Lab 5: Interfacing with I2C and SPI Peripherals

I. Objectives

1. Use a real-time clock and an A/D converter to experiment with I2C/SPI buses
2. Use the DS3231 RTC to set and display the date and time, set alarms, and read measurements from a temperature sensor over I2C bus
3. Use the MCP3202 A/D converter to read an analog input and produce its corresponding digital output over SPI bus
4. Extract the needed information from the datasheets of the components used

II. I2C Bus

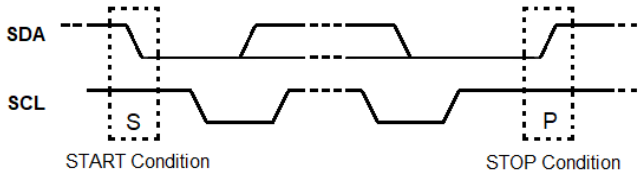
II.1 Overview

I2C is a synchronous communication protocol, meaning that the two devices sharing information must share a common clock signal. Communication should always occur between a Master and a Slave. More than one slave can be connected to a Master. The complete communication takes place through these two wires namely, Serial Clock (SCL) and Serial Data (SDA).

- Serial Clock (SCL): Shares the clock signal generated by the master with the slave
- Serial Data (SDA): Sends and receives data between the master and the slave.

I2C communication protocol is mainly used to communicate with sensors or other devices which have to send information to a master over a short distance. It is very handy when a microcontroller has to communicate with many other slave modules.

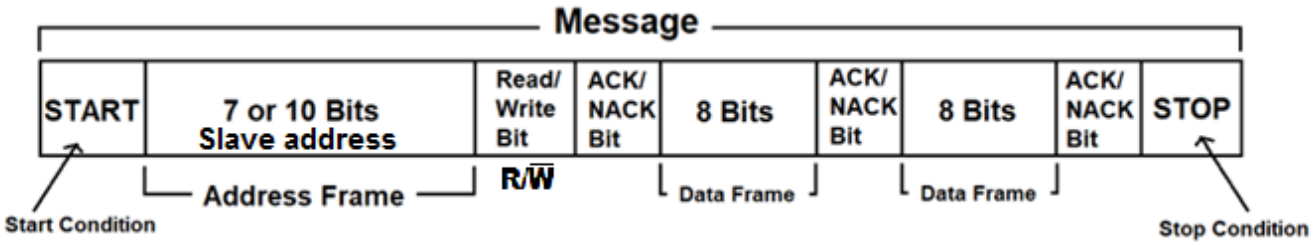
At any given time only the master is able to initiate the communication. Initialization of transmission begins with a falling edge of SDA, which is defined as 'START' condition in the diagram below where the master leaves SCL high while setting SDA low. The falling edge of SDA is the hardware trigger for the START condition. After this all devices on the same bus go into listening mode. In the same manner, a rising edge of SDA stops the transmission which is shown as 'STOP' condition in the diagram, where the master leaves SCL high and also releases SDA to go HIGH. So a rising edge of SDA stops the transmission.



Since there is more than one slave in the bus, the master has to refer to each slave using a different address. When addressed, only the slave with that particular address will reply back with the information while others keep quiet. This way the master can use the same bus to communicate with multiple devices. R/W bit indicates the direction of transmission of following bytes, HIGH means the slave will transmit and LOW means the master will transmit.

Each bit is transmitted on each clock cycle, so it takes 8 clock cycles to transmit a byte. After each byte is sent or received, the ninth clock cycle is held for the ACK/NACK (acknowledged/not acknowledged).

This ACK bit is generated by either slave or master depending upon the situation. For ACK bit, SDA is set to low by master or slave at 9th clock cycle. So if it is low it is considered an ACK, otherwise NACK.



II.2 Real-Time Clock (RTC)

A real-time clock is basically just like a watch - it runs on a battery and keeps time even when there is a power outage. Using an RTC, you can keep track of long timelines, even if you reprogram your microcontroller or disconnect it from USB or a power plug.

The DS3231 is a low-cost, extremely accurate I2C real-time clock (RTC) with an integrated temperature-compensated crystal oscillator (TCXO) and crystal. The device incorporates a battery input, and maintains accurate timekeeping when main power to the device is interrupted. The clock operates in either the 24-hour or 12-hour format with an active-low AM/PM indicator. The DS3231 RTC has a built-in alarm functions as well as a temperature sensor. Address and data are transferred serially through an I2C bidirectional bus.

Key features:

- Highly Accurate RTC Completely manages all timekeeping functions
- Real-Time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year, with leap-year compensation valid up to 2100
- Two time-of-day alarms
- Fast (400kHz) I2C interface
- Battery-backup input for continuous timekeeping
- Digital Temperature Sensor with ±3°C accuracy
- Programmable square-wave output signal
- 5V and 3.3V operation

Pinouts:

- Vin and Gnd to be connected to the microcontroller
- SCL: clock input for the I2C serial interface and is used to synchronize data movement on the serial interface, to be connected to SCL pin of the microcontroller I2C serial interface
- SDA: Serial Data Input/Output, to be connected to SDA pin of the microcontroller I2C serial interface

- INT/SQW: this pin either generates an interrupt due to alarm condition or outputs a square-wave signal. The selection is controlled by the bit *INTCN* in the Control Register (0Eh).

Note: The contents of the time and calendar registers are in the binary-coded decimal (BCD) format.

RTC Alarms:

- The DS3231 contains two time-of-day/date alarms. Alarm 1 for instance can be set by writing to registers 07h to 0Ah.
- The alarms can be programmed (by the alarm enable *A1IE* and *INTCN* bits of the control register 0Eh) to activate the INT/SQW output on an alarm match condition. If the corresponding Alarm Interrupt Enable *A1IE* is set to logic 1 and the *INTCN* bit is set to logic 1, the alarm condition will activate the INT/SQW signal. The *A1IE* bit is disabled (logic 0) when power is first applied.
- Bit 7 of each of the time-of-day/date alarm registers are mask bits (*A1M1* to *A1M4*). When all the mask bits for the alarm are logic 0, an alarm only occurs when the values in the timekeeping registers match the corresponding values stored in the time-of-day/date alarm registers. The alarms can also be programmed to repeat every second, minute, hour, day, or date.

II.3 Experiments

Experiment 1:

This experiment develops an application on the nucleo board that sets the hours, minutes, seconds of DS3231 and displays the time over the UART interface to TeraTerm application.

```
/* USER CODE BEGIN 0 */
#include "stdio.h"
/* USER CODE END 0 */

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART2_UART_Init();

  /* USER CODE BEGIN 2 */

  //check that device is ready to operate
  if (HAL_I2C_IsDeviceReady(&hi2c1, 0xD0, 10, HAL_MAX_DELAY) == HAL_OK)
  {
    for (int i = 1; i<=10;i++)            // indicator of ready device
    {
      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
      HAL_Delay(250);
    }
  }
```

```
  //Transmit via I2C to set clock to 7:15:35 am
  uint8_t secbuffer[2], minbuffer[2], hourbuffer[2];

  // seconds
  secbuffer[0] = 0x00; //register address
  secbuffer[1] = 0x35; //data to put in register --> 35 sec
  HAL_I2C_Master_Transmit(&hi2c1, 0xD0, secbuffer, 2, 10);

  // minutes
  minbuffer[0] = 0x01; //register address
  minbuffer[1] = 0x15; //data to put in register --> 15 min
  HAL_I2C_Master_Transmit(&hi2c1, 0xD0, minbuffer, 2, 10);

  // hours
  hourbuffer[0] = 0x02; //register address
  hourbuffer[1] = 0x47; //data to put in register 01001001 --> 7 am
  HAL_I2C_Master_Transmit(&hi2c1, 0xD0, hourbuffer, 2, 10);

  char uartBuf [100] = {0};

  //Receive via I2C and forward to UART
  while (1)
  {
    //send seconds register address 00h to read from
    HAL_I2C_Master_Transmit(&hi2c1, 0xD0, secbuffer, 1, 10);
    //read data of register 00h to secbuffer[1]
    HAL_I2C_Master_Receive(&hi2c1, 0xD1,  secbuffer+1, 1, 10);

    HAL_I2C_Master_Transmit(&hi2c1, 0xD0, minbuffer, 1, 10);
    HAL_I2C_Master_Receive(&hi2c1, 0xD1,  minbuffer+1, 1, 10);

    HAL_I2C_Master_Transmit(&hi2c1, 0xD0, hourbuffer, 1, 10);
    HAL_I2C_Master_Receive(&hi2c1, 0xD1,  hourbuffer+1, 1, 10);
    hourbuffer[1] = hourbuffer[1] & 0x1F;

    // transmit time to UART
sprintf(uartBuf,"%02x:%02x:%02x\r\n",hourbuffer[1],minbuffer[1],secbuffer[1]);

    HAL_UART_Transmit(&huart2, (uint8_t *)uartBuf, sizeof(uartBuf), 10);

    HAL_Delay(1000);
  }
  /* USER CODE END 2 */
}
```

Experiment 2:

Update the code of experiment 1 to add an alarm capability to the RTC. Set your clock to 12:59 pm and enable an alarm to buzz at 01:00 pm. Use the active buzzer to produce the alarm sound and use alarm 1 on the RTC.

Hints:

- You need to update the values of the control and status registers and the mask bits (check datasheet)
- INT/SQW pin is active-low

III. SPI Bus

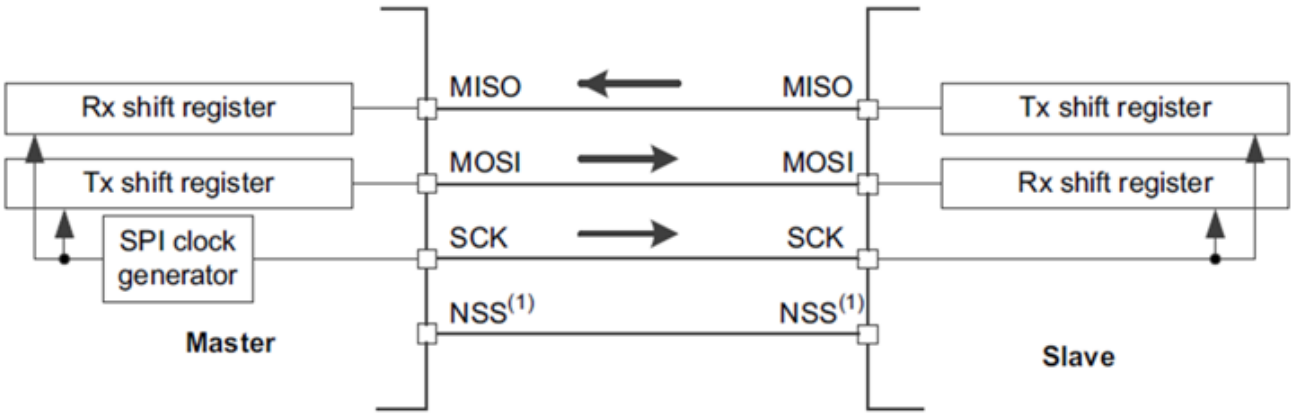
III.1 Overview

The serial peripheral interface (SPI) is a four-wire bus that provides high-speed synchronous data exchange over relatively short distances (typically within a set of connected boards), using a master/slave system with hardware slave selection. SPI is commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

It consists of a serial clock (SCK), master output/slave input (MOSI), master input/slave output (MISO), and a slave select pin (SS). One processor must act as a master, generating the clock. Others act as slaves, using the master clock for timing the data send and receive. The slaves can be other microcontrollers or peripherals with an SPI interface.

To transfer data, the master selects a slave device to talk to, by taking its SS line low. This tells the slave that it should wake up and receive/send data and is also used when multiple slaves are present to select the one the master wishes to talk to. Data bits are then clocked in or out of the slave SPI shift register to or from the master.

The speed of the bus range is much higher than that found in I2C. There is no standard specification defined for SPI — it is a de facto standard. Many microcontrollers have built-in SPI peripherals that handle all the details of sending and receiving data, and can do so at very high speeds.



III.2 MCP3202 12-Bit A/D Converter

MCP3202 is a Dual Channel 12-Bit A/D Converter with SPI® Serial Interface in DIP Package.
Key features:

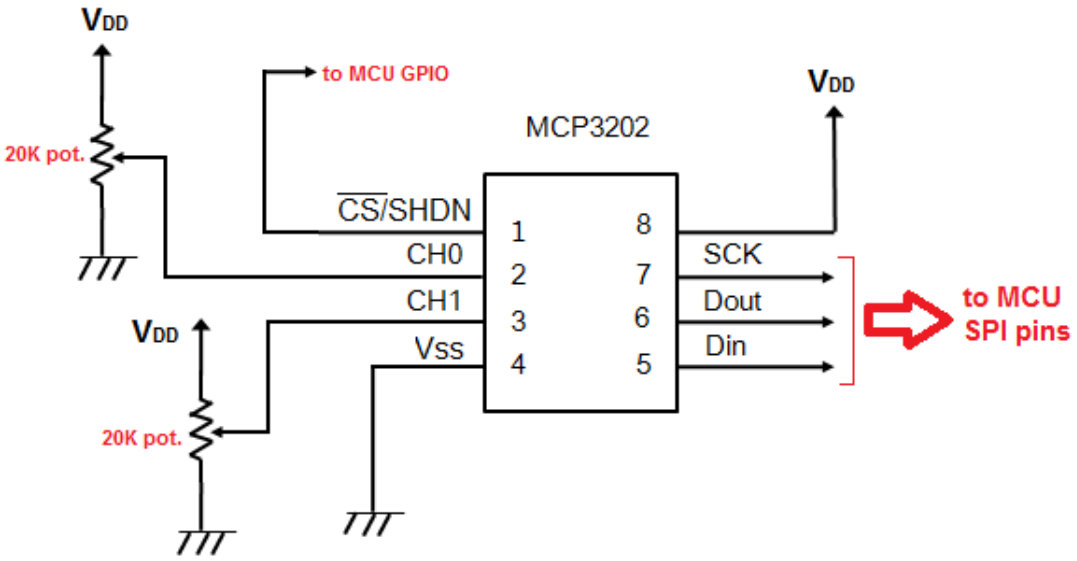
- 12-bit resolution (range 000_{16} to FFF_{16})
- Analog inputs programmable as single-ended or pseudo-differential pairs
- SPI® serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V – 5.5V
- 100ksps max sampling rate at VDD = 5V

$$\text{Digital Output Code} = \frac{4096 * V_{IN}}{V_{DD}}$$

where:
V_{IN} = analog input voltage
V_{DD} = supply voltage

Chip pinouts:

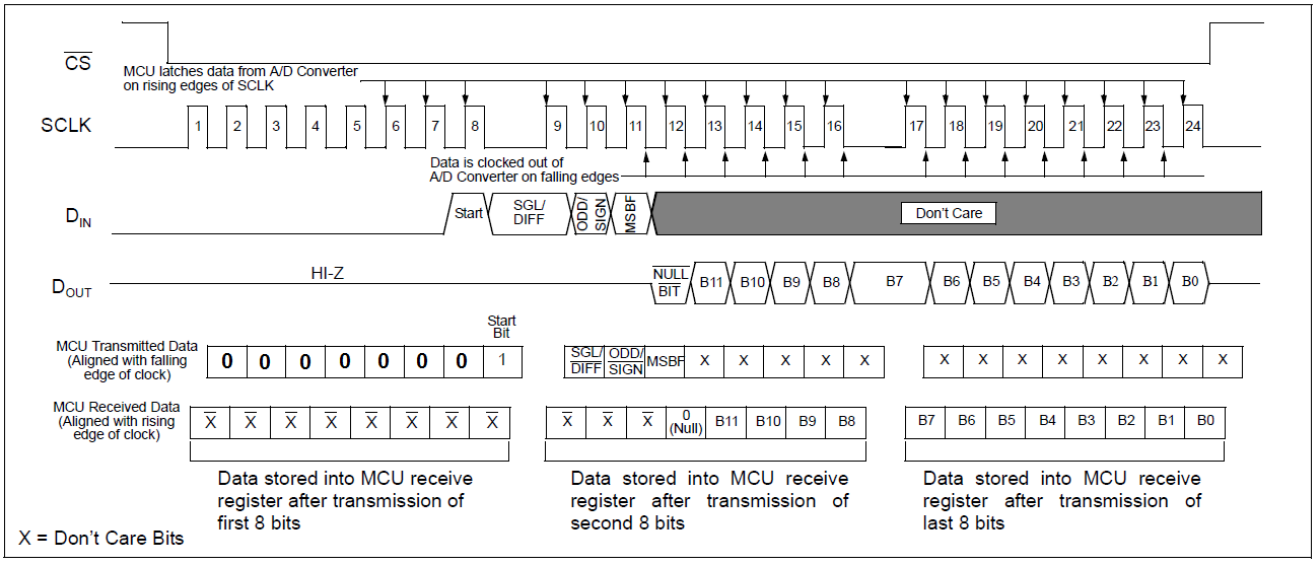
- **CH0** and **CH1** are the analog inputs for channels 0 and 1 respectively. These channels can be programmed to be used as two independent channels in single ended-mode or as a single pseudo-differential input where one channel is IN+ and one channel is IN-. Configuration is done as part of the serial command before each conversion begins. Input Voltage Range for CH0 or CH1 in Single-Ended Mode is $V_{SS} \rightarrow V_{REF}$, and for IN+ in Pseudo-Differential Mode is $IN- \rightarrow V_{DD}+IN-$.
- The **CS/SHDN** pin is used to initiate communication with the device when pulled low and will end a conversion and put the device in low power standby when pulled high. The CS/SHDN pin must be pulled high between conversions.
- Serial Clock (**CLK**): The SPI clock pin is used to initiate a conversion and to clock out each bit of the conversion as it takes place.
- $\mathbf{D_{IN}}$ (Serial Data Input): The SPI port serial data input pin is used to clock in input channel configuration data.
- $\mathbf{D_{OUT}}$ (Serial Data output): The SPI serial data output pin is used to shift out the results of the A/D conversion (serial 12-bit digital output code). Data will always change on the falling edge of each clock as the conversion takes place.



Communication details:

- Initiating communication with the device is done by bringing the CS line low. If the device was powered up with the CS pin low, it must be brought high for t_{CSH} and back low to initiate communication. **The first clock received with CS low and D_{IN} high will constitute a start bit**.
- With most microcontroller SPI ports, it is required to send groups of eight bits. It is also required that the microcontroller SPI port be configured to clock out data on the falling edge of clock and latch data in on the rising edge. It is very possible that the number of clocks required for communication will not be a multiple of eight. Therefore, it may be necessary for the MCU to send more clocks than are actually required. This is usually done by **sending 'leading zeros' before the start bit**, which are ignored by the device.
- **The SGL/DIFF bit and the ODD/SIGN bit follow the start bit** and are used to select the input channel configuration.
- Following the ODD/SIGN bit, **the MSBF bit is transmitted** to and is used to enable/disable the LSB first format for the device. If the MSBF bit is high, then the data will come from the device in MSB first format and any further clocks with CS low will cause the device to output zeros. If the MSBF bit is low, then the device will output the converted word LSB first after the word has been transmitted in the MSB first format.
- On the falling edge of the clock for the MSBF bit, the device will **output a low null bit**. The **next sequential 12 clocks will output the result of the conversion** with MSB first. After completing the data transfer, if further clocks are applied with CS low, the A/D Converter will output zeros indefinitely.

Refer to the figure below adopted from the A/D datasheet to develop the exact sequence of bits to transmit and receive using 8-bit segments.



III.3 Experiments

Experiment 3:

Below is a simple application on the nucleo board that uses two SPI interfaces of the MCU as a master and slave and echoes whatever data sent by one to the other.

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI1_Init();
  MX_SPI3_Init();

  /* USER CODE BEGIN 2 */
  uint8_t txdata = 0, rxdata;
  while (1)
  {
    HAL_SPI_Transmit(&hspi1, &txdata, 1, 100);
    HAL_SPI_Receive(&hspi3, &rxdata, 1, 100);

    if (txdata == rxdata)
    {
       // indicate success by blinking a LED 3 times or sending "Yes" to UART
    }
    else
    {
       // indicate failure by turning the LED off or sending "No" to UART
    }
    HAL_Delay (500);
    txdata +=10;
  }
  /* USER CODE END 2 */
}
```

Notes:

- On NucleoL432 the on-board LED pin conflicts with SPI3 pins, so you can use another pin as GPIO output and connect it to an external LED.
- You can also test a single SPI interface by echoing data on it. Just connect MOSI to MISO, and replace the transmit and receive statements with:
  `HAL_SPI_TransmitReceive(&hspi1,&txdata,&rxdata,1,100);`
- The default value for SPI data size on CubeMX is 4 bits. However, we send and receive 1 byte at a time using HAL APIs, so you need to update this value.

Experiment 4:

Develop an application on the nucleo board that uses a 20K potentiometer to generate a varying signal in the range 0 → V_{DD} to be supplied as the analog input to the MCP3202 ADC. The ADC produces the digital conversion output and communicates it to the microcontroller through the SPI interface. The microcontroller should send the resulting hexadecimal value over one of its UART interfaces to be displayed on TeraTerm.

Hints:

- Use an unused GPIO on the MCU to set/clear the CS/SHDN pin on the ADC
- You can use a special API to transmit and receive simultaneously
- Set the sysclk on CubeMX such that the SPI peripheral speed does not exceed 1.8 Mbps. This is because, as mentioned in the ADC datasheet, the maximum speed it can work on with a VDD of 5v is 1.8 Mbps.
- Set the data size in CubeMX to 8 bits.

IV. References:

- DS3231 datasheet
- MCP3202 datasheet
- UM1884_Description of STM32L4 HAL and LL drivers (dm00173145)

V. Lab Report [10 pts]

(Deadline: Monday of next week 11:59 pm) (Individual submission)

1. [2 pts] Provide your C code of experiments 2 and 4.
2. [3 pts] Modify experiment 2 to set and display the time, date, day, month and year. And set alarm 2 to buzz when day, hours, and minutes match. Provide a small sized video clearly showing your board connections and TeraTerm output in order to get the question grade.
3. [5 pts] Modify experiment 2 such that the alarm time can be input over UART while the DS3231 RTC is running. The user should input 6 digits to indicate the 24-hour format hhmmss for the alarm time. Use interrupts for UART input (no polling allowed). Provide a small sized video clearly showing your board connections and TeraTerm output in order to get the question grade.