Bern University
of Applied Sciences

# Lab – 02

## Read external IC using SPI communication

Andreas Habegger | Adrian Steiner
BTS3230 | Version 1.0.0 of 17.03.2025

Please be aware that the content is subject to change at any time. For the latest version, please check the website.

The **S**erial **P**eripheral **I**nterface (SPI) is one of the most widely used interfaces between the MCU and peripheral ICs such as the MAX 31855 Thermocouple Digital Converter or Operational Amplifier on the temperature logger board. The aim of this lab is to initialise the SPI bus, read the internal temperature from the MAX 31855 Thermocouple Digital Converter in a sample rate of $1\,\mathrm{Hz}$ and set the RGB LED depending on the temperature.

### Objectives

- Initialise SPI communication
- Read data from the MAX31855 IC using SPI
- Extracting temperature data
- Correct handling of temperature data encoded with sign magnitudes

### Outcomes

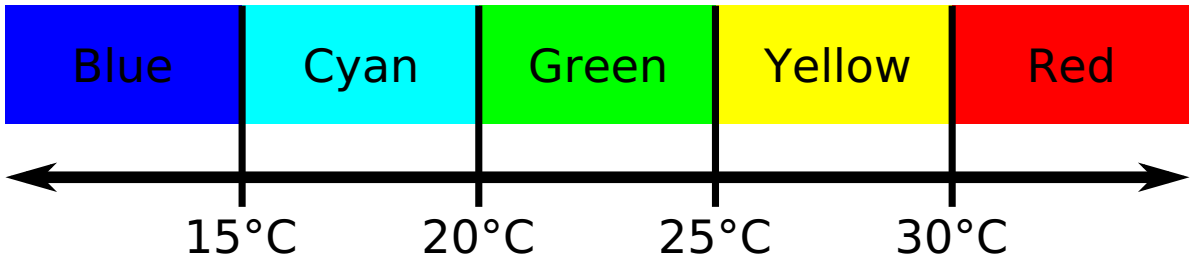
- Understand SPI communication
- Handling data with integer datatypes in different bit lengths
- Interpretation of data in an integer value
- Temperature display with RGB LED

# Description

Until now, external peripherals have been connected directly to the GPIO pins and used as simple inputs and outputs. Your applications are based only on the logic in your MCU and some simple I/O hardware such as the LED and joystick. In the real world, you will never encounter a situation like this. Many special tasks are outsourced and not integrated into the MCU, such as the MAX31855, which reads the thermocouple interface and calculates the effective temperature at the probe tip.

In addition to the thermocouple temperature, the MAX31855 also measures the current internal circuit temperature. The aim of this lab is to obtain this internal temperature data and provide a feedback to the user using the RGB LED with different colours. The colour scheme is based on the current received temperature. The sample frequency is $1\,\mathrm{Hz}$.

**Colorscheme**




LED colour scheme according to temperature

Several communication systems have been developed in recent years to communicate with external peripherals. As they all have different advantages, no general method has been developed for all applications. Rather, they are used in different areas where their advantages are greater or the disadvantages are less. In this exercise, the focus is on SPI, which is explained more in the followed sections.

## SPI bus

The SPI is a synchronous full-duplex serial communication interface that is widely used to exchange data between a MCU and peripheral devices by using four wires. It was developed by Motorola in the late 1970s. Due to the high speed full-duplex communication and simple implementation, several integrated circuit companies began to use SPI as their interface.

Disadvantages:

- Requires more wires than other communication systems
- No hardware to control the flow of data
- No acknowledgement between master and slave (even the master does not send data to any device without noticing)
- Only one master device available at a time
- Slowest device determines transfer speed
- Supports only short distance communication compared to RS485 and CAN
- Requires a slave select (SS) (also called chip select (CS)) line for each slave device

Advantages:

- Supports full-duplex communication at all times
- Provides higher throughput than I2C
- Interface hardware is very simple (consists of a simple shift register)
- Power consumption is lower compared to I2C due to very simple hardware circuitry
- Slaves do not require a precision oscillator as they use a clock from the master device.
- Only one master device is supported, therefore no conflicts when initiating a transfer
- No addresses, START or STOP symbols, therefore no protocol overhead

## SPI bus structure

The bus consists of one clock, two data and one select line per slave.

- **SCLK**: **S**erial **Cl**oc**k**

  Serial clock to synchronise the data transfer (in the range of MHz)

- **MOSI**: **M**aster **O**ut **S**lave **I**n

  Transfer data from the master unit to the selected slave device

- **MISO**: **M**aster **I**n **S**lave **O**ut

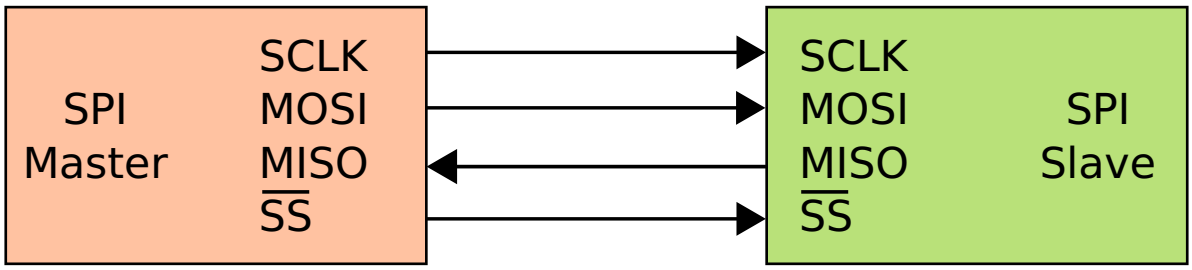
  Transfer data from the selected slave device to the master unit

- **$SS_n$**: **S**lave **S**elect or **C**hip **S**elect (**$CS_n$**)

  Control signal to select the receiving slave on the bus. This is done by pulling the signal low from the master unit.

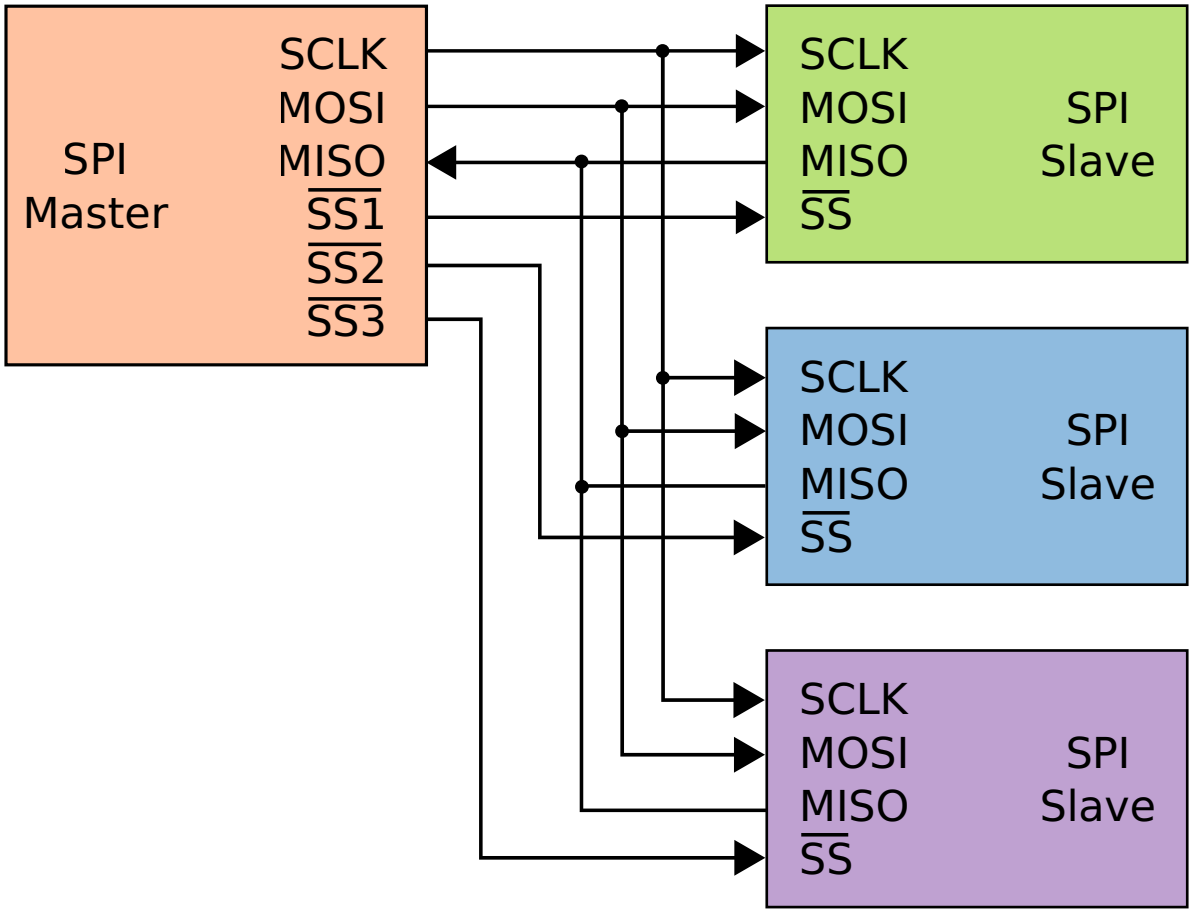
## Device connections

### Single device




Single device connection (src: SPI Wikipedia)

### Multi device




Multi device connection (src: SPI Wikipedia)

A new individual SS line must be connected to the peripherals for each device.

## SPI modes

On a serial bus you have to decide on which edge of the clock to read the current state on the input line or to shift the data to the output. The default state of the clock is also important. This depends on whether it is set to high by default and starts with a falling edge or vice versa. This results in four possible modes, which can be configured with two bits:

- CPOL: Clock Polarity

  Sets the default state of the clock SCK. 1 results in high, otherwise low.

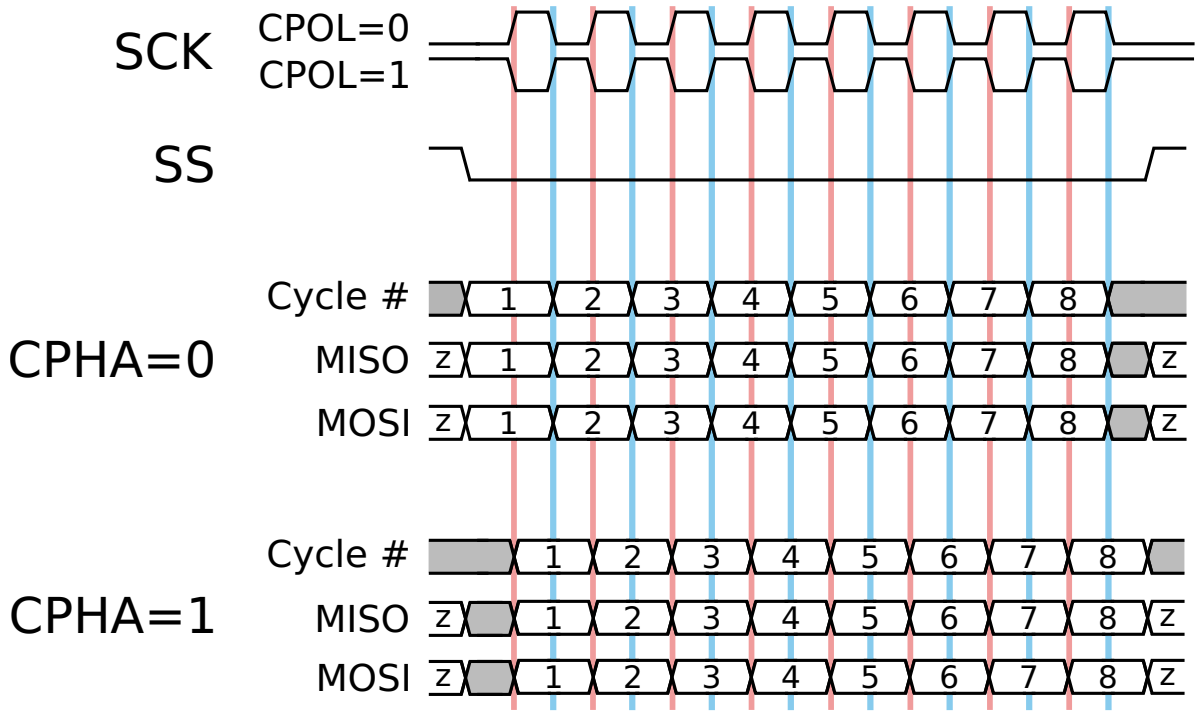
- CPHA: Clock Phase

  Reads the data or shifts the output data on the first clock edge. A zero results in reading the input data, a one results in shifting the output data.

SPI Mode Configurations

| Mode | CPOL | CPHA | Clk idle | Description |
|---|---|---|---|---|
| 0 | 0 | 0 | low | Data sampled on rising edge and shifted out on the falling edge |
| 1 | 0 | 1 | low | Data sampled on the falling edge and shifted out on the rising edge |
| 2 | 1 | 0 | high | Data sampled on the falling edge and shifted out on the rising edge |
| 3 | 1 | 1 | high | Data sampled on the rising edge and shifted out on the falling edge |

**Description SPI Modes**




SPI Modes Description (src: SPI Wikipedia)

## Tasks

### SPI

For peripherals that use GPIO pins, initialisation consists of two steps. The first is the initialisation of the GPIO pins and the second is the initialisation of the peripheral itself.

As mentioned before, the SPI needs several pins to communicate between peripherals. Although only the MAX31855 is used in this lab, all relevant SPI pins must be initialised, i.e. all chip select pins that are used for the same SPI bus.

**Note**

There are MCUs that don't provide SPI peripheral. In such a situation, you can still realise SPI by using GPIOs and implementing the interface using the bit banging technique. The current situation with an included peripheral is by far the most favorable in terms of effort and complexity of implementation.

**Question**

a. Look at the Thermocouple Expansion Board schematic or PCB Description to identify the SPI peripheral you are using.

b. Find all used GPIO pins connected with this SPI bus.

c. Why is it important to initialise and use ALL connected SPI pins, i.e. SS (CS) wires?

**Solution**

a. The used SPI peripheral is **SPI2**

b. Pins used for SPI2 external peripherals:

Used SPI Pins

| Pin name | MCU pin | Description |
|---|---|---|
| SPI2_SCK | PB13 | SPI2 Clock wire |
| SPI2_MISO | PB14 | SPI2 Master In Slave Out |
| MAX_31855CS_I | PA15 | Chip select MAX31855 channel I |
| MAX_31855CS_II | PA11 | Chip select MAX31855 channel II |
| LTC2360_CONV_I | PB7 | Chip Select LTC2360 channel I |
| LTC2360_CONV_II | PB12 | Chip Select LTC2360 channel II |

c. A device with a slave select (SS) or chip select (CS) is activated by a low on the SS line. With this information, the designer must ensure that all slave select lines are set to high.

The GPIO pin requires the alternate function mode for data and clock wiring. The chip select pins can be used as normal general purpose outputs.

**Question**

d. What is the alternate function for the clock and data pins? All the alternate functions for the GPIO pins are described in the DS10693 table 11.

**Solution**

d. The fifth alternate function for both pins (see property **5**) is used.

The SPI peripheral can be configured in the SPI Control Register 1 (SPI_CR1). To be able to complete the configuration, further information are need about the SPI in the used slaves in order to match them. Study the MAX 31855 Thermocouple Digital Converter to find out all the solutions.

**Question**

e. In which mode does the SPI for the MAX31855 have to run? Have a look at the serial interface diagrams in the MAX 31855 Thermocouple Digital Converter.

f. What is the maximum clock frequency of the MAX31855?

g. Is the LSB or MSB bit sent first?

h. What is the size of a data frame? Can this size be read with our MCU?

**Solution**

e. In the MAX 31855 Thermocouple Digital Converter in Figure 2 shows that the clock is low by default and the data is read on a rising edge and shifted on the falling edge. This results in **mode 0**.

f. In the same document on page 4 in the table Serial-Interface Timing Characteristics is the Serial-Clock Frequency described with a max value of **5Mhz**.

g. The **MSB** is sent as the first bit.

h. The data frame is 32 bit. On the SPI peripheral in our MCU only 16 bit can be read. We have to read the frame twice for the complete data frame.

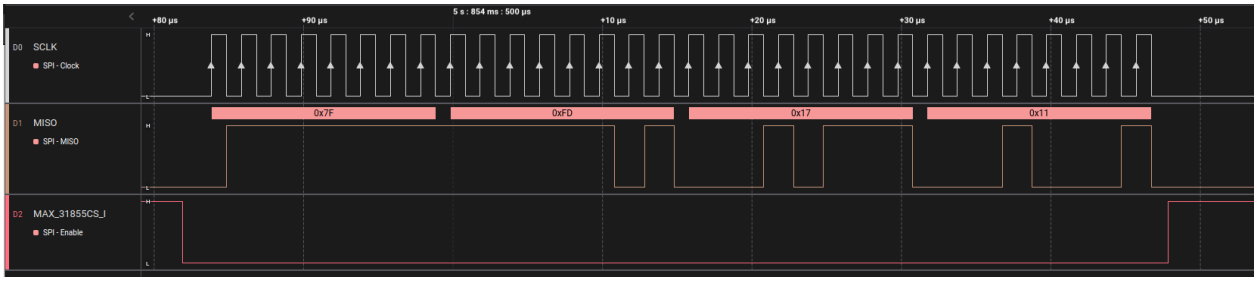
## Conversation

The internal temperature must be extracted from the data frame. The 12 bit twos complement value must be converted to an MCU readable value i.e. 16bit. The aim is to have a signed integer value. To simplify the calculation, the fractional value is ignored for this laboratory.

### Analysis

If you are having problems getting data, an additional method of checking the SPI bus is to use the saleae. This allows you to monitor your communication. The logic2 application also provides an SPI analyser that displays the transmitted values on the SPI data wires. In case of errors, a wrong protocol will be displayed and the recognising of the data will fail.

**Demo**




Correct SPI communication verified using the saleae logic analyzer.

**See also**

For more information on how to use the analyser, please refer to the FAQ Saleae instructions in the protocol analyzer section.

## Implementation

The RGB LED is initialised as it has been several times before.

**RGB-LED initialisation**

1. Initialise all channels of the RGB LED as general purpose outputs.
2. Write a function to set the desired colours.

After the LED configuration, the SPI bus is initialised. First the used GPIO pins are set to the correct configuration.

**SPI GPIO initialisation**

3. Configure the SPI clock and data pins (MISO) as alternate function mode.
4. Configure the same pins as high speed pins (GPIOx_OSPEEDR).

5. Set the correct alternate function in the corresponding register and pin.

> **Note**
>
> The CMSIS GPIO_TypeDef has no member of AFRL and AFRH. CMSIS combines these registers in an array of type `uint32_t` with size 2 called AFR. This results in AFR[0] = AFRL and AFR[1] = AFRH.

6. Initialise all SS pins as general purpose output.
7. Set all SS pins to HIGH.

The next step is to configure the SPI. With the important setting data from the task section all relevant data is known.

**SPI2 initialisation**

8. Initialise the used SPI peripheral with the needed configuration.

> **SPI Init Function**

```
void SPI_init(SPI_TypeDef *spiHandler) {
  // Set SPI off
  CLEAR_BIT(spiHandler->CR1, SPI_CR1_SPE);

  // Set clock polarity and clock phase to zero
  CLEAR_BIT(spiHandler->CR1, SPI_CR1_CPHA);
  CLEAR_BIT(spiHandler->CR1, SPI_CR1_CPOL);

  // Set master mode
  SET_BIT(spiHandler->CR1, SPI_CR1_MSTR);

  // Set baudrate of max 5Mhz: 16MHz / 32 = 0.5 MHz
  MODIFY_REG(spiHandler->CR1, SPI_CR1_BR_Msk, 0b100 << SPI_CR1_BR_Pos);

  // Set recieve only mode
  SET_BIT(spiHandler->CR1, SPI_CR1_RXONLY);
  CLEAR_BIT(spiHandler->CR1, SPI_CR1_BIDIMODE);
  CLEAR_BIT(spiHandler->CR1, SPI_CR1_BIDIOE);

  // Hardware CRC calculation disable
  CLEAR_BIT(spiHandler->CR1, SPI_CR1_CRCEN);

  // Set software slave management and internal slave select
  SET_BIT(spiHandler->CR1, SPI_CR1_SSI);
  SET_BIT(spiHandler->CR1, SPI_CR1_SSM);

  // Set data frame format to 16 bit
  SET_BIT(spiHandler->CR1, SPI_CR1_DFF);
```

```
  // disable ss output
  CLEAR_BIT(spiHandler->CR2, SPI_CR2_SSOE);

  // set spi motorola mode
  CLEAR_BIT(spiHandler->CR2, SPI_CR2_FRF);
}
```

To read data with SPI, several steps are required to get the desired data. The sequence is very important for SPI communication.

**SPI reading**

9. Set the SS pin to low
10. Enable the SPI bus (SPI_CR1_SPE bit).
11. Wait until the SPI_SR_RXNE bit is true.
12. Read out the data from the data register.
13. As the MAX31855 IC sends 4 bytes and the data register of the SPI peripheral only contains 2 bytes, repeat the two previous steps a second time.
14. Disable the SPI bus (SPI_CR1_SPE bit).
15. Set the SS pin to high.

Finally, the chip's current internal temperature must be extracted from the received data, converted to MCU-readable form, and the RGB LED set accordingly.

**Extract temperature**

16. Extract the internal temperature from the received data.
17. Convert the received value in a MCU readable value (i.e. `int16_t`) without the fraction part.
18. Set the desired colour with the temperature.

To finish the laboratory, add a sample frequency with methods you have learnt before.

### Sample frequency

19. Sample the MAX31855 IC every $1\,\mathrm{Hz}$. The system to get this frequency is up to you. Discuss with your colleagues possible solutions.