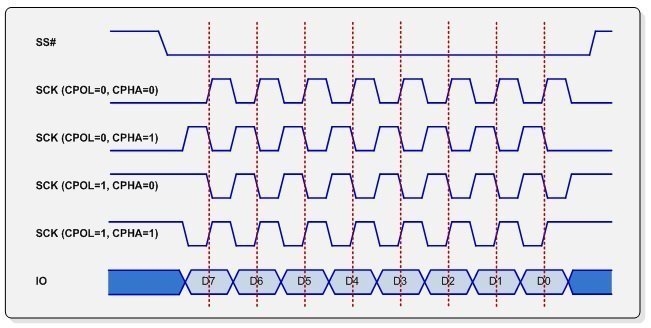
**Interfacing ADXL345 accelerometer sensor using SPI Protocol**

**SPI Communication mode**

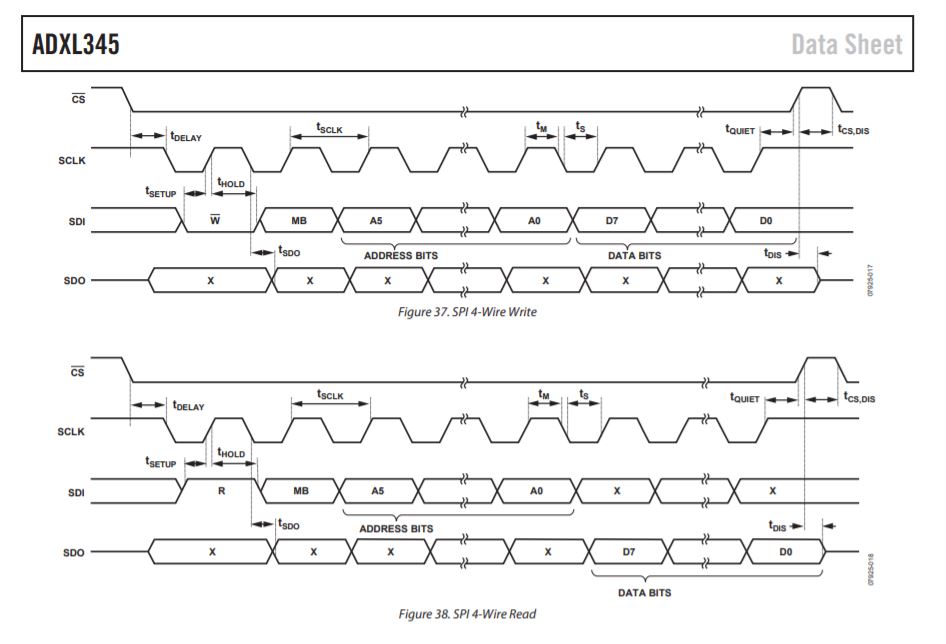
* SPI is a widely used high speed protocol, which can have data transfer data more than 10Mbps, faster than I2C protocol.
* It is a synchronous communication protocol, and can work in both half duplex or full duplex mode based on the master and slave devices.
* Full duplex mode is common and it involves of 4 lines minimum. One for clock (SCK), MISO (Master in Slave out), MOSI (Master out Slave in) and a CS (Chip select pin) which is active low.
* SCK, MOSI and MISO pins are common for all slave devices, if many slave devices are connected in full duplex mode. Every slave device should have a separate CS pin. It should be set LOW to activate the device, and after communication, the slave should be deactivated by setting the Chip Select pin to HIGH.
* So, the number of pins related to SPI in full duplex would be 3+number of slave devices.
* Every slave device would have a maximum baud rate. So, the SPI speed should be lesser or equal to the maximum speed of slave.
* Two other important terms regarding SPI is clock polarity and clock phase. Clock polarity defines the state of clock during inactive time. If clock polarity is 0, then the default state of clock would be low and vice versa if clock polarity is 1.
* Clock phase indicates at what edge of the clock, the data would be sampled. If clock phase is said to be 0 or low or 1 edge, then the data is processed at rising edge of the clock. If the clock phase is 1 or HIGH or 2 edge, then the data is processed at falling edge of the clock.



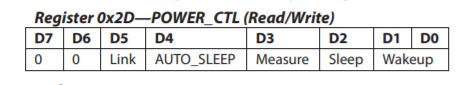
**ADXL345 Accelerometer**

Note: ADXL345 can be used both in SPI and I2C protocols. Here, SPI is being used. If the need is to establish I2C communication, then the SDA and SCL pins can be used as data and clock lines.

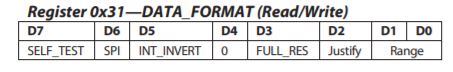
* 3V or 3.3V is the supply voltage which should be provided at the VCC pin of sensor.
* The device works in both half duplex and full duplex mode.
* The maximum speed is 5Mbps, so the SPI speed should be less than this.
* The pins required for establishing connection to master in full duplex mode would be GND, VCC, CS (Chip Select), SDO (Serial Data Out which is to be connected to MISO pin in master), SDA (Serial Data In in case of 4-wire SPI mode which is to be connected to MOSI pin in master) and SCL (Serial Clock or SCK which is to be connected to SCK pin in master).
* If half duplex mode is being used, then VCC, GND, CS, SCL pins would be used same as above and SDA pin would act as SDIO (Serial Data Input Output) pin in SPI connection.
* The CS pin should be low before starting any read/write operation and it should be high after completing the corresponding operations.
* The clock polarity of the device is HIGH/1 and the clock phase is Falling Edge/HIGH/2 edge/1.
* According to the timing diagram, in order to read/write multiple bits in same operation, the 6th bit should be set. That’s why in the code, the register addresses are OR’d with 0x40, which sets the 6th bit high.
* Also according to the timing diagram, the MSB defines whether it is a read or write operation. During write operation, the MSB should be 0 and for read operation, it should be HIGH. That’s why for read operations, the register addresses are OR’d with 0x80, setting the MSB.



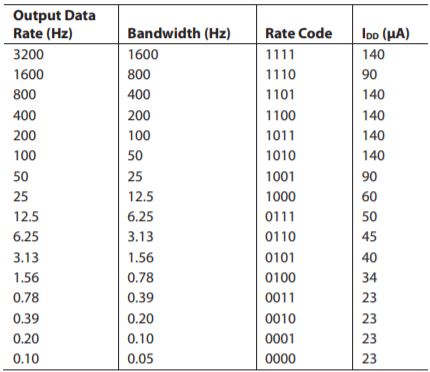
* Now, the appropriate registers in the sensor should be configured, so that it can be used in data transfer. First, the device should be activated. So, the measure bit (D3) should be set to high in order to start the measurement. So, 0x08 should be written into register 0x2D to start the device.



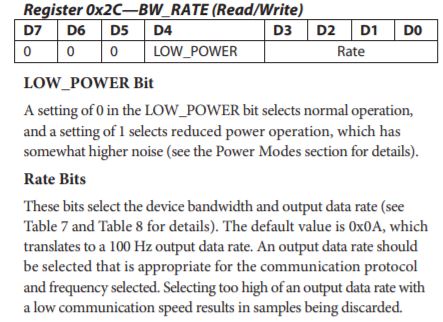
* Next, range of measurement has to be selected. Writing 01 in the LSB sets it to +/-4g. Also, the interrupt configuration by default is active high. So, setting the bit D5 makes it active low. So, if interrupt is needed in active low, 0x21 has to be written in register 0x31.



* Now, sampling rate has to be set.



* The register corresponding to sampling rate is 0x2C. Bits D0 to D3 correspond to sampling rate. So, from the above table taken from the datasheet, we can set the lower nibble so as to get the required sampling rate.

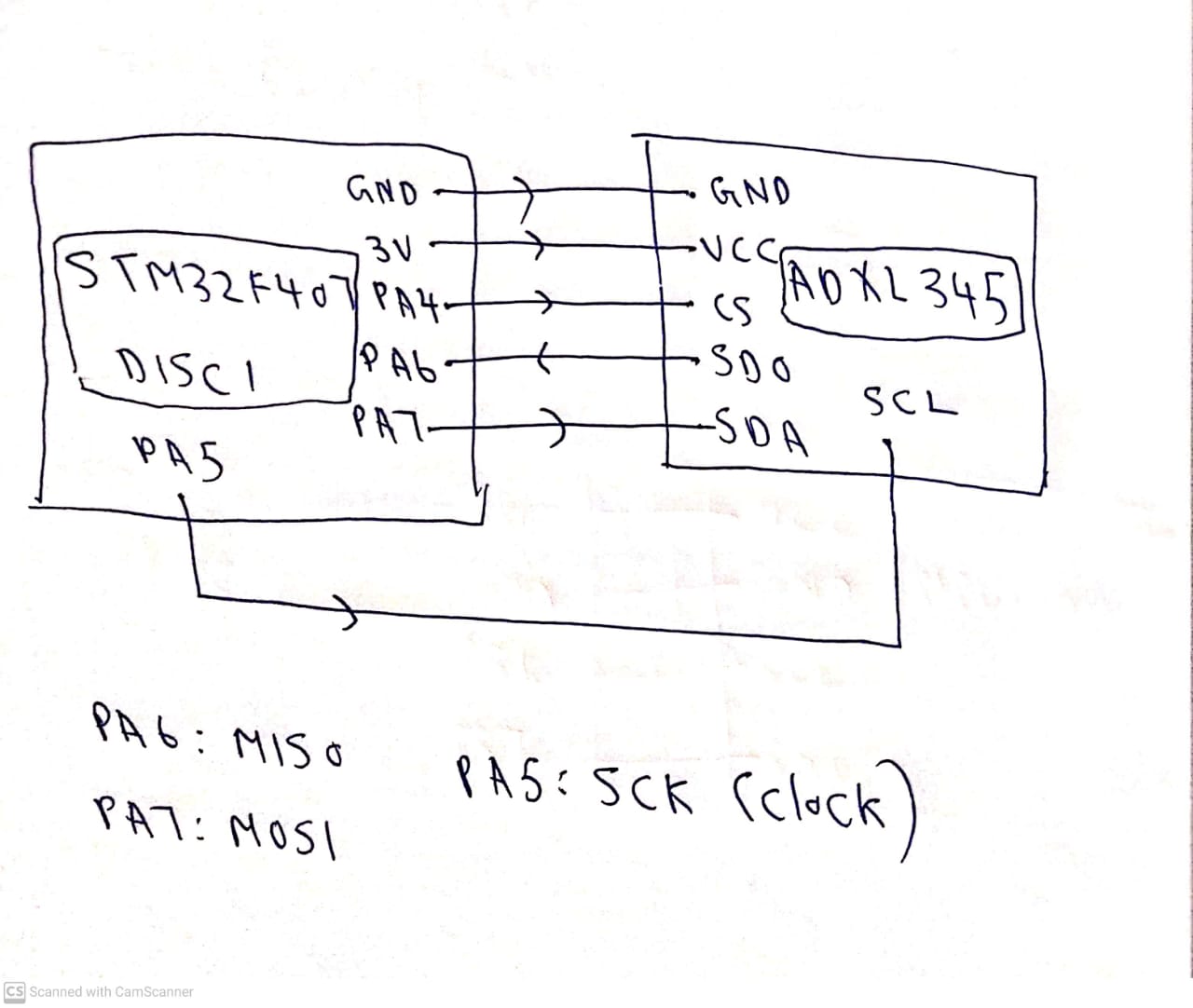


**Microcontroller part**



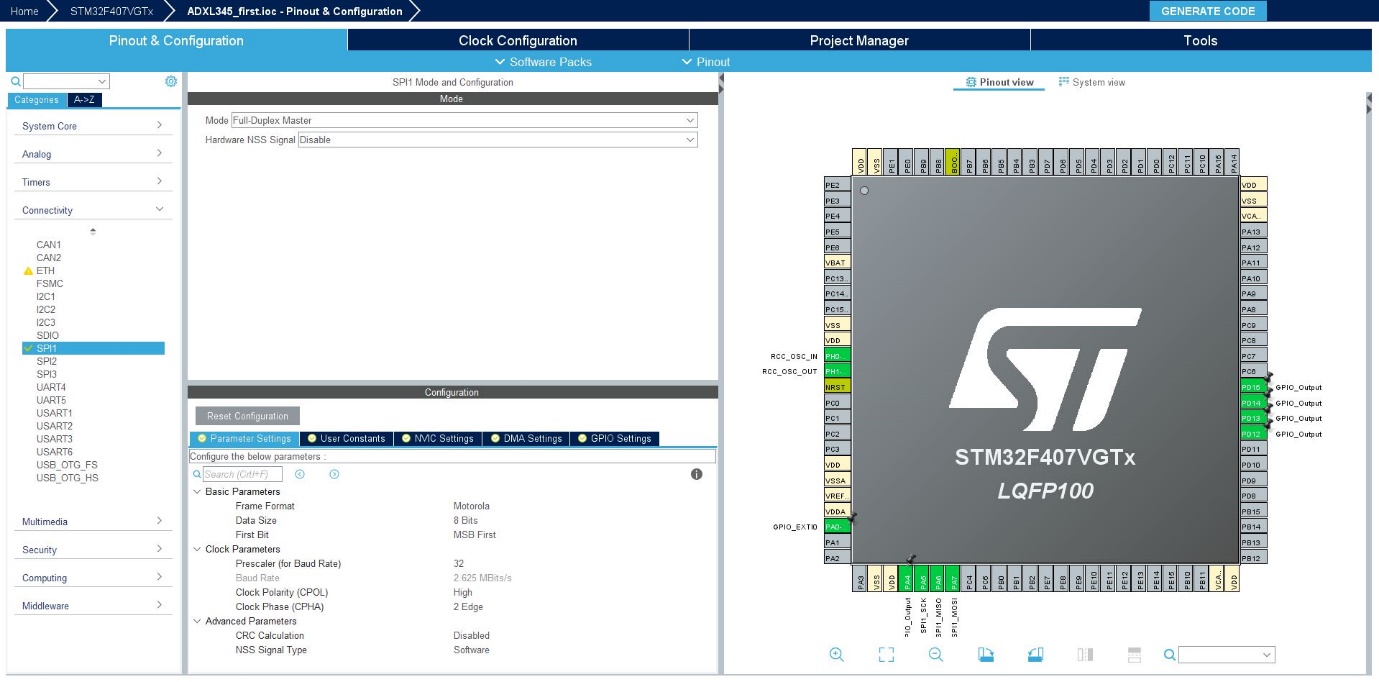
* In stm32f407vgtx, pin PA5 corresponds to SCK, PA6 to MISO and PA7 to MOSI. Any other pin can be configured as chip select (just use it as a GPIO pin as no special function is needed for CS pin).
* In the code, PA4 is configured as an output pin and is used for chip select. If interrupt is required, PA0 can be configured as external interrupt 0.
* Built in LEDs PD12 to PD15 can be configured and used if needed.

**Connections**

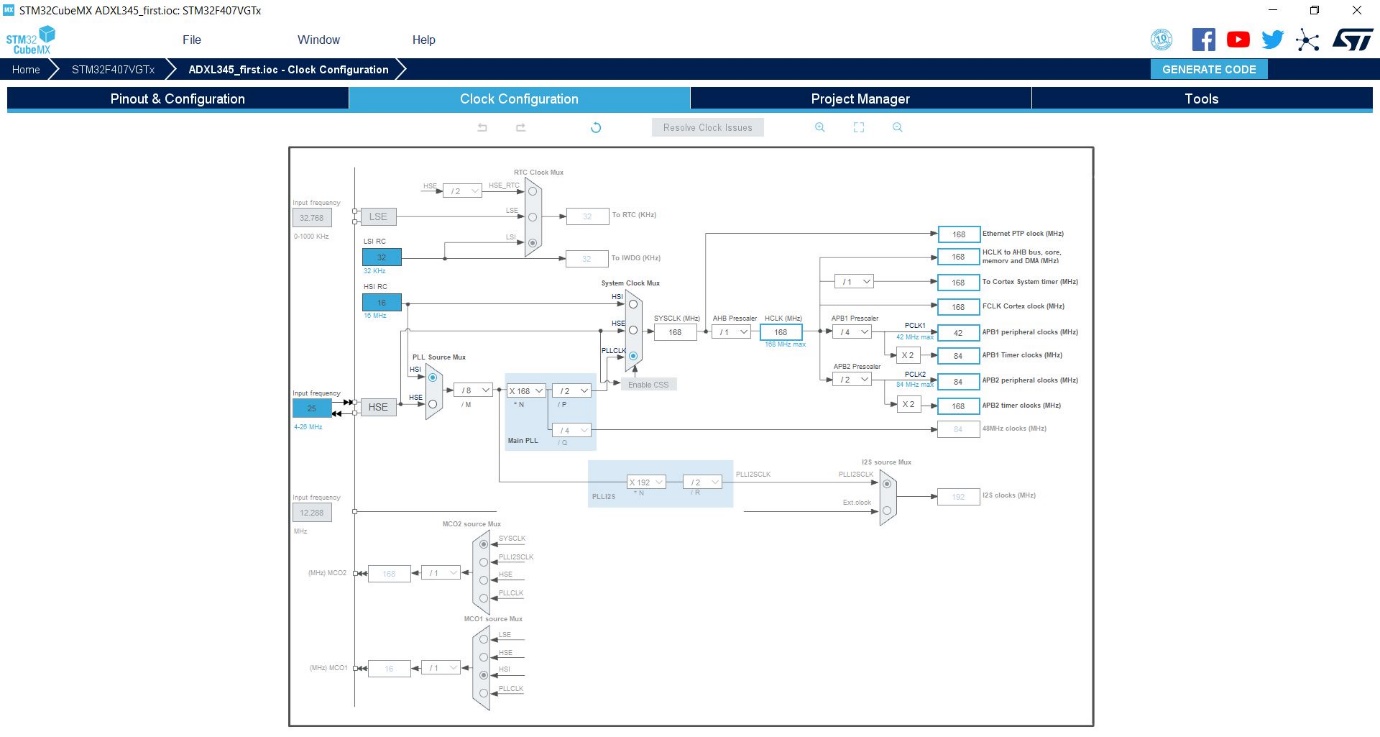


**CubeMX Part**

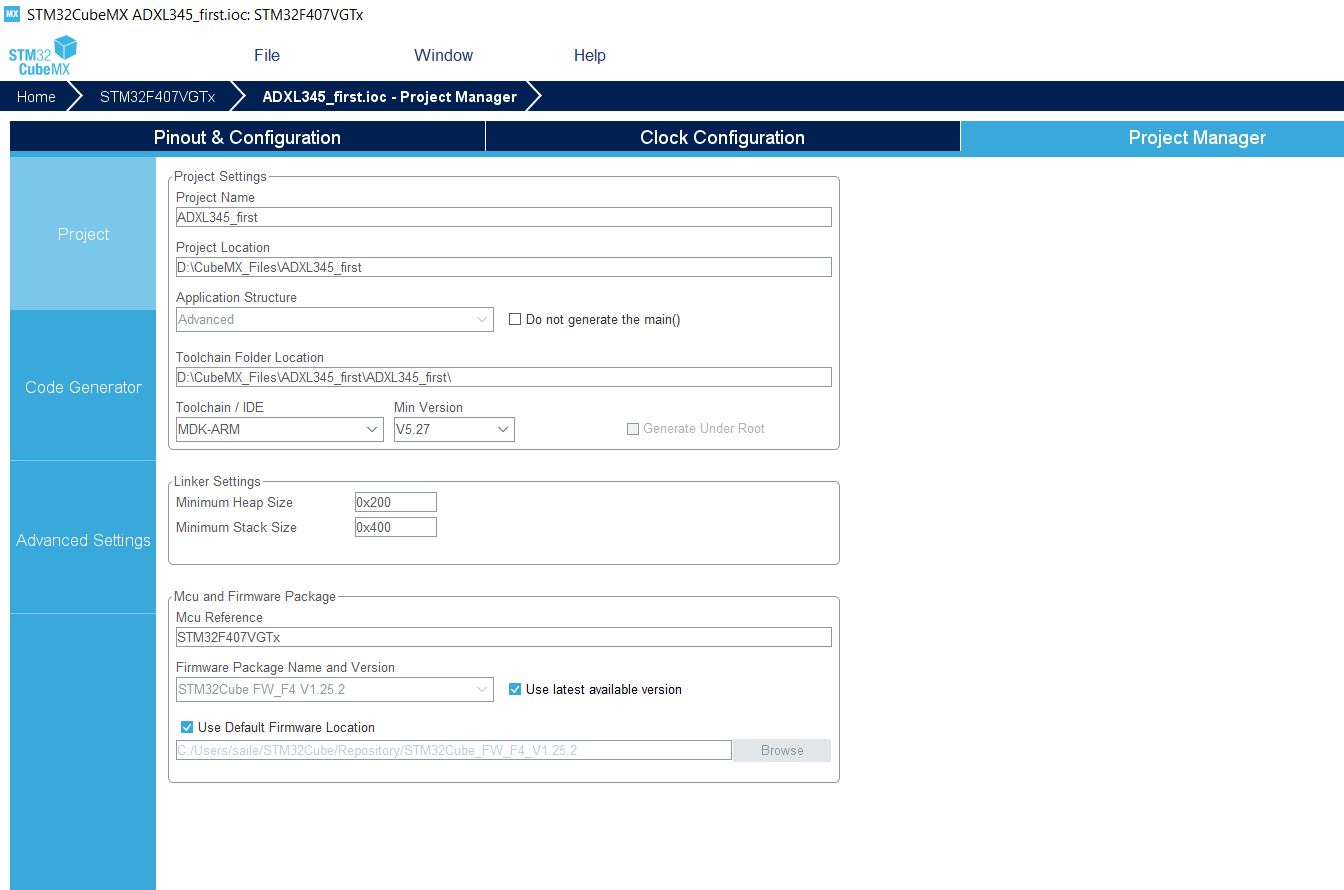
* After selecting the right board in CubeMX, go to connectivity, select SPI1 and enable it in full duplex mode. Next, set the value of prescaler so that the baud rate comes less than 5Mbps. As explained before, ADXL345 has a high clock polarity and clock phase. So set the clock polarity to 1 and clock phase to 2 edge.



* Next, go to system core, select RCC, and enable high speed crystal/ceramic resonator.
* Select NVIC and enable SPI interrupt and external interrupt lines if necessary.
* Go to clock configuration, and set the value of HCLK to 168MHz.



* Next, select project manager and give appropriate project name and project location and choose ARM-MDK as tool IDE if keil is being used as IDE, else choose the appropriate tool.



**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 \*/

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

SPI\_HandleTypeDef hspi1;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

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

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_SPI1\_Init(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

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

/\* USER CODE BEGIN 0 \*/

uint8\_t spibuf[10];

int8\_t x,y,z;

float xacc,yacc,zacc;

void spiwrite(uint8\_t address, uint8\_t value)

{

uint8\_t data[2];

data[0] = address|0x40; //Or with 0x40 to do write/read multiple bits

data[1] = value;

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_RESET); // make chip select low to enable transmission

HAL\_SPI\_Transmit(&hspi1, data, 2, 10); // write data to register

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_SET); // make chip select high at the end of transmission

}

void spiread(uint8\_t address)

{

address |= 0x80; // //Or with 0x80 for read operations

address |= 0x40;

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_RESET);

HAL\_SPI\_Transmit(&hspi1, &address, 1, 1); // send address

HAL\_SPI\_Receive(&hspi1, &spibuf[0], 6, 1); // read the data from device

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_SET);

}

/\* 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\_SPI1\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_GPIO\_TogglePin(GPIOD,GPIO\_PIN\_12);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_4,GPIO\_PIN\_SET);

HAL\_Delay(10);

spiwrite(0x2D,0x00);// Reset before configuring power register

spiwrite(0x2D,0x08);//Configure the power register, and turn on the device

// Configuring the data format register

//The 5th bit corresponds to setting interrupt to active low if set

spiwrite(0x31,0x21);//Lower nibble:Selecting +/- 4g range and 4 wire spi mode

//Configuring sampling rate to 100hz

spiwrite(0x2C,0x0A);

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

spiread(0x32);// Register address corresponding to required data

x=((spibuf[1]<<8)|spibuf[0]);

y=((spibuf[3]<<8)|spibuf[2]);

z=((spibuf[5]<<8)|spibuf[4]);

xacc=x\*0.0078;

yacc=y\*0.0078;

zacc=z\*0.0078;

/\* 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 SPI1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_SPI1\_Init(void)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_HIGH;

hspi1.Init.CLKPhase = SPI\_PHASE\_2EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_32;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 10;

if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_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();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15, 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 pin : PA4 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_4;

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

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

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

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

/\*Configure GPIO pins : PD12 PD13 PD14 PD15 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15;

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 \*/

/\* 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\*\*\*\*/