Laboratory 5: Serial Communication Interface

Objectives

1. Understand the concept of MCUs communication interface
2. Understand the concept of Serial Peripheral Interface Bus (SPI)
3. Understand the concept of Inter-Integrated Circuit Bus (I^2C)

Serial Peripheral Interface Bus (SPI)

The Serial Peripheral Interface (SPI) is a synchronous serial bus standard with full-duplex capability introduced by Motorola to support communications between a master host processor and one or multiple slave peripheral devices.

SPI is one of the simplest synchronous communications protocols ever developed, as it only establishes a basic mechanism to relay packets between a dedicated master and one or more slaves, without specifying any data, session, or higher-level protocol. This simplicity translates into a protocol with little overhead, capable of achieving a high efficiency in the channel usage, particularly in point-to-point connections.

An SPI controller is developed around a single shift register that serves as both, receiver and transmitter, synchronized by the transmission clock. Figure 1 illustrates the structure of an SPI master-slave pair implementing a point-to point channel. The CPU side of the interface connects to the data lines D0-D7 and the selection and control lines like any other interface, omitted in the figure for simplicity,

The channel side features four signals whose functions are:

- SCLK: Serial clock, sent by the master and synchronizing both master and slave.
- SDO: Serial data-out, the serial output stream from the device.
- SDI: Serial data-in, the serial input stream into the device.
- SS: Slave select, a selection line to enable the slave. The SS line is omitted in point-to-point interconnects, grounding the slave SS input.

A character sent by the CPU is stored in the interface data buffer and copied into the shift register. The master initiates the transfer by activating the slave select signal. In a point-to point, the slave select signal can be hardwired, allowing a 3-wire connection with the master.

Since the master and slave shift registers are tied together and synchronized by the same clock, both interfaces simultaneously transmit and receive. As bits are shifted out from the master they are also shifted into the slave and vice versa. A device doing a transmit-only transfer simply discards the received frame.

Since an SPI does not specify an address field, additional hardware is required for managing multiple slaves. The slave selection lines can be implemented with GPIO lines or an external decoder and user software must activate the corresponding slave. Figure 2 illustrates a single master, multi-slave SPI configuration.

The SPI clock signal can be derived from the system clock, from an external clock source, or from a timer channel, depending on the particular implementation. Interrupt-based servicing is possible in most MCUs. The specific details will depend on the manufacturer's design.

Like other serial formats, SPI transfers using the logic levels of the interface is limited to only a few inches. By adding appropriate drivers and receivers, SPI channels could be extended over longer distances, although it is mostly used for short intra-board distances. SPI has no error checking mechanism and is not defined as an standard.

Due to its simplicity, SPI has been adopted by numerous manufacturers of serial EEPROMs, real-time clock modules, data converters, LCDs, and other peripherals. Due to the absence of upper-level protocols and no standardization, each implementation can define its own particularities, and upper-level protocols are left to the application implementer

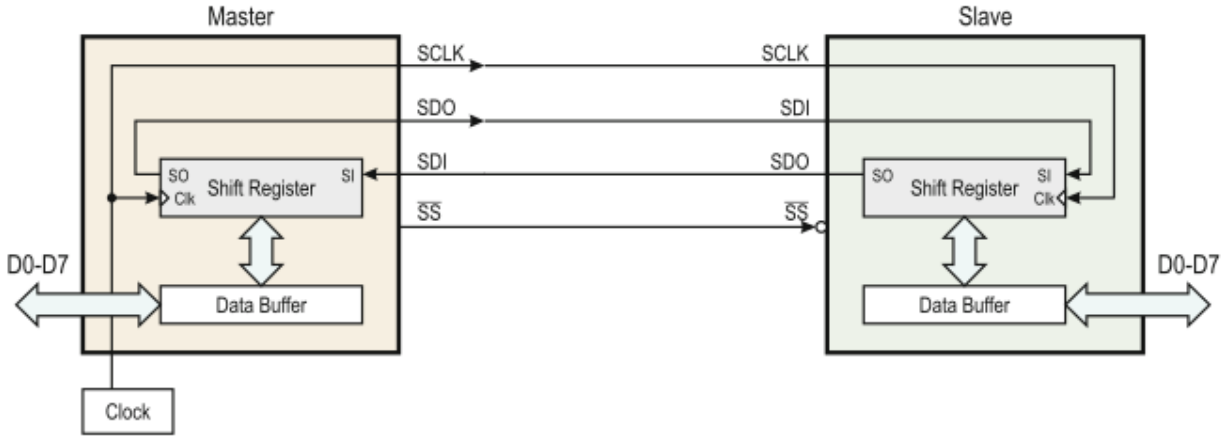


Figure 1: SPI synchronous bus for a point-to-point connection

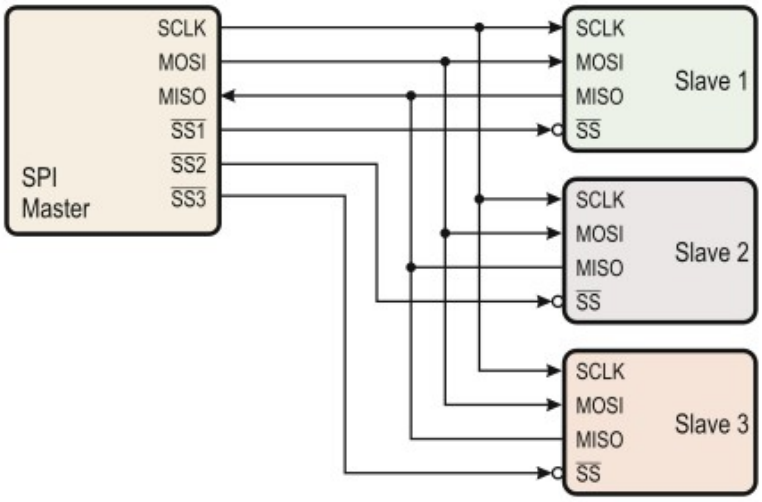


Figure 2: SPI single-master, multi-slave connection

STM32F407 SPI functional description

The block diagram of the SPI is shown in Figure 3.

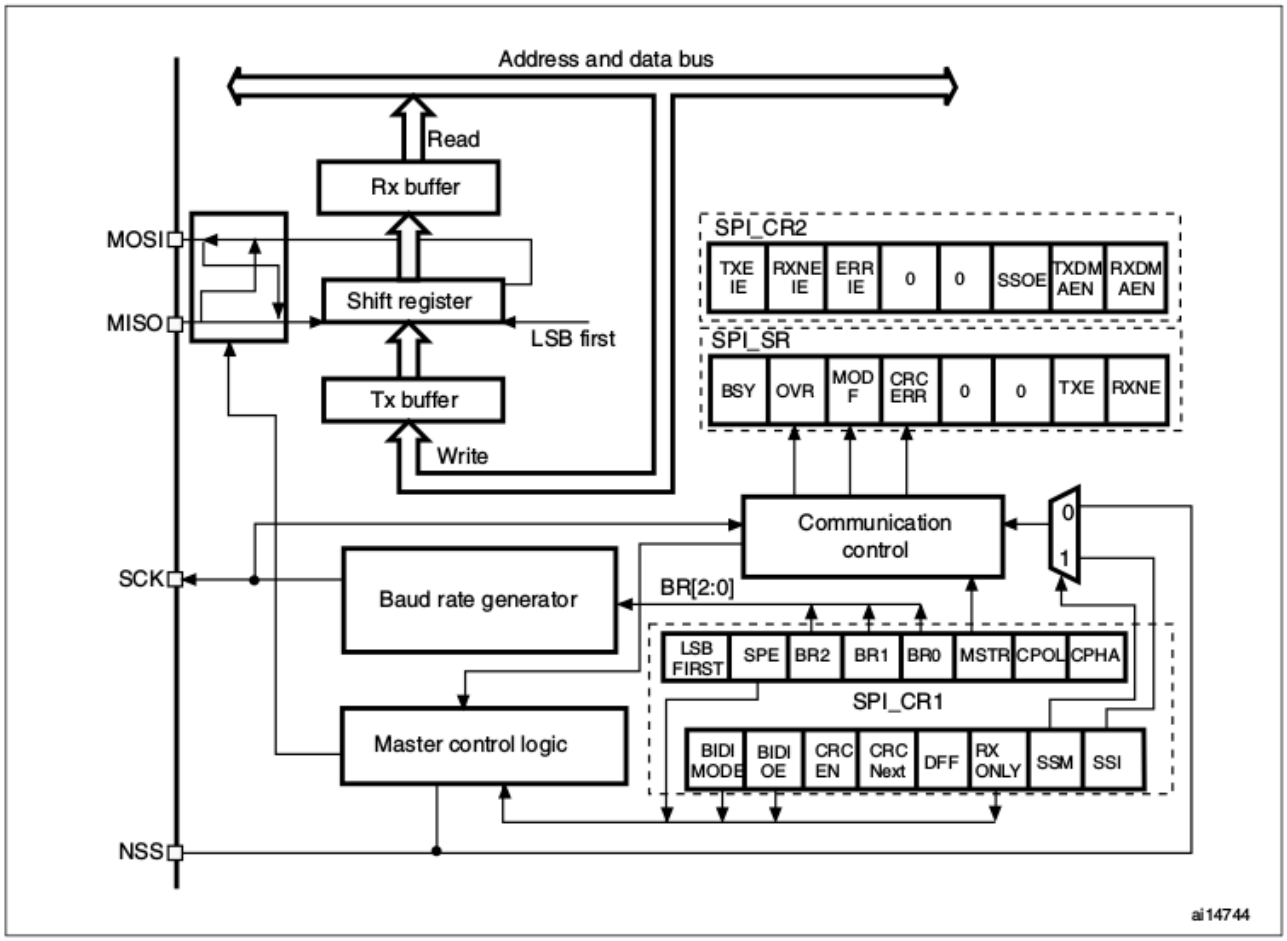
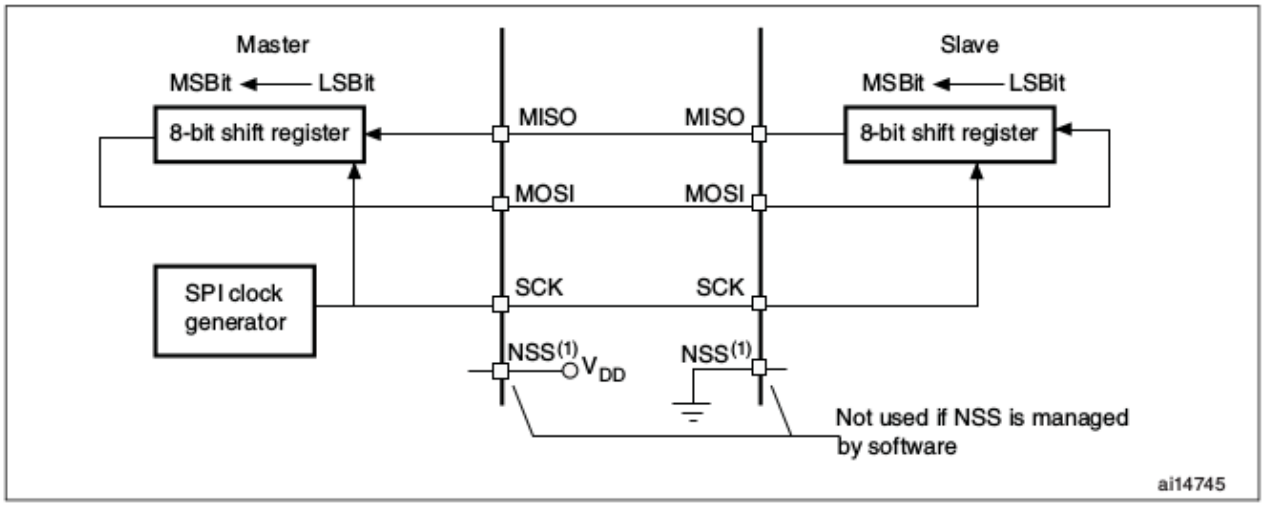


Figure 3: STM32F4 SPI block diagram

Usually, the SPI is connected to external devices through 4 pins:

- MISO: Master In / Slave Out data. This pin can be used to transmit data in slave mode and receive data in master mode.
- MOSI: Master Out / Slave In data. This pin can be used to transmit data in master mode and receive data in slave mode.
- SCK: Serial Clock output for SPI masters and input for SPI slaves.
- NSS: Slave select. This is an optional pin to select a slave device. This pin acts as a 'chip select' to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave NSS inputs can be driven by standard IO ports on the master device. The NSS pin may also be used as an output if enabled (SSOE bit) and driven low if the SPI is in master configuration. In this manner, all NSS pins from devices connected to the Master NSS pin see a low level and become slaves when they are configured in NSS hardware mode. When configured in master mode with NSS configured as an input (MSTR=1 and SSOE=0) and if NSS is pulled low, the SPI enters the master mode fault state: the MSTR bit is automatically cleared and the device is configured in slave mode

A basic example of interconnections between a single master and a single slave is illustrated in Figure 4.



1. Here, the NSS pin is configured as an input.

Figure 4: Single master/ single slave application

The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via the MOSI pin, the slave device responds via the MISO pin. This implies full-duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

Inter-Integrated Circuit Bus (I^2C)

The Inter-Integrated Circuit bus, or I^2C is a synchronous serial protocol developed by Philips Semiconductor (now NXP Semiconductors) in the early 1980s to support board-level interconnection of IC modules and peripherals. The protocol uses two lines, SDA (Serial Data) and SCL (Serial Clock), (and ground) to establish a half-duplex, master/slave, multidrop bus capable of handling multiple masters and

slaves. The serial clock line (SCL) synchronizes all bus transfers, while SDA carries the data being transferred.

I^2C was designed to be an intra-system serial bus capable of accommodating all kind of peripherals found in embedded systems. These include MCUs, data converters (ADCs and DACs), display devices, memories, real-time clock calendars, GPIO modules, etc. A large number of IC peripheral manufacturers offer products compatible with I^2C.

Devices in an I^2C bus are software addressable, with 7- or 10-bit address fields. Although the most common usage establishes a single master/multiple slave topology, the protocol allows for any device to be a master, as it incorporates collision detection and arbitration mechanisms necessary for a multi-master operation. Nominal maximum speeds can reach up to 5 Mbps, although in reality the limit is imposed by the total bus capacitance, with its maximum specified at 400 pF or 500 pF depending on the version. With input capacitances at 10 pF, plus that of cables, practical numbers call for a few dozen devices per bus. Bus extenders might be used to expand that number if the application requires doing so.

Being a bus for interconnecting ICs in a board, distances in I^2C are expected to be short, in the order of a few inches. It might be possible to stretch its length to several feet, at the expense of reduced speed due to the effect of the increased capacitance.

The bidirectional multidrop capability of the SCL and SDA lines is achieved by driving them with open-collector or open-drain drivers. This calls for fitting bus lines pull-up resistors to complete their driver circuit. Figure 5 shows an I^2C bus topology featuring an MCU as master and several other devices as slaves.

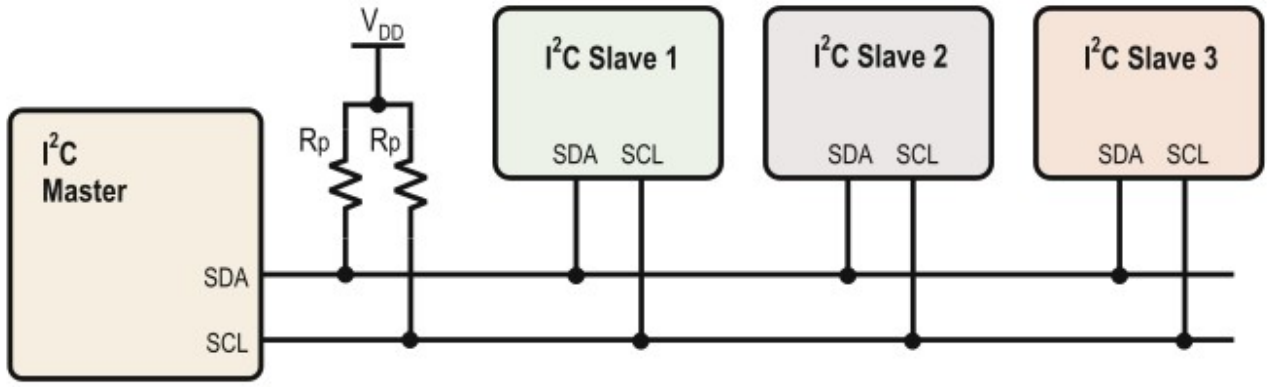


Figure 5: Topology of an I^2C bus connection

STM32F4 I²C functional description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I²C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz) or fast (up to 400 kHz) I²C bus.

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

Communication flow

In Master mode, the I²C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to Figure 6.

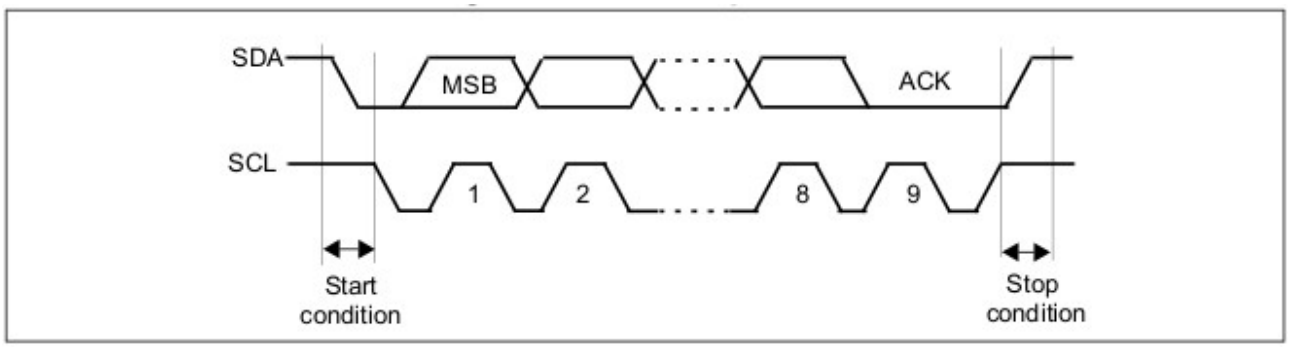


Figure 6: I²C bus protocol

Acknowledge may be enabled or disabled by software. The I²C interface addresses (dual addressing 7-bit/ 10-bit and/or general call address) can be selected by software.

The block diagram of the I²C interface is shown in Figure 7.

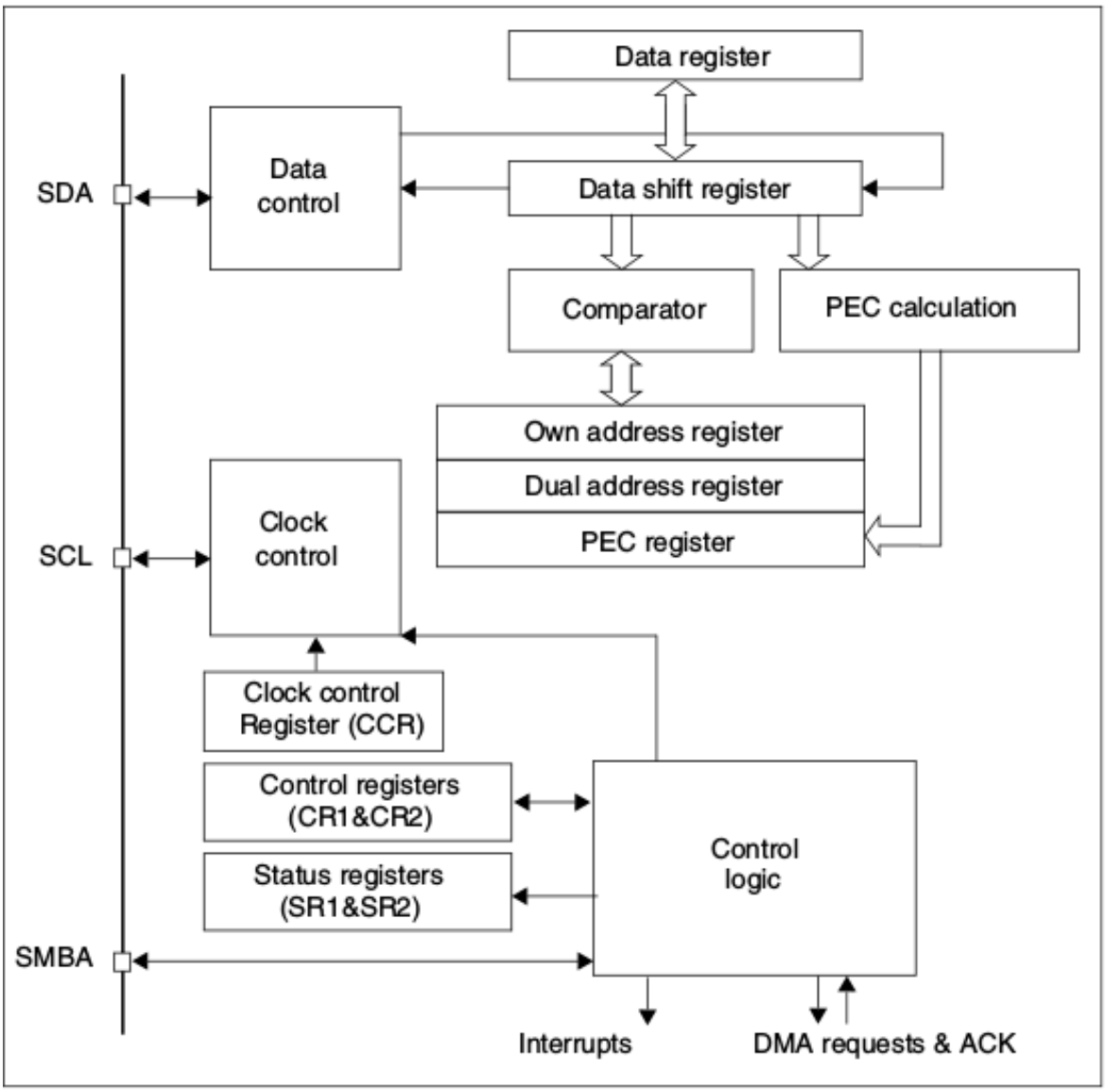


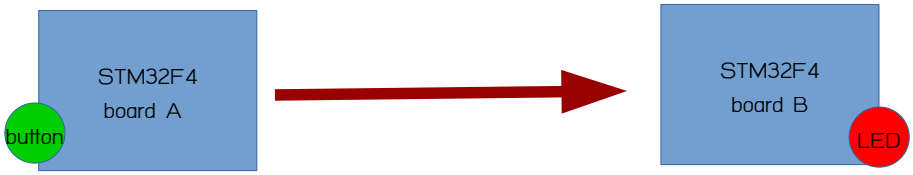
Figure 7: I²C block diagram for STM32F40x/41x

Source: Introduction to Embedded Systems Using Microcontrollers and the MSP430 , Jiménez, Manuel, Palomera, Rogelio, Couvertier, Isidoro

Reference Manuals – RM0090: STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs

For each exercise, you may choose to do one of the following

Lab Exercises – If you have two boards, or come to the lab, you can work in Pair



1. Create a new STM32 Project to transfer a data between MCUs using UART. If User button on STM32 board A is pressed, the LED on STM32 board B must be toggled.
2. Create a new STM32 Project to transfer a data between MCUs using SPI. If User button on STM32 board A is pressed, the LED on STM32 board B must be toggled.
3. Create a new STM32 Project to transfer a data between MCUs using I^2C. If User button on STM32 board A is pressed, the LED on STM32 board B must be toggled.

Lab Exercises – If you stay at home, or want to work alone

1. Create a new STM32 Project to transfer a data between MCUs using two UARTs. If User button on STM32 the board is pressed, the board send data through one UART and a jumper cable to another UART of the STM32, which turn the LED on must be toggled. You must show the code that it is using the data transmission from one UART to another in order to toggle the LED. You may use FreeRTOS, but it would be easier to use the interrupt on the receiver UART.
2. Create a new STM32 Project to transfer a data between MCUs using two SPIs. If User button on STM32 the board is pressed, the board send data through one SPI and jumper cables (SCLK, MOSI, MISO) to another SPI of the STM32, which turn the LED on must be toggled. You must show the code that it is using the data transmission from one SPI to another in order to toggle the LED. You may use FreeRTOS, but it would be easier to use the interrupt on the receiver SPI.

3. Create a new STM32 Project to transfer a data between MCUs using two I^2C. If User button on STM32 the board is pressed, the board send data through one SPI and jumper cables (SCK, SCL) to another I^2C of the STM32, which turn the LED on must be toggled. You must show the code that it is using the data transmission from one I^2C to another in order to toggle the LED. You may use FreeRTOS, but it would be easier to use the interrupt on the receiver I^2C .