Fachhochschule
Münster University of
Applied Sciences

Lab for Semiconductor Devices and Bussystems

Exercise Bussystems v. 04-2014
Tutor: MSc. Francisco Estevez

# Exercise 2
# SPI Management

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select lines. Sometimes SPI is called a four-wire serial bus. SPI is often called SSI (Synchronous Serial Interface).

**Interface**
The SPI bus specifies four logic signals:

- SCLK: serial clock (output from master);
- MOSI: master output, slave input (output from master);
- MISO: master input, slave output (output from slave);
- SS: slave select (active low, output from master).

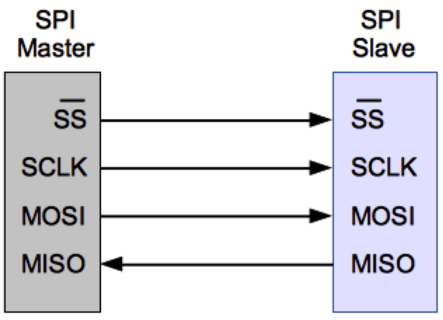


*Image 1: SPI Signals schema*

**Operation**
The SPI bus can operate with a single master device and with one or more slave devices.
If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. With multiple slave devices, an independent SS signal is required from the master for each slave device. Most slave devices have tri-state outputs so their MISO signal becomes high impedance when the device is not selected.

**Data transmission**
A typical hardware setup using two shift registers to form an inter-chip circular buffer. To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1–100 MHz.
The master then transmits the logic 0 for the desired device over slave select line. A logic 0 is transmitted because the slave select line is active low, meaning its off state is a logic 1; on is asserted with a logic 0.
During each SPI clock cycle, a full duplex data transmission occurs:

- the master sends a bit on the MOSI line; the slave reads it from that same line.
- the slave sends a bit on the MISO line; the master reads it from that same line.

Not all transmissions require all four of these operations to be meaningful but they do happen.

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data and the process repeats.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.
Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it needs. However, other word sizes are also common, such as 16-bit words or 12-bit words.
Every slave on the bus that hasn't been activated using its slave select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

**Clock polarity and phase**
A timing diagram showing clock polarity and phase. The red vertical line represents CPHA=0 and the blue vertical line represents CPHA=1. In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. The timing is further described below and applies to both the master and the slave device.

At CPOL=0 the base value of the clock is zero

For **CPHA=0**, data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition).

For **CPHA=1**, data is captured on the clock's falling edge and data is propagated on a rising edge.

At CPOL=1 the base value of the clock is one (inversion of CPOL=0)

For **CPHA=0**, data is captured on clock's falling edge and data is propagated on a rising edge.

For **CPHA=1**, data is captured on clock's rising edge and data is propagated on a falling edge.

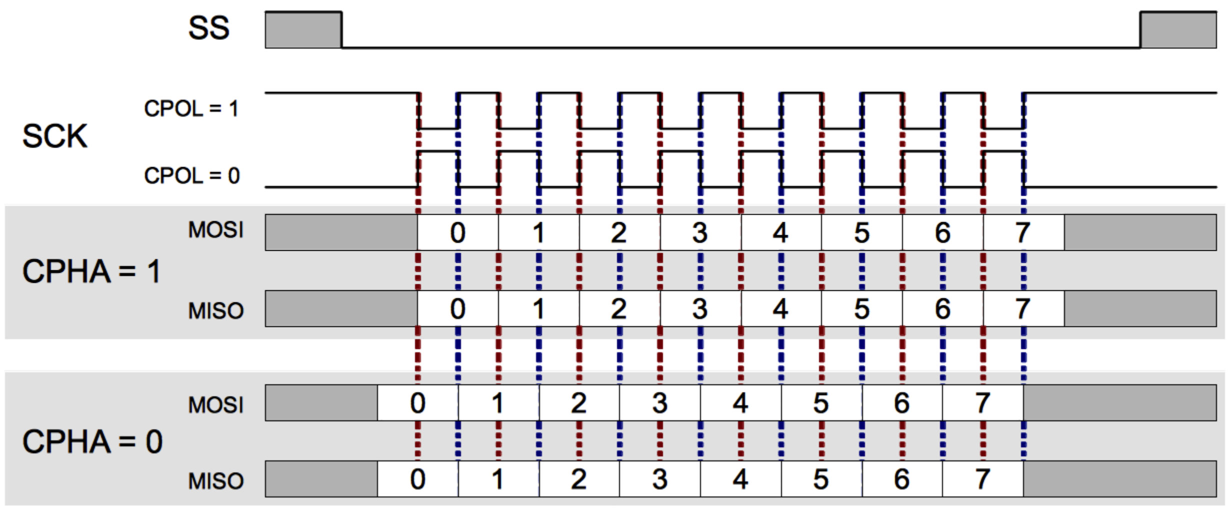


*Image 2: SPI Timing Diagram*

That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling.

Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active.

**Mode numbers**
The combinations of polarity and phases are often referred to as modes which are commonly numbered according to the following convention, with CPOL as the high order bit and CPHA as the low order bit:

- Mode 0: CPOL = 0 and CPHA = 0
- Mode 1: CPOL = 0 and CPHA = 1

- Mode 2: CPOL = 1 and CPHA = 0
- Mode 3: CPOL = 1 and CPHA = 1

**Independent slave SPI configuration (Master and three independent slaves)**
In the independent slave configuration, there is an independent slave select line for each slave. This is the way SPI is normally used. Since the MISO pins of the slaves are connected together, they are required to be tri-state pins.

**Daisy chain SPI configuration (Master and cooperative slaves)**
Some products with SPI bus are designed to be capable of being connected in a daisy chain configuration, the first slave output being connected to the second slave input, etc. The SPI port of each slave is designed to send out during the second group of clock pulses an exact copy of what it received during the first group of clock pulses. The whole chain acts as an SPI communication shift register; daisy chaining is often done with shift registers to provide a bank of inputs or outputs through SPI. Such a feature only requires a single SS line from the master, rather than a separate SS line for each slave.

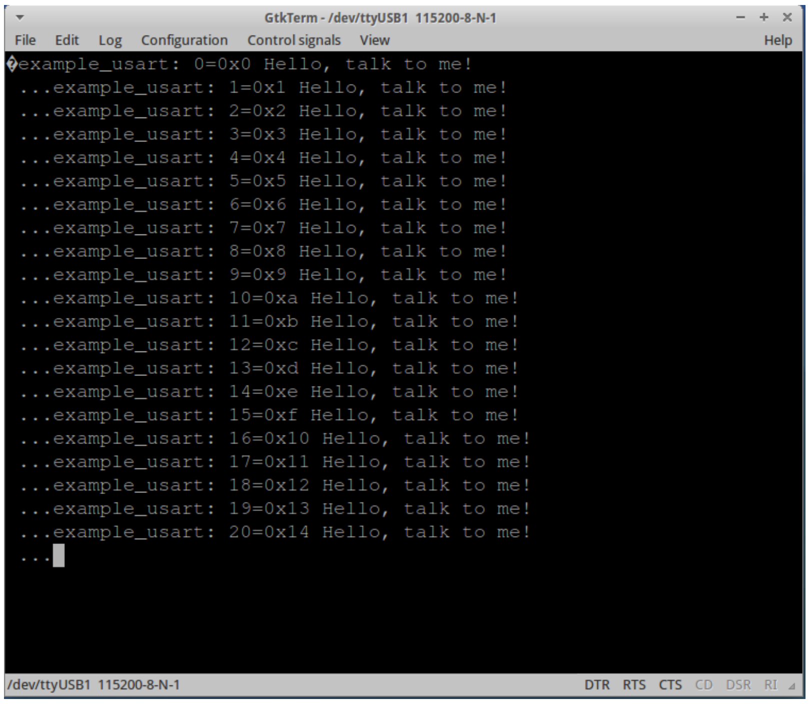
**SPI Exercises:**
To begin to use SPI interface, we will start with our examples in the Toolchain. Later, you should complete exercises depicted below. Files where SPI interface is defined are:

- ttc_spi.h
- ttc_spi_types.h
- ttc_spi.c

To activate SPI library you should do it via *activate_project.sh*, like we saw in latest sessions. You can start using SPI example in the Toolchain, but later you should write your own source code to complete the exercises.

To connect SPI port on Olimex LCD and Olimex P107 you need to find where SPI interface is available. You should review schematic from Olimex, and taking a view on where are mapped these pins (you can find these schematics at https://www.olimex.com/Products/ARM/ST/STM32-P107/resources/STM32-P107-Rev.A-schematic.pdf).
The parameters to configure your UART interface on your virtual terminal are: 115200 bps, no parity, 8 bits, 1 stop bit and no flow control.
You will need to connect your interface and finally you should receive signals from your board on your PC like the following picture.

To manage several interfaces at same time, it is necessary to use several tasks. You can practice it with *example_threads_queues*, this example can give you the keys to work with multiple interfaces and resources. Read it and try to understand it.


GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Control signals View Help

```
�example_usart: 0=0x0 Hello, talk to me!
 ...example_usart: 1=0x1 Hello, talk to me!
 ...example_usart: 2=0x2 Hello, talk to me!
 ...example_usart: 3=0x3 Hello, talk to me!
 ...example_usart: 4=0x4 Hello, talk to me!
 ...example_usart: 5=0x5 Hello, talk to me!
 ...example_usart: 6=0x6 Hello, talk to me!
 ...example_usart: 7=0x7 Hello, talk to me!
 ...example_usart: 8=0x8 Hello, talk to me!
 ...example_usart: 9=0x9 Hello, talk to me!
 ...example_usart: 10=0xa Hello, talk to me!
 ...example_usart: 11=0xb Hello, talk to me!
 ...example_usart: 12=0xc Hello, talk to me!
 ...example_usart: 13=0xd Hello, talk to me!
 ...example_usart: 14=0xe Hello, talk to me!
 ...example_usart: 15=0xf Hello, talk to me!
 ...example_usart: 16=0x10 Hello, talk to me!
 ...example_usart: 17=0x11 Hello, talk to me!
 ...example_usart: 18=0x12 Hello, talk to me!
 ...example_usart: 19=0x13 Hello, talk to me!
 ...example_usart: 20=0x14 Hello, talk to me!
 ...
```

*Image 3: UART example execution*

1. In this exercise you should manage two interfaces at same time. You need to manage USART and SPI interfaces. You should configure USART interface like main system output and use a queue to print everything on your virtual terminal. At same time you should configure your board like SPI master and send exactly the same message via SPI. To pass this exercise you need to check it via our second exercise. The message that you send is free, but you can not repeat it from other groups.

2. For the second exercise you will need to recognize SPI signals on your oscilloscope. You will recognize the message that you sent before via SPI. You need to draw here the signals that you can see on the oscilloscope. For this exercise you must specify, which configuration did you use on your oscilloscope and remark each character over the wave, like we did it before.

MOSI and SCLK Example with "*1\n\r*"

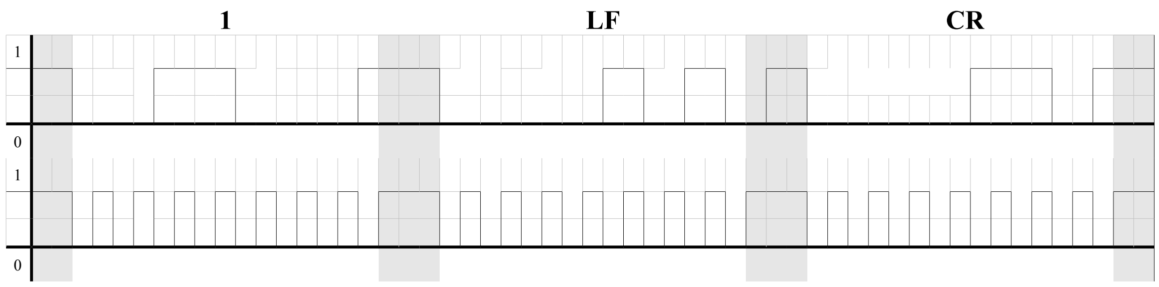
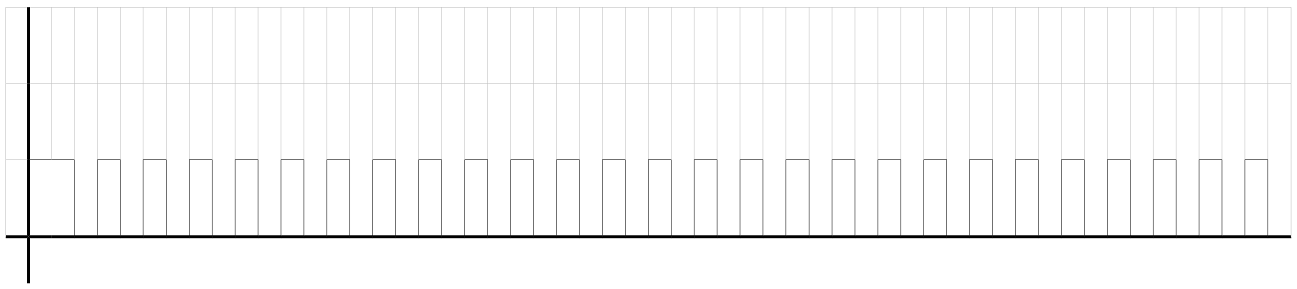


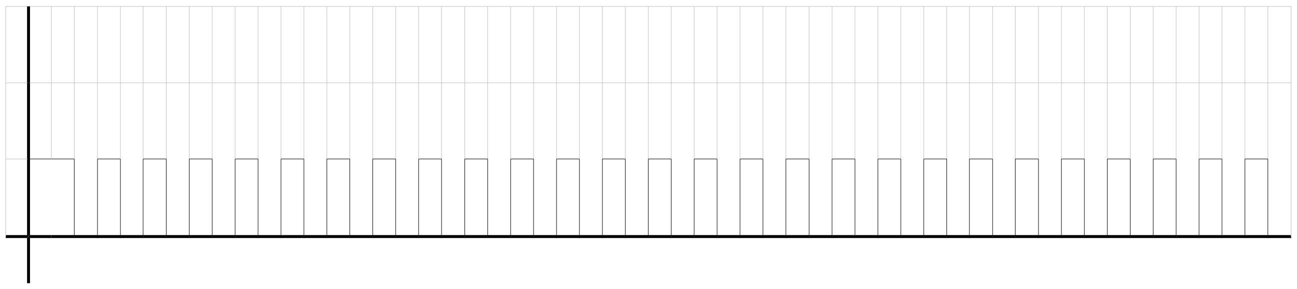
*Chart 1: Number "1" in ASCII code on oscilloscope*

Time division value: 500 ns

MOSI:



Time division value: 500 ns
CPHA: CPOL: MODE:



Time division value: 500 ns
CPHA: CPOL: MODE:

String sent: ___________________

Interesting links to find help with ASCII symbols: http://ascii.cl/
Information from www.en.wikipedia.org

Computer Number: _________ Group: A | B | C

Name: ______________________________ Name: ______________________________

Sign: ______________________________ Sign: ______________________________