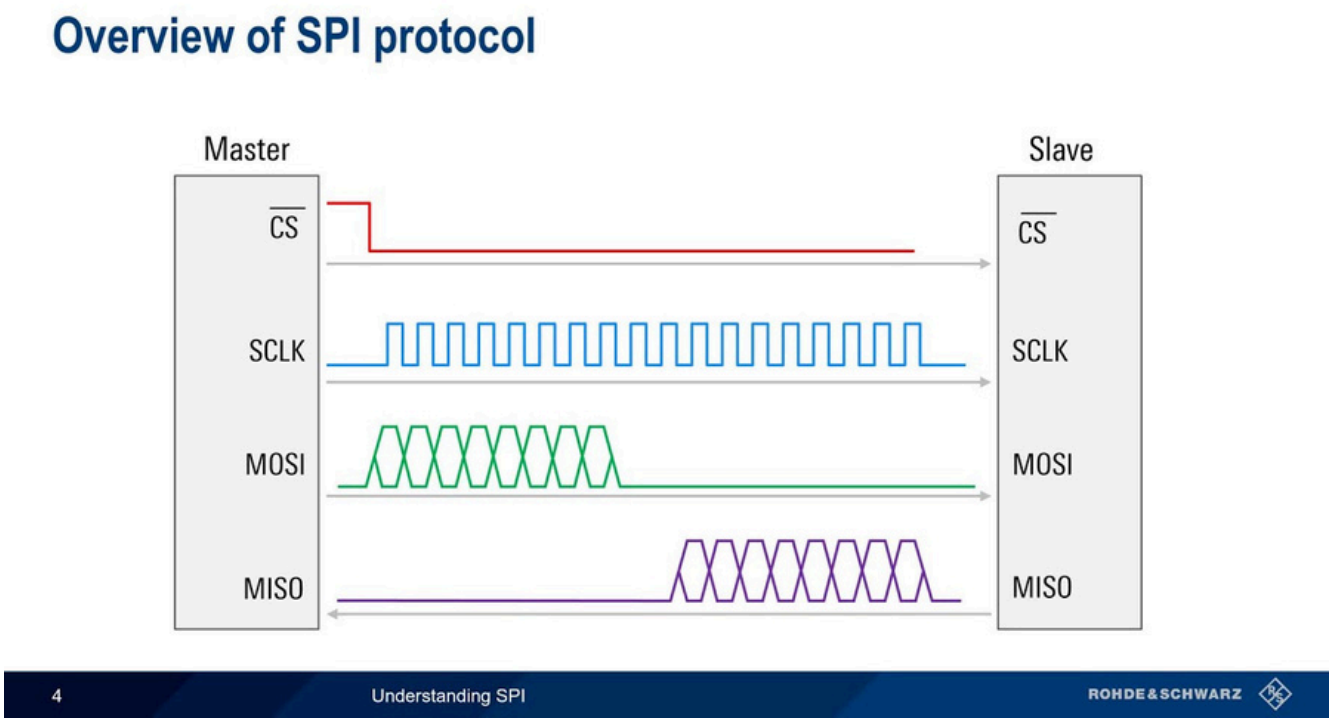
project report

bouchemla cherif osmane

stm32 spi communication with built in accelerometer sensor LIS3DSH

UNIVERSITÉ ÉVRY
PARIS-SACLAY

1- spi protocol :



spi communication happens between a master (mcu) and a slave (sensor) the master is the only one who can start the conversation by setting the cs to low, and ends it by setting the cs to high, both the master and the slave must be synchronized (have the same clock)
the master must send a command first to the slave, and the command is coded in a byte (r or w)&addr
if the msb is 1, then its a read command, if its 0 then its a write command, note that the address of each register in the sensor has a 7 bit address, most standard sensors are like that

- spi sensors mostly have 7 bit address bus for their registers
- spi has 4 wires between the master and the slave
  - cs (active low)
  - clk
  - miso
  - mosi
- the slave cant start the conversation, the master must set cs to low and start the clk to start data transfer
- spi is faster than i2c and uart
- because its full duplex, after the first command sent, the sensor sends garbage data because it takes a cycle to process what the cpu wants
- if we are using dma, spirxbuff[0] will be 00 or FF(dummy data)

2- LIS3DSH:

- its a black chip on the stm32 that measures linear acceleration
- how it works is it outputs a 16 bit signed integer value so the range is -32,768 to +32,767
- if we leave it in the 2G scale, every integer step is equal to 0.06 mg/digit (mg = mili gravity)
- to use it we need to configure the registers write and read from the registers inside it :
  - who_am_i register: Address `0x0F`, id of the sensor, it is burned into the silicon of the sensor so we cant change, we need it to confirm if the sensor is connected or not
  - ctrl4 : we set this to 01101111, according to the datasheet, we set the odr(outputdatarate) to 100hz, good enough for the computer, enable x y and z axes reading, and bdu to 1 so the values arent updated unless the data has been read

CTRL_REG4 (20h)

Control register 4.

Table 53. Control register 4

ODR3	ODR2	ODR1	ODR0	BDU	Zen	Yen	Xen

Table 54. CTRL_REG4 register description

ODR 3:0	Output data rate and power mode selection. Default value: 0000 (see *Table 55*)
BDU	Block data update. Default value: 0 (0: continuous update; 1: output registers not updated until MSB and LSB have been read)
Zen	Z-axis enable. Default value: 1 (0: Z-axis disabled; 1: Z-axis enabled)
Yen	Y-axis enable. Default value: 1 (0: Y-axis disabled; 1: Y-axis enabled)
Xen	X-axis enable. Default value: 1 (0: X-axis disabled; 1: X-axis enabled)

- Data Registers (`0x28` - `0x2D`) xlow to zhigh
- cntrl_reg3 :
- we set it to C8 meaning, int1 active high data ready connected to int1 no vector filter and no latching

CTRL_REG3 (23h)

Control register 3.

Table 60. Control register 3

DR_EN	IEA	IEL	INT2_EN	INT1_EN	VFILT	-	STRT

Table 61. CTRL_REG3 register description

DR_EN	DRDY signal enable to INT1. Default value: 0 (0: data ready signal not connected; 1: data ready signal connected to INT1)
IEA	Interrupt signal polarity. Default value: 0 (0: interrupt signals active LOW; 1: interrupt signals active HIGH)
IEL	Interrupt signal latching. Default value: 0 (0: interrupt signal latched; 1: interrupt signal pulsed)
INT2_EN	Interrupt 2 enable/disable. Default value:0 (0: INT2 signal disabled; 1: INT2 signal enabled)
INT1_EN	Interrupt 2 enable/disable. Default value: 0 (0: INT1/DRDY signal disabled; 1: INT1/DRDY signal enabled)
VFILT	Vector filter enable/disable. Default value: 0 (0: vector filter disabled; 1: vector filter enabled)
STRT	Soft reset bit (0: no soft reset; 1: soft reset (POR function)

- when the sensor is done preparing the data is sends an interrupt which is connected to pin PE0 on the board
we set the data transfer rate to 100 hz meaning it will send an interrupt every 10ms

method 1 : polling

the most basic way to use the sensor, cons of this method is evry time the processor wants to read it will sleep for 10 ms, so to read from the sensor itll be over 10ms

this is just simple spi communication, i have also made the pc receive the data through uart serial communication

sensor source file

```
// --- Basic Read/Write Functions ---
void LIS3DSH_Write(uint8_t reg, uint8_t data) {
    uint8_t txData[2] = {reg | WRITE_CMD, data};
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET); // CS Low
    HAL_SPI_Transmit(&hspi1, txData, 2, 10);
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);   // CS High
}

uint8_t LIS3DSH_Read(uint8_t reg) {
    uint8_t txData = reg | READ_CMD;
    uint8_t rxData = 0;
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET); // CS Low
    HAL_SPI_Transmit(&hspi1, &txData, 1, 10);
    HAL_SPI_Receive(&hspi1, &rxData, 1, 10);
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);   // CS High
    return rxData;
}
```

```c
// Initialize the sensor (Reset, Configure, Verify)
void LIS3DSH_Init(void) {
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET); // Safety set CS High
    HAL_Delay(10);

    // 1. Force Reset (Write 0x00)
    LIS3DSH_Write(CTRL_REG4_ADDR, 0x00);
    HAL_Delay(50);

    // 2. Enable Sensor (100Hz, X Y Z enabled)
    LIS3DSH_Write(CTRL_REG4_ADDR, 0x6F);
    HAL_Delay(50);

    // 3. Optional: Verify (Check if it stuck)
    if (LIS3DSH_Read(CTRL_REG4_ADDR) != 0x6F) {
        // Retry once if failed
        LIS3DSH_Write(CTRL_REG4_ADDR, 0x6F);
    }
}
```

```c
// Read raw data and convert to G-force
void LIS3DSH_Update_Acceleration(void) {
    // Read raw 16-bit values
    int16_t x_raw = (LIS3DSH_Read(0x29) << 8) | LIS3DSH_Read(0x28);
    int16_t y_raw = (LIS3DSH_Read(0x2B) << 8) | LIS3DSH_Read(0x2A);
    int16_t z_raw = (LIS3DSH_Read(0x2D) << 8) | LIS3DSH_Read(0x2C);

    // Convert to G-force
    xg = x_raw * 0.06f / 1000.0f;
    yg = y_raw * 0.06f / 1000.0f;
    zg = z_raw * 0.06f / 1000.0f;
}
```

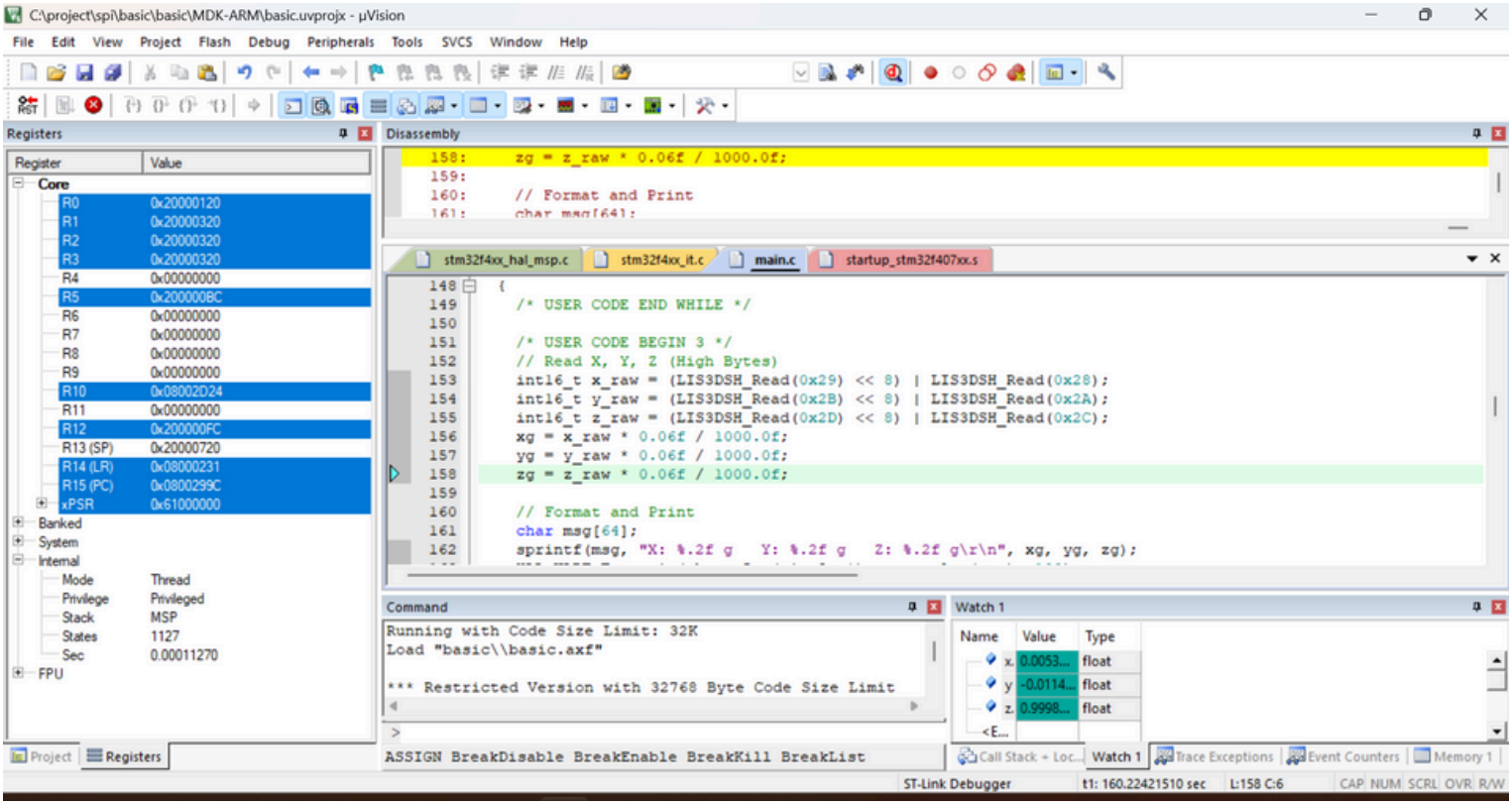
uart sourse file

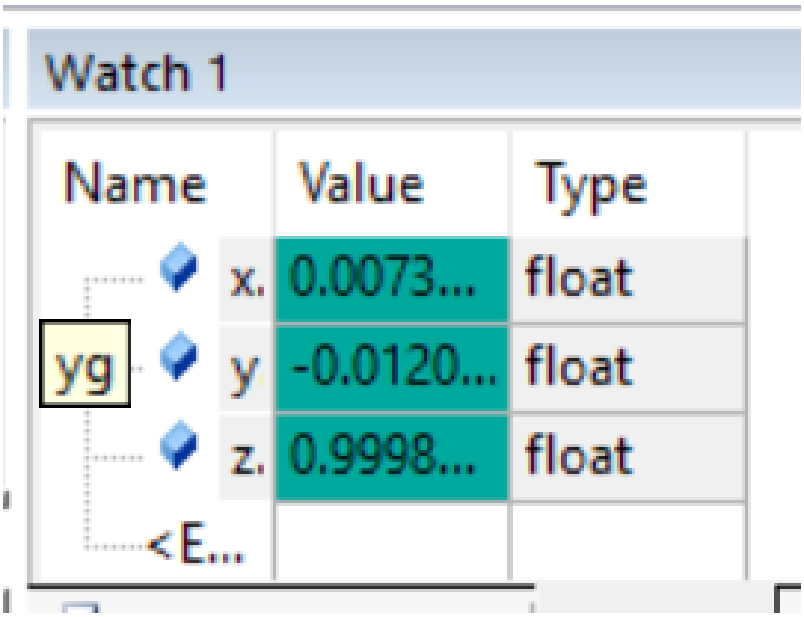
```c
extern UART_HandleTypeDef huart5;

void MyUART_PrintAccel(float x_val, float y_val, float z_val) {
    char msg[100];

    // Format the text
    sprintf(msg, "X: %5.2f g   Y: %5.2f g   Z: %5.2f g\r\n", x_val, y_val, z_val);

    // Use &huart5 directly!
    HAL_UART_Transmit(&huart5, (uint8_t*)msg, strlen(msg), 100);
}
```

the main

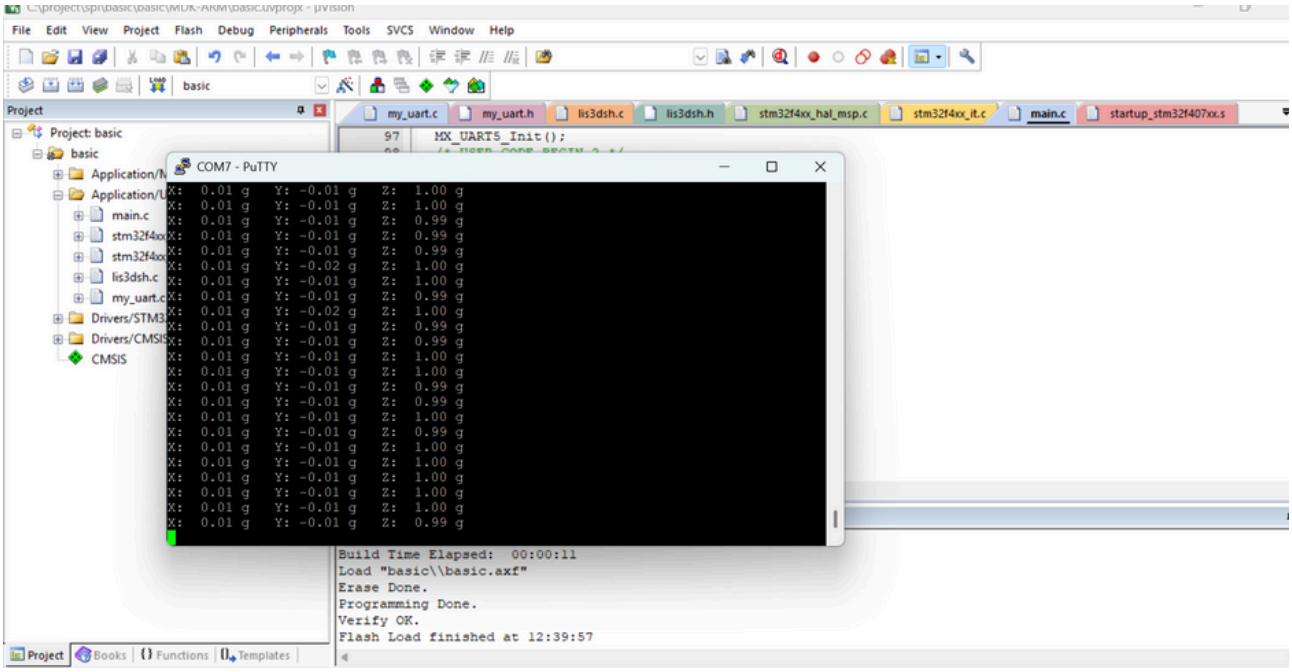
```c
  /* USER CODE BEGIN 2 */
  LIS3DSH_Init();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    LIS3DSH_Update_Acceleration();

    MyUART_PrintAccel(xg, yg, zg);

    HAL_Delay(100);
  }
  /* USER CODE END 3 */
}
```

using the debug mode in keil we can monitor our variables

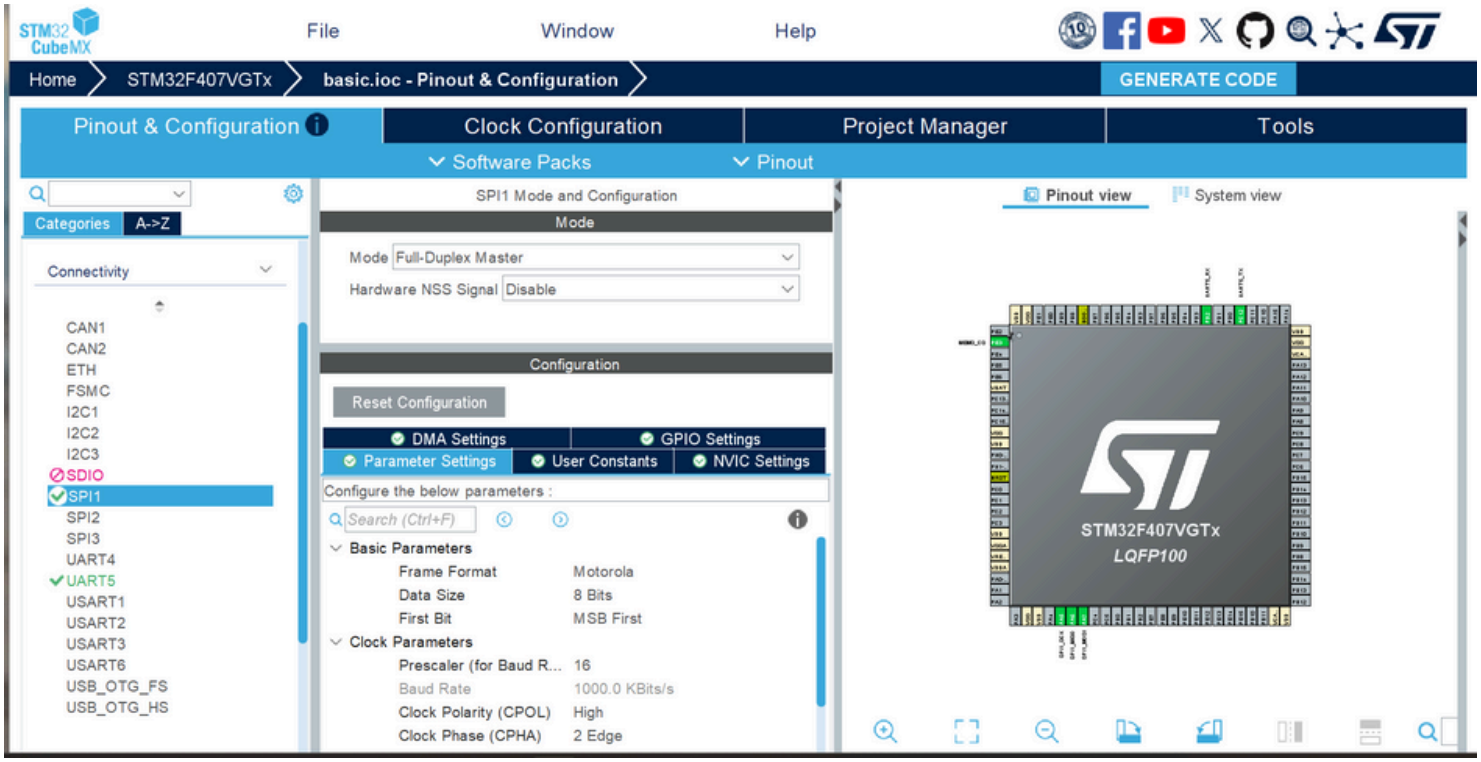




uart communication



cube mx setup



method2 : interrupts

the problem with the first method is the cpu has to wait for the sensor to respond after sending the read command, so its just sleeping for 10ms for no reason, we can fix this by making the cpu read the data every time the sensor sends an interrupt and tells it its ready

in the previous method we used a delay so the cpu can guess when the data was ready, the cpu couldve read the same data twice or takes too long to read and misses new samples

by using an the interrup signal the cpu and sensor are synchronized

new interrupt functions in the sensor library

```
// Called by HAL_GPIO_EXTI_Callback in main.c
void LIS3DSH_DataReady_Callback(void) {
    internal_drdy_flag = 1;
}

// Called by main while(1) loop
// Returns 1 if data is ready, 0 if not. Automatically clears flag.
uint8_t LIS3DSH_IsDataReady(void) {
    if (internal_drdy_flag == 1) {
        internal_drdy_flag = 0; // Reset flag immediately
        return 1; // True
    }
    return 0; // False
}
```

```
LIS3DSH_Write(CTRL_REG3_ADDR, 0xC8);
```

new initilization

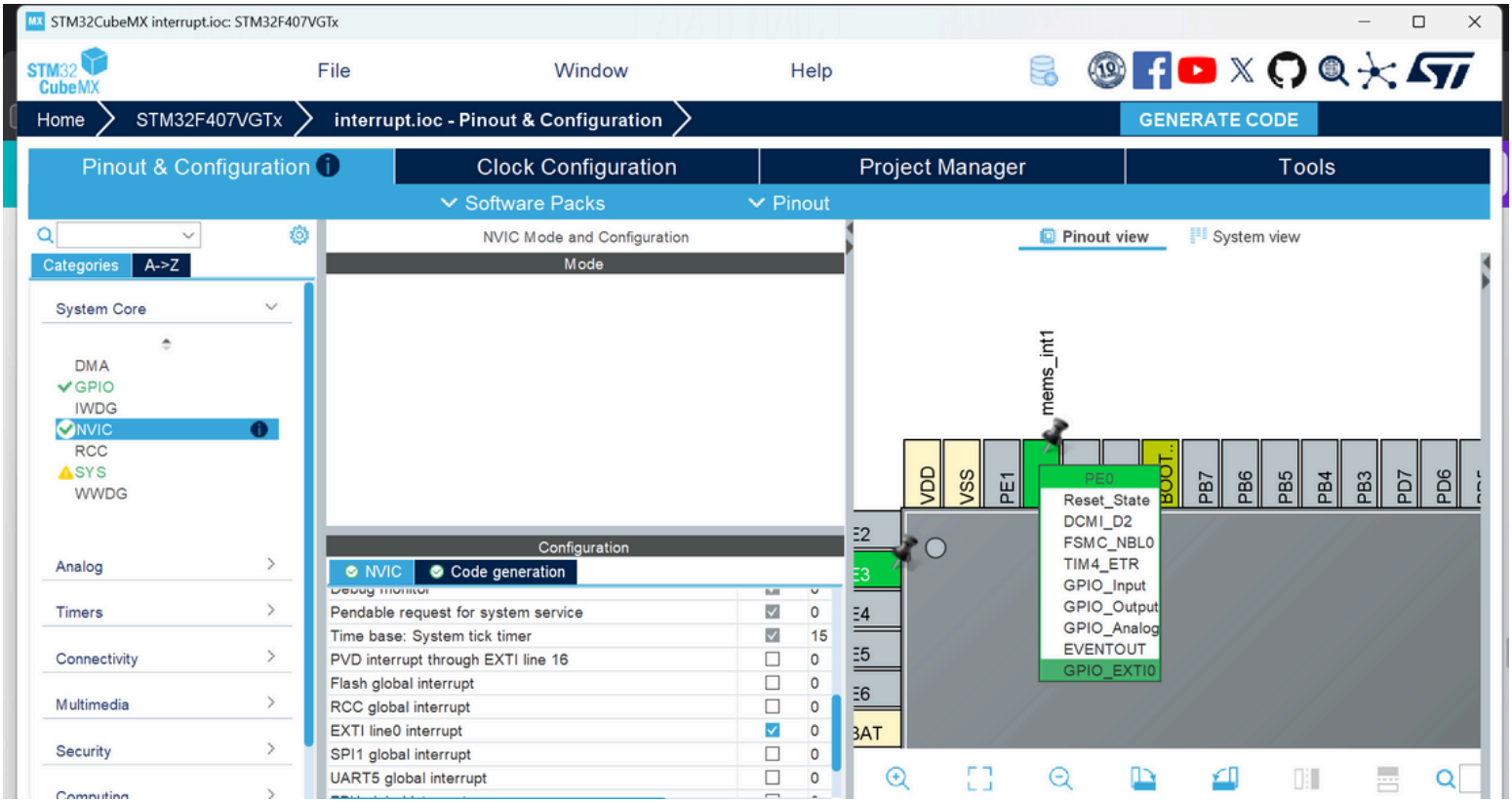
interrupt in the main

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin == mems_int1_Pin) {
        LIS3DSH_DataReady_Callback(); // Forward signal to driver
    }
}
```

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  // Only read if the flag is raised
  if (LIS3DSH_IsDataReady())
  {
      LIS3DSH_Update_Acceleration();
      MyUART_PrintAccel(xg, yg, zg);
  }


}
/* USER CODE END 3 */
/* USER CODE END 3 */
```

observation

data is being read faster



cubemx setup

method 3 : dma

dma :

- dmas are mini processors whos job is simply to do data transfer between peripherals and memory
- the cpu is the main processing unit and we should offload some of its tasks (like spi communication or any data transfer between perifpherals) so it can focus on more important tasks, so we make the dmas read data from the sensor and when it finished it writes it to the memory
- whent the dma is done trasfering it sends an interrupt signal to the cpu telling it its done, and the cpu just reads from the memory location the dma was programmed to write to
- when using dma, we set up a memory section to data transfer spitxbuff[] and spirxbuff[]
- txbuff contains the data that will be sent to the sensor sequencially
- rx buff contains the data that will be recieved by the sensor sequencialy

- recieving and transmiting is done at the same time
- when we call the function in our program we tell the dma where the tx and rx start and how long they are
- when the dma is done with the transfer of tx and rx, it sends and interrupt to the cpu

new dma functions

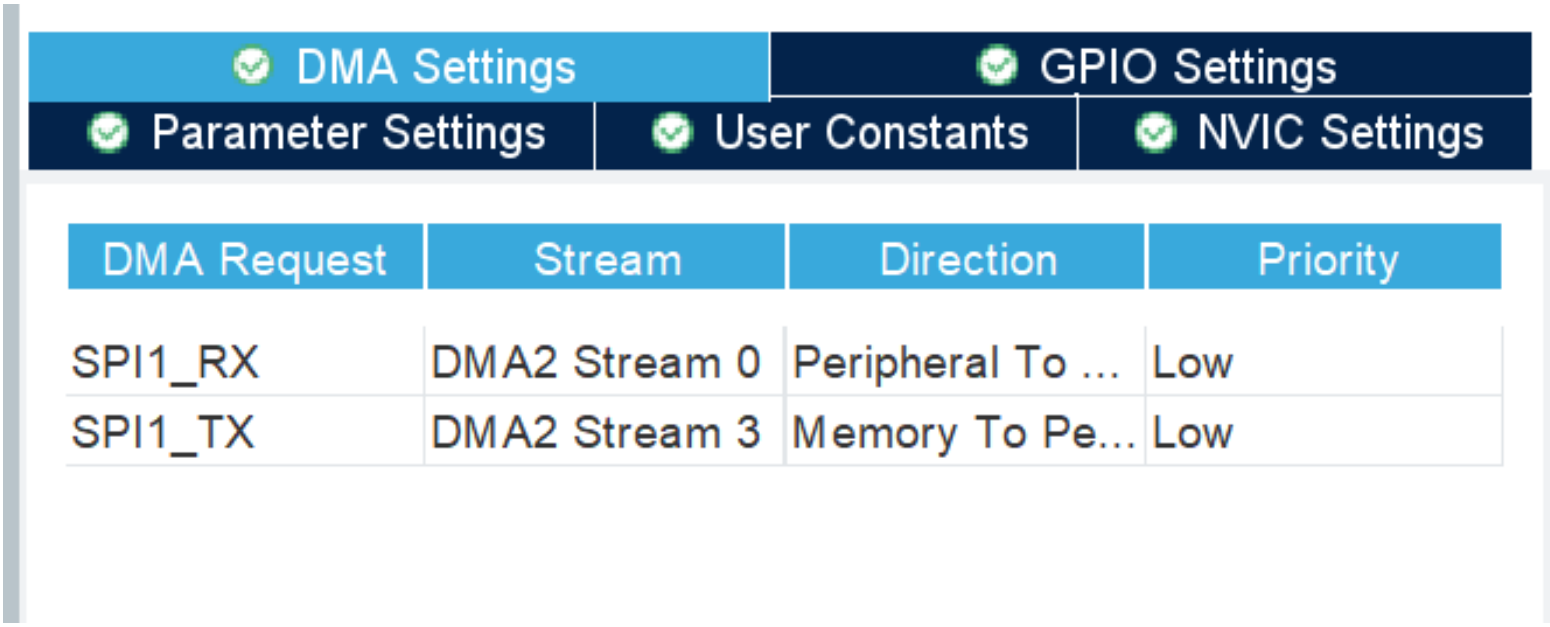
```
void LIS3DSH_Start_DMA_Reading(void) {

    // 1. Prepare the Command: Read (0x80) starting at OUT_X_L (0x28)
    // The Auto-increment feature is on by default, so it will read 0x28 to 0x2D
    spiTxBuf[0] = 0x28 | 0x80;

    // 2. Pull CS LOW to start the conversation
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);

    // 3. Launch DMA!
    // We send 7 bytes (1 Command + 6 Dummy)
    // We receive 7 bytes (1 Garbage + 6 Data)
    HAL_SPI_TransmitReceive_DMA(&hspi1, spiTxBuf, spiRxBuf, 7);
}
```

```c
// --- NEW: FINISH THE DMA TRANSFER ---
// This is called when the DMA shouts "I'm done!" (DMA Callback)
void LIS3DSH_DMA_Complete_Callback(void) {

    // 1. Pull CS HIGH to end the conversation
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);

    // 2. Raise the flag for the Main Loop
    dma_data_ready = 1;
}
```

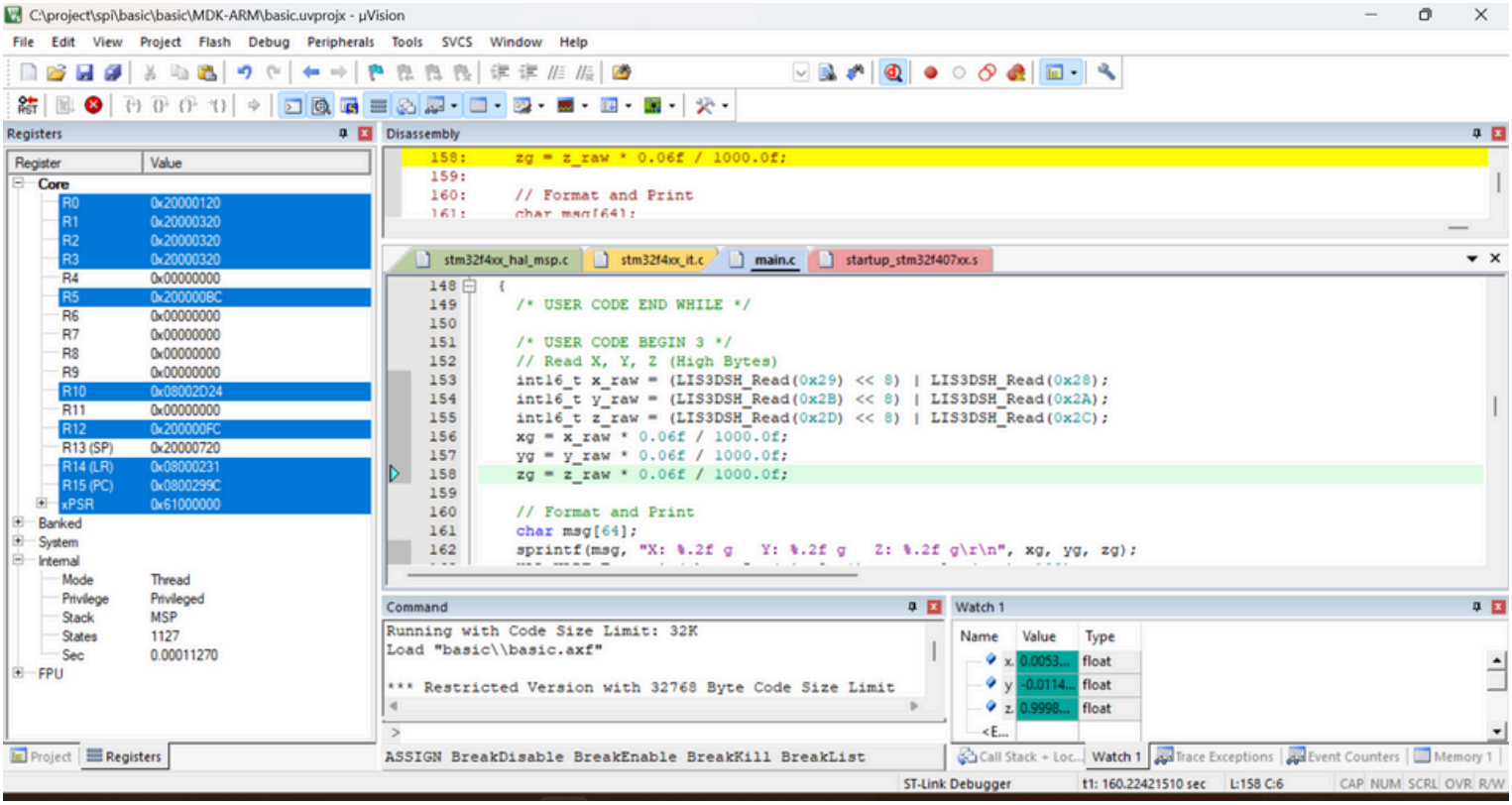
```c
// --- CHECKER ---
uint8_t LIS3DSH_IsDataReady(void) {
    if (dma_data_ready == 1) {
        dma_data_ready = 0;
        return 1;
    }
    return 0;
}
```

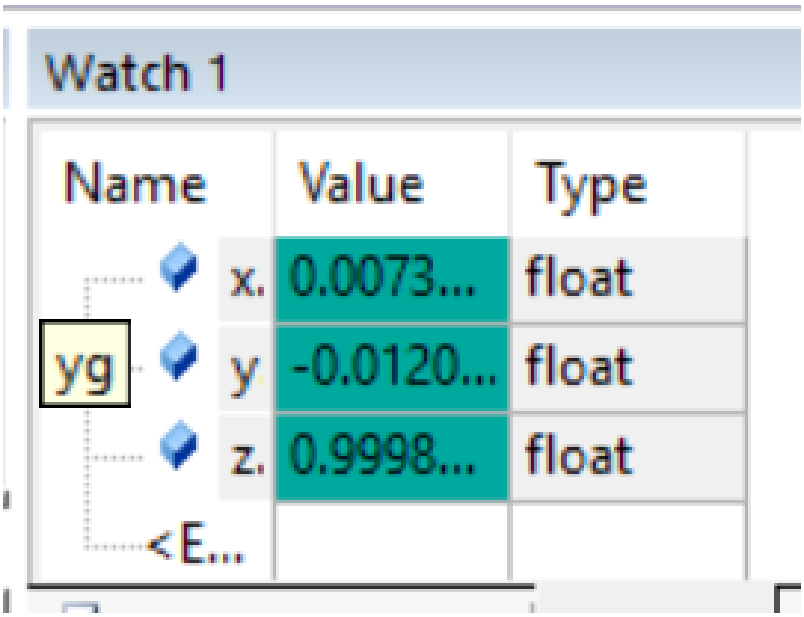
new interrupts in the main

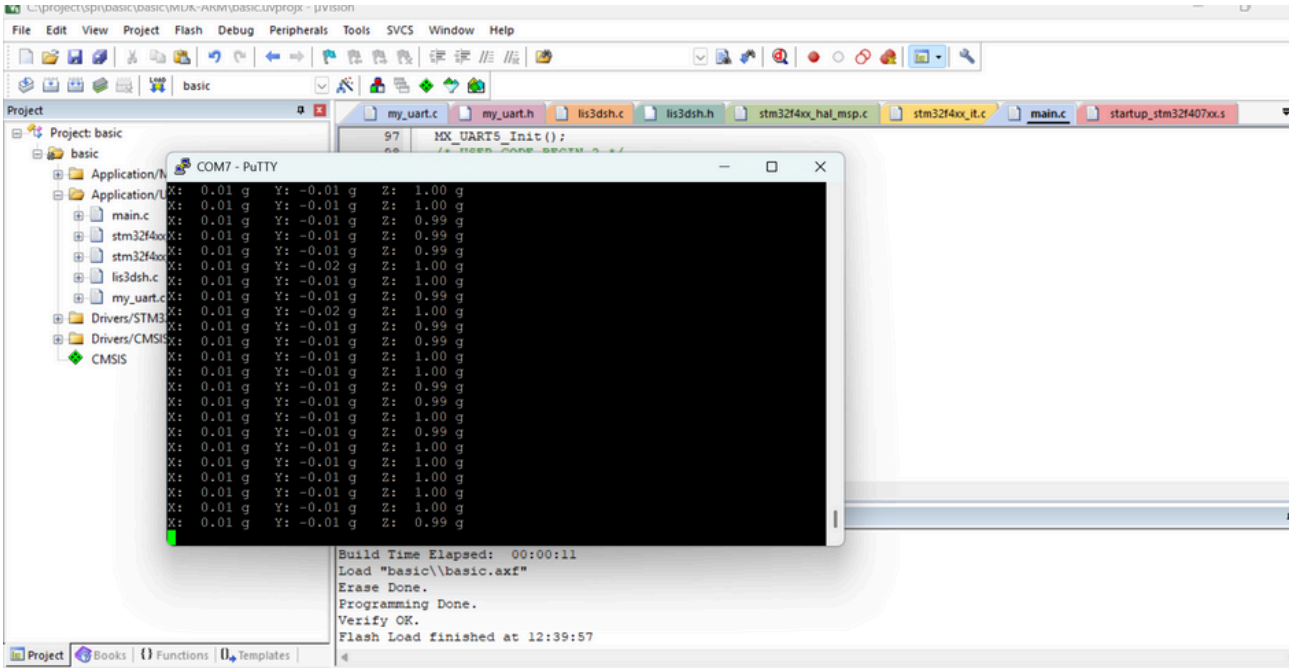
```c
72  // 1. SENSOR INTERRUPT (Starts the Chain)
73  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
74  {
75      if(GPIO_Pin == GPIO_PIN_0) {
76          LIS3DSH_Start_DMA_Reading();
77      }
78  }
79
80  // 2. DMA INTERRUPT (Ends the Chain)
81  // This function is called automatically by HAL when DMA finishes
82  void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)
83  {
84      if(hspi->Instance == SPI1) {
85          LIS3DSH_DMA_Complete_Callback();
86      }   void LIS3DSH_DMA_Complete_Callback()
87  }
88  /* USER CODE END 0 */
```



cubemx setup


C:\project\spi\basic\basic\MDK-ARM\basic.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Core
R0 0x20000120
R1 0x20000320
R2 0x20000320
R3 0x20000320
R4 0x00000000
R5 0x200000BC
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x08002D24
R11 0x00000000
R12 0x200000FC
R13 (SP) 0x20000720
R14 (LR) 0x08000231
R15 (PC) 0x0800299C
xPSR 0x61000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 1127
Sec 0.00011270
FPU
Disassembly
158: zg = z_raw * 0.06f / 1000.0f;
159:
160: // Format and Print
stm32f4xx_hal_msp.c
stm32f4xx_it.c
main.c
startup_stm32f407xx.s
148 {
149 /* USER CODE END WHILE */
150
151 /* USER CODE BEGIN 3 */
152 // Read X, Y, Z (High Bytes)
153 int16_t x_raw = (LIS3DSH_Read(0x29) << 8) | LIS3DSH_Read(0x28);
154 int16_t y_raw = (LIS3DSH_Read(0x2B) << 8) | LIS3DSH_Read(0x2A);
155 int16_t z_raw = (LIS3DSH_Read(0x2D) << 8) | LIS3DSH_Read(0x2C);
156 xg = x_raw * 0.06f / 1000.0f;
157 yg = y_raw * 0.06f / 1000.0f;
158 zg = z_raw * 0.06f / 1000.0f;
159
160 // Format and Print
161 char msg[64];
162 sprintf(msg, "X: %.2f g Y: %.2f g Z: %.2f g\r\n", xg, yg, zg);
Command
Running with Code Size Limit: 32K
Load "basic\\basic.axf"
*** Restricted Version with 32768 Byte Code Size Limit
ASSIGN BreakDisable BreakEnable BreakKill BreakList
Watch 1
Name Value Type
x. 0.0053... float
y -0.0114... float
z. 0.9998... float
<E...
Project Registers
Call Stack + Loc... Watch 1 Trace Exceptions Event Counters Memory 1
ST-Link Debugger t1: 160.22421510 sec L:158 C:6 CAP NUM SCRL OVR R/W


Watch 1
Name
Value
Type
x.
0.0073...
float
yg
y
-0.0120...
float
z.
0.9998...
float
<E...


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
basic
Project
Project: basic
basic
main.c
lis3dsh.c
my_uart.c
CMSIS
my_uart.c
my_uart.h
lis3dsh.c
lis3dsh.h
stm32f4xx_hal_msp.c
stm32f4xx_it.c
main.c
startup_stm32f407xx.s
97 MX_UART5_Init();
COM7 - PuTTY
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.02 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.02 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 1.00 g
X: 0.01 g Y: -0.01 g Z: 0.99 g
Build Time Elapsed: 00:00:11
Load "basic\\basic.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 12:39:57
Project
Books
Functions
Templates