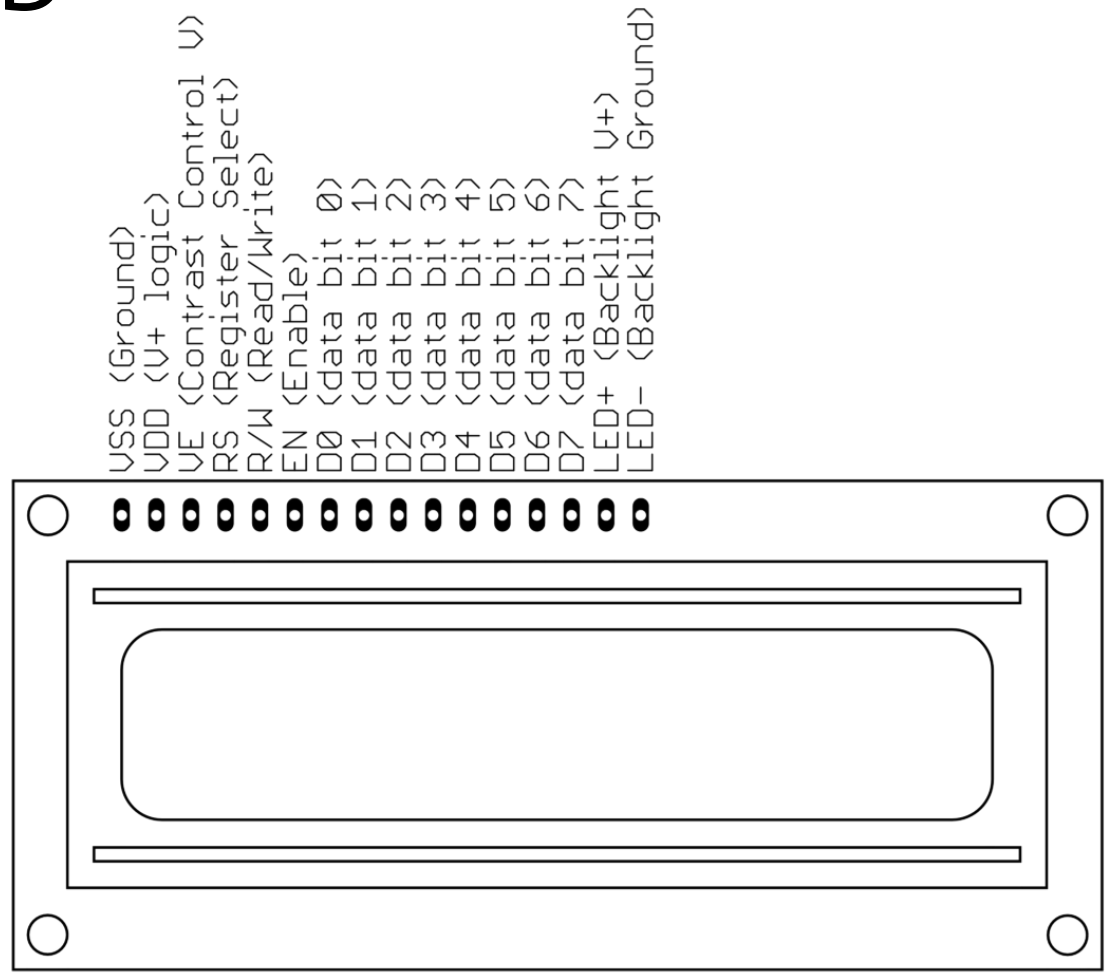
I2C LCD

- Character LCD
  - 16x2 characters
  - Parallel Connection
  - HD44780U LCD Controller
- I/O Expander
  - I2C
  - PCF8574





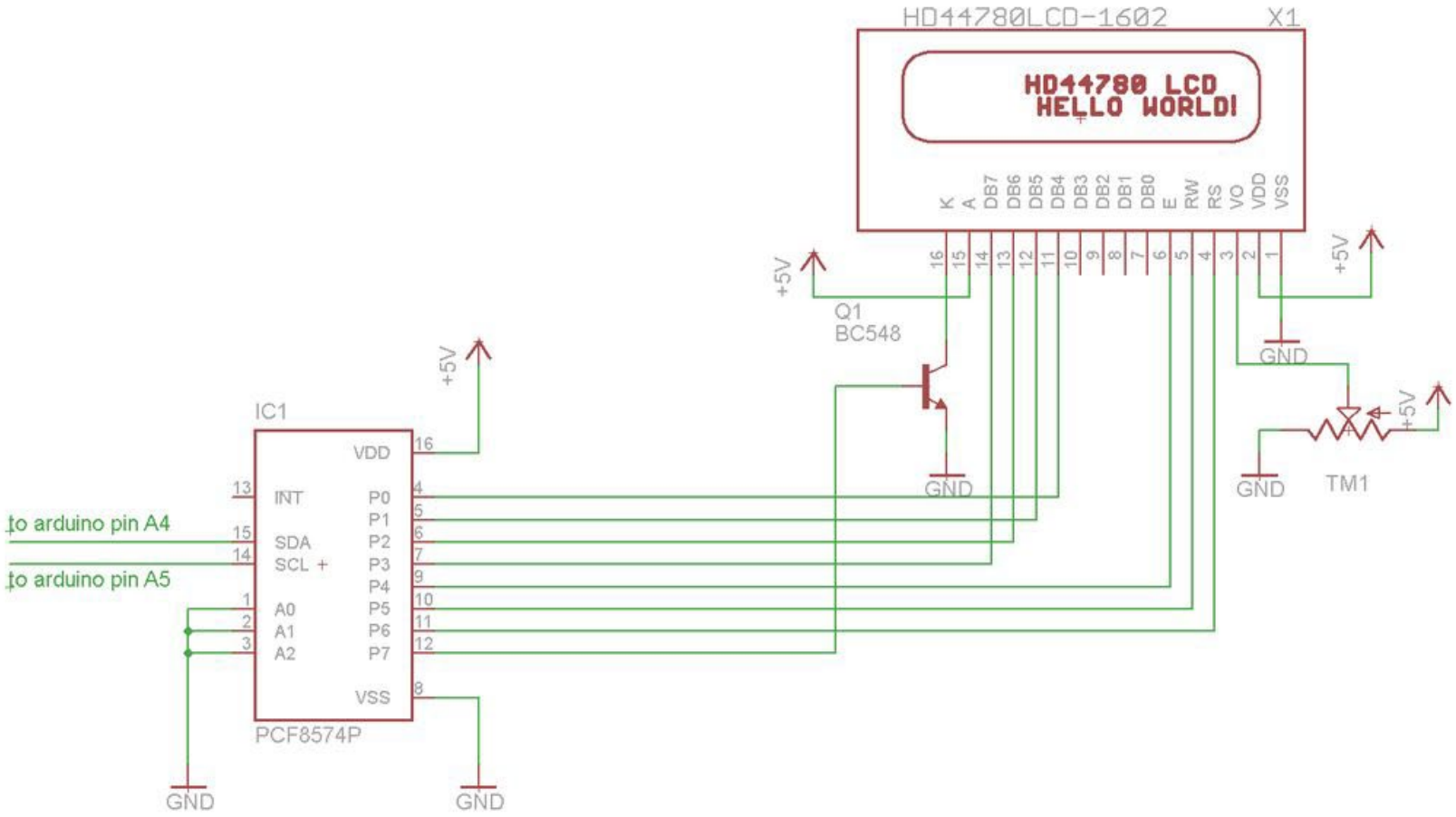
LCD



HD44780 EagleCad part by Adafruit Industries.

VSS (Ground)] POWER
VDD (V+ logic)]
VE (Contrast Control V)] CONTRAST
RS (Register Select)] REGISTER CONTROL
R/W (Read/Write)]
EN (Enable)]
D0 (data bit 0)] NOT USED
D1 (data bit 1)]
D2 (data bit 2)]
D3 (data bit 3)]
D4 (data bit 4)] DATA CTRL (4 bits)
D5 (data bit 5)]
D6 (data bit 6)]
D7 (data bit 7)]
LED+ (Backlight V+)] LED POWER
LED- (Backlight Ground)]

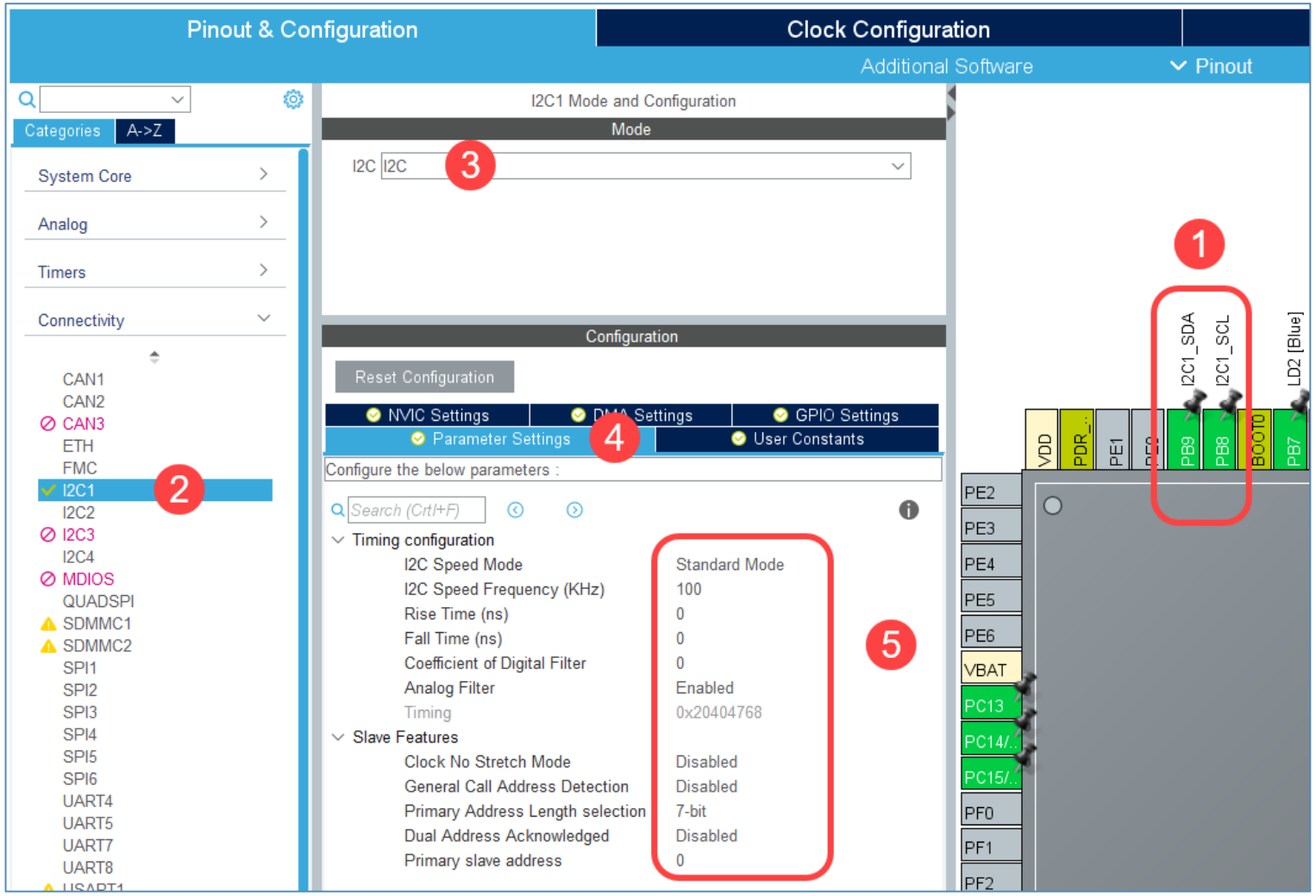
LCD Schematic



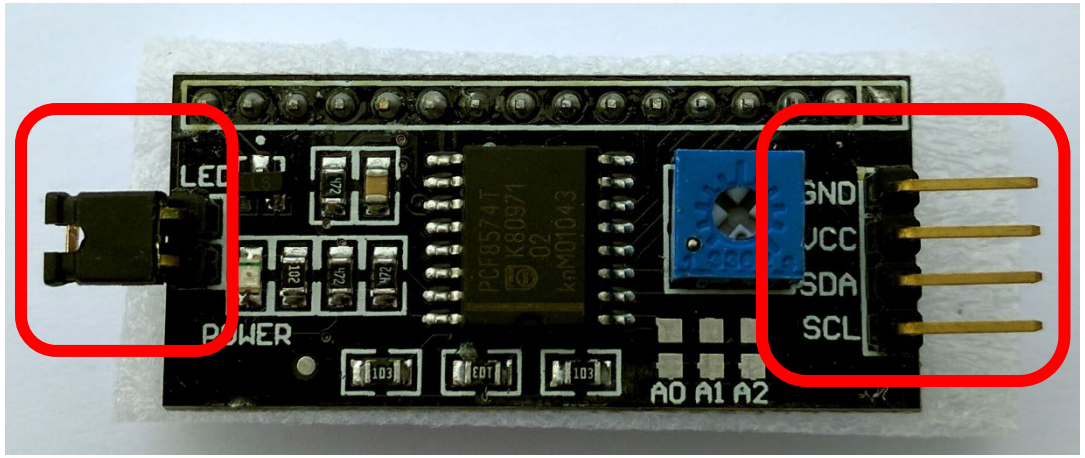
HD44780U LCD Controller

- Open Datasheet

Setting STM32Cube for I2C



การเชื่อมต่อกับ I2C ของ LCD กับ STM32



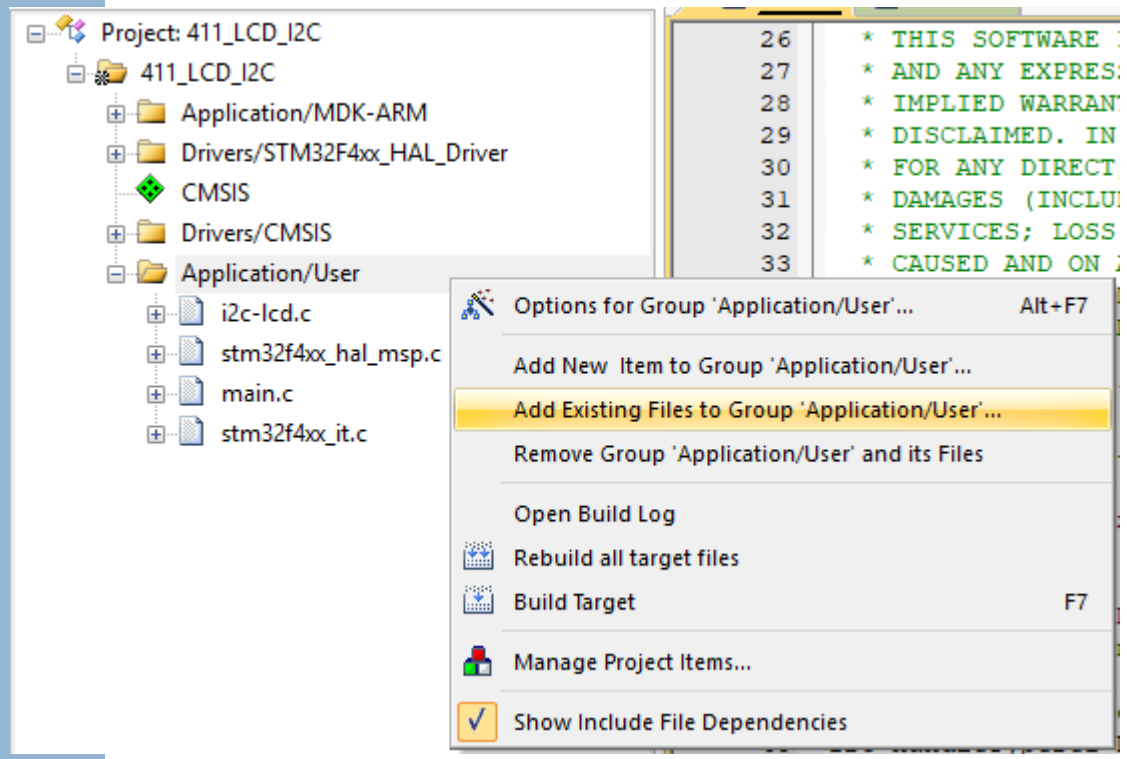
I2C LCD	STM32
GND	GND
Vcc	5V
SDA	PB8 (D14, I2C1_SDA)
SCL	PB9 (D15, I2C1_SCL)

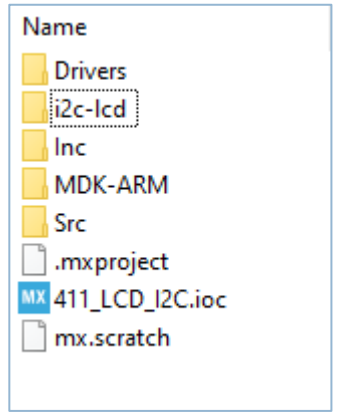
คง Jumper : Backlight ติด

ถอด Jumper : Backlight ดับ

Adding Library into Keil

เพิ่ม i2c-lcd.c เข้าไปใน Project

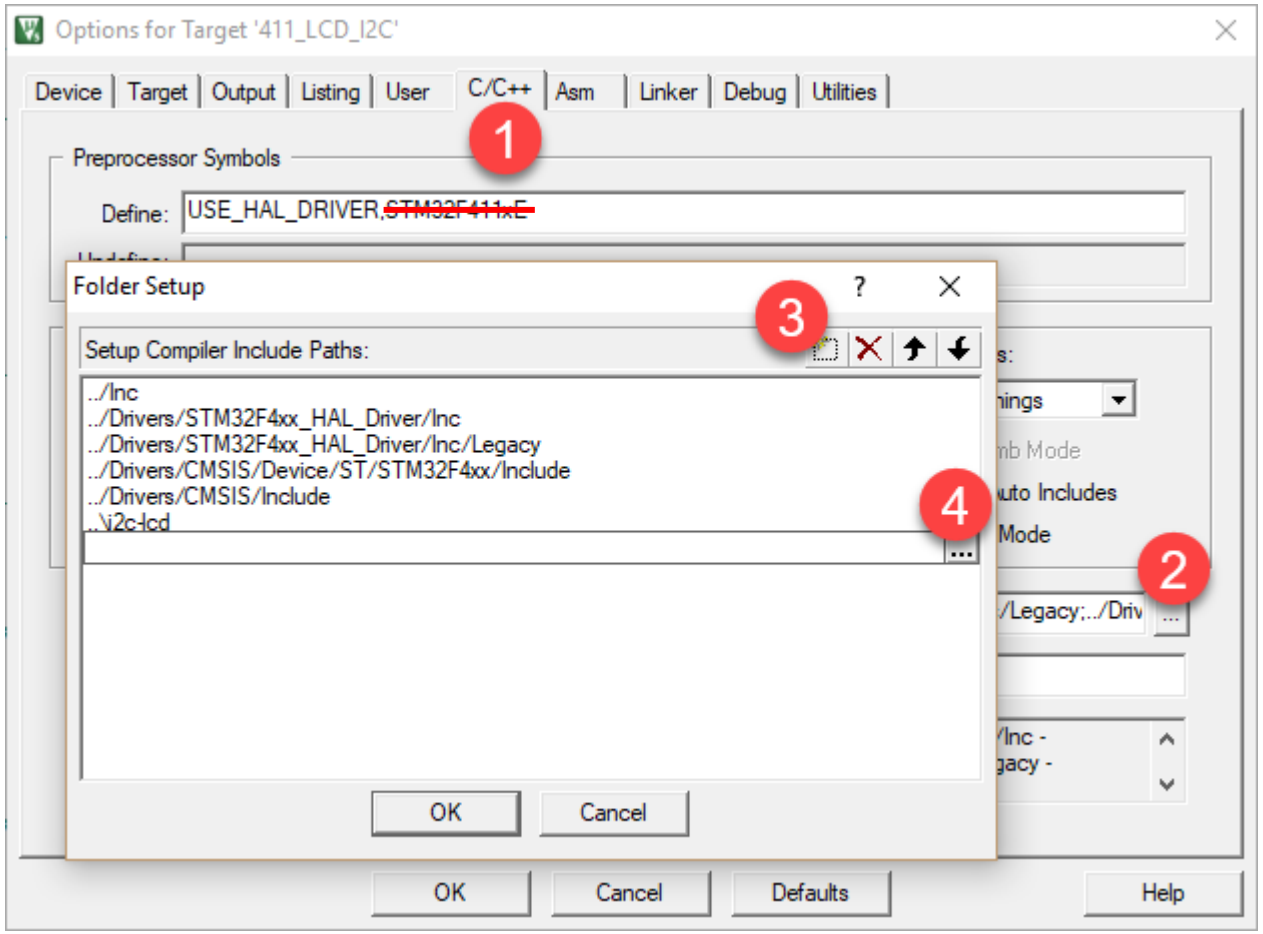


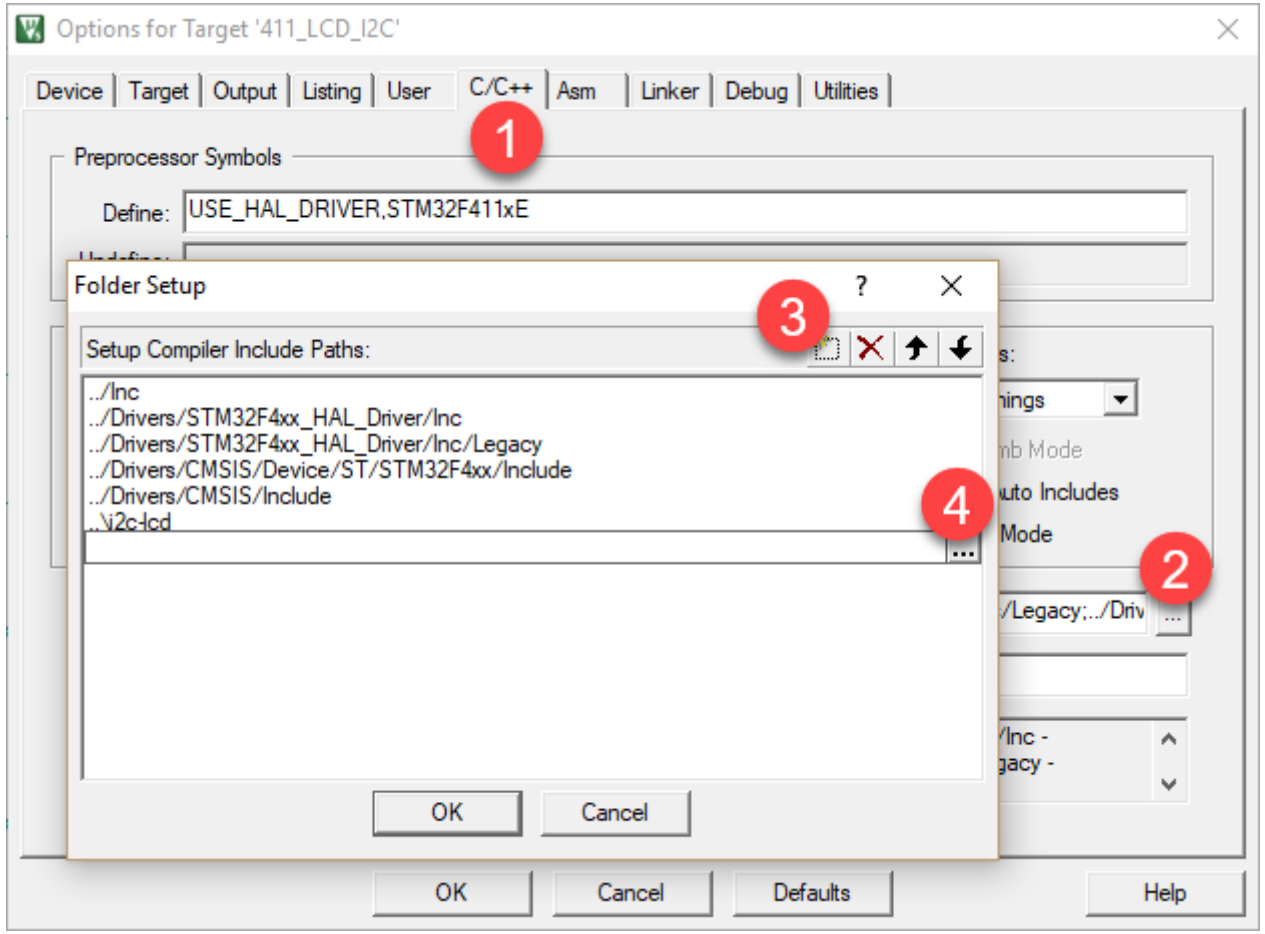


- คลิกขวาที่โฟลเดอร์ Application/User
- เลือก Add Existing Files
- เลือกไฟล์ i2c-lcd.c
- กด Add

เพิ่ม Path สำหรับ i2c-lcd.h

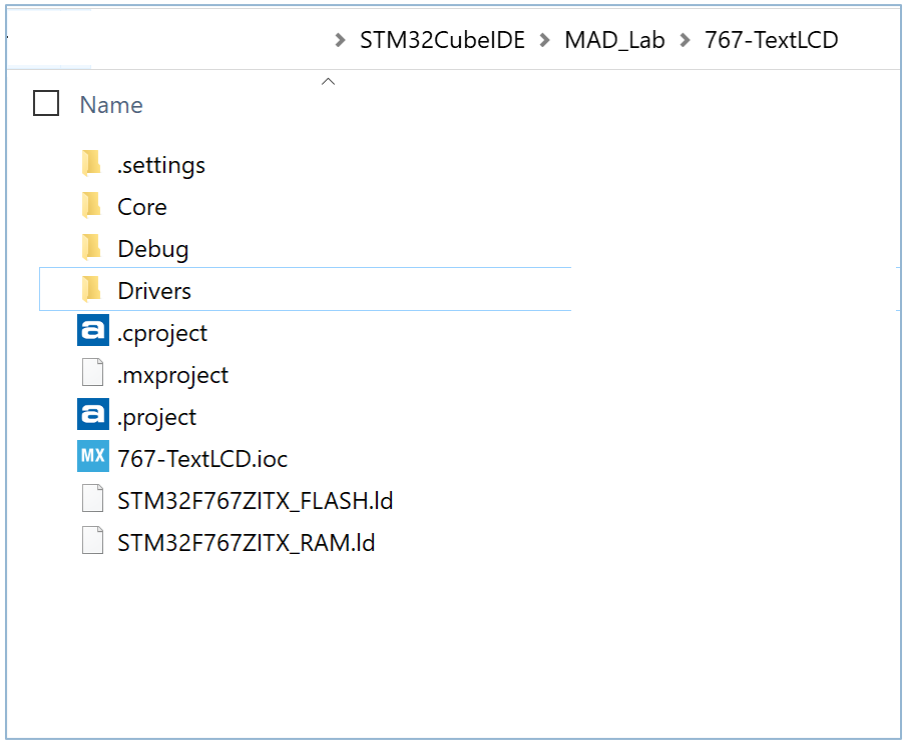
- เมนู Project -> Options for Target
- ทำขั้นตอน 1-4
- เลือกโฟลเดอร์ที่บรรจุไฟล์ i2c-lcd.h

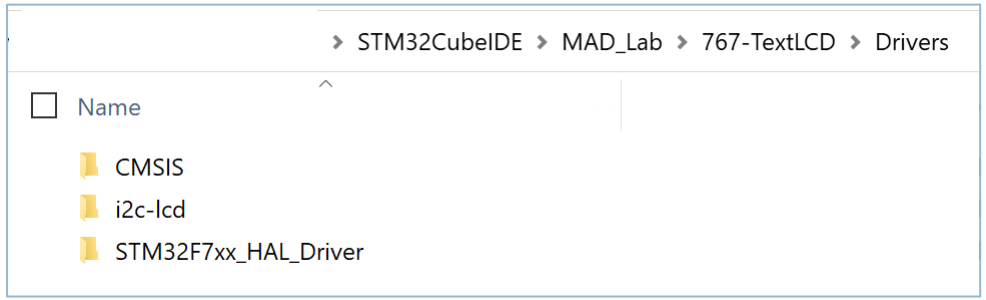


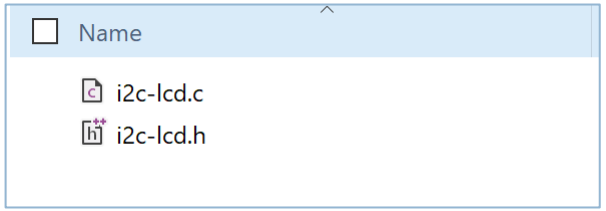

Options for Target '411_LCD_I2C'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
1
Preprocessor Symbols
Define:
USE_HAL_DRIVER,STM32F411xE
Folder Setup
3
Setup Compiler Include Paths:
../Inc
../Drivers/STM32F4xx_HAL_Driver/Inc
../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F4xx/Include
../Drivers/CMSIS/Include
..\i2c-lcd
4
OK
Cancel
2
OK
Cancel
Defaults
Help

Adding Library into STM32CubeMX

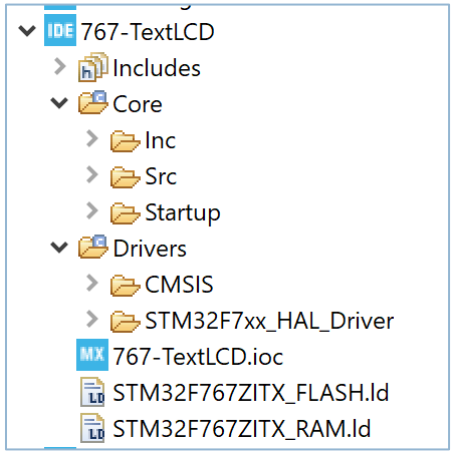
i2c-lcd Library

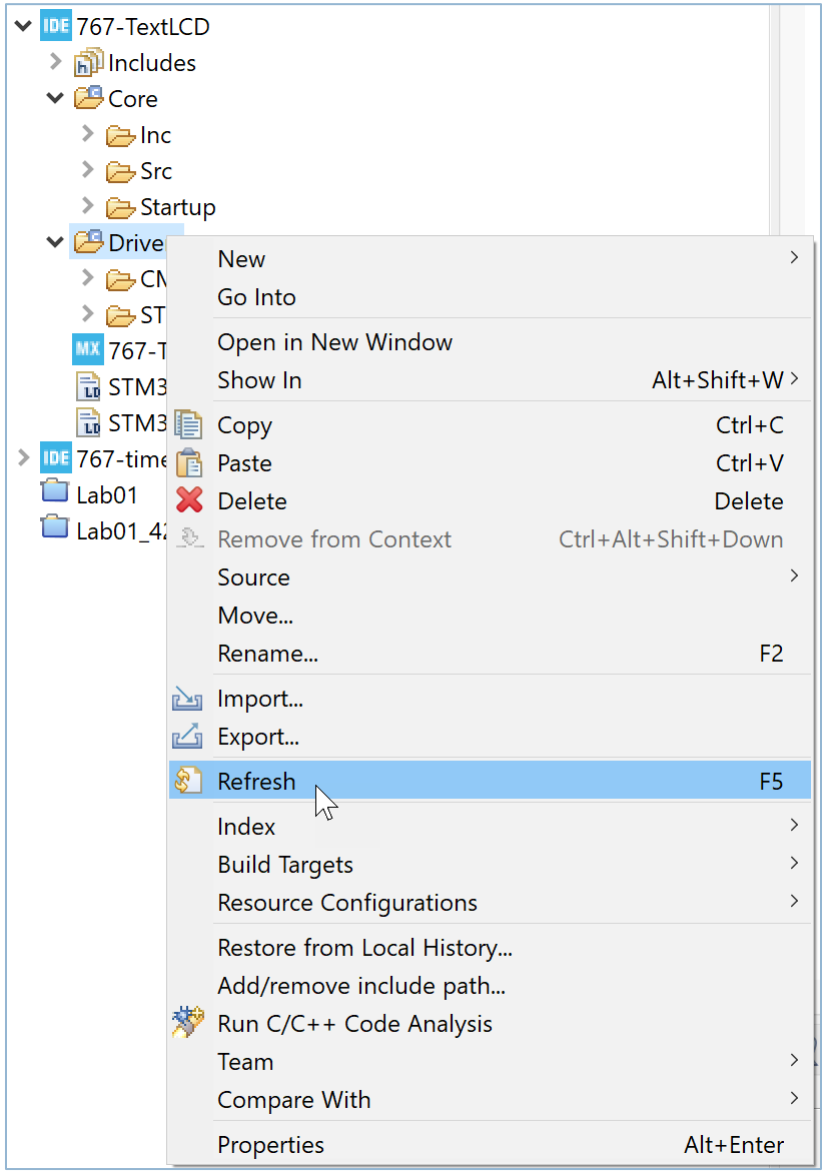


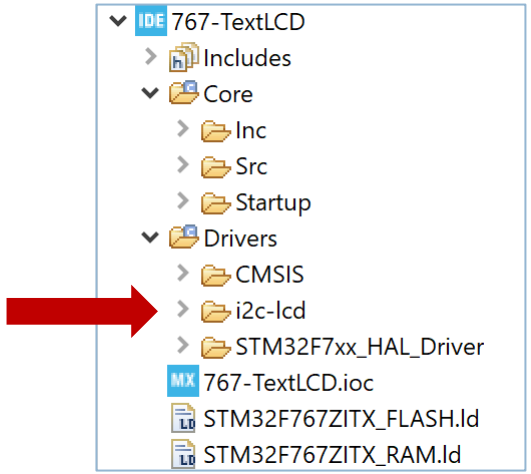




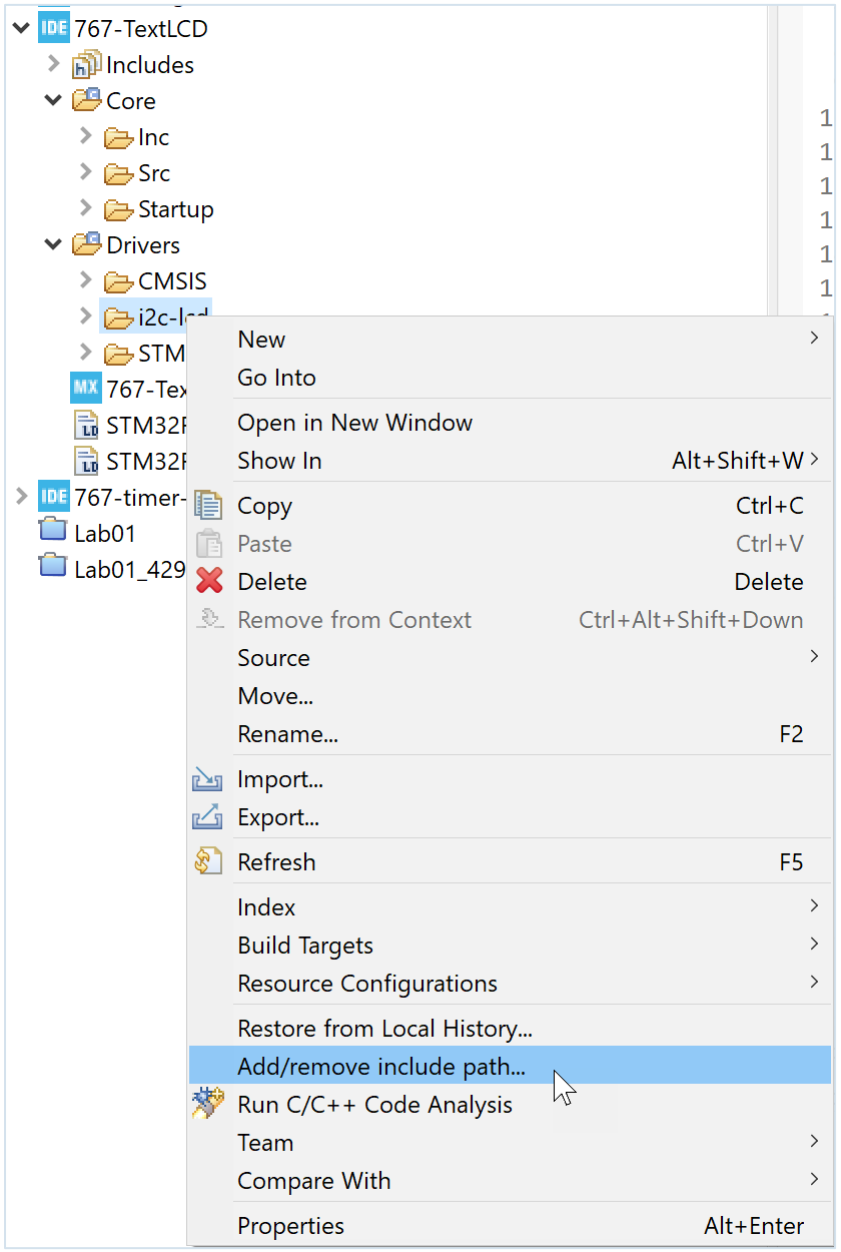
Refresh

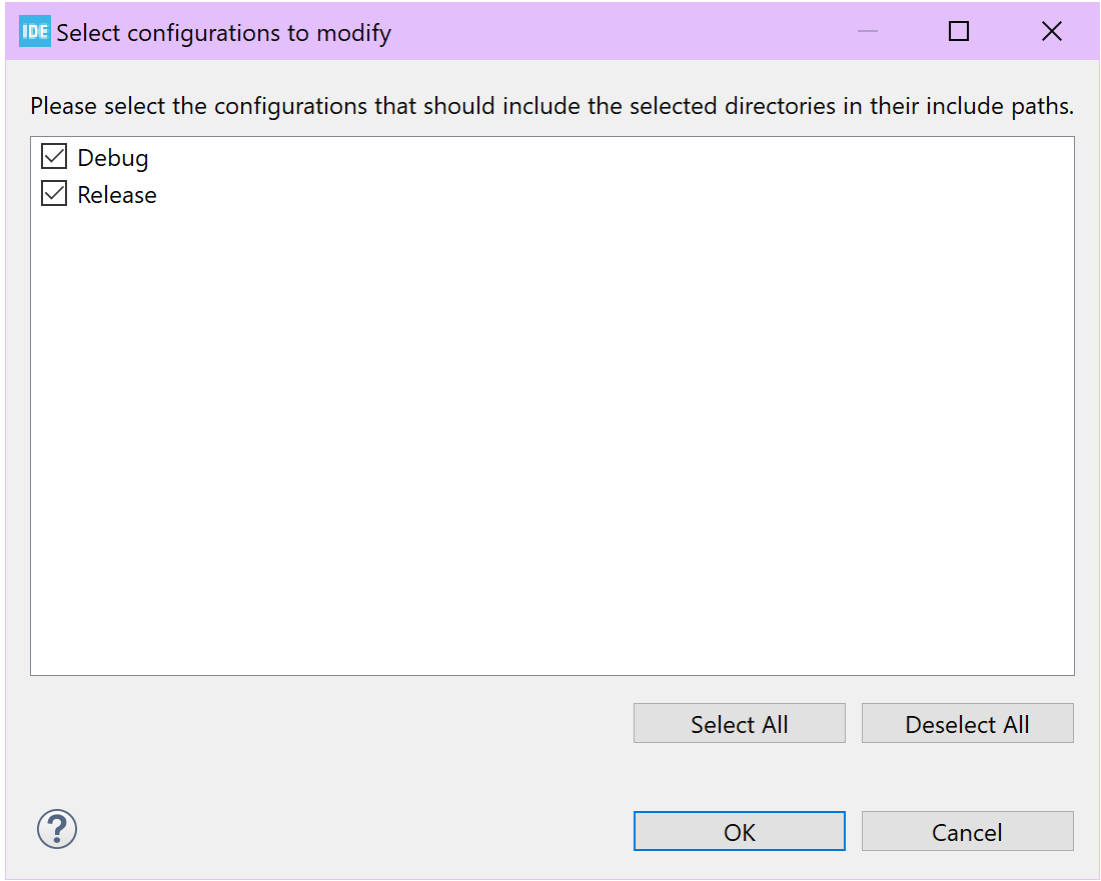

767-TextLCD
Includes
Core
Inc
Src
Startup
Drivers
CMSIS
STM32F7xx_HAL_Driver
767-TextLCD.ioc
STM32F767ZITX_FLASH.ld
STM32F767ZITX_RAM.ld


767-TextLCD
Includes
Core
Inc
Src
Startup
Drive
CM
ST
767-T
STM3
STM3
767-time
Lab01
Lab01_4
New
Go Into
Open in New Window
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Source
Move...
Rename... F2
Import...
Export...
Refresh F5
Index
Build Targets
Resource Configurations
Restore from Local History...
Add/remove include path...
Run C/C++ Code Analysis
Team
Compare With
Properties Alt+Enter

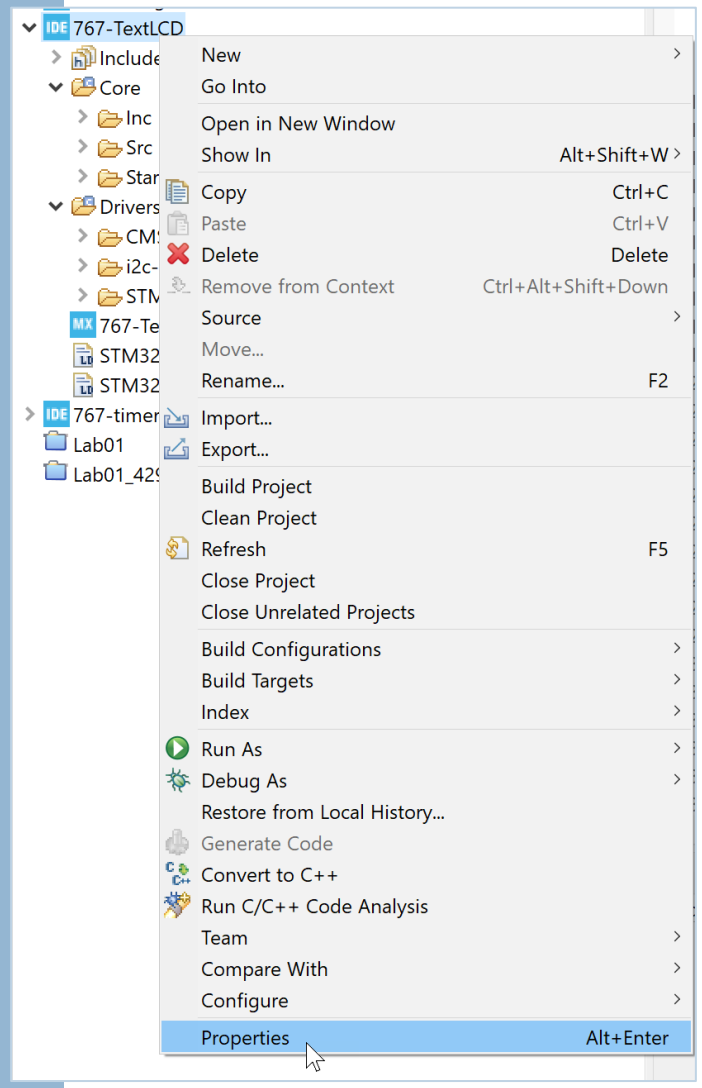

767-TextLCD
Includes
Core
Inc
Src
Startup
Drivers
CMSIS
i2c-lcd
STM32F7xx_HAL_Driver
767-TextLCD.ioc
STM32F767ZITX_FLASH.ld
STM32F767ZITX_RAM.ld

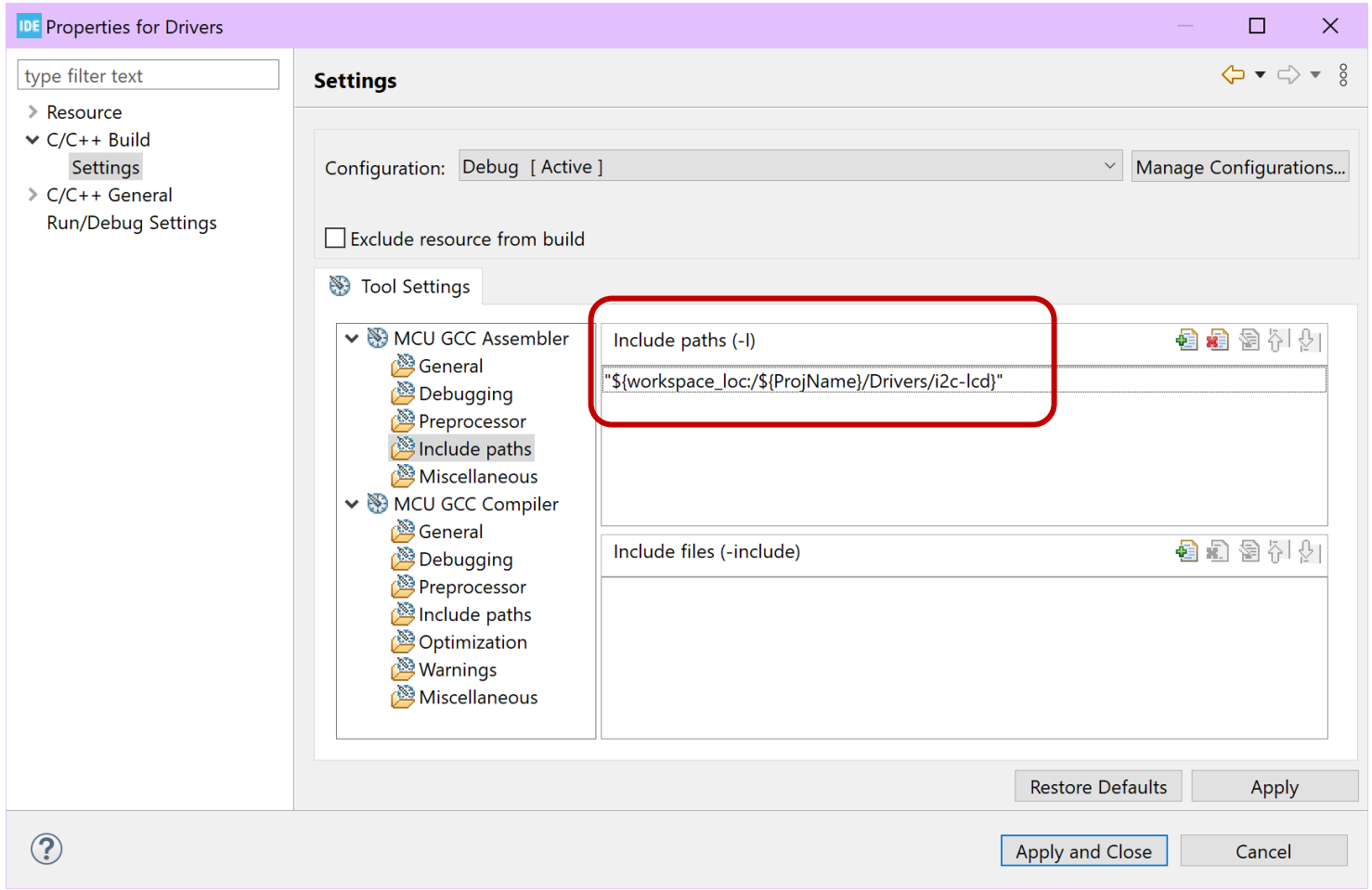
Add “Include Path”



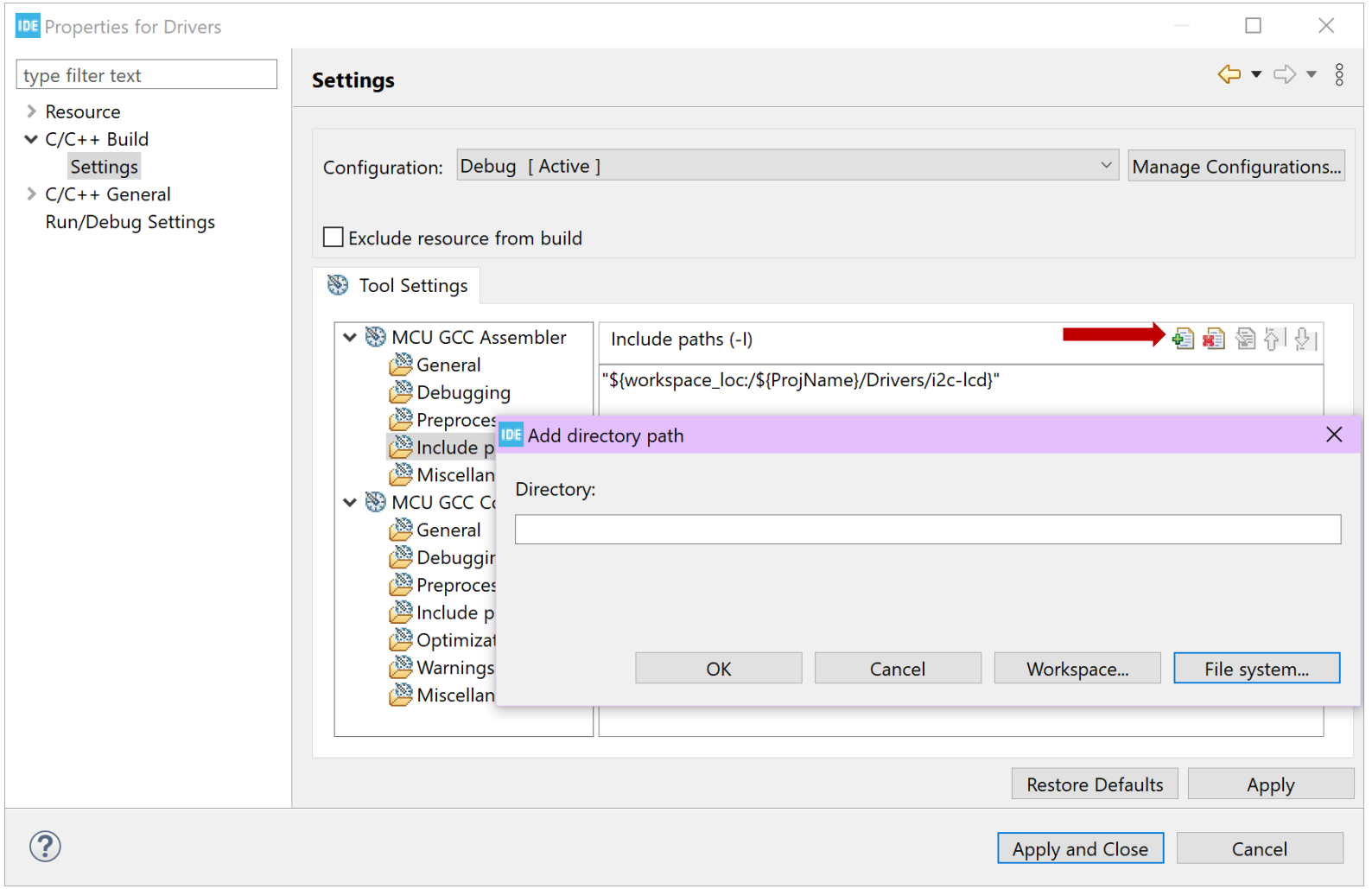


Check


767-TextLCD
Include
Core
Inc
Src
Star
Drivers
CM
i2c-
STM
767-Te
STM32
STM32
767-timer
Lab01
Lab01_42
New
Go Into
Open in New Window
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Projects
Build Configurations
Build Targets
Index
Run As
Debug As
Restore from Local History...
Generate Code
Convert to C++
Run C/C++ Code Analysis
Team
Compare With
Configure
Properties Alt+Enter


Properties for Drivers
type filter text
Resource
C/C++ Build
Settings
C/C++ General
Run/Debug Settings
Settings
Configuration: Debug [Active]
Manage Configurations...
Exclude resource from build
Tool Settings
MCU GCC Assembler
General
Debugging
Preprocessor
Include paths
Miscellaneous
MCU GCC Compiler
General
Debugging
Preprocessor
Include paths
Optimization
Warnings
Miscellaneous
Include paths (-I)
"${workspace_loc:/${ProjName}/Drivers/i2c-lcd}"
Include files (-include)
Restore Defaults
Apply
Apply and Close
Cancel

Add Manually



Coding

เพิ่ม Code ก่อนเข้า while loop

```
39  /* Includes ------------------------
40  #include "main.h"
41  #include "stm32f4xx_hal.h"
42
43  /* USER CODE BEGIN Includes */
44  #include "i2c-lcd.h"
45  /* USER CODE END Includes */
46
47  /* Private variables ----------------
48  I2C_HandleTypeDef hi2c1;
49
```

```
96   /* Initialize all configured peripherals */
97   MX_GPIO_Init();
98   MX_I2C1_Init();
99   /* USER CODE BEGIN 2 */
100  lcd_init ();
101  HAL_Delay(500);
102  lcd_send_cmd (0x01);  // clear the display
103  HAL_Delay(500);
104
105  lcd_send_string ("HELLO WORLD !!");
106  HAL_Delay(500);
107
108  lcd_send_cmd (0x01);  // clear the display
109  HAL_Delay(500);
110  /* USER CODE END 2 */
111
112  /* Infinite loop */
113  /* USER CODE BEGIN WHILE */
114  while (1)
115  {
116
```

while loop code

```
112   while (1)
113   {
114
115   /* USER CODE END WHILE */
116
117   /* USER CODE BEGIN 3 */
118     lcd_send_cmd (0x80);  // cursor goes to line:1 col:1
119
120     lcd_send_string ("subscribe"); //display string
121
122     lcd_send_cmd (0xc0); // cursor goes line:2 col:1
123
124     lcd_send_string ("to this channel"); //diaplay string
125
126     HAL_Delay (2000);  // wait for 2 sec
127
128     lcd_send_cmd (0x01);  // clear the display
129
130     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
131
132     HAL_Delay (1000);
133
134
135     //Display Line 1
136     for(int i=0; i<=15; i++)
137     {
138       lcd_send_cmd (0x80+i);
139       lcd_send_string ("1");
140       HAL_Delay(200);
141
142     }
```

```
144     //Display Line 2
145     for(int i=0; i<=15; i++)
146     {
147
148       lcd_send_cmd (0xc0+i);
149       lcd_send_string ("2");
150       HAL_Delay(200);
151
152     }
153
154     lcd_send_cmd (0x01);  // clear the display
155     HAL_Delay(500);
156
157     lcd_send_cmd (0xc0+0xd);
158     lcd_send_string ("XYZ");
159     HAL_Delay(500);
160
161     //Shift Left
162     lcd_send_cmd (0x18);
163     HAL_Delay(500);
164
165     //Shift Left
166     lcd_send_cmd (0x18);
167     HAL_Delay(500);
168
169     //Shift Left
170     lcd_send_cmd (0x18);
171     HAL_Delay(500);
172
173     //Shift Right
174     lcd_send_cmd (0x1c);
175     HAL_Delay(500);
176
177     //Shift Right
178     lcd_send_cmd (0x1c);
179     HAL_Delay(500);
180
181     //Shift Right
182     lcd_send_cmd (0x1c);
183     HAL_Delay(500);
184
185   }
```