**Serial peripheral interface (SPI)**

**Modification history of the document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date / Version** | **Status** | **Section** | **Type of modification** | **Author** |
| 16.12.2024 v0.1 | Draft | All | Created initial version of the document. | M. Teletin |

**Index of Contents**

1. Document Management 4

1.1. Purpose of the Document 4

1.2. Referenced Documents 4

1.3. Glossary 4

1.4. Contact Persons 4

2. Workshop description 5

2.1. What is a serial pheriperal interface (SPI) 5

2.1.1. Synchronous serial communication 5

2.1.2. Terminology description 7

2.1.3. Topology 7

2.1.4. Data Transmission 8

2.1.5. Clock Polarity and Clock Phase 9

2.2. Implementation SPI communication using STM Cube IDE 10

2.3. Exercises 10

3. Attachments 12

3.1. Reference 12

# Document Management

## Purpose of the Document

This document describes the activities that will be performed during a specific workshop organized by Preh Romania. It is a summary of the subjects / discussions addressed during the workshop, it also contains links to different useful resources that can be used for gathering additional information on the subjects or for gaining a deeper understanding of the subjects.

## Referenced Documents

|  |  |
| --- | --- |
| **Reference-ID** | **Name of the document, if necessary, incl. version and link** |
| [DS\_STM32F091RC] | [Datasheet STM32F091xB STM32F091xC](https://www.st.com/resource/en/datasheet/stm32f091rc.pdf) |
| [RM0091] | [Reference manual STM32F0x1/STM32F0x2/STM32F0x8 advanced Arm®-based 32-bit MCUs](https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf) |
| [MB1136] | [MB1136-DEFAULT-C05 Board schematic](https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group2/5a/85/d6/9a/34/e2/47/1d/MB1136-DEFAULT-C05_Schematic/files/MB1136-DEFAULT-C05_Schematic.pdf/jcr:content/translations/en.MB1136-DEFAULT-C05_Schematic.pdf) |
| [UM1724] | [User manual STM32 Nucleo-64 boards (MB1136) (UM1724)](https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf) |
| [UM1785] | [User Manual - Description of STM32F0 HAL and low-layer drivers](https://www.st.com/resource/en/user_manual/um1785-description-of-stm32f0-hal-and-lowlayer-drivers-stmicroelectronics.pdf) |

**Table 1: Referenced Documents**

## Glossary

|  |  |
| --- | --- |
| **Abbreviation / Term** | **Explanation** |
| GPIO | General Purpose Input Output |
| HAL | Hardware Abstraction Layer |
| HW | Hardware |
| LED | Light Emitting Diode |
| MCU | Microcontroller unit |
| UM | User Manual |
| VCC | Positive Supply Voltage |

**Table 2: Glossary**

## Contact Persons

|  |  |
| --- | --- |
| **Name** | **Email** |
| Bota Radu | radu.bota@preh.ro |
| Lupu Emanuel | emanuel.lupu@preh.ro |
| Sovaiala Alexandru | alexandru.sovaiala@preh.ro |
| Smailovic Alen | alen.smailovic@preh.ro |
| Capatina Ioana | ioana.capatina@preh.ro |
| Teletin Mihai | mihai.teletin@Preh.ro |
| Iosub Alexandru | mihai-alexandru.iosub@preh.ro |

**Table 3: Contact Person**

# Workshop description

The goal of this workshop is to work with [development board STM32 Nucleo-64](https://www.st.com/en/evaluation-tools/nucleo-f091rc.html#documentation) and understanding the communication interface called SPI.

**Objectives:**

1. Introduce participants to the fundamentals of serial communication buses.
2. Understanding and configurating a serial communication bus(particularly SPI interface).
3. Explore concepts such as master behavior/slave behavior, bus topology.
4. Engage students in practical exercises to reinforce theoretical concepts.

## What is a serial pheriperal interface (SPI)

One of the most widely used interfaces is the serial peripheral interface (SPI), which is frequently used in embedded systems and computers to provide short-range communication between a microcontroller (Mcu) and one or more peripheral integrated circuits. The SPI chips are connected by several transmission lines or signal wires, which allow the microcontroller to communicate with the peripherals and exchange data.   
An interface bus called Serial Peripheral Interface (SPI) is frequently used to transfer data between microcontrollers and tiny peripherals like SD cards, shift registers, and sensors. It makes use of distinct clock and data lines in addition to a select line for selecting the device you want to communicate with.

* Synchronous serial communication protocol
* Master-slave architecture
* Four-wire interface (sometimes three)
* Full-duplex communication (optional)
* Short-distance communication

### Synchronous serial communication

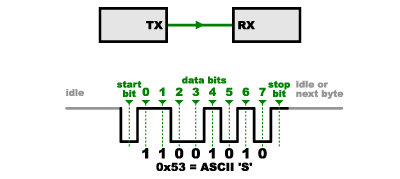
* Serial. Data is sent across a single line, one bit at a time, in sequential order. This is in contrast to parallel communication, in which data is transferred simultaneously across numerous lines. Serial interfaces simplify wiring, allow for longer cables, and reduce interference between wires.
* Synchronous. Data is transmitted in a continuous data stream that is synchronized with a timing signal used by both the transmitter and receiver. Asynchronous transmissions, on the other hand, might be random or irregular and are not timed using a common clock. Synchronous communications are faster and have less overheard than asynchronous communications.

**Asincronus vs syncronus comunicatuion**

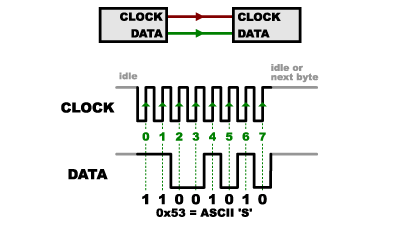
A common serial port with TX and RX lines is referred to as "**asynchronous**" since there is no control over when data is delivered or any guarantee that both sides are running at the same rate. Because computers generally rely on everything being synchronized to a single "clock" (the primary crystal attached to a computer that drives everything), it can be difficult for two systems with slightly differing clocks to interact with one another.

To solve this issue, asynchronous serial connections add extra start and stop bits to each byte, allowing the receiver to sync up with the data as it arrives. **Both parties must** also **agree** on a transmission speed (e.g., 9600 bits per second) in advance.

Asynchronous serial works properly, but there is a significant **overhead** due to the **extra start and stop bits** provided with each byte, as well as the sophisticated gear necessary to send and receive data. If both sides aren't configured to the same speed, the received data is garbage. This is due to the receiver sampling the bits at precise intervals (the arrows in the diagram below). If the receiver is looking at the wrong moment, it will see incorrect bits.

[](https://cdn.sparkfun.com/assets/f/c/6/2/4/52ddb2d5ce395f59658b4567.png)

SPI operates in a somewhat different approach. It's a "**synchronous**" data bus, which means it has **separate lines for data and a "clock**" to maintain both sides in perfect time. The clock is an oscillating signal that instructs the receiver when to sample the bits on the data line. This could be the rising (low to high) or falling (high to low) edge of the clock signal. When the receiver senses that edge, it will instantly check the data line for the next bit (see the arrows in the image below). Because the clock is provided along with the data, defining the speed isn't required. However, devices will have limitations.

[](https://cdn.sparkfun.com/assets/d/6/b/f/9/52ddb2d8ce395fad638b4567.png)

SPI's popularity stems from the fact that the receiving hardware can be as simple as a shift register. This is a much simpler (and less expensive!) piece of hardware than the full-featured UART (Universal Asynchronous Receiver / Transmitter) that asynchronous serial requires.

### Terminology description

SPI components are commonly implemented in a 4-wire configuration. Each wire carries a specific type of signal between the controller and the peripherals. The naming conventions used to describe this configuration varies from one source to other.

* Serial clock (**SCK**) synchronizes data transmission. The clock signal that is generated by the master to provide the timing signal necessary to keep the controller and peripheral in sync. The clock is also referred to as CLK, SCLK or other terms.
* Master Out/Slave In (**MOSI**): The master sends data to the slave., or PICO (peripheral in/controller out) The signal that carries the data from the controller to the peripheral.
* Master In/Slave Out (**MISO**): Slave sends data to the master or (POCI) peripheral out/controller in. The signal that carries the data from the peripheral to the controller
* Chip select (**CS**). The signal determines which peripheral should be made active when the controller needs to communicate with one of multiple peripherals. The signal was commonly referred to as SS (slave select).

### Topology

The simplest SPI configuration consists of one controller and one peripheral. The clock signal (SCK), outbound data signal (MOSI) and chip selection signal transmit data from the controller to the peripheral, as shown in figure below. The inbound data signal (MISO) carries data from the peripheral to the controller. The inclusion of both the MOSI and MISO signal wires makes it possible to support full-duplex communications.

A green and black rectangles

Description automatically generated

SPI Bus –Simple Point-to-Point Topology

SPI supports two multi-device topologies, **daisy-chain** and **star**. Daisy-chain topology splits the clock to route in parallel to the slaves. But, data remains point-to-point. The MISO of one slave goes to the MOSI of another, chaining them together. Data for all the devices clocks through all the devices in a chain similar to boundary scan, each device just picks out the data addressed to them. The final device in the chain drives its MISO to the master.

A black background with green squares with black text

Description automatically generated

SPI Bus – Daisy Chain Topology

In Star topology all the signals are split and routed to each slave in parallel, except chip select. Multiple chip select are used to select individual slave devices. More devices support this mode than daisy-chain.

A black background with green squares with black text

Description automatically generated

SPI Bus – Paralel Topology

### Data Transmission

To initiate communication, the SPI master selects a subdevice by pulling its CS low. (Note: the bar above indicates it is an active low signal, therefore a low voltage means "selected" and a high voltage implies "not selected")  
  
If a waiting period is required, such as during an analog-to-digital conversion, the master processor must wait at least that long before delivering clock cycles.

Each SPI clock cycle, a single bit is transmitted full-duplex. The master sends a bit on the MOSI line, while the slave sends a bit on the MISO line, and then both read their respective incoming bits. This sequence is kept even when just one-directional data transport is intended.

Transmission using a single slave involves one **shift register** in the master and one shift register in the slave, both of some given word size (e.g. 8 bits), connected in a virtual ring topology. Data is usually shifted out with the most-significant bit (MSB) first. On the clock edge, both master and slave shift out a bit to its counterpart.

A close-up of a sign

Description automatically generated

### Clock Polarity and Clock Phase

In SPI, the master can select the clock polarity and clock phase. The CPOL bit sets the polarity of the clock signal during the idle state. The CPHA bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data. The master must select the clock polarity and clock phase, as the requirements of the slave. Depending on the CPOL and CPHA bit selection, four SPI modes are available.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SPI mode** | **Clock polarity (CPOL)** | **Clock phase (CPHA)** | **Data is shifted out on** | **Data is sampled on** |
| 0 | 0 | 0 | falling SCLK, and when CS activates | rising SCLK |
| 1 | 0 | 1 | rising SCLK | falling SCLK |
| 2 | 1 | 0 | rising SCLK, and when CS activates | falling SCLK |
| 3 | 1 | 1 | falling SCLK | rising SCLK |

A diagram of a diagram

Description automatically generated with medium confidence

A diagram of a diagram

Description automatically generated with medium confidence

## Implementation SPI communication using STM Cube IDE

Before starting the implementation, a new project must be created in STM32 Ice Cube IDE based on the previous workshops or the template project from the first workshop must be imported.

The guide uses different functions from the Hardware Abstraction Library provided by the manufacturer of the microcontroller, all the features of this library are described by [UM1785].

## Exercises

1. Configure one of the SPI channels as master- full duplex, and the other channel as slave full-duplex.
   1. Disable Slave Select line(NSS signal), since we are both salve and master, we wont need to select any salve
   2. Select the same parameters for both slave/master (Frame Format – Motorola, Data size: 8 bit, First bit: MSB first).  
      Clock parameters CPOL/CPHA - 0/0 (Mode 0), and for master select a prescaler(e.g 64)  
      Advanced Parameters CRC Calculation – disabled, NSSP Mode – disabled, NSS Signal Type – Software  
      Disabled interrupts for both SPI
   3. Make connections on the Nucleo board(check GPIO settings for each SPI)
   4. Create two buffers, one with data that will be transmitted and one for data received,   
      transmit the “transmit\_buffer” and save it in the “receive\_buffer”, after compering those two buffers, turn on the user LED if the data transmitted is the same as data received.
2. Configure one of the SPI channels as master- full duplex, and the other channel as slave full-duplex.
   1. Disable Slave Select line(NSS signal), since we are both salve and master, we wont need to select any salve
   2. Select the same parameters for both slave/master (Frame Format – Motorola, Data size: 8 bit, First bit: MSB first).  
      Clock parameters CPOL/CPHA - 0/0 (Mode 0), and for master select a prescaler (e.g 64)  
      Advanced Parameters CRC Calculation – disabled, NSSP Mode – disabled, NSS Signal Type – Software  
      Disabled interrupts for master SPI and enable for slave SPI
   3. Activate FreeRTOS and create 3 tasks
   4. In task\_1, verify that the button was pressed
   5. In task\_2, start transmitting on SPI, if the button was pressed in task\_1
   6. Inside the callback Rx interrupt, verify data transmitted on SPI, and set the flag for the task\_3 if received data is the same as transmitted data
   7. In task\_3 togle an led if flag is true

# Attachments

## Reference

1. https://www.techtarget.com/whatis/definition/serial-peripheral-interface-SPI#:~:text=A%20serial%20peripheral%20interface%20(SPI)%20is%20an%20interface%20commonly%20used,peripheral%20integrated%20circuits%20(ICs).
2. https://en.wikipedia.org/wiki/Serial\_Peripheral\_Interface
3. https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all
4. https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html
5. https://practicalee.com/spi/