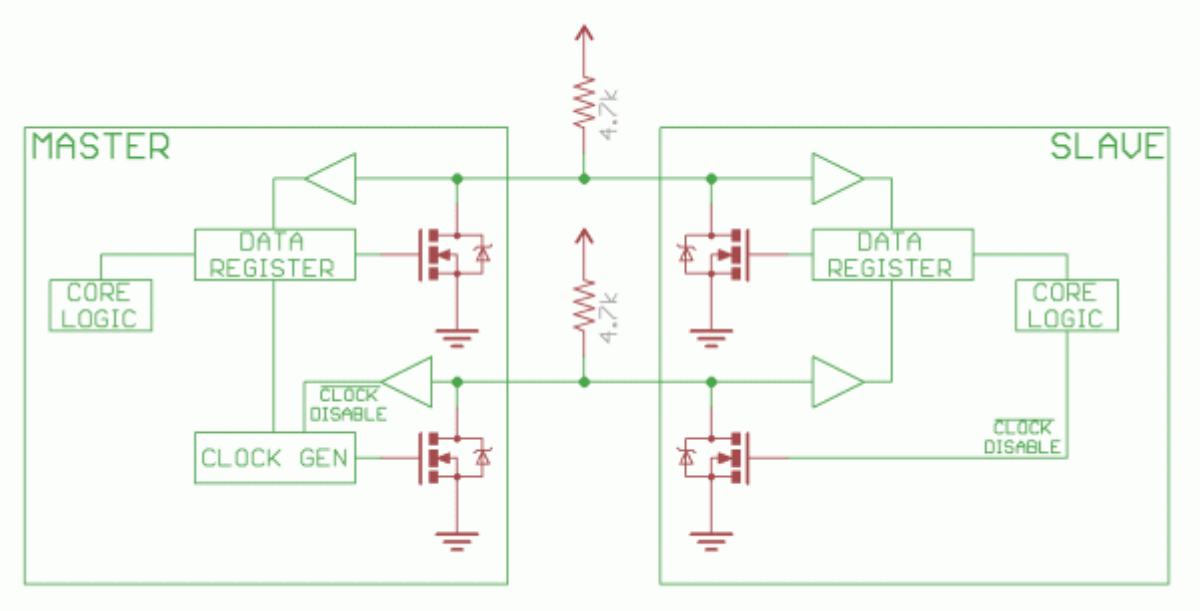
Giao thức I2C:

Giao thức I2C là chuẩn giao tiếp đồng bộ được Phillips phát triển, đặc điểm của chuẩn giao tiếp này là chỉ cần dùng 2 dây nhưng có thể giao tiếp với nhiều nhất là 127 thiết bị, đồng thời chuẩn I2C cũng được sử dụng rất phổ biến trong giao tiếp giữa các cảm biến, chip nhớ.

Đặc điểm:

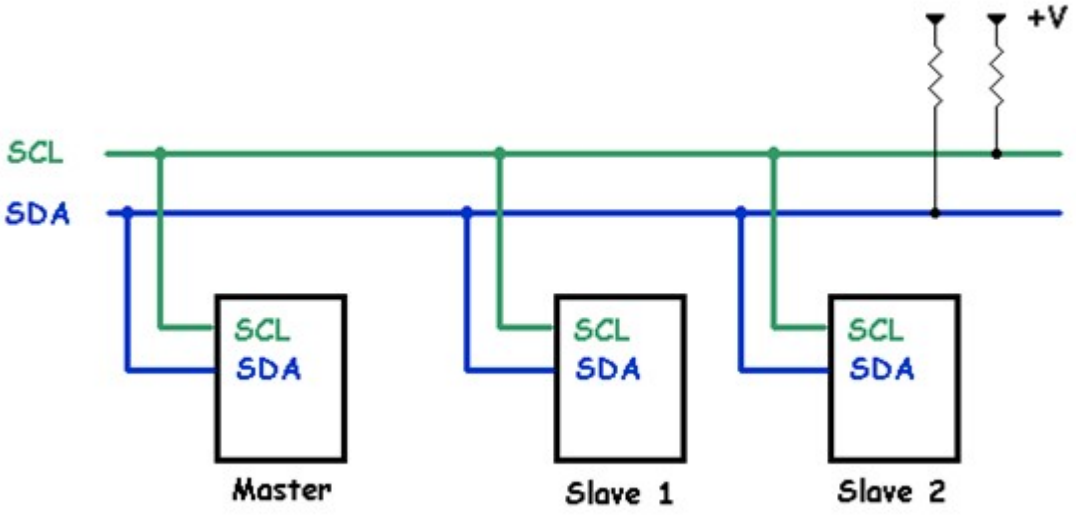
Gồm 2 dây SDA (serial data) và dây SCL (serial clock), dây data để truyền dữ liệu, còn dây clock để đồng bộ giữa 2 thiết bị và tạo xung nhịp cho giao tiếp, xung nhịp của I2C có các mức là 100kHz hoặc 400kHz, chế độ fast mode I2C có thể lên đến 1MHz, high speed mode 3.4MHz, ultra fast mode lên đến 5MHz.

Thiết kế của phần cứng I2C là thiết kế cực hở (open drain) nên mỗi dây SDA và SCL cần được kéo lên cao bằng một điện trở khoảng 2.2k đến 10k, thiết kế này chỉ cho điều khiển mức điện áp xuống 0 nên bắt buộc phải có trở kéo, tuy vậy ta có thể kéo điện trở lên bằng mức mà mình mong muốn thay vì phải chuyển mức điện áp. Các module bán sẵn thường có kéo trở từ trước.

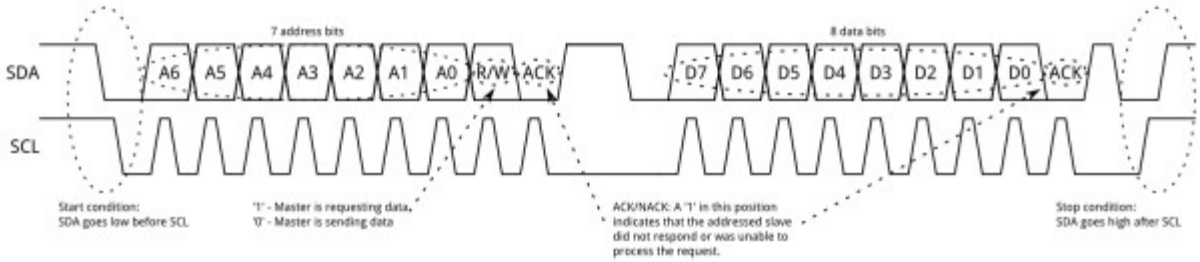


Khi I2C hoạt động, các thiết bị được nối vào cùng một đường dây, mỗi thiết bị sẽ có một địa chỉ riêng 7 bitsr45tt trong khoảng từ 0000000 – 1111111 (0x00 – 0xFE), tuy nhiên chính thức chỉ có 112 địa chỉ có thể sử dụng, ngoài ra còn có chế độ địa chỉ 10 bits.

Khi I2C hoạt động, một thiết bị sẽ đóng vai trò là master, điều khiển quá trình giao tiếp và phát clock, còn một thiết bị sẽ đóng cai trò slave, nhận và trả lại tín hiệu điều khiển từ master.



Quá trình truyền nhận của giao thức I2C:



Trạng thái bắt đầu:

Khi bắt đầu truyền, master sẽ kéo chân SDA xuống low, sau đó kéo SCL xuống low 1chu kì để báo hiệu việc bắt đầu truyền tin.

Sau đó, master truyền 7 bit địa chỉ của thiết bị cần kết nối + 1 bit R/W để cho biết mình đang cần đọc hay ghi dữ liệu, trong trường hợp ban đầu thường là bit 0.

Sau khi truyền xong bit địa chỉ, master sẽ chờ cho slave nhận được thông tin và trả về bit dữ liệu ACK (đã nhận được, thực hiện thành công), nếu hết thời gian timeout mà chưa nhận được tín hiệu trả về thì master sẽ báo timeout.

Trạng thái truyền tin:

Sau khi truyền được địa chỉ, một kết nối được tạo lập giữa master và slave, lúc này master sẽ truyền các byte dữ liệu qua slave để thực hiện, ứng với mỗi xung clock trên SCL sẽ là một bit dữ liệu truyền qua SDA,(sau khi truyền master sẽ chờ tín hiệu ACK, ACK này có thể là kết quả đo đạc của cảm biến, etc,...) trả về để xác nhận truyền thành công, nếu không nhận được ACK master sẽ cho là đã xảy ra lỗi.

Trạng thái STOP:

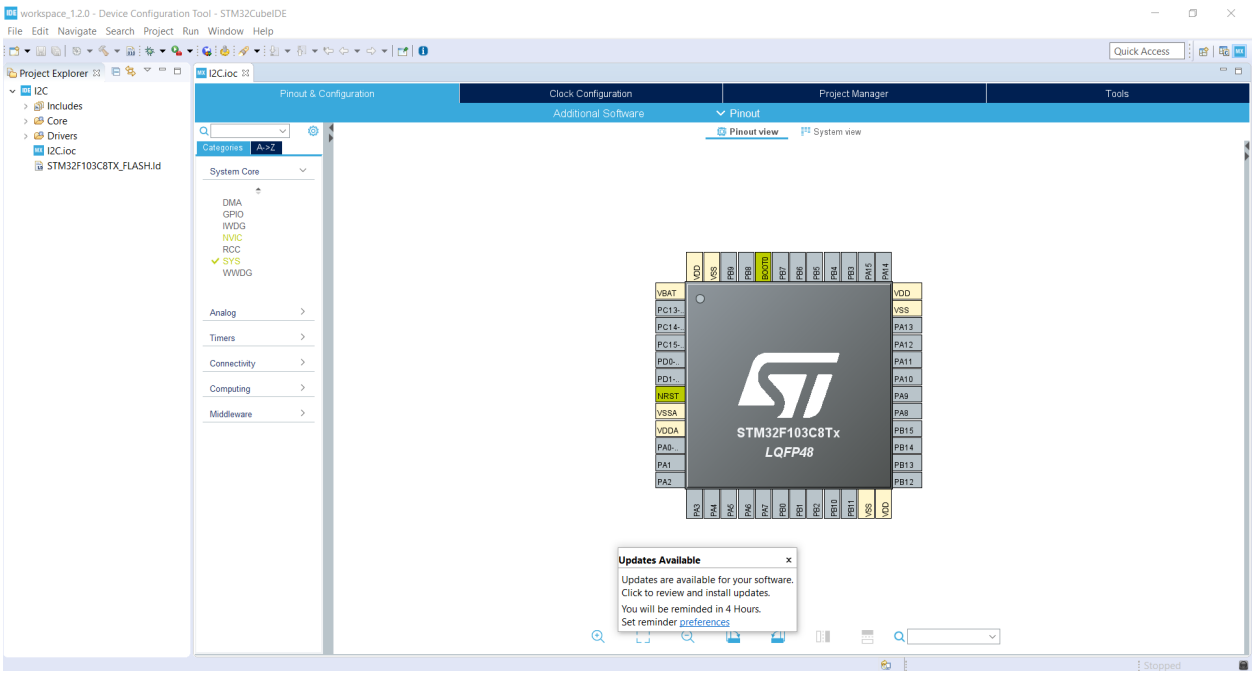
Để kết thúc truyền tin master kéo SCL lên cao trước sau đó kéo SDA lên cao sau.

*Câu hỏi: Nếu như khi đang truyền data mà có một master nữa muốn truyền data thì chuyện gì sẽ xảy ra?

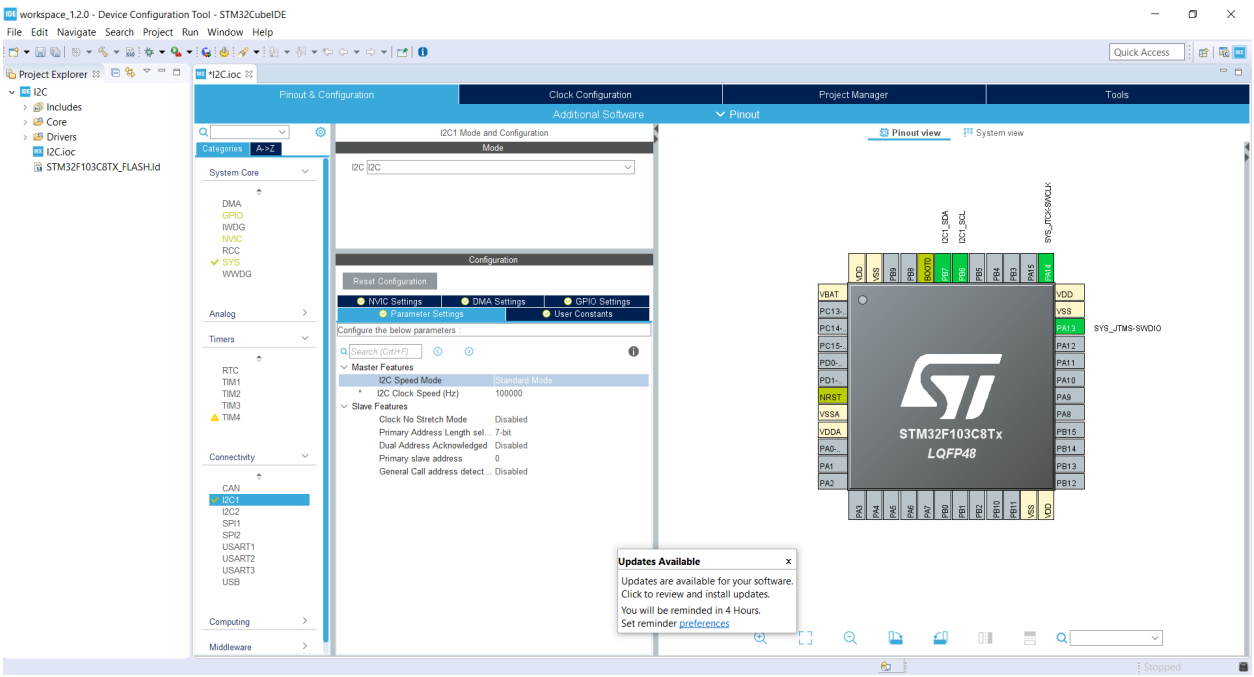
I2C trên STM32

Trong thực tế, I2C thường được sử dụng cho các cảm biến có nhiều chế độ, thanh ghi, do đó cách sử dụng giao thức này thường khá phức tạp.

Tạo project



Set up I2C như sau:



Về Stm32f103c8t6 có hỗ trợ I2C standard và fast mode, standard speed có clock là 100kHz, fast mode có clock là 400kHz

Giờ thì generate code và ta có cấu hình I2C được set up như sau:

Struct hi2c1 kiểm soát I2C 1:

```
43 /* Private variables ---------------------------------------------------------*/
44 I2C_HandleTypeDef hi2c1;
```

Hàm setup các giá trị của I2C 1:

```
149 static void MX_I2C1_Init(void)
150 {
151
152   /* USER CODE BEGIN I2C1_Init 0 */
153
154   /* USER CODE END I2C1_Init 0 */
155
156   /* USER CODE BEGIN I2C1_Init 1 */
157
158   /* USER CODE END I2C1_Init 1 */
159   hi2c1.Instance = I2C1;
160   hi2c1.Init.ClockSpeed = 100000;
161   hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
162   hi2c1.Init.OwnAddress1 = 0;
163   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
164   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
165   hi2c1.Init.OwnAddress2 = 0;
166   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
167   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
168   if (HAL_I2C_Init(&hi2c1) != HAL_OK)
169   {
170     Error_Handler();
171   }
172   /* USER CODE BEGIN I2C1_Init 2 */
173
174   /* USER CODE END I2C1_Init 2 */
175
176 }
```

Một số hàm HAL cần biết cho giao thức I2C:

HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout) :
Hàm này là hàm truyền byte từ master sang slave, gồm các biến:
I2C_HandleTypeDef *hi2c: Tham trị trỏ đến đối tượng i2c đang xét đến.

uint16_t DevAddress: Địa chỉ slave 7 bits được kết nối đến dịch trái or với bit read/write, trong hầu hết trường hợp, để đọc ta thêm bit 1 vào cuối, để write ta thêm bit 0.

uint8_t *pData: pointer trỏ đến data cần được gửi đi.

uint16_t Size: Kích thước data cần gửi đi.

uint32_t Timeout: timeout của master, tính theo đơn vị mili giây.

*Chú ý: Hàm này và các hàm khác đều trả về kiểu HAL_StatusTypeDef

Ví dụ:

```
uint8_t data[2] = {0xFF,0xAB};
HAL_I2C_Master_Transmit(&hi2c1, (uint16_t) 0xD0, (uint8_t *) pData, 2, 100);
```

Gửi 2 byte 0xFF, 0xAB đến slave ở địa chỉ 0x68 (0xD0 = 0x68<<1 | 0), yêu cầu đọc (bit 0), timeout 100ms.

`HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
Hàm này là hàm nhận data của master, khi được gọi nó sẽ chờ cho slave từ địa chỉ DevAddress gửi data đến, lưu Size byte vào các ô nhớ địa chỉ của *pData trỏ tới.

Tương tự, các hàm
`HAL_I2C_Slave_Transmit(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
`HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
Thực hiện các tác vụ tương ứng với các hàm trên cho chế độ Slave.

`HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)`

Hàm này kiểm tra xem device ở địa chỉ DevAdress có hoạt động không, trong đó Trials là số lần thử kết nối lại. Nếu hàm thực hiện thành công sẽ trả về HAL_OK

Ví dụ:
```
if(HAL_I2C_IsDeviceReady(hi2c1, (uint16_t) 0xD0, 3, 1000) == HAL_OK){

}
```

*Bài tập: Viết hàm scanI2C() thực hiện việc tìm tất cả các thiết bị có trong mạng I2C và in kết quả ra serial port.

Ví dụ thực tế:

MPU6050:

Đây là loại cảm biến gia tốc kế và con quay hồi chuyển và cảm biến nhiệt độ 3 trong một với giá rẻ, dễ sử dụng và cho hiệu quả tốt, được ứng dụng trong điện thoại, drone rất nhiều.

Cảm biến này sử dụng giao thức I2C, trong các loại drone người ta hay sử dụng giao thức SPI do tốc độ cao hơn, nhưng với lý do học tập ta sẽ xài I2C :3

1. Datasheet:
   Datasheet của cảm biến:
   https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
   Danh sách thanh ghi:
   https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf
   Chỉ cần hai món này là các bạn có thể code được rồi.

Board MPU6050 bán sẵn trên thị trường đã có kéo trở lên nên chỉ cần kết nối SCL – SCL, SDA – SDA và không quên nối nguồn là dùng được.

2. Quy trình đọc dữ liệu:

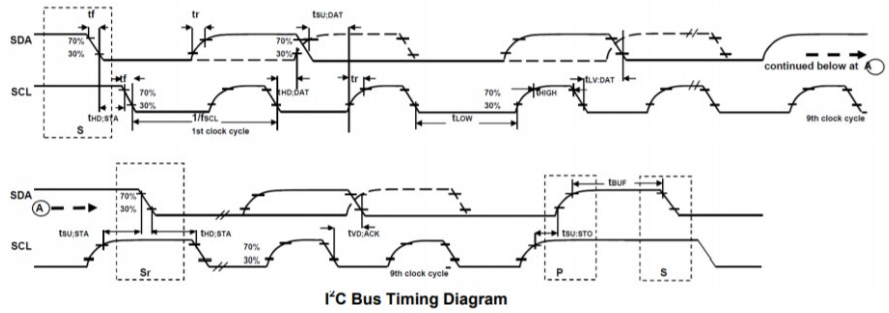
Theo datasheet, timing protocol I2C của MPU6050 như sau:

6.7 I²C Timing Characterization

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING	**I²C FAST-MODE**					
f_{SCL}, SCL Clock Frequency				400	kHz	
$t_{HD.STA}$, (Repeated) START Condition Hold Time		0.6			µs	
t_{LOW}, SCL Low Period		1.3			µs	
t_{HIGH}, SCL High Period		0.6			µs	
$t_{SU.STA}$, Repeated START Condition Setup Time		0.6			µs	
$t_{HD.DAT}$, SDA Data Hold Time		0			µs	
$t_{SU.DAT}$, SDA Data Setup Time		100			ns	
t_r, SDA and SCL Rise Time	C_b bus cap. from 10 to 400pF	20+0.1C_b		300	ns	
t_f, SDA and SCL Fall Time	C_b bus cap. from 10 to 400pF	20+0.1C_b		300	ns	
$t_{SU.STO}$, STOP Condition Setup Time		0.6			µs	
t_{BUF}, Bus Free Time Between STOP and START Condition		1.3			µs	
C_b, Capacitive Load for each Bus Line			< 400		pF	
$t_{VD.DAT}$, Data Valid Time				0.9	µs	
$t_{VD.ACK}$, Data Valid Acknowledge Time				0.9	µs	

Note: Timing Characteristics apply to both Primary and Auxiliary I²C Bus



I²C Bus Timing Diagram

Có thể thấy đây là giao thức I2C khá bình thường, tốc độ tối đa có thể đạt được là 400kHz.

Phần lớn các IC sử dụng kết nối I2C đều có một hệ thống thanh ghi phức tạp (ví dụ IC DS1307, ROM,...) để truy cập các thanh ghi này, các IC thường có thêm một bộ địa chỉ thanh ghi.

Sau khi thiết lập kết nối với cảm biến, cảm biến sẽ tự hiểu là byte data tiếp theo sẽ là địa chỉ thanh ghi và truy cập vào trong thanh ghi.

Ví dụ:

```
#define  PWR_MGMT_1 0x6B

 uint8_t d[2];

 uint8_t device_address =0xD0;

d[0] = PWR_MGMT_1;

d[1] = 0;
```

while(HAL_I2C_Master_Transmit(&hi2c1,(uint16_t)device_address , (uint8_t *)d, 2, 1000) != HAL_OK);

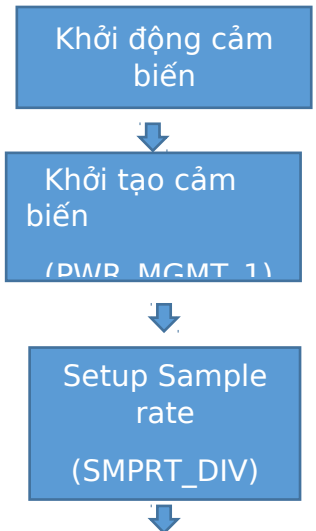
Câu lệnh này ghi vào thanh ghi PWR_MGMT_1 giá trị 0x00, câu lệnh while đảm bảo lặp lại việc ghi dữ liệu cho đến khi thành công.

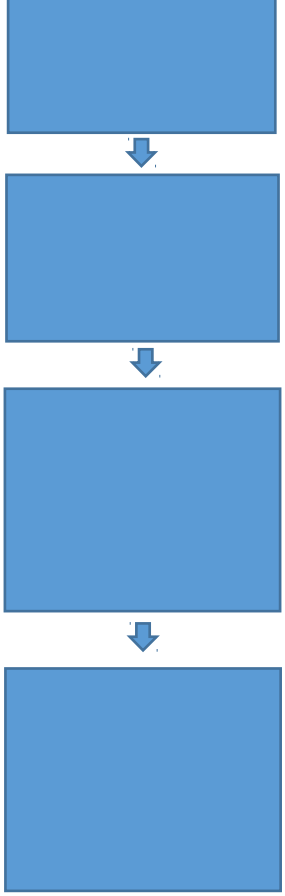
Để setup và sử dụng MPU6050 ta cần tham khảo Register Map và Datasheet:

The register map for the MPU-60X0 is listed below.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0D	13	SELF_TEST_X	R/W	XA_TEST[4-2]			XG_TEST[4-0]				
0E	14	SELF_TEST_Y	R/W	YA_TEST[4-2]			YG_TEST[4-0]				
0F	15	SELF_TEST_Z	R/W	ZA_TEST[4-2]			ZG_TEST[4-0]				
10	16	SELF_TEST_A	R/W	RESERVED		XA_TEST[1-0]		YA_TEST[1-0]		ZA_TEST[1-0]	
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	-	-	-	FS_SEL [1:0]		-	-	-
1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]				
23	35	FIFO_EN	R/W	TEMP _FIFO_EN	XG _FIFO_EN	YG _FIFO_EN	ZG _FIFO_EN	ACCEL _FIFO_EN	SLV2 _FIFO_EN	SLV1 _FIFO_EN	SLV0 _FIFO_EN
24	36	I2C_MST_CTRL	R/W	MULT _MST_EN	WAIT _FOR_ES	SLV_3 _FIFO_EN	I2C_MST _P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	R/W	I2C_SLV0 _RW	I2C_SLV0_ADDR[6:0]						
26	38	I2C_SLV0_REG	R/W	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	R/W	I2C_SLV0 _EN	I2C_SLV0 _BYTE_SW	I2C_SLV0 _REG_DIS	I2C_SLV0 _GRP	I2C_SLV0_LEN[3:0]			
28	40	I2C_SLV1_ADDR	R/W	I2C_SLV1 _RW	I2C_SLV1_ADDR[6:0]						
29	41	I2C_SLV1_REG	R/W	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	R/W	I2C_SLV1 _EN	I2C_SLV1 _BYTE_SW	I2C_SLV1 _REG_DIS	I2C_SLV1 _GRP	I2C_SLV1_LEN[3:0]			
2B	43	I2C_SLV2_ADDR	R/W	I2C_SLV2 _RW	I2C_SLV2_ADDR[6:0]						
2C	44	I2C_SLV2_REG	R/W	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	R/W	I2C_SLV2 _EN	I2C_SLV2 _BYTE_SW	I2C_SLV2 _REG_DIS	I2C_SLV2 _GRP	I2C_SLV2_LEN[3:0]			
2E	46	I2C_SLV3_ADDR	R/W	I2C_SLV3 _RW	I2C_SLV3_ADDR[6:0]						
2F	47	I2C_SLV3_REG	R/W	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	R/W	I2C_SLV3 _EN	I2C_SLV3 _BYTE_SW	I2C_SLV3 _REG_DIS	I2C_SLV3 _GRP	I2C_SLV3_LEN[3:0]			
31	49	I2C_SLV4_ADDR	R/W	I2C_SLV4 _RW	I2C_SLV4_ADDR[6:0]						
32	50	I2C_SLV4_REG	R/W	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	R/W	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	R/W	I2C_SLV4 _EN	I2C_SLV4 _INT_EN	I2C_SLV4 _REG_DIS	I2C_MST_DLY[4:0]				
35	53	I2C_SLV4_DI	R	I2C_SLV4_DI[7:0]							
36	54	I2C_MST_STATUS	R	PASS_ THROUGH	I2C_SLV4 _DONE	I2C_LOST _ARB	I2C_SLV4 _NACK	I2C_SLV3 _NACK	I2C_SLV2 _NACK	I2C_SLV1 _NACK	I2C_SLV0 _NACK
37	55	INT_PIN_CFG	R/W	INT_LEVEL	INT_OPEN	LATCH _INT_EN	INT_RD _CLEAR	FSYNC_ INT_LEVEL	FSYNC _INT_EN	I2C _BYPASS _EN	-

=)) cảm giác khá là đau đầu đúng không, thực sự mà nói thì datasheet của con MPU6050 này không thực sự tốt lắm, nhưng không sao, quy trình thực hiện như sau:





Các giá trị và thanh ghi này các bạn có thể tham khảo trong Register Map, tuy nhiên các bạn có thể sử dụng các setup mà các bạn thấy trong các code Arduino, nhưng bây giờ có lẽ các bạn đã hiểu tại sao lại làm vậy =))

Các giá trị mình sử dụng:

Thanh ghi	Địa chỉ thanh ghi	Giá trị cần ghi
PWR_MGMT_1	0x6B	0x00
SAMPLE_RATE	0x19	0x07
GYRO_CONFIG	0x1B	0x1B
ACCEL_CONFIG	0x1C	0x00
INT_ENABLE	0x38	0x01
USER_CTRL	0x6A	0x00

Vậy là chúng ta đã setup xong MPU6050, bước tiếp theo là đọc dữ liệu từ cảm biến:

MPU6050 có một bộ ADC 16 bit nên data trả về là 1 số nguyên 16 bits, mà I2C chỉ gửi theo byte, chính vì vậy cần phải đọc 2 byte cao và thấp sau đó ghép chúng lại theo công thức:

Data = (Byte cao <<8|byte thấp)/trọng số.

Trong đó trọng số tùy thuộc vào loại cảm biến và tầm đo của cảm biến đã được config.

Đồng thời MPU6050 có chế độ ACK liên tục, tức khi ta đọc một thanh ghi trong bộ thanh ghi dữ liệu, lần lượt các thanh ghi sẽ đưa dữ liệu về cho master. Dữ liệu bao gồm các thành phần theo thứ tự:

- Dữ liệu gia tốc theo 3 trục x,y,z, mỗi trục gồm 2 bytes, tổng cộng 6 bytes.
- Dữ liệu cảm biến nhiệt độ, 2 bytes.
- Dữ liệu cảm biến con quay hồi chuyển theo 3 trục mỗi trục 2 bytes, tổng công 6 bytes.

Như vậy khi ta đọc thanh ghi đầu tiên của gia tốc kế, cảm biến sẽ trả về tổng cộng 14 bytes.

Ví dụ:

```
uint8_t data[14];

float Acc_x;

while(HAL_I2C_Master_Transmit(&hi2c1,(uint16_t)0xD0, 0x3B, 1, 1000) != HAL_OK);

while(HAL_I2C_Master_Receive(&hi2c1,(uint16_t)0xD0, data,14, 1000) != HAL_OK);

Acc_x = (int16_t)(data[0] << 8 | data[1]); // chú ý là giá trị của gia tốc có thể âm.
```

4.17 Registers 59 to 64 – Accelerometer Measurements ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Description:

These registers store the most recent accelerometer measurements.

Accelerometer measurements are written to these registers at the Sample Rate as defined in Register 25.

The accelerometer measurement registers, along with the temperature measurement registers, gyroscope measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the accelerometer sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit accelerometer measurement has a full scale defined in *ACCEL_FS* (Register 28). For each full scale setting, the accelerometers' sensitivity per LSB in *ACCEL_xOUT* is shown in the table below.

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Parameters:

ACCEL_XOUT 16-bit 2's complement value.

Stores the most recent X axis accelerometer measurement.

ACCEL_YOUT 16-bit 2's complement value.

Stores the most recent Y axis accelerometer measurement.

ACCEL_ZOUT 16-bit 2's complement value.

Stores the most recent Z axis accelerometer measurement.

*Bài tập: Viết code cho MPU6050 trên board Chicken, gửi data về serial port.