Management and analysis of physics datasets, Part. 1

Eighth Laboratory

Stefano Pavinato
9/1/2019

INFN
Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali di Legnaro

Outline

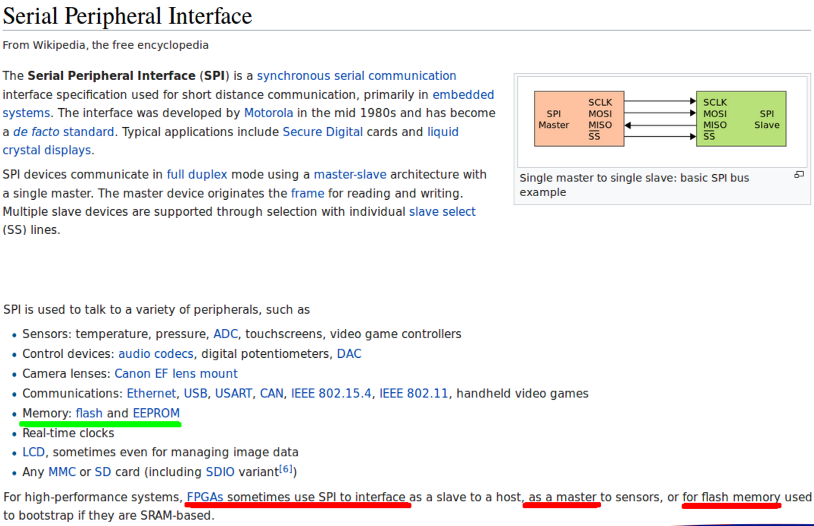
Outline

Goals

- Become familiar with SPI protocol.
- Serial Flash Memory as SPI slave.

Outline

SPI - Wiki

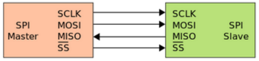


Serial Peripheral Interface

From Wikipedia, the free encyclopedia

The **Serial Peripheral Interface** (**SPI**) is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid 1980s and has become a *de facto* standard. Typical applications include Secure Digital cards and liquid crystal displays.

SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.



Single master to single slave: basic SPI bus example

SPI is used to talk to a variety of peripherals, such as

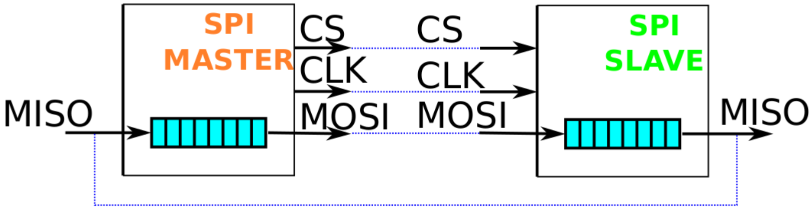
- Sensors: temperature, pressure, ADC, touchscreens, video game controllers
- Control devices: audio codecs, digital potentiometers, DAC
- Camera lenses: Canon EF lens mount
- Communications: Ethernet, USB, USART, CAN, IEEE 802.15.4, IEEE 802.11, handheld video games
- Memory: flash and EEPROM
- Real-time clocks
- LCD, sometimes even for managing image data
- Any MMC or SD card (including SDIO variant[6])

For high-performance systems, FPGAs sometimes use SPI to interface as a slave to a host, as a master to sensors, or for flash memory used to bootstrap if they are SRAM-based.

SPI (1)

- SPI (Serial Peripheral Interface) :
  - it is serial: the data is transmitted one bit at a time;
  - it is synchronous: the transmission of the data is imposed by the clock;
  - it has an architecture master/slave(s);
- It is a basic serial protocol (not the most).
- It is mostly used for the communication between μC or FPGA and ICs like: A/D, D/A converters, sensors, memories ...

SPI (2)



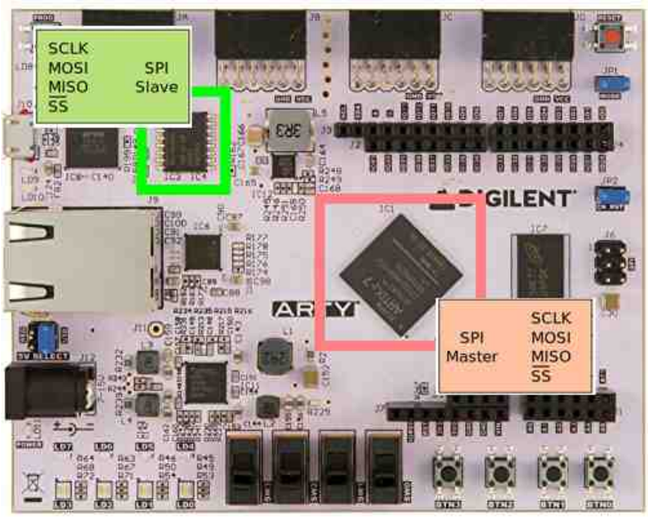
- The data are exchanged between a master and a slave.
- Essentially each device has inside it a shift register with the data. The data transmission "exchanges" the data between the shift registers.
- The Master addresses the Slave and it manages the data transmission with the clock (on the **rising edge**).

SPI (3)

SPI signals (from the slave side):

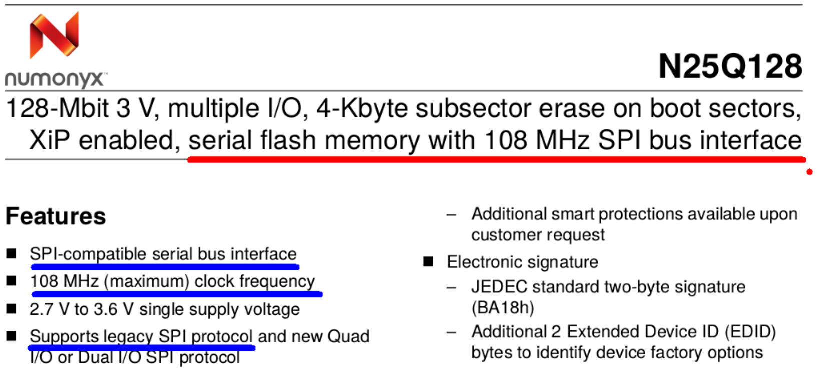
- **CS** or **SS**: Chip (Slave) Select. When this input signal is low, the device is selected.
- **CLK**: Clock. This input signal provides the timing for the serial interface. Instructions, addresses, or data present at the MOSI are latched (evaluated) on the rising edge of the clock.
- **MOSI**: Master Output Slave Input. This input signal is used to transfer data serially into the device. It receives instructions, addresses, and the data to be programmed.
- **MISO**: Master Input Slave Output. This output signal is used to transfer data serially out of the device.

SPI (4)


SCLK
MOSI
MISO
$\overline{SS}$
SPI Slave
SPI Master
SCLK
MOSI
MISO
$\overline{SS}$
DIGILENT
ARTY

Outline

Flash Memory



numonyx

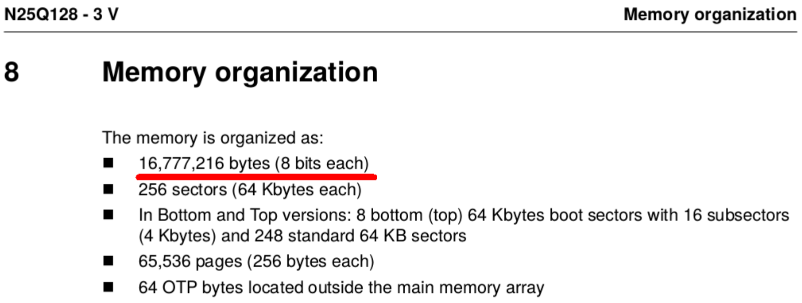
N25Q128

128-Mbit 3 V, multiple I/O, 4-Kbyte subsector erase on boot sectors, XiP enabled, serial flash memory with 108 MHz SPI bus interface

Features

- SPI-compatible serial bus interface
- 108 MHz (maximum) clock frequency
- 2.7 V to 3.6 V single supply voltage
- Supports legacy SPI protocol and new Quad I/O or Dual I/O SPI protocol
  - Additional smart protections available upon customer request
- Electronic signature
  - JEDEC standard two-byte signature (BA18h)
  - Additional 2 Extended Device ID (EDID) bytes to identify device factory options

Flash Memory - Organization



N25Q128 - 3 V   **Memory organization**

8 Memory organization

The memory is organized as:

- 16,777,216 bytes (8 bits each)
- 256 sectors (64 Kbytes each)
- In Bottom and Top versions: 8 bottom (top) 64 Kbytes boot sectors with 16 subsectors (4 Kbytes) and 248 standard 64 KB sectors
- 65,536 pages (256 bytes each)
- 64 OTP bytes located outside the main memory array

- $16777216 = 2^{24}$;
- Therefore the address of each 8-bit register is a 24-bit address.

Flash Memory - Read (1)

To read the memory content in SPI protocol different instructions are available: READ, Fast Read, Dual Output Fast Read, Dual Input Output Fast Read, Quad Output Fast Read and Quad Input Output Fast read, allowing the application to choose an instruction to send addresses and receive data by one, two or four data lines.

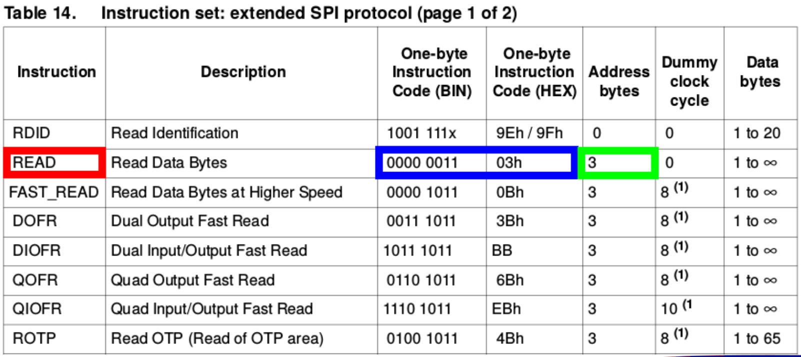
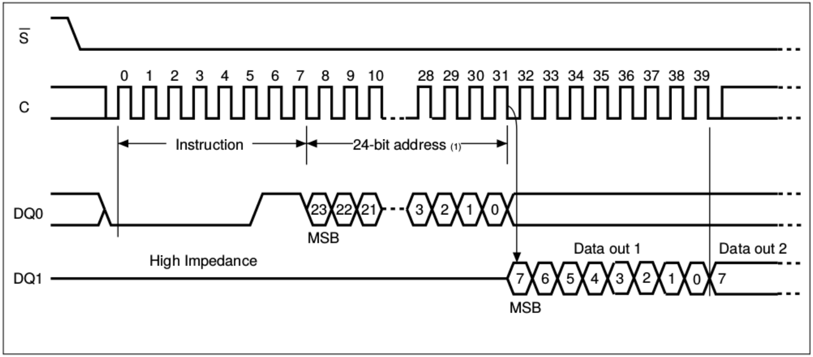


Table 14. Instruction set: extended SPI protocol (page 1 of 2)

Instruction	Description	One-byte Instruction Code (BIN)	One-byte Instruction Code (HEX)	Address bytes	Dummy clock cycle	Data bytes
RDID	Read Identification	1001 111x	9Eh / 9Fh	0	0	1 to 20
READ	Read Data Bytes	0000 0011	03h	3	0	1 to ∞
FAST_READ	Read Data Bytes at Higher Speed	0000 1011	0Bh	3	8 (1)	1 to ∞
DOFR	Dual Output Fast Read	0011 1011	3Bh	3	8 (1)	1 to ∞
DIOFR	Dual Input/Output Fast Read	1011 1011	BB	3	8 (1)	1 to ∞
QOFR	Quad Output Fast Read	0110 1011	6Bh	3	8 (1)	1 to ∞
QIOFR	Quad Input/Output Fast Read	1110 1011	EBh	3	10 (1	1 to ∞
ROTP	Read OTP (Read of OTP area)	0100 1011	4Bh	3	8 (1)	1 to 65

Flash Memory - Read (2)

Figure 11. Read Data Bytes instruction and data-out sequence



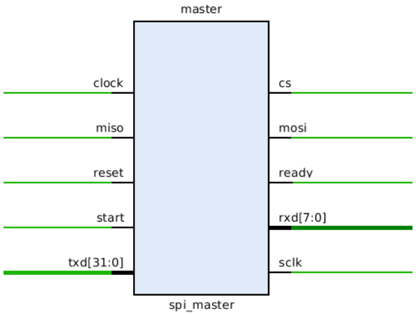
Flash Memory - Read (3)

- The device is selected by driving the CS low.
- The instruction code for the READ instruction (00000011_2) is sent. Each bit is latched-in (evaluated) on the rising edge of the clock.
- The 3 bytes address (23,22,...,1,0) are then sent. Each bit is latched-in (evaluated) on the rising edge of the clock.
- Then the memory content, at that address, is shifted out on the MISO, each bit is sent, on the falling edge of Serial Clock (C). But evaluated by the Master on the following rising-edge.
- The READ instruction is terminated by driving the CS high.

Outline

Homework

- **You have to write the VHDL code to implement an SPI master, in order to read the data stored in a 8-bit register of the serial flash memory.**



Hmw - Master signals (1)

- clock: the clock provided by the oscillator mounted on the evaluation board;
- miso;
- reset: provided by the VIO. It is used to reset the FSM;
- start: provided by the VIO. It is used to start the READ operation, on the rising edge of the signal;
- txd: a 32 bit signals. It is formed by the the instruction byte and the address 3 bytes. The address (or rather the last 4 significant bits) is provided by the VIO;

Hmw - Master signals (2)

- cs: when you have received the 8 bits stored in a register, after the read operation, the CS has to switch from '0' to '1';
- mosi;
- ready: the ready is a pulse '0' $\rightarrow$ '1' $\rightarrow$ '0' that happens when you have received the 8 bits stored in a register, after the read operation;
- rxd: a 8 bit signals. It is formed by the 8 bits obtained by the read operation;
- sclk: is the clock generated by the master, that synchronizes the data transmission.

Memory Configuration

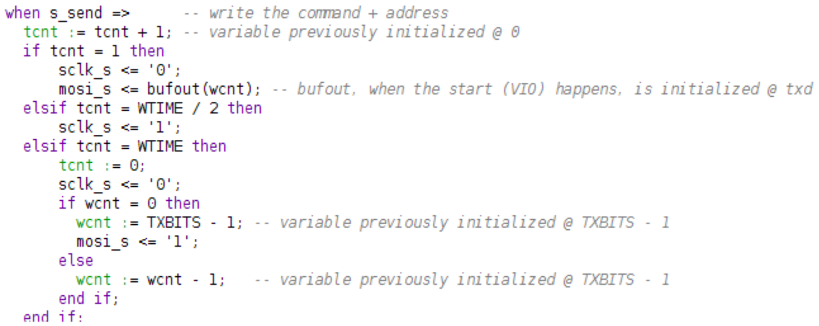
- The flash memory must be configured using the .mcs file in the folder *flash_configuration*. With this .mcs file you are going to write the first six flash memory locations with these values:
  1. @ address 0x000000 $\Rightarrow$ 0x00;
  2. @ address 0x000001 $\Rightarrow$ 0x01;
  3. @ address 0x000002 $\Rightarrow$ 0x02;
  4. @ address 0x000003 $\Rightarrow$ 0x03;
  5. @ address 0x000004 $\Rightarrow$ 0x04;
  6. @ address 0x000005 $\Rightarrow$ 0x05.
- **0x** stands for hexadecimal value.
- The configuration of the flash memory using in the .mcs file is described in the section 3 of the third laboratory.

Hints (1)

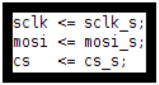
- You can download all the folder *Lab8*, open the project (the file .xpr) and work there.
- You have to write the code **only** inside the file *spi_master.vhd*.
- The SPI master can be implemented as a 3 state FSM.
- **Remember** the testbenches !!!

Hints (2)

You can use this code as an inspirational source for you code. It is a **partial** implementation of a state of the FSM. In particular in this state the txd word is transmitted to the slave.



```
when s_send =>       -- write the command + address
  tcnt := tcnt + 1; -- variable previously initialized @ 0
  if tcnt = 1 then
      sclk_s <= '0';
      mosi_s <= bufout(wcnt); -- bufout, when the start (VIO) happens, is initialized @ txd
  elsif tcnt = WTIME / 2 then
      sclk_s <= '1';
  elsif tcnt = WTIME then
      tcnt := 0;
      sclk_s <= '0';
      if wcnt = 0 then
        wcnt := TXBITS - 1; -- variable previously initialized @ TXBITS - 1
        mosi_s <= '1';
      else
        wcnt := wcnt - 1;   -- variable previously initialized @ TXBITS - 1
      end if;
  end if;
```

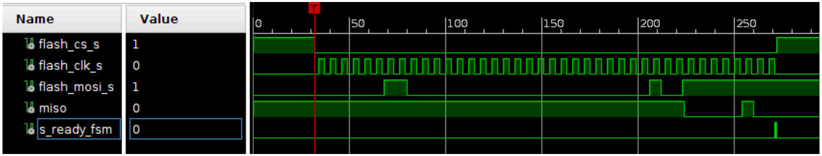


```
sclk <= sclk_s;
mosi <= mosi_s;
cs   <= cs_s;
```

Hints (3)

It might be very useful exploit the ILA core(s) in order to monitor the state of the FSM, the internal signals and the signals representing the SPI master inputs and output.

For the final check instantiates an ILA core, where you monitor the CS, the MOSI, the MISO and the SCLK. You must get a temporal diagram very similar to the diagram of the picture/ slide 15. Monitor also the "ready" signal.


Name	Value
flash_cs_s	1
flash_clk_s	0
flash_mosi_s	1
miso	0
s_ready_fsm	0