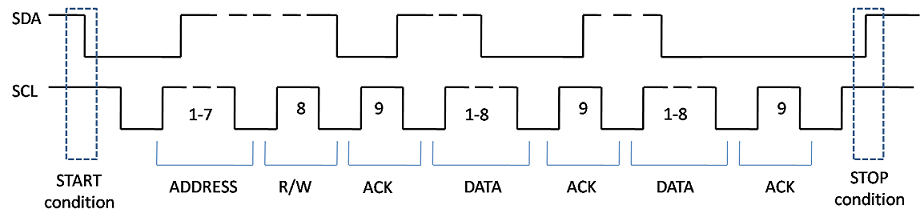
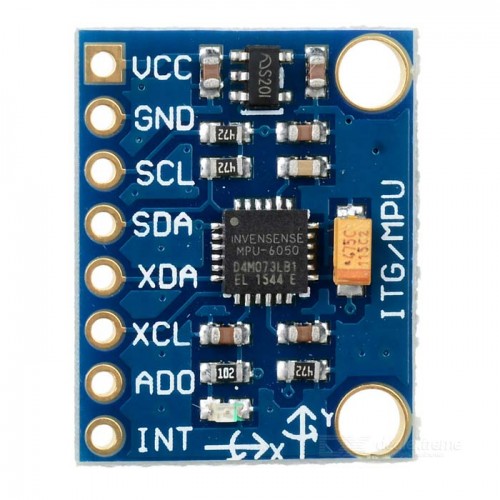
**Interfacing MPU6050 sensor with STM32F407 using I2C Protocol**

**I2C Protocol**

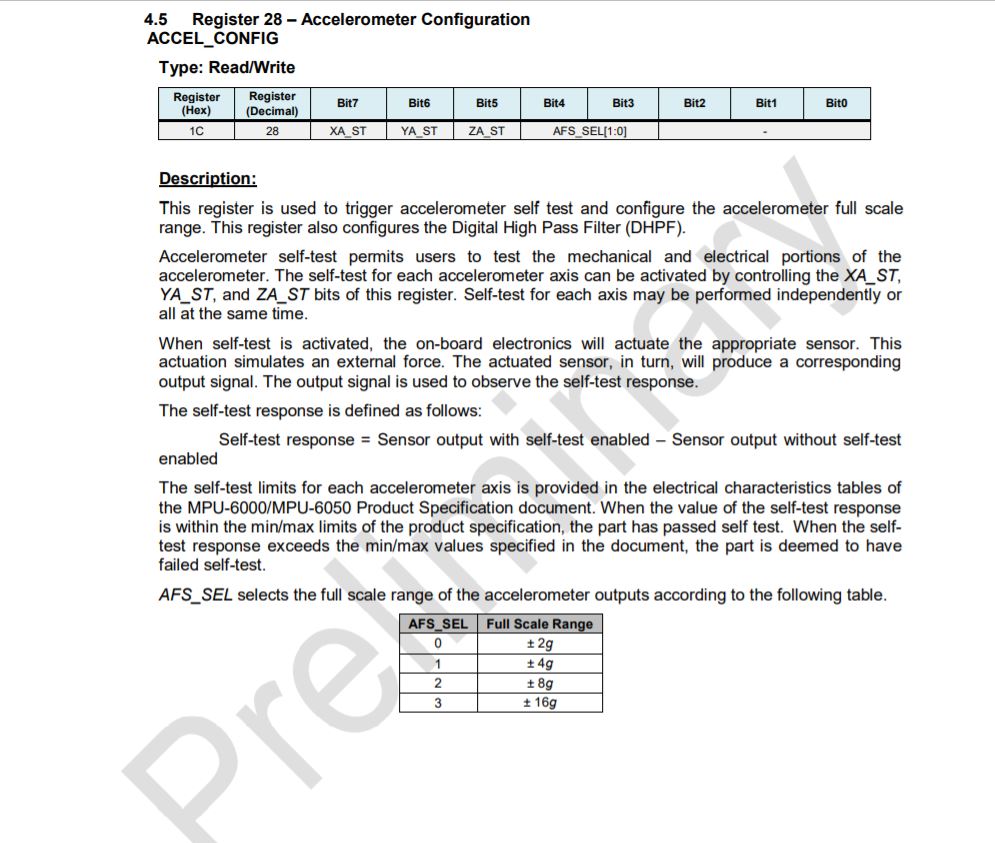
* I2C protocol is based on master-slave communication.
* There’ll be one master and many slaves.
* It’s a half-duplex and synchronous mode of communication.
* It has 2 lines, one for clock (SCL or Serial Clock) and another for data line (SDA or Serial Data)
* The maximum speed of I2C is 400KHz. It is also used at 100KHz speed in many applications.
* Each slave device should have a unique I2C slave address, based on which communication would happen between master and slave.



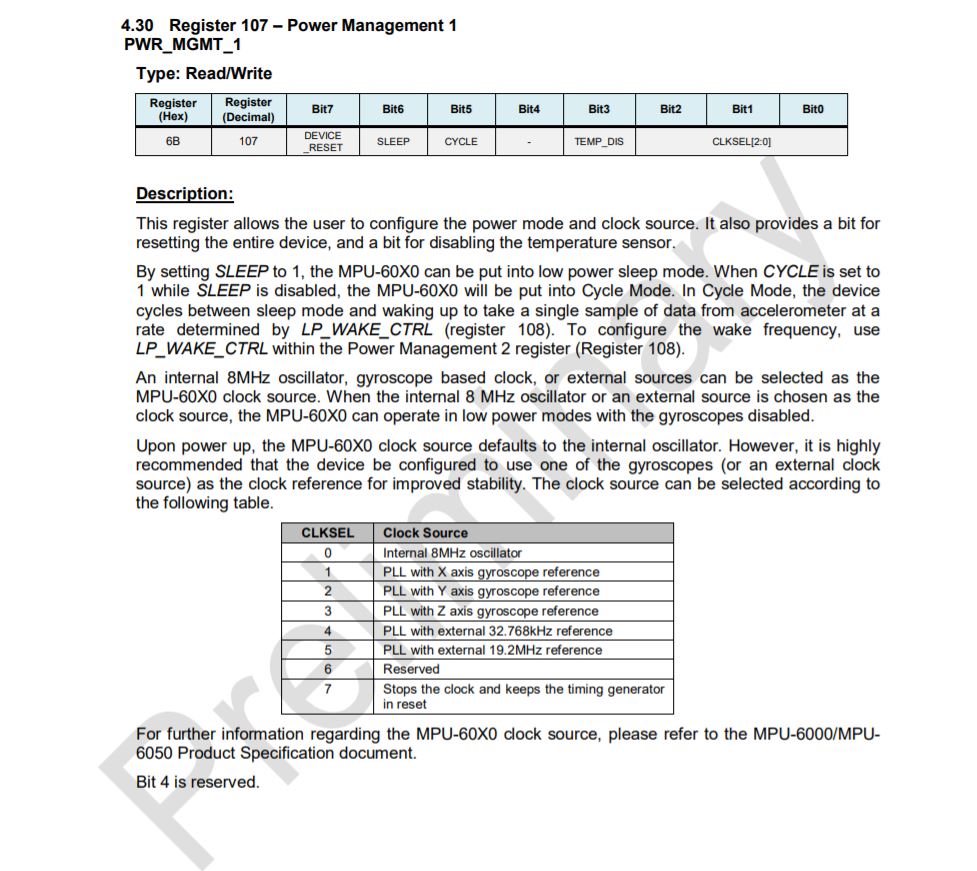
**Things to know about MPU6050:**



* The slave address of MPU6050 is 0xD0.
* MPU6050 contains many registers from 0x0D to 0x75.
* We need to configure these registers according to our need.
* Register 27 (0x1B) corresponds to gyroscope configuration.
* Register 28 (0x1C) corresponds to accelerometer configuration. These can be seen in the data sheet.
* Next, we have to configure the above registers (27 if we are using gyroscope. 28 if we are using accelerometer) and write the values in the registers for configuring it. The value which we write, is to specify the full-scale range. Full scale range is the range of force that can be measured. Lesser the range, more the sensitivity.



* Select the full-scale range that is required and write the appropriate value into the register using I2C write operation. For ex, we have to write 0x08 to configure it in 4g range.
* One of the important registers that needs to be configured is register 107 (0x6B).



* Register 6B is power management register. The sensor MPU6050 by default is in sleep mode when powered up. So, we have to write to this register first, to activate the sensor before we do any other write or read operation. We have to write 0x00 to this register to make it work using its internal clock of 8MHz.
* Next, we must know about the sample rate. The default output rate of the gyroscope and accelerometer is 8KHz. To vary the sample rate, we can write to Sample rate divider register of the accelerometer (0x19).

According to the data sheet,

Sampling rate = (output rate) / (1+Value to be written)

We have to write 0 to get a sampling rate of 8KHz and 7 to get a sampling rate of 1KHz.

We can change the output rate of the accelerometer by enabling the DLPF in register 0x1A as clearly given in the data sheet.

So, if the register is not written into, the sampling

* Register 0x1A corresponds to output rate of the accelerometer. To configure it, we have to write in the DLPF register. To get output rate as 8KHz, we have to write 0 or 7.
* To get output rate of 1KHz, we have to enable the DLPF register by writing values from 1 to 6.
* To use the interrupt of the sensor, we have to enable the interrupt register(0x38).
* Next, we have to configure the interrupt register. Interrupt should be enabled whenever there is a data available. For that, we have to 0x01 to the interrupt register(0x38).
* The sensor should be supplied with 3V.
* The accelerometer output value will be from 0x3B register to 0x40 register.
* For every x, y and z axis, there are 2-byte values, one corresponding to High and other low. So, we have to join the 2 bytes into a single 16-bit signed integer number.
* The value read from the accelerometer will be the raw value. We have to divide it by the sensitivity of the sensor to get the output acceleration in terms of g.
* For +/- 4g range, the sensitivity is 8192. So, we have to divide the raw 16-bit value by 8192 to get the required output.
* The output of gyroscope can be read from registers 0x43 to 0x48

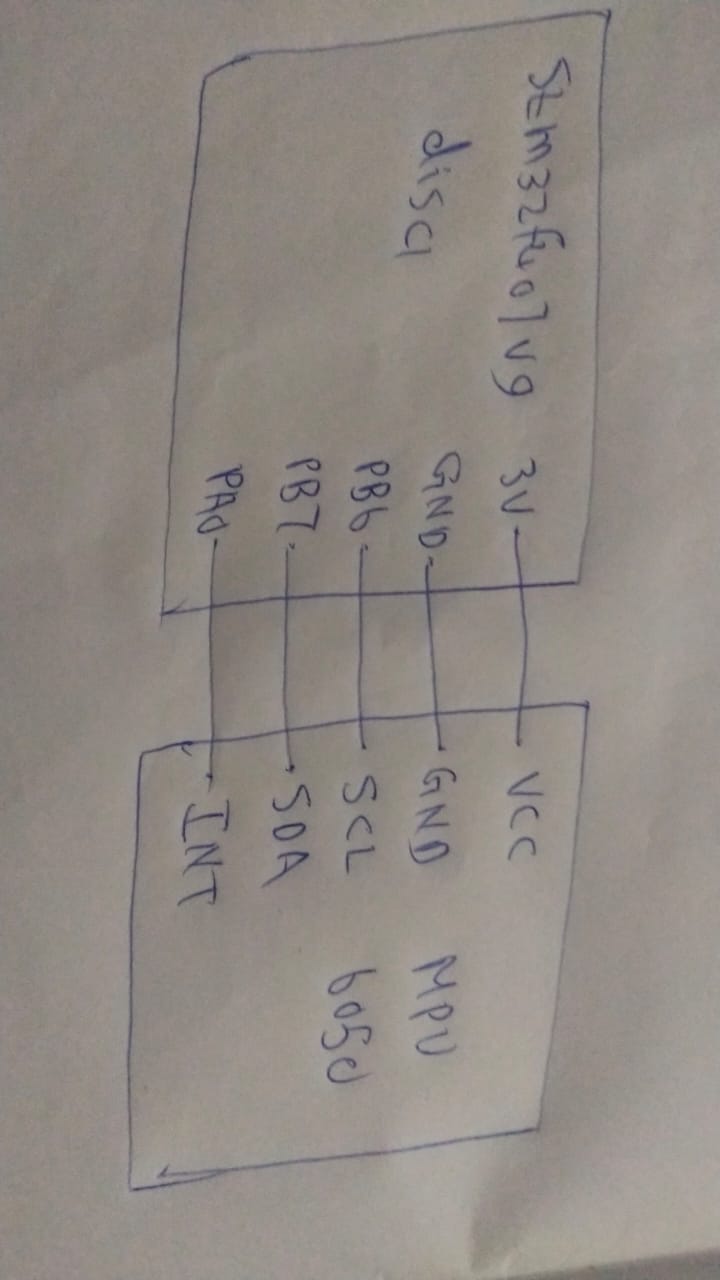
**Microcontroller Part:**



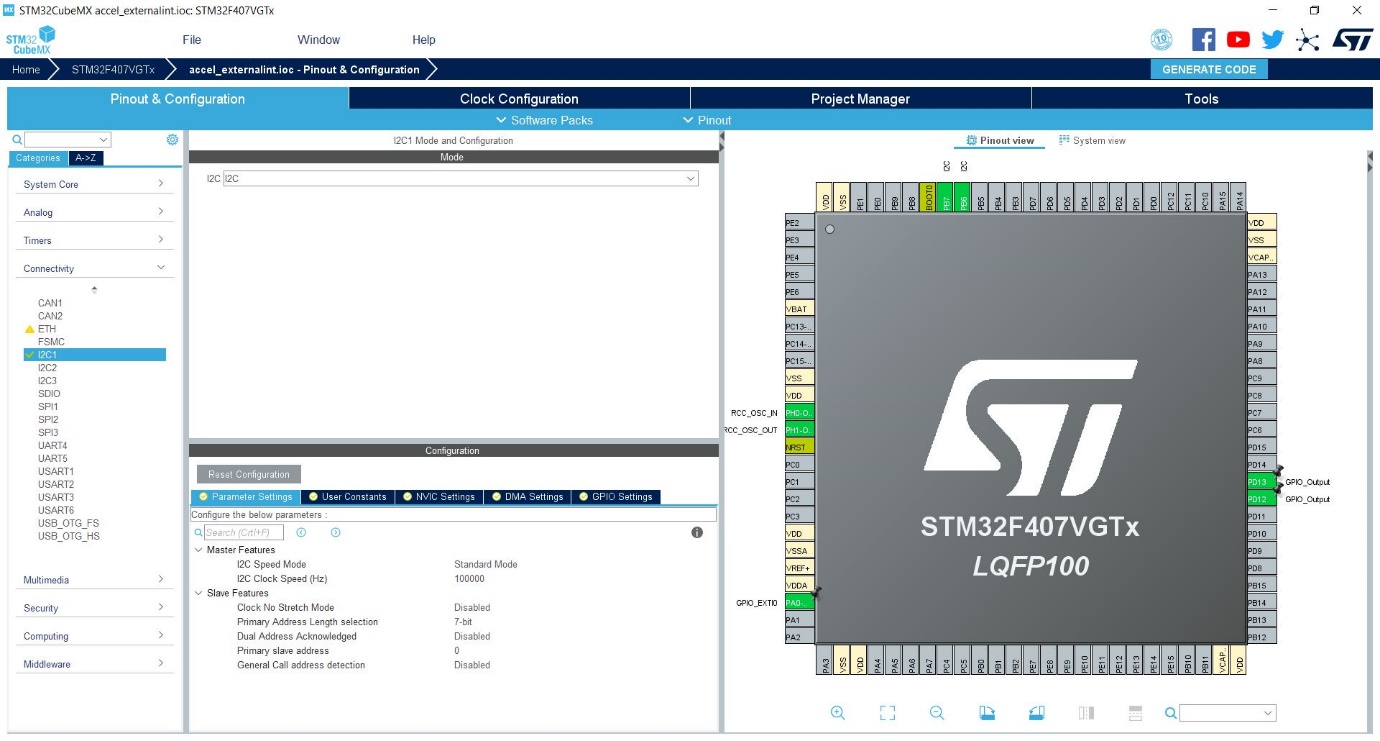
* I2C communication speed is set to 100KHz.
* The pins PB6 and PB7 corresponds to one set of I2C pins in the stm32f407vg disc1 board. PB6 is the SCL pin and PB7 is the SDA pin.
* We have to enable these pins in pullup mode, which is a must for I2C communication.
* We also have to enable external interrupt 0. This corresponds to PA0 pin in the discovery board.

Connection:

* Connect SCL pin in the sensor to PB6 and SDA pin to PB7.
* Power up the sensor using the 3V pin in the board.
* Connect the ground in the sensor to ground in the board.
* Connect the INT pin in the sensor to PA0.

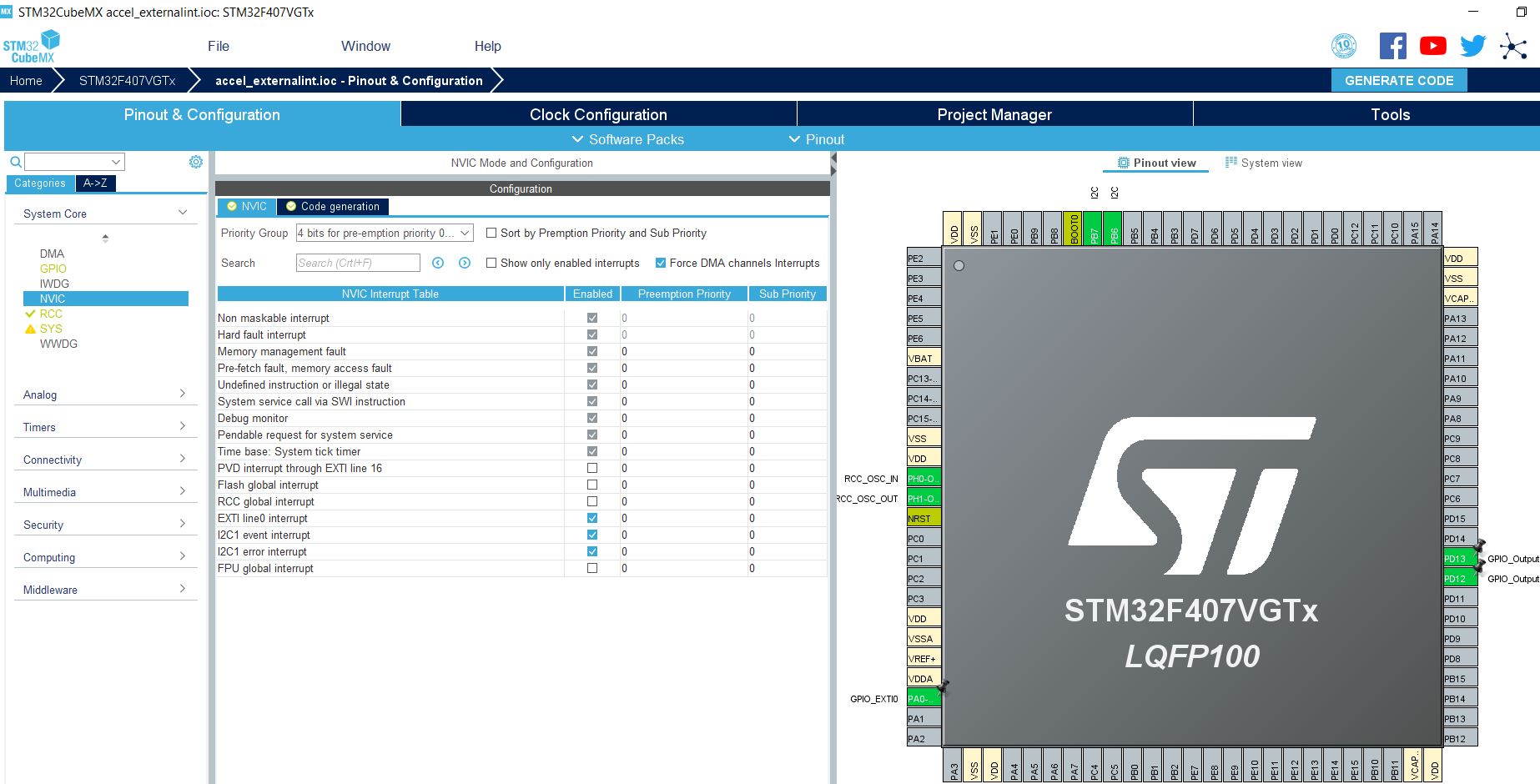


**CubeMX configurations:**



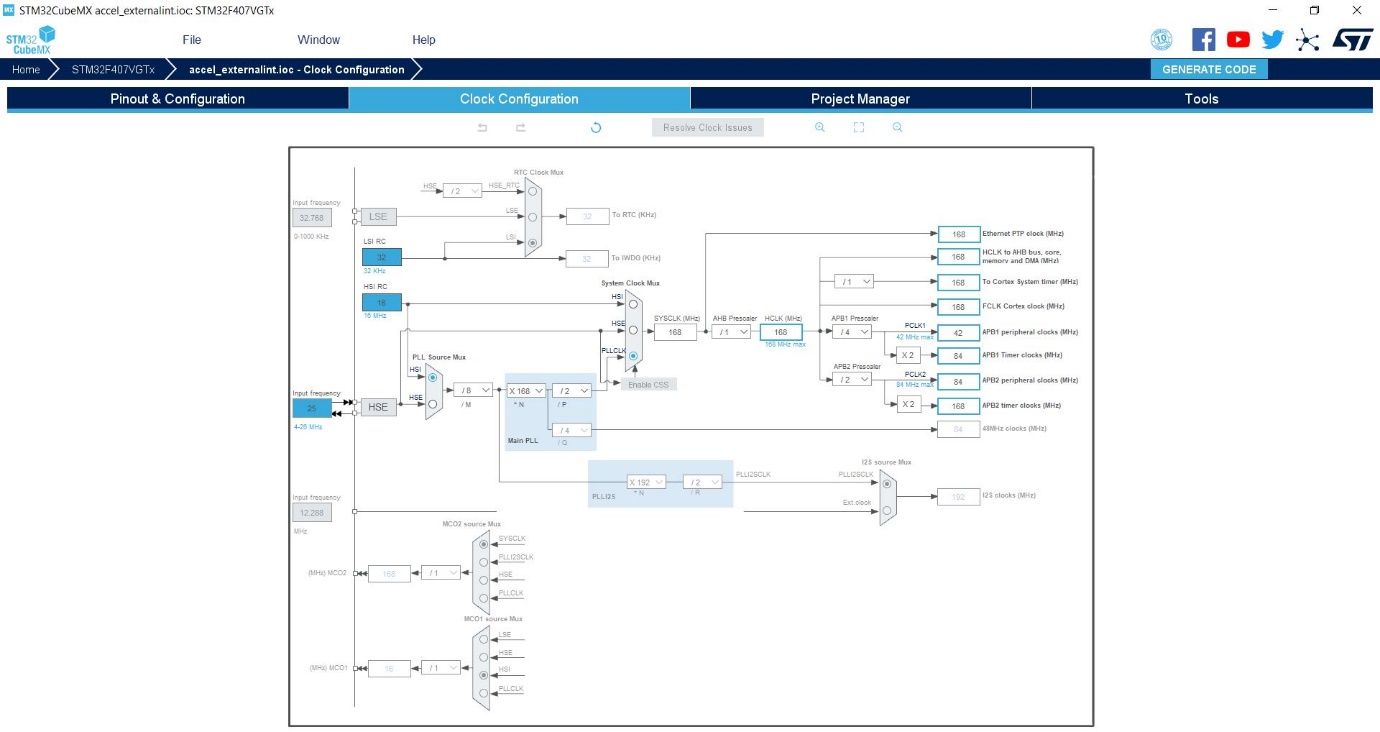
First, enable the I2C mode under connectivity and set the required I2C speed. Next, select the pin PA0 and make it as pin for external interrupt 0. Enable PD12, LED for checking purposes.

Under system core, click on RCC and enable high speed crystal/ceramic oscillator.

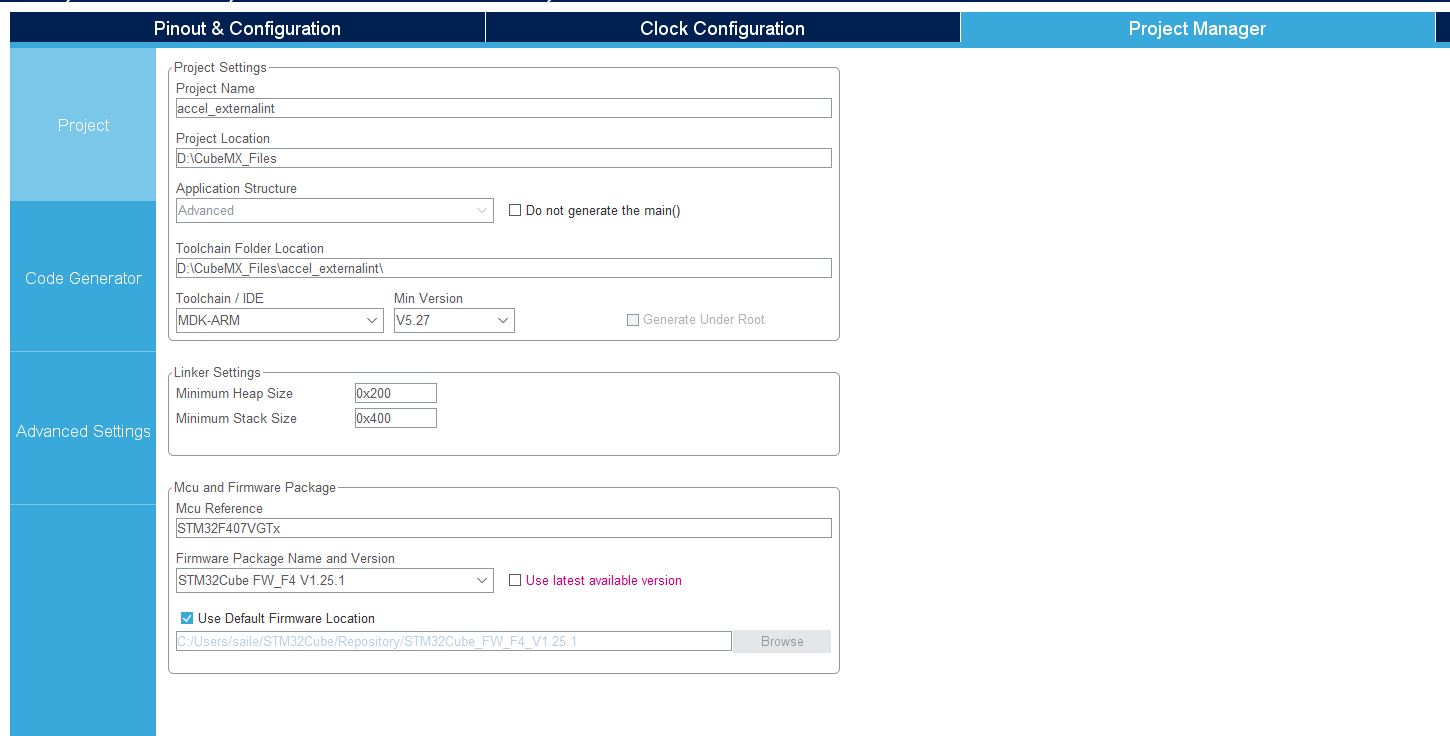


Next, under system core, go to NVIC and make sure to enable the line for external interrupt. Enabling I2C interrupt is optional.

After enabling interrupt line, go to clock configuration and change HCLK to 168 MHz which is the clock in which the microcontroller would work.



After completing all the configurations, go to project manager, give an appropriate title for project, choose its location. Select the tool chain IDE as ARM-MDK if keil is the IDE being used. If not, select the corresponding IDE.



According to the code, if the connection is established with the sensor, when executed, the LED corresponding to PD12 would glow.

**Code:**

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.

\* All rights reserved.</center></h2>

\*

\* This software component is licensed by ST under BSD 3-Clause license,

\* the "License"; You may not use this file except in compliance with the

\* License. You may obtain a copy of the License at:

\* opensource.org/licenses/BSD-3-Clause

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

#define slaveaddress 0xD0 // Address of MPU6050 sensor

int16\_t ax,ay,az; // To save the 16 bit data from sensor

//float xaccel[4000],yaccel[4000],zaccel[4000];// Which will contain the final output values

float xaccel,yaccel,zaccel;

uint8\_t i2cBuf[8]; //Array which is used in reading and writing of values to sensor using I2C

uint16\_t i=0;

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

I2C\_HandleTypeDef hi2c1;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_I2C1\_Init(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_I2C1\_Init();

/\* USER CODE BEGIN 2 \*/

// Loop to check if the sensor is connected to the board and to verify its address

for(uint8\_t i=0;i<255;i++)

{

if(HAL\_I2C\_IsDeviceReady(&hi2c1,i,1,10)==HAL\_OK)

{

HAL\_GPIO\_TogglePin(GPIOD,GPIO\_PIN\_12);// Inboard LED toggles when sensor is detected

break;

}

}

// Powering up the sensor

i2cBuf[0]=0x6B;//Corresponds to power configuration register

i2cBuf[1]=0x00;//Enabling internal 8MHz clock of sensor

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,2,1);

i2cBuf[0]=28;// Corresponds to accelerometer configuration register

i2cBuf[1]=0x08;// Selecting full-scale range of +/- 4g

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,2,1);

//According to data sheet, sampling rate=(gyroscope output rate/(1+sample rate divider))

i2cBuf[0]=0x19;// Corresponds to sample rate divider register

i2cBuf[1]=0x07;// Setting the sample rate

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,2,1);

i2cBuf[0]=0x1A;//Corresponds to output rate of accelerometer

i2cBuf[1]=0x02;//Setting output rate to 1KHz

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,2,1);

i2cBuf[0]=0x38;//Interrupt enable register

i2cBuf[1]=0x01;//Sets interrupt when data is ready

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,2,1);

// Following lines can be used to verify if the read and write operations are working fine

/\*i2cBuf[0]=28;

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,1,10);

i2cBuf[1]=0x00;

// If i2cBuf[1] has 0x08 after read operation, the read and write operation works fine

HAL\_I2C\_Master\_Receive(&hi2c1,slaveaddress,&i2cBuf[1],1,10);\*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

//HAL\_Delay(1000);

//for(uint8\_t i=0;i<255;i++);

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 8;

RCC\_OscInitStruct.PLL.PLLN = 168;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 4;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief I2C1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_I2C1\_Init(void)

{

/\* USER CODE BEGIN I2C1\_Init 0 \*/

/\* USER CODE END I2C1\_Init 0 \*/

/\* USER CODE BEGIN I2C1\_Init 1 \*/

/\* USER CODE END I2C1\_Init 1 \*/

hi2c1.Instance = I2C1;

hi2c1.Init.ClockSpeed = 100000;

hi2c1.Init.DutyCycle = I2C\_DUTYCYCLE\_2;

hi2c1.Init.OwnAddress1 = 0;

hi2c1.Init.AddressingMode = I2C\_ADDRESSINGMODE\_7BIT;

hi2c1.Init.DualAddressMode = I2C\_DUALADDRESS\_DISABLE;

hi2c1.Init.OwnAddress2 = 0;

hi2c1.Init.GeneralCallMode = I2C\_GENERALCALL\_DISABLE;

hi2c1.Init.NoStretchMode = I2C\_NOSTRETCH\_DISABLE;

if (HAL\_I2C\_Init(&hi2c1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN I2C1\_Init 2 \*/

/\* USER CODE END I2C1\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : PA0 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_RISING;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\*Configure GPIO pins : PD12 PD13 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_12|GPIO\_PIN\_13;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOD, &GPIO\_InitStruct);

/\* EXTI interrupt init\*/

HAL\_NVIC\_SetPriority(EXTI0\_IRQn, 0, 0);

HAL\_NVIC\_EnableIRQ(EXTI0\_IRQn);

}

/\* USER CODE BEGIN 4 \*/

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_PIN)

{

i2cBuf[0]=0x3B;// Corresponds to register containing values of x axis

// Request to read from 3B register

HAL\_I2C\_Master\_Transmit(&hi2c1,slaveaddress,i2cBuf,1,1);

// The values of x,y and z axis are in 2 bytes (high and low) separately in the sensor from register 3B

// So we have to read 6 bytes starting from register 3B to 40

HAL\_I2C\_Master\_Receive(&hi2c1,slaveaddress,&i2cBuf[1],6,1);

// Combining the 2 bytes and saving it in a 16 bit number

ax=(i2cBuf[1]<<8|i2cBuf[2]);

ay=(i2cBuf[3]<<8|i2cBuf[4]);

az=(i2cBuf[5]<<8|i2cBuf[6]);

// Sensitivity of accelerometer in +/- 4g is 8192/g

// Converting the raw data in terms of acceleration in g

xaccel=ax/8192.0;

yaccel=ay/8192.0;

zaccel=az/8192.0;

/\*xaccel[i]=ax/8192.0;

yaccel[i]=ay/8192.0;

zaccel[i]=az/8192.0;

i++;\*/

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/