Beverly Yee

Prelab 05

u0770041

1. Describe two differences between I2C master and slave devices?

**A slave cannot start a transaction on its own.**

**The master initiates a transaction with a slave by sending the address of the**

1. What are the two connections in an I2C bus? Describe their purpose.

**Serial Data (SDA) – for when the master device uses clock transitions.**

**Serial Clock (SCL) – allows for both the master and the slave to produce data, depending on the direction of the communication.**

1. What is the difference between open-drain and push-pull outputs?

**Push-pull – depends on the voltage of external system.**

**Open-drain – contains a single transistor, can only pull to a low state. Requires an external connection to return a line to a high state.**

1. What is the purpose of the I2C restart condition?

**To allow a master to continue with a new transaction without having to stop and risk the chance of having other devices take control.**

1. What peripheral register would you use to set the read/write direction of the next I2C transaction?

**CR2 Register**

1. The 10-bit SADD bit-field holds the slave device address. Since standard I2C addresses only use 7 bits, to which bits in the bit-field would you write the shorter address?

**[7:1]**

1. Name one thing you found confusing or unclear in the lab.

**How to successfully bitmask an address without losing the data.**

Materials

* STM32F072 Discovery Board
* 2x 1kΩ resistor
* Jumper wires
* Logic analyzer

I2C Peripheral Registers

Control Register 1 (I2C\_CR1)

Manages the overall operation of the I2C peripheral.

* **SMBus Configuration** – SMBus is a more restrictive protocol designed for greater reliability than conventional I2C; configuration bits and registers related to SMBus should remain at their default values.
* **Slave Mode Configuration –** The I2C peripheral operates as both a slave and master device.
* **Noise Filters –** The I2C peripheral has both analog and digital noise filters; the analog filter is enabled by default and is sufficient for normal operation.
* **Interrupt Enables –** Many I2C conditions can generate interrupts; this register controls interrupts for events such as transmission errors, completed transmissions, and when the bus is free.
* **Peripheral Enable –** You must enable the I2C peripheral in this register after initialization: many initialization settings become read-only once you set this bit.

Control Register 2 (I2C\_CR2)

Contains settings for current I2C transaction. Use this register whenever communicating with a slave device. Understanding this register is critical to using the peripheral; the lab exercises extensively use the bits that we describe here.

* **AUTOEND** – When set, the peripheral will automatically generate a stop condition at the end of a transaction. This setting is undesirable when performing chained writes and reads—which is necessary in the lab assignment.
* **NBYTES[7:0]** – This group of 8-bits set the number of bytes to transmit in the next transaction. Modify these bits using bitwise operations (bit-masking) to prevent overwriting the rest of the register!
* **STOP & START** – These bits generate start and stop conditions on the bus. The user starts a new transaction by writing to the START bit. Release the bus by setting the STOP bit.
* **RD\_WRN** – This bit sets the direction of data transfer for the next transaction; its state controls the read/write bit in the address frame.
* **SADD[9:0]** – This group of 10-bits sets the slave address used in the next transaction. Unfortunately, the default 7-bit addressing mode uses bits [7:1] within the center of the bit field—often leading to an easy user error when learning the peripheral. Again, modify these bits using bitwise operations to prevent overwriting the rest of the register!

Timing Register (I2C\_TIMINGR)

The I2C peripheral has a very flexible timing system which allows the user to specify slew-rates and sampling delays. These settings can adjust the peripheral to operate reliably under non-ideal conditions. The configurable timings for the peripheral are:

* **PRESC** – This field sets the prescaler for the internal timers within the I2C peripheral. These timers generate the clock signal and control when it samples the data.
* **SCLL & SCLH** – These fields set the high and low periods of the clock signal (SCL). Since you can set these independently, the peripheral allows the user to specify an asymmetric clock.
* **SDADEL & SCLDEL** – These fields determine the data setup and hold timing used when transmitting and receiving.

Interrupt and Status Register (I2C\_ISR)

The read-only interrupt and status register indicates the state of every interrupt condition in the peripheral. These flag bits are also used by blocking drivers to determine the state of the bus.

The peripheral hardware clears many of these flags automatically whenever the user completes the appropriate action—always examine the documentation to determine the specific conditions required for each bit.

* **TXIS** – Transmit Interrupt Status. Address frame completed successfully. The peripheral is requesting new data to be written into the transmit data (TDXR) register.
* **NACKF** – Not Acknowledge Received Flag. The slave device did not acknowledge the address frame. There is likely a configuration issue; the current transaction has been aborted. Clear the NACKF flag, revise the initialization, and attempt to start a new transaction.
* **RXNE** – Receive Data Register Not Empty. Data has been received and is waiting to be read from the receive data (RXDR) register.
* **TC** – Transfer Complete. Set when the peripheral determines that the transaction is complete and is waiting for the user to perform a restart or stop condition. To restart, return to the top of the process for transmitting or receiving data from the slave.

Transmit & Receive Data Registers (I2C\_TXDR\I2C\_RXDR)

These registers contain the data that will transmit as well as the data that it has received from its slave device.

L3GD20 Digital Gyroscope Registers

Use the datasheet (<http://www.st.com/resource/en/datasheet/l3gd20.pdf>) for configuration.

Chip ID Register (WHO\_AM\_I)

Most devices contain an ID register with a known value documented in the datasheet. The primary purpose of the ID register is to provide something to compare against when testing driver code for the device.

**Address: 0x0F** (should contain the value 0xD3).

Control Register 1 (CTRL\_REG1)

The control register 1 enables the axes of the sensor and sets the output bandwidth and data-rate.

**Address: 0x20**

Status Register (STATUS\_REG)

The status register indicates whether new data has been produced by the sensor and is ready to be accessed. It also contains overrun error flags which indicate that previous data was overwritten before it was read. Many of these events can be configured to produce interrupt requests on the two interrupt output pins of the chip.

**Address: 0x27**

X-Axis Data Registers (OUT\_X\_L & OUT\_X\_H)

The L3GD20 produces 16-bit signed data for each axis. The two 8-bit registers for each axis must be shifted and bitwise OR’d together to produce the actual sensor output. These registers contain the x-axis data.

* OUT\_X\_L (Data Low Bytes) **Address: 0x28** (**0xA8** when reading both registers in same transaction)
* OUT\_X\_H (Data High Bytes) **Address: 0x29**

These registers should be read together in the same transaction. When reading multiple bytes, the L3GD20 automatically advances to the next register if the highest bit was set in the first address.

Y-Axis Data Registers (OUT\_Y\_L & OUT\_Y\_H)

These registers contain the 16-bit y-axis data.

* OUT\_X\_L (Data Low Bytes) **Address: 0x2A** (**0xAA** when reading both registers in same transaction)
* OUT\_X\_H (Data High Bytes) **Address: 0x2B**

These registers should be read together in the same transaction. When reading multiple bytes, the L3GD20 automatically advances to the next register if the highest bit was set in the first address.

Lab Assignment

Check Section 25.4 of the peripheral reference document for the complete initialization process.

Part 1

1. Connect the sensor pins.

A black and white diagram of a circuit

Description automatically generated

1. Set the GPIO modes.
   1. Enable GPIOB and GPIOC in the RCC.
   2. Set PB11 to alternate function mode, open-drain output type, and select I2C2\_SDA as its alternate function.
   3. Set PB13 to alternate function mode, open-drain output type, and select I2C2\_SCL as its alternate function.
   4. PB14 controls the slave address when in I2C mode. Set PB14 to output mode, push-pull output type, and initialize/set the pin high.
   5. Pin PC0 is connected to the SPI/I2C mode select pin. Set PC0 to output mode, push-pull output type, and initialize/set the pin high.
   6. Leave PB15 in input mode since it is connected to PB11 through a jumper wire. Modifying the mode of pin PB15 could cause a conflict if the two pins try to output different logic states.
2. Initialize the I2C peripheral.
   1. Enable the I2C2 peripheral in the RCC. The I2C2 peripheral is simpler and requires less configuration than I2C1.
   2. Set the parameters in the TIMINGR register to use 100 kHz standard-mode I2C.

A red circle with white text

Description automatically generated

* 1. Enable the I2C peripheral using the PE bit in the CR1 register.
     1. Make sure to do this after everything else has been initialized. Setting it locks all of the system-wide configuration bits and registers to prevent accidental modification during transmission. Clearing the PE bit after it’s been set will perform a peripheral reset and clears all configuration registers.

1. Set the transaction parameters in the CR2 register.
   1. Clear the NBYTES and SADD bit fields.
      1. I2C2->CR2 &= ~((0x7F << 16) | (0x3FF << 0));
   2. Set the slave address in the SADD[7:1] bit field. **Address = 0x69**.
   3. Set the number of data byte to be transmitted = 1 in the NBYTES[7:0] bit field.
   4. Configure the RD\_WRN to indicate a *write* operation.
2. Set the START bit to begin the address frame.
   1. Set the START bit in the CR2 register after configuring the slave address and transaction length. Setting the START bit locks the transaction parameters until the peripheral has completed the address frame.
   2. Multi-byte reads are accomplished by repeatedly polling on the status flags. Once the number of bytes has been read as was set in the NBYTES register, the user can issue a restart or stop condition. Same for multi-byte transmissions.
3. Wait until either of the TXIS or NACKF flags are set.
   1. If the NACKF flag is set, the slave did not respond to the address frame. You may have a wiring or configuration error.
   2. Continue if the TXIS flag is set.
4. Write the address of the “WHO\_AM\_I” register into the I2C TXDR register.
5. Wait until the TC flag is set.
6. Reload the CR2 register with the same parameters as before but set the RD\_WRN bit to indicate a *read* operation. (repeat step 4, but modify 4.d)
7. Set the START bit to begin the address frame.
8. Wait until either of the RXNE or NACKF flags are set.
   1. Continue if the RXNE flag is set.
9. Wait until the TC flag is set.
10. Check the contents of the RXDR register to see if it matches 0xD3 (expected value of the “WHO\_AM\_I” register).
11. Set the STOP bit in the CR2 register to release the I2C bus.
12. After successfully verifying that the “WHO\_AM\_I” register matches the expected value, include a logic analyzer screenshot of the I2C transaction in your postlab.

Part Two

1. Initialize the gyroscope.
   1. Set transaction parameters to transmit two bytes. (Part One, step 4, but with NBYTES = 2).
   2. Poll for TXIS for the first bit and set the TXDR register to **0x20** (CTRL\_REG1 address of the gyroscope).
   3. Poll for TXIS for the second bit and set the TXDR register to **0x0B** (enabling the PD bit for the register).
   4. Wait for the TC flag to be set.
   5. Set transaction parameters to read one byte. (Not sure if needed)
   6. Wait for the TC flag to be set.
   7. Set the STOP bit.
2. Set a threshold value for the gyroscope noise.
   1. This will prevent the LEDs in the next stops from triggering due to small vibrations.
3. In the main while loop, read and save the value of the X & Y registers every 100ms.
   1. Set the transaction parameter to transmit a single byte.
   2. Poll for TXIS and set the TXDR register to an axis register (0xAA or 0xA8)
   3. Wait for the TC flag.
   4. Set the transaction parameter to read two bytes.
   5. Poll for the first RXNE and grab the lower 8 bytes of the axis read.
   6. Poll for the second RXNE and grab the higher 8 bytes of the axis read.
   7. Wait for the TC flag.
   8. Put the lower & higher bytes together for the final result.
4. Use the 4 LEDs to indicate whether each measured axis is positive or negative.
   1. Design your application such that the LED nearest the direction of rotation lights up.

A circuit board with arrows pointing to the sides

Description automatically generated

Beverly Yee

u0770041

Postlab 5

1. What does the AUTOEND bit in the CR2 register do? Why don’t you want to use it when you’ll be needing a restart condition?

**Automatically generates a stop condition at the end of a transaction, as soon as NBYTES are transferred. Enabling this means you can’t do chained writes and reads.**

1. This lab used standard-mode 100 kHz I2C speed. What values would you write in the TIMINGR if we were using 400 kHz fast-mode?

**PRESC = 0**

**SCLL = 0x9**

**SCLH = 0x3**

**SDADEL = 0x1**

**SCLDEL = 0x3**

1. This lab used blocking code. To implement it completely as non-blocking you would replace all of the wait loops with interrupts. Most flags in the I2C peripheral can trigger an interrupt if the proper enable bit is set. Find the interrupt enable bits that match the following flags:

* TC – **TCIE, I2C\_CR1 bit 6**
* NACKF – **NACKIE, I2C\_CR1 bit 4**
* TXIS – **TXIE, I2C\_CR1 bit 1**
* ARLO – **ERRIE, I2C\_CR1 bit 7, (generic/all other flags)**

1. The gyro can operate in three full-scale/measurement ranges, measured in degrees-per-second (dps). What are these three ranges?

**245, 500, 2000 dps**

1. What is the I2C address of the gyro when the SDO pin is low? The lab has the pin set high, read the I2C section of the gyro datasheet.

**0x68**