#include "stm32f4xx.h"

int main(){

SPI\_HandleTypeDef SPI\_Params; // Declares the structure handle for the parameters of SPI1

GPIO\_InitTypeDef GPIOA\_Params; // Declares the structure handle for the parameters of GPIOA

GPIO\_InitTypeDef GPIOE\_Params; // Declares the structure handle for the parameters of GPIOE

GPIO\_InitTypeDef GPIOE\_Params\_I; // Declares the structure handle for the parameters of the interrupt pin on GPIOE

uint8\_t data\_to\_send[1]; //Declares an array to store the required LIS3DSH

//register address in. It has a single element since we will only be

//accessing a single address in each SPI transaction.

uint16\_t data\_size=1; //Declares a variable that specifies that only a

//single address is accessed in each transaction.

uint32\_t data\_timeout=1000; //Sets a maximum time to wait for the SPI

//transaction to complete in - this mean that our program won’t freeze if

//there is a problem with the SPI communication channel.

uint8\_t CTRL\_REG4; //Declares the variable to store the who\_am\_I register value in

// Read the value from the Who\_am\_I register of the LIS3DSH

// Code to initialise the SPI

RCC->APB2ENR |= RCC\_APB2ENR\_SPI1EN; // Enables the clock for SPI1

SPI\_Params.Instance = SPI1; // Selects which SPI interface to use

SPI\_Params.Init.Mode = SPI\_MODE\_MASTER; // Sets the STM32F407 to act as the master

SPI\_Params.Init.NSS = SPI\_NSS\_SOFT; // Sets the slave to be controlled by software

SPI\_Params.Init.Direction = SPI\_DIRECTION\_2LINES; // Sets the SPI to fullduplex

SPI\_Params.Init.DataSize = SPI\_DATASIZE\_8BIT; // Sets the data packet size to 8-bit

SPI\_Params.Init.CLKPolarity = SPI\_POLARITY\_HIGH; // Sets the idle polarity for the clock line to high

SPI\_Params.Init.CLKPhase = SPI\_PHASE\_2EDGE; // Sets the data line to change on the second transition of the clock line

SPI\_Params.Init.FirstBit = SPI\_FIRSTBIT\_MSB; // Sets the transmission to MSB first

SPI\_Params.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_32; // Sets the clock prescaler to divide the main APB2 clock (previously set to 84MHz) by

//32 to give a SPI clock of 2.625MHz, which is less the maximum value of 10MHz for the SPI.

HAL\_SPI\_Init(&SPI\_Params); // Configures the SPI using the specified parameters

// Code to initialise pins 5-7 of GPIOA

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; //Enable the clock for GPIOA

GPIOA\_Params.Pin = GPIO\_PIN\_5 | GPIO\_PIN\_6 | GPIO\_PIN\_7; // Selects pins5,6 and 7

GPIOA\_Params.Alternate = GPIO\_AF5\_SPI1; //Selects alternate function 5 which corresponds to SPI1

GPIOA\_Params.Mode = GPIO\_MODE\_AF\_PP; //Selects alternate function push-pull mode

GPIOA\_Params.Speed = GPIO\_SPEED\_FAST; //Selects fast speed

GPIOA\_Params.Pull = GPIO\_NOPULL; //Selects no pull-up or pull-down activation

HAL\_GPIO\_Init(GPIOA, &GPIOA\_Params); // Sets GPIOA into the modes specified in GPIOA\_Params

// Code to initialise pin 3 of GPIOE

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOEEN; //Enable the clock for GPIOE

GPIOE\_Params.Pin = GPIO\_PIN\_3; // Selects pin 3

GPIOE\_Params.Mode = GPIO\_MODE\_OUTPUT\_PP; //Selects normal push-pull mode

GPIOE\_Params.Speed = GPIO\_SPEED\_FAST; //Selects fast speed

GPIOE\_Params.Pull = GPIO\_PULLUP; //Selects pull-up activation

HAL\_GPIO\_Init(GPIOE, &GPIOE\_Params); // Sets GPIOE into the modes specified in GPIOE\_Params

GPIOE->BSRR |= GPIO\_PIN\_3; //Sets the serial port enable pin CS high (idle)

\_\_HAL\_SPI\_ENABLE(&SPI\_Params); //Enable the SPI

//Code to initialise GPIOE pin 0 for the interrupt

GPIOE\_Params\_I.Pin = GPIO\_PIN\_0; // Selects pin 0

GPIOE\_Params\_I.Mode = GPIO\_MODE\_IT\_RISING; // Selects the interrupt mode and configures the interrupt to be signalled on a rising edge (low to high transition)

GPIOE\_Params\_I.Speed = GPIO\_SPEED\_FAST; //Selects fast speed

HAL\_GPIO\_Init(GPIOE, &GPIOE\_Params\_I); // Sets GPIOE into the modes specified in GPIOE\_Params\_I

\_\_HAL\_SPI\_ENABLE(&SPI\_Params); //Enable the SPI

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER14\_0; // Port D.14 output - red LED

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

GPIOD->MODER |= GPIO\_MODER\_MODER15\_0; // Port D.15 output - blue LED

GPIOD->MODER |= GPIO\_MODER\_MODER13\_0; // Port D.13 output - orange

data\_to\_send[0] = 0x00|0x20; // Address for CTRL\_REG4 register of LIS3DSH and fisrt bit been set to write

GPIOE->BSRR |= GPIO\_PIN\_3<<16; // Set the SPI communication enable line low to initiate communication(I copied!)

HAL\_SPI\_Transmit(&SPI\_Params,data\_to\_send,data\_size,data\_timeout); // Send the address of the register to be read on the LIS3DSH

data\_to\_send[0] =0x00;// Set a blank address because we are waiting to receive data

HAL\_SPI\_Receive(&SPI\_Params,data\_to\_send,data\_size,data\_timeout); // Get the data from the LIS3DSH through the SPI channel

data\_to\_send[0]=0x14; //Sent value 00010100 in Hex form to address 0x20.

HAL\_SPI\_Transmit(&SPI\_Params,data\_to\_send,data\_size,data\_timeout); // Send the address of the register to be read on the LIS3DSH

GPIOE->BSRR |= GPIO\_PIN\_3; // Set the SPI communication enable line high to signal the end of the communication process

CTRL\_REG4 = \*SPI\_Params.pRxBuffPtr; //Write tem value into CTRL\_REG4 register variable

if (CTRL\_REG4 == 0x14){ //to exam the value in that variable is 0x14

GPIOD->BSRR |= (1<<12); //turn green led on if it matches

}

else{

GPIOD->BSRR |=(1<<14); //else it will be red

}

}