Objective

Design an API that can be used to configure an ADXL345 Accelerometer and read data from it using the **I2C** protocol.

Understanding the I2C Bus

Initialising I2C

The I2C1 module on the STM32 Nucleo Dev board for this project.

For I2C1, pin PB8 functions as SCL and PB9 as PB9.

GPIO configuration (Refer to GPIO section of the reference manual for the registers)

1. Enable clock for GPIOB (AHB1 bus) using RCC (Refer to reference manual).
2. Set PB8 and PB9 to alternate mode *AF04* (Refer alternate function table in datasheet).
3. Set the output type of the pins to open drain (OTYPER register) and enable pull up (PUPDR register).

I2C1 module configuration (Refer to I2C section of the reference manual for the registers)

1. Enable clock for I2C1 module (APB1 bus) using RCC (Refer to reference manual).
2. Reset the module by enabling and disabling reset (CR1 register).
3. Select peripheral clock frequency to 16MHz (CR2 register).
4. Set SCL frequency (CCR register)
5. Set maximum rise time (TRISE register)
6. Enable I2C1 module (CR1 register)

Setting SCL frequency in I2C:

Desired clock frequency: 100kHz

Period (P_SCL) = 1/F = 10us

T high = T Low = 5us

Period of peripheral clock (P_periph) = 1/16MHz = 62.5ns

P_periph / P_SCL will give us the desired scale that we can set in the *CCR register.*

Setting max rise time:

In I2C, the feedback loop is a process by which the master ensures that the clock has transitioned from low to high before proceeding with communication. The TRISE register configures the *maximum rise time*, allowing the I2C master to wait long enough for the SCL signal to transition from low to high, ensuring reliable communication.

Explanation of TRISE value:

TRISE = (MRT / f_PERIPH) + 1 = 9

Number of PCLK cycles=Maximum rise time for SCL/Period of PCLK + 1

We are finding out the number of clock cycles the peripheral clock takes for the SCL signal to go from low to high.

I2C Byte read function

Write a function to read one byte of data from a slave device.
Arguments to the function:

1. Slave address
2. Memory address within the slave that we want to read from.
3. Pointer to the variable that stores the data

Code design to read a byte of data from a slave device:

1. Wait until the bus *busy* flag in SR2 register is cleared.
2. Master generates start condition (I2C_SR1).
    a. Wait till the start flag in SR1 is set.
3. Master sends slave address in *write mode* (Write addr to data register (DR) after left shifting by 1.
    a. Wait until the ADDR bit is set in SR1 (This happens after ACK).
4. Clear addr flag in SR1 by reading SR2.
    a. Address flag is cleared as per the reference manual (I2C functional description section).
5. Master sends address of internal data register of slave (Write to DR).
    a. Wait until the transmitter is empty.
6. Master restarts start condition.

7. Master <u>sends</u> slave address in *read mode* and waits till addr flag is set.
8. Disable acknowledgement in CR1 (Masters sends a NACK to indicate that one byte has been received).
9. Clear addr flag in SR1 by reading SR2.
10. Master generates a stop condition.
11. Wait till RXNE flag is set in SR1.
12. Master reads data from the data register and stores it in a variable.

I2C Burst read function

Write a function to read multiple bytes of data from a slave device.

<u>Arguments to the function:</u>

1. Slave address
2. Slave register address to read data from
3. Data size?
4. Pointer to the variable that stores the data

<u>Code design to read a byte of data from a slave device:</u>

Follow steps (1-9) in the previous section.

1. Enable the acknowledgement bit in CR1.
2.

In Burst read mode, the acknowledgement bit must be enabled because we want the master to send an acknowledgement after every byte of data. If the data size is 1 byte, then a NACK is sent by the master to stop the slave from sending any data and the stop condition is enabled.

I2C Burst write function

1. Wait until the bus busy flag in SR2 register is cleared.
2. Master generates start condition (I2C_SR1).
    a. Wait till the start flag in SR1 is set.
3. Master <u>sends</u> slave address in *write mode* (Write addr to data register (DR) after left shifting by 1.
    a. Wait until the ADDR bit is set in SR1 (This happens after ACK).
4. Clear addr flag in SR1 by reading SR2.

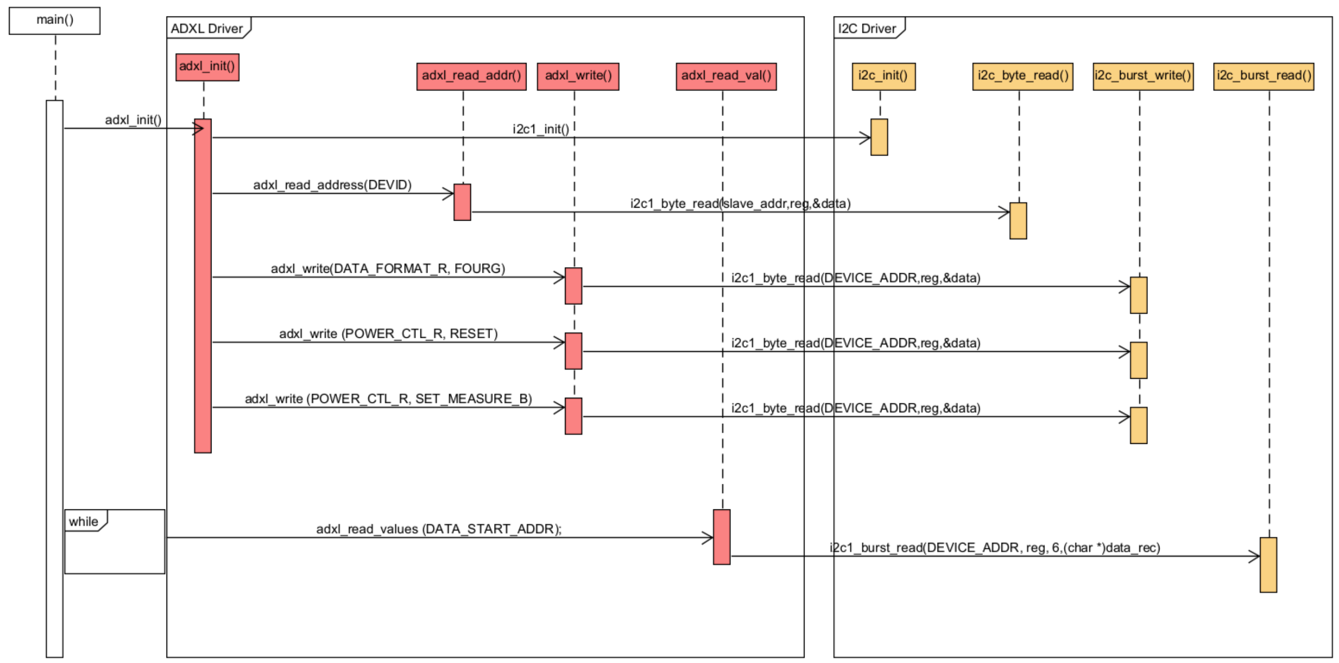
5. Master sends address of internal data register of slave (Write to DR).
    a. Wait until the transmitter is empty.
6. Wait until the data register is empty and send data by populating the data register of I2C1 (DR).
7. Wait until transfer finished by checking

Configuring and reading from the ADXL345 Accelerometer

Parameters for the accelerometer have to be configured before we can start using it.

1. Set data format to +/- 4g
2. Reset all bits.
3. Configure power control to set it to measuring mode.

Sequence diagram depicting the interaction of the ADXL driver with the I2C driver


main()
ADXL Driver
adxl_init()
adxl_read_addr()
adxl_write()
adxl_read_val()
I2C Driver
i2c_init()
i2c_byte_read()
i2c_burst_write()
i2c_burst_read()
adxl_init()
i2c1_init()
adxl_read_address(DEVID)
i2c1_byte_read(slave_addr,reg,&data)
adxl_write(DATA_FORMAT_R, FOURG)
i2c1_byte_read(DEVICE_ADDR,reg,&data)
adxl_write (POWER_CTL_R, RESET)
i2c1_byte_read(DEVICE_ADDR,reg,&data)
adxl_write (POWER_CTL_R, SET_MEASURE_B)
i2c1_byte_read(DEVICE_ADDR,reg,&data)
while
adxl_read_values (DATA_START_ADDR);
i2c1_burst_read(DEVICE_ADDR, reg, 6,(char *)data_rec)

Function description:

1. adxl_read_address: Reads the address of the slave device to ensure communication is with the right device
2. adxl_write: Writes data to the slave device using the i2c api.
3. adxl_read_values: Reads a continuous stream of values using the i2c_burst_read function.

Connecting ADXL345 to the Nucleo board

ADXL345	STM32F401RE Nucleo
GND	GND
VCC	5V
SCL	PB8 (D15)
SDA	PB9 (D14)
SDO	GND

Miscellaneous

1. What is dual addressing mode?
2. How does the slave know that the data sent after transmitting the slave address is the register address (Checks are performed on the slave side).
3. Why is 10-bit addressing used?
4. Why is restart required in read mode?
    a. In the context of reading data from a specific register within the slave device, the master initially writes the address of the register it wants to read from.
    b. In I2C communication, a repeated start condition is used to indicate a new transaction without releasing the bus. This is particularly useful in scenarios where the master needs to change the direction of communication (from write to read) or when performing sequential operations without letting other devices use the bus.