Micro-Controller Experiment

Week11

Teacher: 廖裕評 Yu-Ping Liao

TA: 陳大荃 Da-chuan Chen, 陳恩妮 En-ni Chen

Class Rules

1. No drink besides water.
2. Bring a laptop and breadboard if needed.
3. Ask us TAs to sign and borrow development boards. Do not sign or ask others to sign for you without TAs' permission.
4. Arriving 10 minutes after the bell rings will be regarded as absent.
5. If you damage any borrowed equipment, you have to pay for it.

Homework Rules

1. Includes: A. Class content, B. Class exercise, C. Homework (screenshot or video)
2. Editing software: MS PowerPoint
3. File format: PDF
4. Filename: "date_group_studentID_name.pdf", like "0916_第1組_11028XXX_陳OO.pdf"
5. The homework deadline is 23:59 of the day before the next class. If you are late, then your grade will be deducted.

Contact

If you encounter any problems with this class, please get in touch with us with the following

E-mails:

1. Teacher, Prof. Yu-Ping Liao 廖裕評： lyp@cycu.org.tw
2. TA, Da-chuan Chen 陳大荃： dachuan516@gmail.com
3. TA, En-ni Chen 陳恩妮： anna7125867@gmail.com

Or visit 篤信 Lab353 for further questions.

Outline of the Week

1. MPU6500 introduction.
2. MPU6500 project.
3. Homework 11-1.
4. C debugging.

MPU6500 Introduction

MPU6500

- MPU6500 module
- Module num:GY-6500
- Using IC:MPU6500
- Power:3-5V
- Communication protocol: SPI / I2C
- Gyroscope ranges: ±250/500/1000/2000°/s
- Accelerometer ranges: ±2/4/8/16g
- Pin spacing: 2.54mm(standard)
- Module size: 15mm*25mm

front

back

How to research the MPU6500 and use it.

The application example using MPU6500:

- Gesture control, somatosensory game control, balance car, indoor positioning, wearable devices...etc(手勢控制,體感遊戲控制,平衡車,室內定位,可穿戴設備..等等)
- When we want to load the MPU6500 data, we need to know the following 4 keys.

  1. Basic communication concepts of I2C (I2C的基本通訊概念)

  2. MPU6500 specifications, basic register application(MPU6500的規格、基本暫存器應用)

  3.I2C communication method of HT32F52352 (微處理機的I2C通訊方式)

  4.HT32F52352 communicates with MPU6500 module of using I2C(微處理機透過I2C通訊MPU6500模塊)

Basic communication concepts of I2C:

- The condition of start and stop(START和STOP條件)
- Data validity(數據的有效性)
- Addressing format(尋址格式)
- Using 7 bits address(7-bit 地址格式)
- Data Transfer and Acknowledge(傳輸資料/確認資料)
- Host send data or receive(主機傳送/接收資料)
- Complete data transmission timing diagram(完整的數據傳輸時序圖)

☆ Reference week5's material.

Acceleration Characteristics(加速度特性)

Characteristics of a three-axis accelerometer:

- User-programmable precision (±2, 4, 8, 16g) with 16-bit ADC acceleration data output for each axis in a three-axis accelerometer.
- Normal operating current for the accelerometer: 450uA.
- Low-power mode current: 0.98Hz – 8.4uA, 31.25Hz – 19.8uA.
- Sleep mode current: 8uA.
- Enable interrupt wake-up function.

Acceleration Sensitivity(加速度靈敏度)

- Product Specification p.8

3.2 ACCELEROMETER SPECIFICATIONS

Typical Operating Circuit of section 4.2, VDD = 1.8V, VDDIO = 1.8V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	3
	AFS_SEL=1		±4		*g*	3
	AFS_SEL=2		±8		*g*	3
	AFS_SEL=3		±16		*g*	3
ADC Word Length	Output in two's complement format		16		bits	3
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	3
	AFS_SEL=1		8,192		LSB/*g*	3
	AFS_SEL=2		4,096		LSB/*g*	3
	AFS_SEL=3		2,048		LSB/*g*	3
Initial Tolerance	Component-level		±3		%	2
Sensitivity Change vs. Temperature	-40°C to +85°C AFS_SEL=0 Component-level		±0.026		%/°C	1
Nonlinearity	Best Fit Straight Line		±0.5		%	1
Cross-Axis Sensitivity			±2		%	1

Precision

Data length

Sensitivity

Serial Interface Electrical Specifications (連接埠頻率表)

- Product Specification p.12

3.3.3 Other Electrical Specifications

Typical Operating Circuit of section 4.2, VDD = 1.8V, VDDIO = 1.8V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
SERIAL INTERFACE						
SPI Operating Frequency, All Registers Read/Write	Low Speed Characterization		100 ±10%		kHz	1
	High Speed Characterization		1 ±10%		MHz	1
SPI Operating Frequency, Sensor and Interrupt Registers Read Only			20 ±10%		MHz	1
I^2C Operating Frequency	All registers, Fast-mode			400	kHz	1
	All registers, Standard-mode			100	kHz	1

Table 5. Other Electrical Specifications

Absolute Maximum Ratings(最大額定值)

Exceeding these maximum ratings may result in permanent damage to the chip. Under such extreme conditions, it is highly likely to cause destruction to the chip itself, not to mention obtaining accurate data.

- Product Specification p.16

Parameter	Rating
Supply Voltage, VDD	-0.5V to +4V
Supply Voltage, VDDIO	-0.5V to +4V
REGOUT	-0.5V to 2V
Input Voltage Level (AUX_DA, AD0, FSYNC, INT, SCL, SDA)	-0.5V to VDD + 0.5V
Acceleration (Any Axis, unpowered)	10,000g for 0.2ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Electrostatic Discharge (ESD) Protection	2kV (HBM); 250V (MM)
Latch-up	JEDEC Class II (2),125°C, ±100mA

Table 9. Absolute Maximum Ratings

Pin Out Diagram and Signal Description (引腳功能說明)

➢ Product Specification p.17

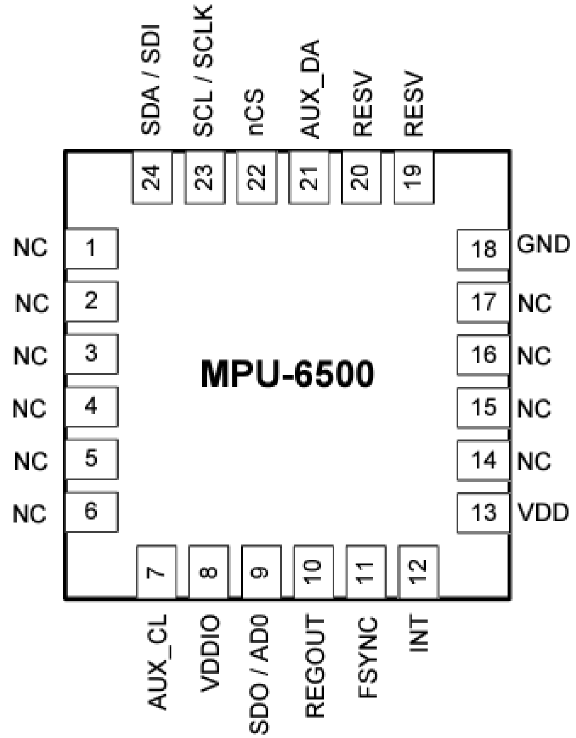


Figure 3. Pin out Diagram for MPU-6500 3.0x3.0x0.9mm QFN

Pin Number	Pin Name	Pin Description
7	AUX_CL	I^2C Master serial clock, for connecting to external sensors
8	VDDIO	Digital I/O supply voltage
9	AD0 / SDO	I^2C Slave Address LSB (AD0); SPI serial data output (SDO)
10	REGOUT	Regulator filter capacitor connection
11	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	INT	Interrupt digital output (totem pole or open-drain) ***Note: The Interrupt line should be connected to a pin on the Application Processor (AP) that can bring the AP out of suspend mode.***
13	VDD	Power supply voltage and Digital I/O supply voltage
18	GND	Power supply ground
19	RESV	Reserved. Do not connect.
20	RESV	Reserved. Connect to GND.
21	AUX_DA	I^2C master serial data, for connecting to external sensors
22	nCS	Chip select (SPI mode only)
23	SCL / SCLK	I^2C serial clock (SCL); SPI serial clock (SCLK)
24	SDA / SDI	I^2C serial data (SDA); SPI serial data input (SDI)
1 – 6, 14 - 17	NC	No Connect pins. Do not connect.

Table 10. Signal Descriptions

Typical Operating Circuit(典型電路圖)

- Product Specification p.18

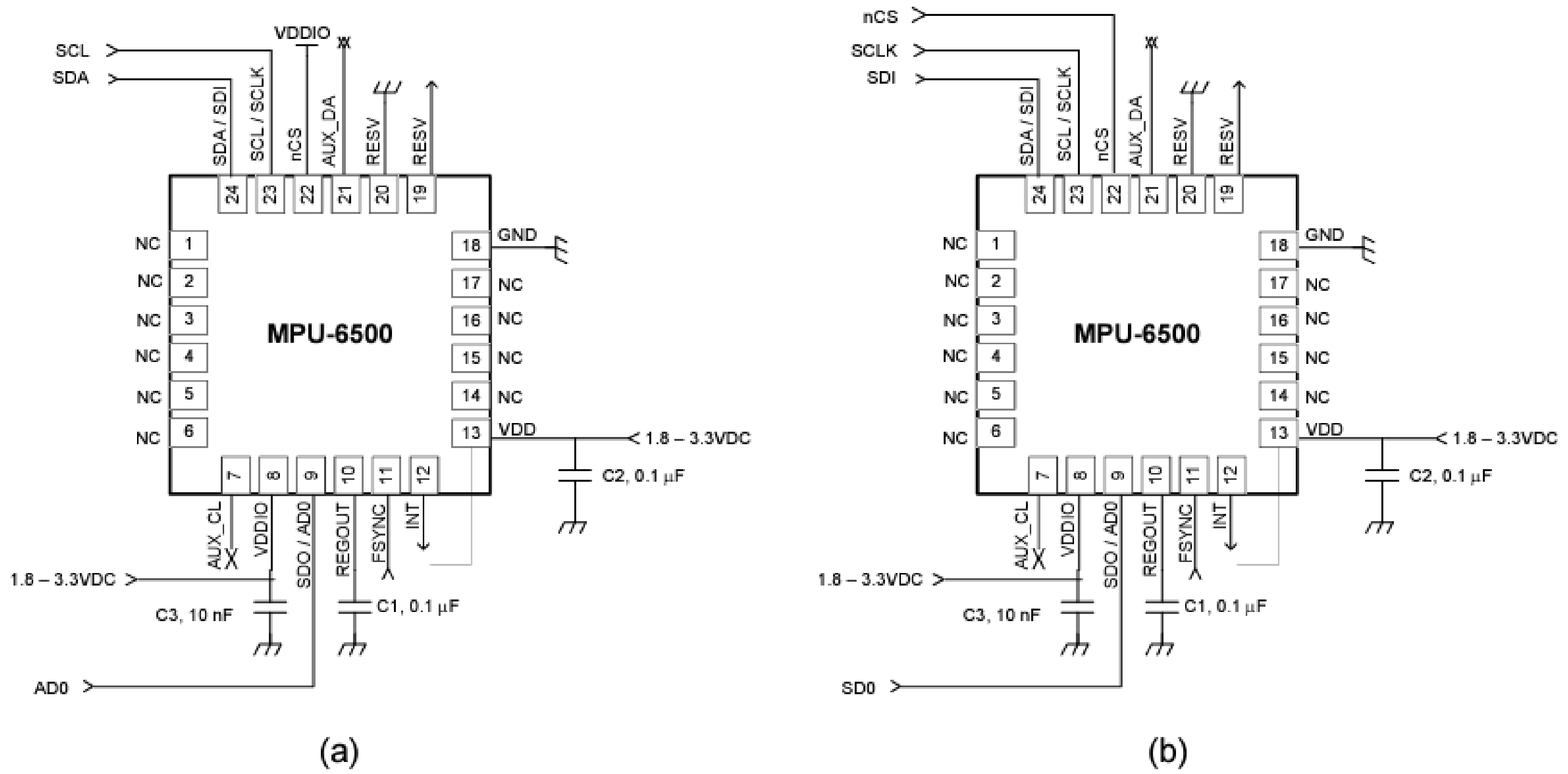


Figure 4. MPU-6500 QFN Application Schematic. (a) I^{2}C operation, (b) SPI operation.

Block Diagram(硬體架構圖)

- Product Specification p.19

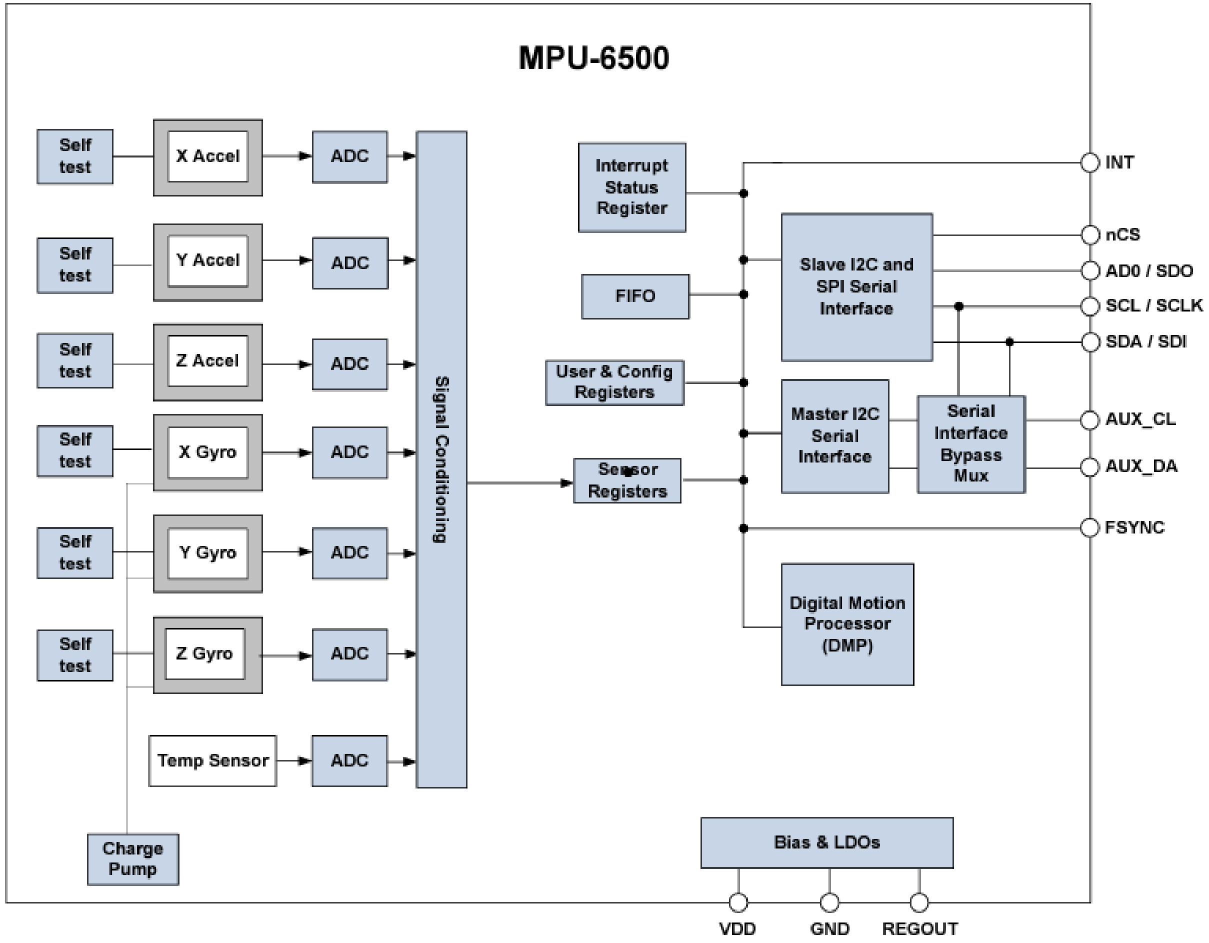


Figure 5. MPU-6500 Block Diagram

16-bit ADC three-axis acceleration signal output and conditioning.

- The three-axis acceleration of the MPU6500 is measured separately for each axis.
- Measure the bias of each axis based on the capacitance on each axis.
- When placed on a flat surface, it measures a gravitational acceleration of 0g on the X and Y axes and 1g on the Z axis, effectively reducing measurement bias caused by various factors in its structure.
- The calibration of the accelerometer is set based on the factory standards, and the power supply voltage may vary from what you are using.
- Each sensor has a dedicated ADC to provide digital output.
- The output precision is programmable to 2g, 4g, 8g, 16g.

MPU6500_I2C Communication

- I2C is a dual-line communication method, consisting of SDA (data) and SCL (clock) lines. Typically, these two interfaces are bidirectional open-drain interfaces. When connecting devices, they can function as either a master or a slave. In slave mode, communication is achieved by matching the address.
- The MPU6500 is typically configured as a slave when connected to a control chip. SDA and SCL usually require pull-up resistors to VDD, and the maximum communication speed can reach 400 KHz.
- When configured as a slave, the MPU6500 has a 7-bit address of 110100X in binary. The LSB of this address is determined by the level of the AD0 pin, allowing two MPU6500 devices to be connected simultaneously in a system. (X is 0 when AD0 is at a low level and 1 when AD0 is at a high level).

➢ Product Specification p.10

I²C ADDRESS	AD0 = 0 AD0 = 1		1101000 1101001			

Method of read / write the data from register of MPU6500 (讀取/寫入MPU6500的暫存器的方法)

Signal name	Function
s	Start signal: When SCL is at a high level, SDA transitions to a low level.
AD	Slave I2C Address
W/R	Write/Read
ACK	Response: When SCL is at a high level, SDA remains at a low level.
NACK	No Response: SDA remains high during the 9th clock cycle.
RA	The addresses of internal registers in MPU6500.
DATA	Send or Receive
P	Stop Signal: When SCL is at a high level, SDA generates a rising edge.

Method of read / write the data from register of MPU6500 (讀取/寫入MPU6500的暫存器的方法)

- Write to the register of MPU5600:

The host sends the start signal followed by the 7-bit address of the slave and an additional 1-bit for write. When at the 9th clock signal, the IC generates an ACK. At this point, the host outputs the register address, and the slave generates another ACK response. The transmission can be stopped at any time by sending a stop signal. After the ACK response, data can continue to be input unless a stop bit is generated. The IC's internally embedded incrementing register can automatically write data to the corresponding register. The transmission order for single-byte and double-byte is listed below.

單字節傳輸時序圖

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

多字節傳輸時序圖

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Method of read / write the data from register of MPU6500 (讀取/寫入MPU6500的暫存器的方法)

- Read from the register of MPU5600:
- The host sends a start signal and the 7-bit address of the slave device plus a read bit (1). At this point, the register address becomes readable. The IC responds with an ACK signal. Then, the host sends a start signal and the address again. The IC responds with an ACK signal and the data. Communication stops when the host sends a NACK or a stop bit. The NACK signal is the 9th clock pulse, and SDA remains high. The timing diagrams for single-byte and double-byte read sequences are shown in the following figure.

單字節讀取時序圖

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

多字節讀取時序圖

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

The register of WHO_AM_I

- This register is used to inform the user about the currently accessed device.

Address

75	117	WHO_AM_I	R	WHOAMI[7:0]

4.38 Register 117 – Who Am I

Name: WHOAMI

Serial IF: READ

Reset value: 0x70

BIT	NAME	FUNCTION
[7:0]	WHOAMI	Register to indicate to user which device is being accessed.

This register is used to verify the identity of the device. The contents of *WHO_AM_I* is an 8-bit device ID. The default value of the register is 0x70 for MPU-6500. This is different from the I2C address of the device as seen on the slave I2C controller by the applications processor. The I2C address of the MPU-6500 is 0x68 or 0x69 depending upon the value driven on AD0 pin.

➢ Register Map p.44

Acceleration Configuration Register

- Registers used for configuring the accelerometer.

Address(Hex)

1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	ACCEL_FS_SEL[1:0]	-

4.7 Register 28 – Accelerometer Configuration

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	XA_ST	X Accel self-test
[6]	YA_ST	Y Accel self-test
[5]	ZA_ST	Z Accel self-test
[4:3]	ACCEL_FS_SEL[1:0]	Accel Full Scale Select: ±2g (00), ±4g (01), ±8g (10), ±16g (11)
[2:0]	-	Reserved

➢ Product Specification p.14

Accelerometer Measurements Register

This register is used to store the sampled acceleration data from the sensor.

- Address: 0x3B ~ 0x40.
- Each register can only store 8 bits.
- Since acceleration has three axes, with each axis having 16 bits, combining data from OUT_H and OUT_L gives the complete 16-bit data.

Address(Hex)

3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]

➢ Product Specification p.7

MPU6500 Project

Download project

1. Download from i-learning.

2. Decompression
3. path: example/I2C/I2C_MPU6500/MDK_ARMv537

The master sends the device address to the slave.(I2CTAR)

```c
/* 傳送START信號、目標設備位址，寫入模式
Send I2C START & I2C slave address for write*/
I2C_TargetAddressConfig(HT_I2C0, I2C_SLAVE_ADDRESS, I2C_MASTER_WRITE);
```

F12

```c
void I2C_TargetAddressConfig(HT_I2C_TypeDef* I2Cx, I2C_AddressTypeDef I2C_Address, u32 I2C_Direction)
{
  /* Check the parameters
  Assert_Param(IS_I2C(I2Cx));
  Assert_Param(IS_I2C_ADDRESS(I2C_Address));
  Assert_Param(IS_I2C_DIRECTION(I2C_Direction));

  /* Make sure the prior stop command has been finished
  while (I2Cx->CR & 0x2);

  if (I2C_Direction != I2C_MASTER_WRITE)
  {
    I2Cx->TAR = I2C_Address | I2C_MASTER_READ;
  }
   else
  {
    I2Cx->TAR = I2C_Address | I2C_MASTER_WRITE;
  }
}
```

- User Manual(sc) p.455

I²C 目标寄存器 – I2CTAR

该寄存器定义了要与之通信的目标设备地址。

偏移量: 0x01C
复位值: 0x0000_0000

	31	30	29	28	27	26	25	24
	保留位							
类型 / 复位								

	23	22	21	20	19	18	17	16
	保留位							
类型 / 复位								

	15	14	13	12	11	10	9	8
	保留位					RWD	TAR	
类型 / 复位						RW 0	RW 0	RW 0

	7	6	5	4	3	2	1	0
	TAR							
类型 / 复位	RW 0	RW 0	RW 0	RW 0	RW 0	RW 0	RW 0	RW 0

位	字段	描述
[10]	RWD	读或写方向位 0：写入目标从机地址 1：读取目标从机地址 如果在 10-bit 主机接收器模式此位被置 1，那么 I²C 接口将在第一个头帧中发起一个值为 11110XX0b 的字节，并由硬件继续在第二个头帧中提供一个值为 11110XX1b 的字节。
[9:0]	TAR	目标从机地址 一旦数据写入该寄存器，I²C 接口将会自动发送一个 START 信号和一个目标从机地址。当系统想要发送一个重复的 START 信号给 I²C 总线时，建议在一个字节传输完成之后再设置 I2CTAR 寄存器。不允许在地址帧设置 TAR。I2CTAR[9:7] 在 7-bit 寻址模式中不可用。

Send/Receive data(I2CDR)

main.c | ht32f5xxxx_i2c.c

```c
  * @retval None
  ***********************************************************************
void I2C_SendData(HT_I2C_TypeDef* I2Cx, u8 I2C_Data)
{
  /* Check the parameters
  Assert_Param(IS_I2C(I2Cx));

  I2Cx->DR = I2C_Data;
}

/***********************************************************************
  * @brief Return the received data by the I2Cx peripheral.
  * @param I2Cx: where I2Cx is the selected I2C from the I2C
  * @retval The value of the received data.
  ***********************************************************************
u8 I2C_ReceiveData(HT_I2C_TypeDef* I2Cx)
{
  /* Check the parameters
  Assert_Param(IS_I2C(I2Cx));

  return (u8)I2Cx->DR;
}
```

- User Manual(sc) p.454

I^2C 数据寄存器 – I2CDR

该寄存器定义了由 I^2C 模块发送和接收的数据。

偏移量： 0x018

复位值： 0x0000_0000

	31	30	29	28	27	26	25	24
	保留位							
类型 / 复位								

	23	22	21	20	19	18	17	16
	保留位							
类型 / 复位								

	15	14	13	12	11	10	9	8
	保留位							
类型 / 复位								

	7	6	5	4	3	2	1	0
	DATA							
类型 / 复位	RW 0	RW 0	RW 0	RW 0	RW 0	RW 0	RW 0	RW 0

位	字段	描述
[7:0]	DATA	I^2C 数据寄存器 在发送器模式中，发送到从机的一个数据字节可以分配给这些位。如果软件分配新的数据给 I2CDR 寄存器时，TXDE 标志位将被清零。 在接收器模式中，一个数据字节从 MSB 到 LSB 逐位的通过 I^2C 接口被接收且被储存在数据移位寄存器中。一旦发送了确认位，当 RXDNE 标志位为 0 时，数据移位寄存器的值将被发送到 I2CDR 寄存器。

Send STOP signal

main.c | ht32f5xxxx_i2c.c

```c
 ****************************************************
void I2C_GenerateSTOP(HT_I2C_TypeDef* I2Cx)
{
  /* Check the parameters
  Assert_Param(IS_I2C(I2Cx));

  I2Cx->CR |= 0x2;
}
```

- User Manual(sc) p.446

[1]	STOP	STOP 条件控制位 0：无动作 1：在主机模式下发送 STOP 条件 此位被软件置 1 来产生一个 STOP 条件，通过硬件自动清零。STOP 位只用于主机。

Read the value of the specified register in the I2C.

```c
u32 I2C_ReadRegister(HT_I2C_TypeDef* I2Cx, u8 I2C_Register)
{
  vu32 tmp = 0;

  /* Check the parameters */
  Assert_Param(IS_I2C(I2Cx));
  Assert_Param(IS_I2C_REGISTER(I2C_Register));

  tmp = (u32)I2Cx;
  tmp += I2C_Register;
  return (*(u32 *)tmp);
}
```

表 47. I^2C 寄存器列表

寄存器	偏移量	描述	复位值
I2CCR	0x000	I^2C 控制寄存器	0x0000_2000
I2CIER	0x004	I^2C 中断使能寄存器	0x0000_0000
I2CADDR	0x008	I^2C 地址寄存器	0x0000_0000
I2CSR	0x00C	I^2C 状态寄存器	0x0000_0000
I2CSHPGR	0x010	I^2C SCL 高电平周期发生寄存器	0x0000_0000
I2CSLPGR	0x014	I^2C SCL 低电平周期发生寄存器	0x0000_0000
I2CDR	0x018	I^2C 数据寄存器	0x0000_0000
I2CTAR	0x01C	I^2C 目标寄存器	0x0000_0000
I2CADDMR	0x020	I^2C 地址屏蔽寄存器	0x0000_0000
I2CADDSR	0x024	I^2C 地址捕获寄存器	0x0000_0000
I2CTOUT	0x028	I^2C 超时寄存器	0x0000_0000

➢ User Manual(sc) p.444

```c
typedef struct
{
                                 /* I2C2: 0x40008000                                      */
                                 /* I2C0: 0x40048000                                      */
                                 /* I2C1: 0x40049000                                      */
  __IO uint32_t CR;              /*< 0x000        Control Register                        */
  __IO uint32_t IER;             /*< 0x004        Interrupt Enable Register               */
  __IO uint32_t ADDR;            /*< 0x008        Address Register                        */
  __IO uint32_t SR;              /*< 0x00C        Status Register                         */
  __IO uint32_t SHPGR;           /*< 0x010        SCL High Period Generation Register     */
  __IO uint32_t SLPGR;           /*< 0x014        SCL Low Period Generation Register      */
  __IO uint32_t DR;              /*< 0x018        Data Register                           */
  __IO uint32_t TAR;             /*< 0x01C        Target Register                         */
  __IO uint32_t ADDMR;           /*< 0x020        Address Mask Register                   */
  __IO uint32_t ADDSR;           /*< 0x024        Address Snoop Register                  */
  __IO uint32_t TOUT;            /*< 0x028        Timeout Register                        */
} HT_I2C_TypeDef;
```

Enable or disable external configuration of I2C.

```c
void I2C_Cmd(HT_I2C_TypeDef* I2Cx, ControlStatus NewState)
{
  /* Check the parameters */
  Assert_Param(IS_I2C(I2Cx));
  Assert_Param(IS_CONTROL_STATUS(NewState));

  if (NewState != DISABLE)
  {
    I2Cx->CR |= CR_ENI2C_SET;
  }
  else
  {
    I2Cx->CR &= CR_ENI2C_RESET;
  }
}
```

- User Manual(sc) p.444

[3]	I2CEN	I^2C 接口使能位 0: I^2C 接口除能 1: I^2C 接口使能

Enable or disable sending ACK on I2C.

```c
void I2C_AckCmd(HT_I2C_TypeDef* I2Cx, ControlStatus NewState)
{
  /* Check the parameters
  Assert_Param(IS_I2C(I2Cx));
  Assert_Param(IS_CONTROL_STATUS(NewState));

  if (NewState != DISABLE)
  {
    I2Cx->CR |= CR_ACK_SET;
  }
  else
  {
    I2Cx->CR &= CR_ACK_RESET;
  }
}
```

- User Manual(sc) p.444

[0]	AA	确认位 0：在一个字节接收后发送一个未确认信号 (NACK) 1：在一个字节接收后发送一个确认信号 (ACK) 当 I2CEN 位清零，AA 位由硬件自动清零。

Check the I2C flag status.

```c
ErrStatus I2C_CheckStatus(HT_I2C_TypeDef* I2Cx, u32 I2C_Status)
{
  /* Check the parameters
  Assert_Param(IS_I2C(I2Cx));
  Assert_Param(IS_I2C_STATUS(I2C_Status));

  if (I2Cx->SR == I2C_Status)
  {
    return (SUCCESS);
  }
  else
  {
    return (ERROR);
  }
}
```

- User Manual(sc) p.449

I^2C 状态寄存器 – I2CSR

该寄存器包含了 I^2C 的工作状态。

偏移量: 0x00C

复位值: 0x0000_0000

	31	30	29	28	27	26	25	24
	保留位							
类型 / 复位								
	23	22	21	20	19	18	17	16
	保留位		TXNRX	MASTER	BUSBUSY	RXBF	TXDE	RXDNE
类型 / 复位			RO 0	RO 0	RO 0	RO 0	RO 0	RO 0
	15	14	13	12	11	10	9	8
	保留位				TOUTF	BUSERR	RXNACK	ARBLOS
类型 / 复位					WC 0	WC 0	WC 0	WC 0
	7	6	5	4	3	2	1	0
	保留位				GCS	ADRS	STO	STA
类型 / 复位					RC 0	RC 0	RC 0	RC 0

Interrupts used in this experiment:

[17]	TXDEIE	发送器模式下数据寄存器空中断使能位 0：中断除能 1：中断使能 当 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。
[16]	RXDNEIE	接收器模式下数据寄存器非空中断使能位 0：中断除能 1：中断使能 当 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。
[11]	TOUTIE	超时中断使能位 0：中断除能 1：中断使能 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。
[10]	BUSERRIE	总线错误中断使能位 0：中断除能 1：中断使能 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。
[9]	RXNACKIE	接收未确认信号中断使能位 0：中断除能 1：中断使能 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。
[8]	ARBLOSIE	I^2C 多主机模式下仲裁丢失中断使能位 0：中断除能 1：中断使能 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。
[2]	ADRSIE	从机地址匹配中断使能位 0：中断除能 1：中断使能 当 I2CCR 寄存器中的 I2CEN 位清零时，此位将由硬件清零。

I2C Configure:

There are 6 parameters to configure for I2C:

1. I2C_GeneralCall（CCR Register）
2. I2C_AddressingMode（CCR Register ）
3. I2C_Acknowledge（CCR Register ）
4. I2C_OwnAddress（ADDR Register ）
5. I2C_Speed（SHPGR Register ）
6. I2C_SpeedOffset（SHPGR Register ）

```c
/* Private constants ---------------------
#define I2C_MASTER_ADDRESS      0x0A
#define I2C_SLAVE_ADDRESS       0x68
#define ClockSpeed              400000

```

```c
/* 配置I2C暫存器 */

I2C_InitStructure.I2C_GeneralCall = DISABLE;
I2C_InitStructure.I2C_AddressingMode = I2C_ADDRESSING_7BIT;
I2C_InitStructure.I2C_Acknowledge = DISABLE;
I2C_InitStructure.I2C_OwnAddress = I2C_MASTER_ADDRESS;
I2C_InitStructure.I2C_Speed = ClockSpeed;
I2C_InitStructure.I2C_SpeedOffset = 0;
I2C_Init(HT_I2C0, &I2C_InitStructure);
```

Configure Transmission Speed:

```c
/* 配置I2C暂存器 */

I2C_InitStructure.I2C_GeneralCall = DISABLE;
I2C_InitStructure.I2C_AddressingMode = I2C_ADDRESSING_7BIT;
I2C_InitStructure.I2C_Acknowledge = DISABLE;
I2C_InitStructure.I2C_OwnAddress = I2C_MASTER_ADDRESS;
I2C_InitStructure.I2C_Speed = ClockSpeed;
I2C_InitStructure.I2C_SpeedOffset = 0;
I2C_Init(HT_I2C0, &I2C_InitStructure);
```

```c
/* Private constants -------------------
#define I2C_MASTER_ADDRESS      0x0A
#define I2C_SLAVE_ADDRESS       0x68
#define ClockSpeed              400000
```

- User Manual(sc) p.453

表 48. I^2C 时钟设置范例

I^2C 时钟	$T_{SCL} = T_{PCLK} \times [(SHPG + d) + (SLPG + d)]$ (d = 6) PCLK 时钟下 SHPG + SLPG 的值			
	8MHz	24MHz	48MHz	72MHz
100 kHz (标准模式)	68	228	468	708
400 kHz (快速模式)	8	48	108	168
1 MHz (快速 + 模式)	x	12	36	60

Configure Addressing Mode and Acknowledge Bit.

```c
/* 配置I2C暫存器 */

I2C_InitStructure.I2C_GeneralCall = DISABLE;
I2C_InitStructure.I2C_AddressingMode = I2C_ADDRESSING_7BIT;
I2C_InitStructure.I2C_Acknowledge = DISABLE;
I2C_InitStructure.I2C_OwnAddress = I2C_MASTER_ADDRESS;
I2C_InitStructure.I2C_Speed = ClockSpeed;
I2C_InitStructure.I2C_SpeedOffset = 0;
I2C_Init(HT_I2C0, &I2C_InitStructure);
```

- User Manual(sc) p.446

[7]	ADRM	寻址模式 0： 7-bit 寻址模式 1： 10-bit 寻址模式 当 I^2C 主机 / 从机模块工作在 7-bit 寻址模式时，它只能发出和响应一个 7-bit 地址，反之亦然。当 I2CEN 除能，ADRM 位会由硬件自动清零。
[0]	AA	确认位 0： 在一个字节接收后发送一个未确认信号 (NACK) 1： 在一个字节接收后发送一个确认信号 (ACK) 当 I2CEN 位清零，AA 位由硬件自动清零。

Set Device Address.

```c
/* 配置I2C暂存器 */

I2C_InitStructure.I2C_GeneralCall = DISABLE;
I2C_InitStructure.I2C_AddressingMode = I2C_ADDRESSING_7BIT;
I2C_InitStructure.I2C_Acknowledge = DISABLE;
I2C_InitStructure.I2C_OwnAddress = I2C_MASTER_ADDRESS;
I2C_InitStructure.I2C_Speed = ClockSpeed;
I2C_InitStructure.I2C_SpeedOffset = 0;
I2C_Init(HT_I2C0, &I2C_InitStructure);
```

```c
/* Private constants ------------------
#define I2C_MASTER_ADDRESS        0x0A
#define I2C_SLAVE_ADDRESS         0x68
#define ClockSpeed                400000

```

- User Manual(sc) p.448

位	字段	描述
[9:0]	ADDR	设备地址 该寄存器定义了 I^2C 设备地址。当 I^2C 设备用在 7-bit 寻址模式时，只有 ADDR[6:0] 位与 I^2C 主机发送的地址相比较。

Connect pull-up resistors to I2C

As mentioned in the previous I2C class, when using I2C communication, SDA and SCL need to be connected to pull-up resistors; otherwise, successful communication may not occur.

```
void I2CMaster_Configuration(void)
{
  I2C_InitTypeDef  I2C_InitStructure;
  CKCU_PeripClockConfig_TypeDef CKCUClock = {{0}};

  /* 配置系統時鐘 */
  CKCUClock.Bit.I2C0 = 1;
  CKCUClock.Bit.AFIO = 1;
  CKCUClock.Bit.PA   = 1;
  CKCU_PeripClockConfig(CKCUClock, ENABLE);
  /* 配置AFIO */
  AFIO_GPxConfig(GPIO_PA, AFIO_PIN_0, AFIO_MODE_7);
  AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_MODE_7);
  /* 配置上拉電阻 */
  GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_0, GPIO_PR_UP);
  GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_1, GPIO_PR_UP);
```

Read data from MPU6500 registers via I2C.

- The I2C transmission process follows the timing diagram of MPU6500.
- The argument is the address of the register.
- The return value is the 8-bit data of that register.

```c
u8 MPU6500_I2C_Read_OneByte(u8 reg_addr)
{
    u8 receive_data = 0;

    /* 等待閒置狀態 */
    while (I2C_ReadRegister(HT_I2C0, I2C_REGISTER_SR)&0x80000);
    /* 啟用I2C發送ACK信號 */
    I2C_AckCmd(HT_I2C0, ENABLE);
    /* 發送START信號、目標設備位址，寫入模式 */
    I2C_TargetAddressConfig(HT_I2C0, I2C_SLAVE_ADDRESS, I2C_MASTER_WRITE);
    /* 檢查START信號是否傳輸完成 */
    while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_SEND_START));
    /* 檢查目標設備位址、讀寫模式位是否傳輸完成 */
    while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_TRANSMITTER_MODE));
    /* 檢查發送模式下數據暫存器是否為空 */
    while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_TX_EMPTY));
    /* 發送暫存器位址 */
    I2C_SendData(HT_I2C0, reg_addr);
    /* 發送START信號、目標設備位址，接收模式 */
    I2C_TargetAddressConfig(HT_I2C0, I2C_SLAVE_ADDRESS, I2C_MASTER_READ);
    /* 檢查START信號是否傳輸完成 */
    while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_SEND_START));
    /* 檢查目標設備位址、讀寫模式位是否傳輸完成 */
    while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_RECEIVER_MODE));
    /* 關閉I2C發送ACK信號 */
    I2C_AckCmd(HT_I2C0, DISABLE);
    /* 檢查接收模式下數據暫存器是否為空 */
    while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_RX_NOT_EMPTY));
    /* 接收數據 */
    receive_data = I2C_ReceiveData(HT_I2C0);
    /* 發送I2C的停止信號 */
    I2C_GenerateSTOP(HT_I2C0);
    return receive_data;
}
```

單字節讀取時序圖

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Write data to MPU6500 registers via I2C.

- I2C transmission process follows the timing diagram of MPU6500.
- Argument 1 is the address of the register.
- Argument 2 is the value to be transmitted.

```
void MPU6500_I2C_Write_OneByte(u8 reg_addr, u8 register_value)
{
  /* 等待閒置狀態 */
  while (I2C_ReadRegister(HT_I2C0, I2C_REGISTER_SR)&0x80000);
  /* 啟用I2C發送ACK信號 */
  I2C_AckCmd(HT_I2C0, ENABLE);
  /* 傳送START信號、目標設備位址，寫入模式
  Send I2C START & I2C slave address for write*/
  I2C_TargetAddressConfig(HT_I2C0, I2C_SLAVE_ADDRESS, I2C_MASTER_WRITE);
  /* 檢查START信號是否傳輸完成
  Check on Master Transmitter STA condition and clear it*/
  while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_SEND_START));
  /* 檢查目標設備位址、讀寫模式位是否傳輸完成
  Check on Master Transmitter ADRS condition and clear it*/
  while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_TRANSMITTER_MODE));
  /* 檢查發送模式下數據暫存器是否為空
  Check on Master Transmitter TXDE condition*/
  while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_TX_EMPTY));
  /* 發送暫存器位址 */
  I2C_SendData(HT_I2C0, reg_addr);
  /* 檢查發送模式下數據暫存器是否為空 */
  while (!I2C_CheckStatus(HT_I2C0, I2C_MASTER_TX_EMPTY));
  /* 發送數據 */
  I2C_SendData(HT_I2C0, register_value);
  /* 發送I2C的停止信號 */
  I2C_GenerateSTOP(HT_I2C0);
}
```

單字節傳輸時序圖

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Combine the data from two 8-bit registers.

```c
u16 Get_HL_Value(u8 high_reg, u8 low_reg)
{
    u8 h_Value, l_Value;

    h_Value = MPU6500_I2C_Read_OneByte(high_reg);
    l_Value = MPU6500_I2C_Read_OneByte(low_reg);

    /* h_value left 8 bits or l_value = (h_value+l_value)*/
    return (((u16)h_Value << 8) | l_Value);
}
```

- Argument 1 is the address of the H register.
- Argument 2 is the address of the L register.
- The return value is the combined value of these two registers (16-bit).

Address(Hex)

3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]

➢ Product Specification p.7

Display the current acceleration range status.

```c
void Show_Acc_Scale_Select(void)
{
  u8 scale;

  scale = MPU6500_I2C_Read_OneByte(Acc_Config_Reg) & 0x18;

  switch(scale)
  {
    case mpu6500_range_2G:
      printf("Acc_Scale_Status: +-2G\r\n");
      break;
    case mpu6500_range_4G:
      printf("Acc_Scale_Status: +-4G\r\n");
      break;
    case mpu6500_range_8G:
      printf("Acc_Scale_Status: +-8G\r\n");
      break;
    case mpu6500_range_16G:
      printf("Acc_Scale_Status: +-16G\r\n");
      break;
    default:
      printf("Acc_Scale_Status: error\r\n");
      break;
  }
}
```

```c
typedef enum
{
    mpu6500_range_16G  = 0x18,
    mpu6500_range_8G   = 0x10,
    mpu6500_range_4G   = 0x08,
    mpu6500_range_2G   = 0x00,
} MPU6500_Acc_Scale;
```

- Product Specification p.14

4.7 Register 28 – Accelerometer Configuration

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	XA_ST	X Accel self-test
[6]	YA_ST	Y Accel self-test
[5]	ZA_ST	Z Accel self-test
[4:3]	ACCEL_FS_SEL[1:0]	Accel Full Scale Select: ±2g (00), ±4g (01), ±8g (10), ±16g (11)
[2:0]	-	Reserved

Set acceleration scale

```c
void Set_Acc_Scale_Select(u8 scale)
{
  u8 scale_data;

  scale_data = (MPU6500_I2C_Read_OneByte(Acc_Config_Reg) & 0xE7) | scale;
  MPU6500_I2C_Write_OneByte(Acc_Config_Reg, scale_data);
}
```

假如加速度配置暫存器內的值是0x18（16G）

想設成0x10（8G）

00011000（0x18）先把[4:3]位元歸零

& 11100111（0xE7）其餘位元維持原值

00000000（0x00）把處理後的值

| 00010000（0x10）跟設定的值或閘

00010000（0x10）最後將值設成8G

Convert the 16-bit raw value read from the acceleration data register to the range value.

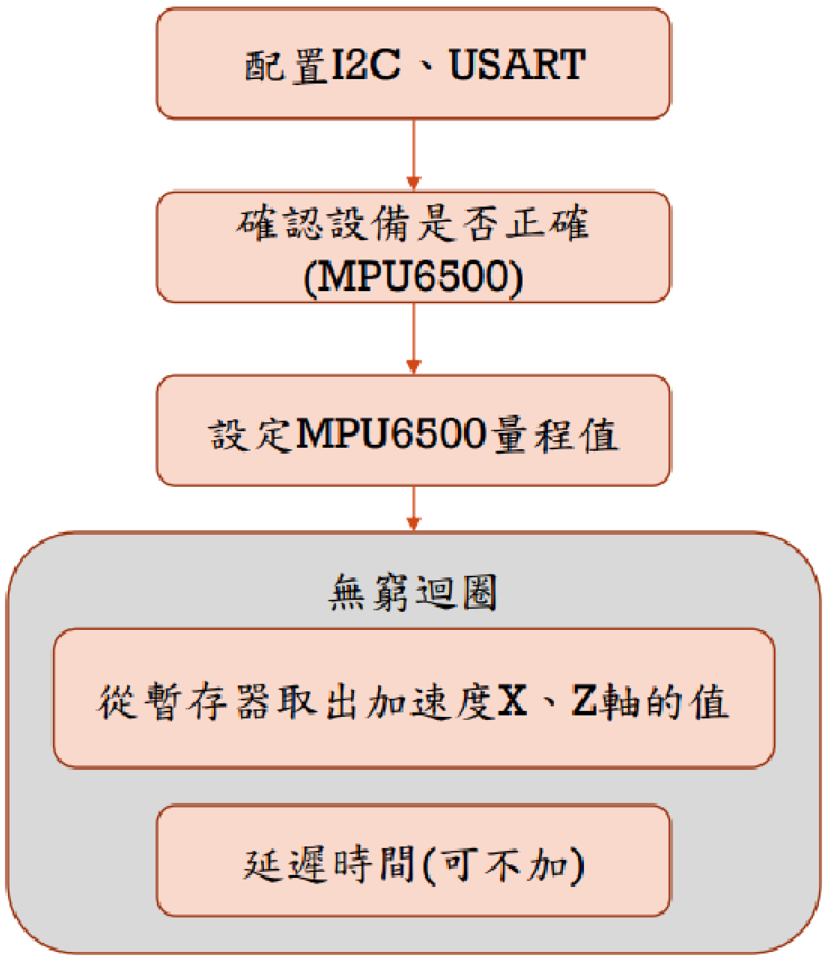
```c
double Acc_Scale(u16 raw_value)
{
  u8 scale;
  double per_digit;

  scale = MPU6500_I2C_Read_OneByte(Acc_Config_Reg) & 0x18;
  switch(scale)
  {
    case mpu6500_range_2G:
      per_digit = 2.0/0x8000;
      break;
    case mpu6500_range_4G:
      per_digit = 4.0/0x8000;
      break;
    case mpu6500_range_8G:
      per_digit = 8.0/0x8000;
      break;
    case mpu6500_range_16G:
      per_digit = 16.0/0x8000;
      break;
    default:
      break;
  }
  return ((signed short)raw_value * per_digit);
}
```

- Parameter is the raw 16-bit value read from the acceleration data register.
- The return value is the range value.

main

```c
int main(void)
{
  /* 配置I2C、USART */
  RETARGET_Configuration();
  I2CMaster_Configuration();

  /* 確認設備正確，不正確就不往下執行 */
  Who_Am_I_Value = MPU6500_I2C_Read_OneByte(0x75);
  printf("Who am I Value : %02x\r\n", Who_Am_I_Value);
  while(!(Who_Am_I_Value ==  0x70));
  /* 設定加速度量程為2G */
  Set_Acc_Scale_Select(mpu6500_range_2G);
  Show_Acc_Scale_Select();
  while (1){
  /* 得到加速度的x軸、z軸放入陣列內 */
    Acc_Raw_Value[0] = Get_HL_Value(AccXH_Reg , AccXL_Reg);
    Acc_Raw_Value[2] = Get_HL_Value(AccZH_Reg , AccZL_Reg);

    printf("Acc%c --> Raw_Value_DEC: %05d , Raw_Value_HEX: %04x , Acc_Scale: %f\r\n",
               Axis[0], Acc_Raw_Value[0], Acc_Raw_Value[0], Acc_Scale(Acc_Raw_Value[0]));
    printf("Acc%c --> Raw_Value_DEC: %05d , Raw_Value_HEX: %04x , Acc_Scale: %f\r\n",
               Axis[2], Acc_Raw_Value[2], Acc_Raw_Value[2], Acc_Scale(Acc_Raw_Value[2]));

    printf("\r\n");

    for(i=0 ; i<0xFFFFF ; i++);
  }
}
```

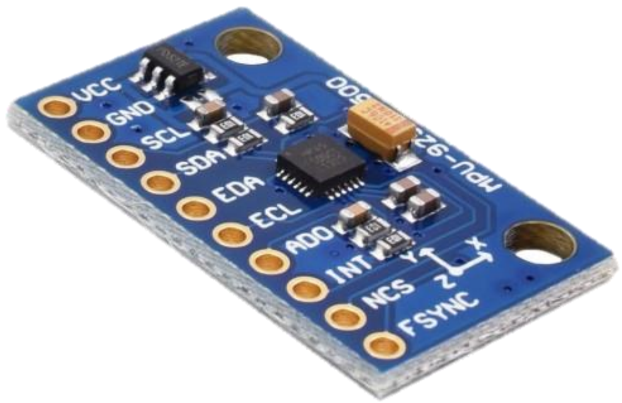
- flow

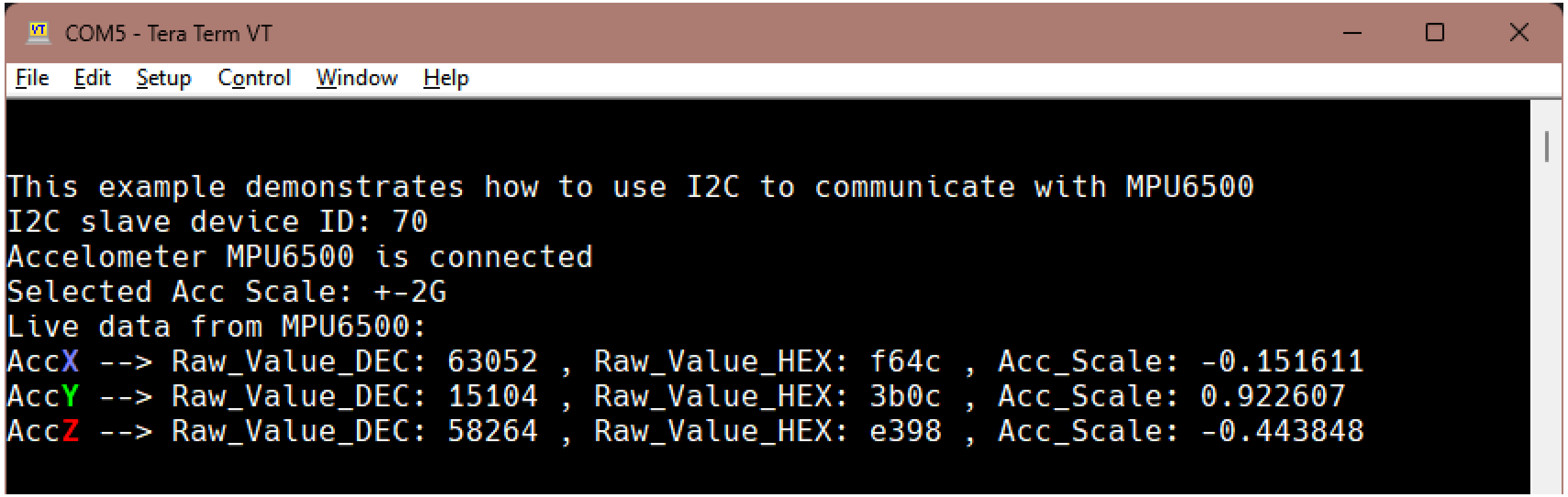


Circuit diagram

HT32F52352 I2C0(Master)	MPU6500(Slave)
VDD(3.3V)	VCC
GND	GND
SCL(PB0)	SCL
SDA(PB1)	SDA

Acceleration Output





Move it along X axis, acceleration of X axis should vary the most.

Move it along Y axis, acceleration of Y axis should vary the most.

Move it along Z axis, acceleration of Z axis should vary the most.

Homework W11-1.

https://github.com/CYCU-AIoT-System-Lab/Microcontroller-Experiment/blob/main/w11/I2C-I2C_MPU6500-Experiment_Steps.md

Execute the example, add data of Y axis

- Objective: Display Y axis data.
- Hint: Modify code in while loop of main function.

```
This example demonstrates how to use I2C to communicate with MPU6500
I2C slave device ID: 70
Accelometer MPU6500 is connected
Selected Acc Scale: +-2G
Live data from MPU6500:
AccX --> Raw_Value_DEC: 65516 , Raw_Value_HEX: ffec , Acc_Scale: -0.001221
AccZ --> Raw_Value_DEC: 63952 , Raw_Value_HEX: f9d0 , Acc_Scale: -0.096680
```

```
This example demonstrates how to use I2C to communicate with MPU6500
I2C slave device ID: 70
Accelometer MPU6500 is connected
Selected Acc Scale: +-2G
Live data from MPU6500:
AccX --> Raw_Value_DEC: 63052 , Raw_Value_HEX: f64c , Acc_Scale: -0.151611
AccY --> Raw_Value_DEC: 15104 , Raw_Value_HEX: 3b0c , Acc_Scale: 0.922607
AccZ --> Raw_Value_DEC: 58264 , Raw_Value_HEX: e398 , Acc_Scale: -0.443848
```

☆ PS. Please record.

☆ Link to Text Formating.

https://youtu.be/Pkrd-i2eFxs

C debugging

https://github.com/CYCU-AIoT-System-Lab/Microcontroller-Experiment/blob/main/w11/c_debugging.md

Class Dismissed