Answer for question in slide 32

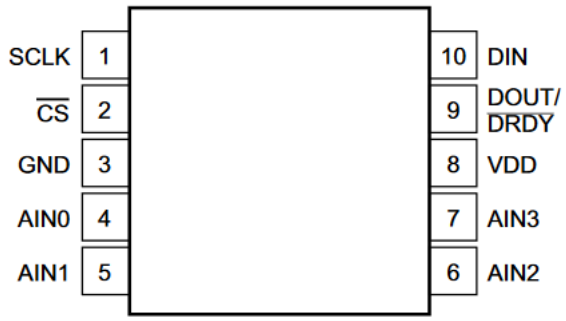
Design a system based on STM32F407VG MCU to connect a sensor that require ADC resolution of 16 bits. Th designers are suggested to use TI's ADS1118 ADC which provide SPI compatible interface. Read the data sheet to understand the scheme of interfacing. Use STM32CubeMx to generate HAL for SPI. Develop functions to read digital value from ADS118. Write application code to test the same.

ADS1118 ADC Features:

The ADS1118 is a precision, low power, 16-bit1 analog-to-digital converter (ADC).

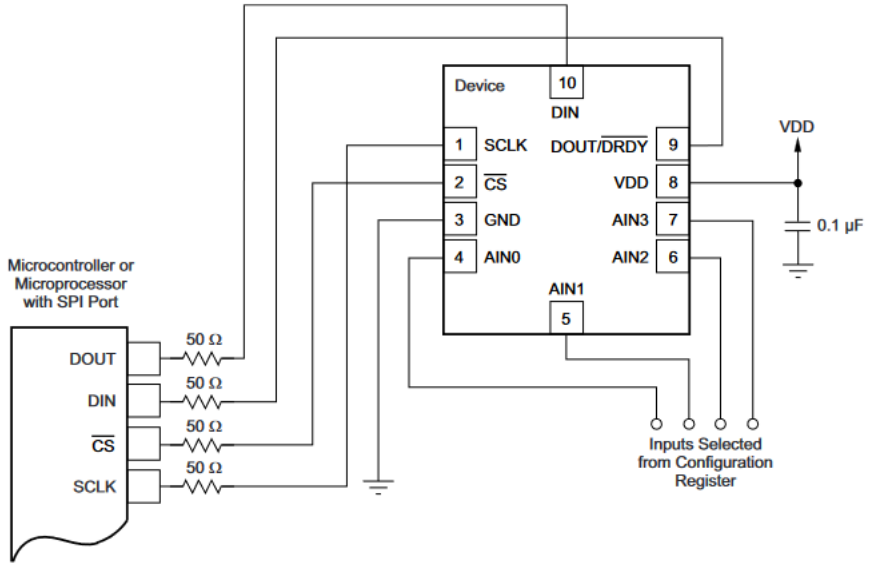
Data are transferred through a serial peripheral interface (SPI).

Pin diagram:



PIN		TYPE	DESCRIPTION
NO.	NAME		
1	SCLK	Digital input	Serial clock input
2	$\overline{CS}$	Digital input	Chip select; active low. Connect to GND if not used.
3	GND	Supply	Ground
4	AIN0	Analog input	Analog input 0. Leave unconnected or tie to VDD if not used.
5	AIN1	Analog input	Analog input 1. Leave unconnected or tie to VDD if not used.
6	AIN2	Analog input	Analog input 2. Leave unconnected or tie to VDD if not used.
7	AIN3	Analog input	Analog input 3. Leave unconnected or tie to VDD if not used.
8	VDD	Supply	Power supply. Connect a 100-nF power supply decoupling capacitor to GND.
9	DOUT/$\overline{DRDY}$	Digital output	Serial data output combined with data ready; active low
10	DIN	Digital input	Serial data input

Typical connection between SPI master and ADS1118 ADC



STM32F407VG terminology:

DOUT: MOSI, DIN:MISO CS:NSS, SCLK:SCLK

Most microcontroller SPI peripherals can operate with the ADS1118. The interface operates in SPI mode with CPOL = 0 and CPHA = 1

Two registers in ADS1118 ADC:

Conversion register hold 16-bit converted value:

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Config register:

15	14	13	12	11	10	9	8
SS	MUX[2:0]			PGA[2:0]			MODE
R/W-0h	R/W-0h			R/W-2h			R/W-1h
7	6	5	4	3	2	1	0
DR[2:0]			TS_MODE	PULL_UP_EN	NOP[1:0]		Reserved
R/W-4h			R/W-0h	R/W-1h	R/W-1h		R-1h

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

We will work with default configuration settings except TS_MODE!

Bit	Field	Type	Reset	Description
15	SS	R/W	0h	**Single-shot conversion start** This bit is used to start a single conversion. SS can only be written when in power-down state and has no effect when a conversion is ongoing. When writing: 0 = No effect 1 = Start a single conversion (when in power-down state) Always reads back as '0' (default).
14:12	MUX[2:0]	R/W	0h	**Input multiplexer configuration** These bits configure the input multiplexer. 000 = AIN_P is AIN0 and AIN_N is AIN1 (default) 001 = AIN_P is AIN0 and AIN_N is AIN3 010 = AIN_P is AIN1 and AIN_N is AIN3 011 = AIN_P is AIN2 and AIN_N is AIN3 100 = AIN_P is AIN0 and AIN_N is GND 101 = AIN_P is AIN1 and AIN_N is GND 110 = AIN_P is AIN2 and AIN_N is GND 111 = AIN_P is AIN3 and AIN_N is GND

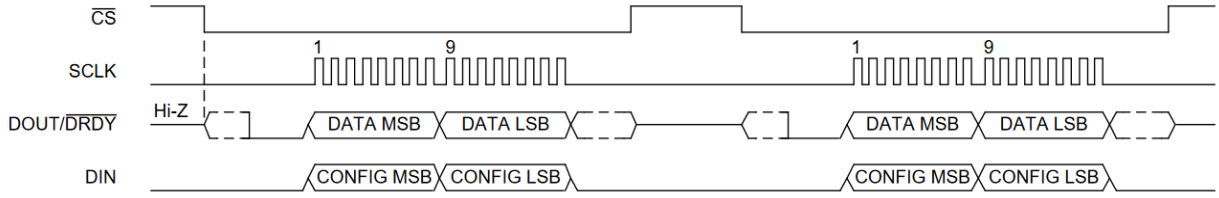
Bit	Field	Type	Reset	Description
11:9	PGA[2:0]	R/W	2h	**Programmable gain amplifier configuration** These bits configure the programmable gain amplifier. 000 = FSR is ±6.144 V(1) 001 = FSR is ±4.096 V(1) 010 = FSR is ±2.048 V (default) 011 = FSR is ±1.024 V 100 = FSR is ±0.512 V 101 = FSR is ±0.256 V 110 = FSR is ±0.256 V 111 = FSR is ±0.256 V
8	MODE	R/W	1h	**Device operating mode** This bit controls the ADS1118 operating mode. 0 = Continuous conversion mode 1 = Power-down and single-shot mode (default)
7:5	DR[2:0]	R/W	4h	**Data rate** These bits control the data rate setting. 000 = 8 SPS 001 = 16 SPS 010 = 32 SPS 011 = 64 SPS 100 = 128 SPS (default) 101 = 250 SPS 110 = 475 SPS 111 = 860 SPS

4	TS_MODE	R/W	0h	**Temperature sensor mode** This bit configures the ADC to convert temperature or input signals. 0 = ADC mode (default) 1 = Temperature sensor mode
3	PULL_UP_EN	R/W	1h	**Pullup enable** This bit enables a weak internal pullup resistor on the DOUT/$\overline{\text{DRDY}}$ pin only when $\overline{\text{CS}}$ is high. When enabled, an internal 400-kΩ resistor connects the bus line to supply. When disabled, the DOUT/$\overline{\text{DRDY}}$ pin floats. 0 = Pullup resistor disabled on DOUT/$\overline{\text{DRDY}}$ pin 1 = Pullup resistor enabled on DOUT/$\overline{\text{DRDY}}$ pin (default)
2:1	NOP[1:0]	R/W	1h	**No operation** The NOP[1:0] bits control whether data are written to the Config register or not. For data to be written to the Config register, the NOP[1:0] bits must be '01'. Any other value results in a NOP command. DIN can be held high or low during SCLK pulses without data being written to the Config register. 00 = Invalid data, do not update the contents of the Config register 01 = Valid data, update the Config register (default) 10 = Invalid data, do not update the contents of the Config register 11 = Invalid data, do not update the contents of the Config register
0	Reserved	R	1h	**Reserved** Always write 1h Reads back either 0h or 1h

So, the config register data will be,

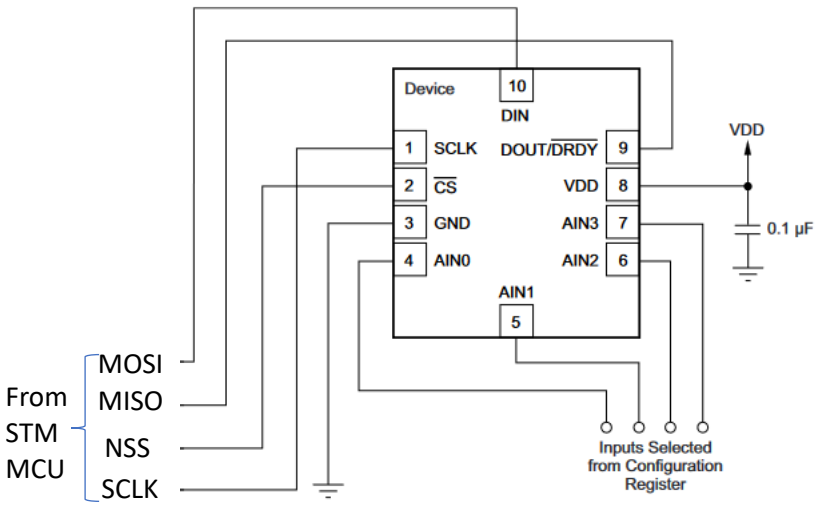
0 000 010 0 100 0 1 11 1 = 048Fh

16-Bit Data Transmission Cycle:



Answer for the question:

Interfacing diagram:



Use STM32CubeMx to generate HAL for SPI:

Th following functions and declarations will be generated:

SPI_HandleTypeDef hspi1;

void MX_SPI1_Init(**void**);

HAL_StatusTypeDef HAL_SPI_Transmit(**SPI_HandleTypeDef** *hspi, **uint8_t** *pData, **uint16_t** Size, **uint32_t** Timeout)

HAL_StatusTypeDef HAL_SPI_Receive(**SPI_HandleTypeDef** *hspi, **uint8_t** *pData, **uint16_t** Size, **uint32_t** Timeout)

Program: Application code(main.c)

```
#include "main.h"
#include "accelo.h"
SPI_HandleTypeDef hspi1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
uint8_t config[]={0x04,0x8F};          //048Fh   configuration
unit8_t data[2];                       //buffer to read ADC value
```

```
int main(void)
{
        HAL_Init();
        SystemClock_Config();
        MX_GPIO_Init();
        MX_SPI1_Init();

        //Write configuration data
        HAL_SPI_Transmit(&hspi1, config, 2, 100)


  while (1)
        {
                //To receive ADC value
              HAL_SPI_Receive(&hspi1,data,2,100)

        }
      }


    }
```