FYS4220/9220

Lab Exercise 3
VHDL – ADC control

Introduction to the lab

The purpose of this third lab exercise is to get experience with a complete VHDL design, including the design of the ASM chart for your state machine. You will also have to extract some of the information from a data sheet.

You are going to design an ADC (analog-to-digital converter) controller, using a state machine in VHDL.

Timing analysis is described in the document called "***tut_timing.pdf***".

A lab report must be created, including the required material. The required hand-in for each part of the lab is specified in the lab text.

1. ADC control using a state machine

In this assignment you are going to make a controller for the analog-to-digital converter (ADC) ***MCP3204*** from Microship, see the datasheet for this ADC. This is a 4 channel, 12-bit successive approximation (SAR) ADC.

The Communication with the ADC is accomplished using a simple serial interface compatible with the SPI protocol, and the serial interface timing is shown in Figure1.

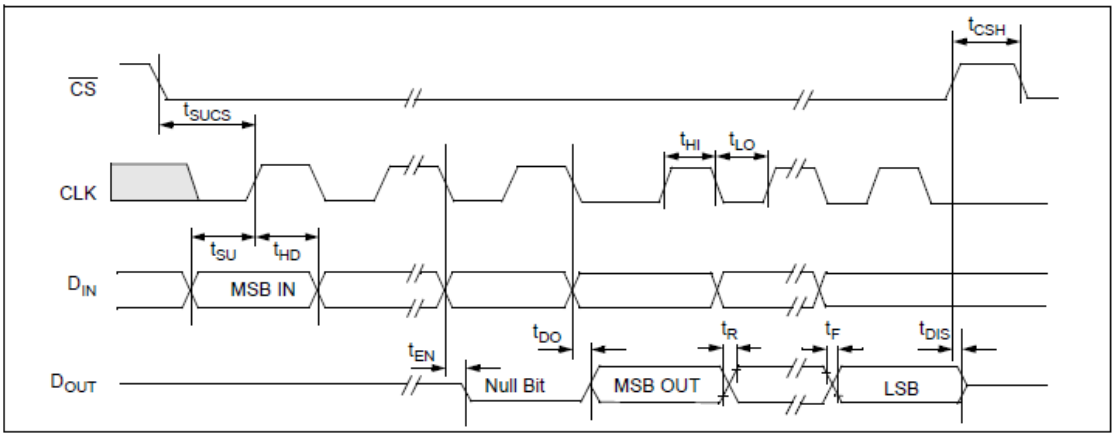

FIGURE 1-1: *Serial Interface Timing.*

Figure 1

The serial communication for controlling the ADC (setup) and reading data from the ADC is described in section 5.0 of the data sheet, and illustrated in Figure 5-1 (Figure 2 below).

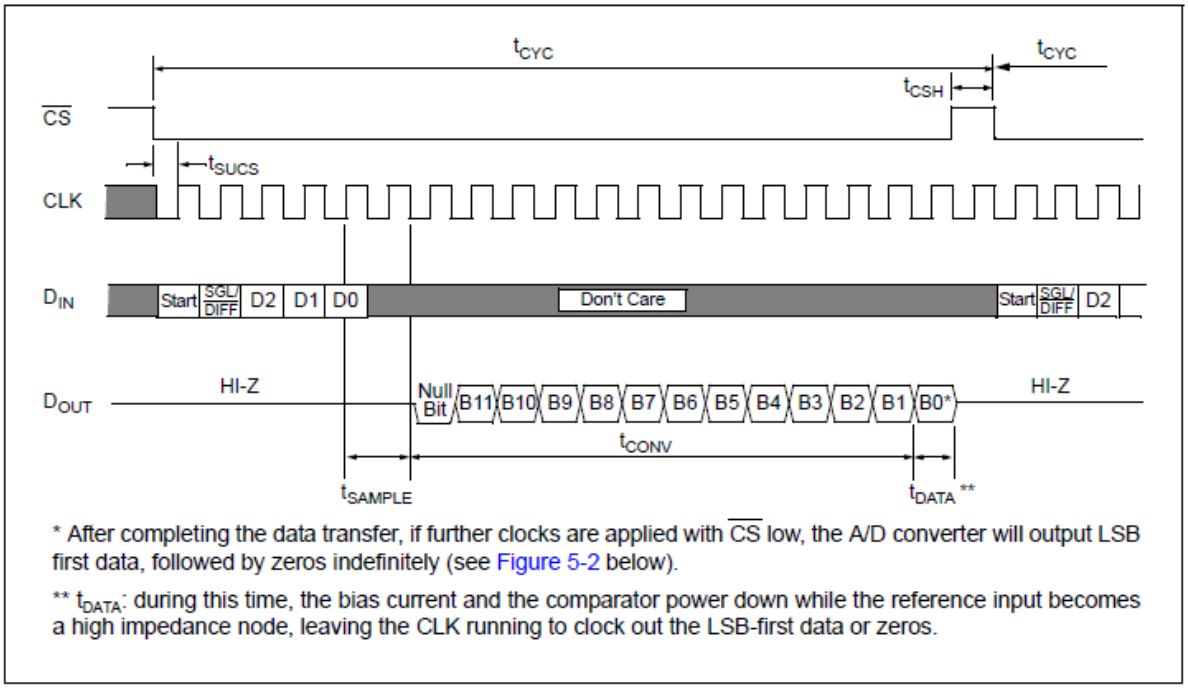

FIGURE 5-1: *Communication with the MCP3204 or MCP3208.*

Figure 2

The ADC is going to work with the following setup:

- Single ended configuration (Single/Diff set to '1')
- The controller must read data from CH0, CH1 and CH2 (3 channels)
- When a conversion is initiated CH2 should be read first, then CH1 and finally CH0

The entity of the controller is shown below:

```
entity MCP3204 is

  port
    (
    sclk        : in    std_logic;                      -- System clock 50 MHz
    start_conv  : in    std_logic;                      -- Start conversion when high
    Dout        : in    std_logic;                      -- Serial data from ADC
    nCS         : out   std_logic;                      -- CS is active low
    Din         : out   std_logic;                      -- Data in to ADC (setup)
    clk         : out   std_logic;                      -- ADC clock 1 MHz
    data_CH0    : out   std_logic_vector(11 downto 0);  -- Data from ADC CH0
    data_CH1    : out   std_logic_vector(11 downto 0);  -- Data from ADC CH1
    data_CH2    : out   std_logic_vector(11 downto 0);  -- Data from ADC CH2
    DataReady   : out   std_logic                       -- High when all data CH  ready
    );
end MCP3204;
```

The system clock called **sclk** (input to the controller) is assumed to be 50 MHz, and it is assumed that the required ADC clock (clock signal out to the ADC) called **clk** is 1 MHz. (You can see in the data sheet that the maximum input clock frequency of this ADC is 2 MHz). Therefore, a clock divider must be inserted between the sclk input and the clk output. The **start_conv** input is an active high signal that is used to start the conversion. Therefore, the controller must wait for this input to go high, and then make the ADC start the conversion process. Since the data from the ADC is on a serial format with 12 bits per sample a 12 bit shift register has to be used on the **Dout** input (serial data from the ADC). When all 12 bits from a sample are loaded into the shift register these data are to be made available at the output signals called **data_CHO**, **data_CH1** and **data_CH2** , respectively. In addition, the signal **DataReady** has to be set to a logic high value ('1'), in order to indicate that a new set of data samples are ready from the three data channels. **DataReady** should not be set to a logic high value before data are available at all three channels (**data_CHO**, **data_CH1** and **data_CH2)**. **DataReady** should be logic high ('1') only for one clock period, and then go back to logic low ('0').

A block diagram of the ADC controller implementation is shown in Figure 3. The data from the ADC, which has been read into the serial-in-parallel-out shift register called **SR_SerIn_redge** (see Figure 3) can be routed to the parallel data outputs of the controller in the following way (in a state in the FSM):

```
Data_CH0 <= Pdata;
DataReady <= '1';
```

The controller is going to be implemented as a ***state machine***, and the state machine has to reproduce the waveform in Figure 5-1 in the data sheet (Figure 2 in the lab text) with the requirements as described above. Note that **nCS** should be pulled high again when the last bit (B0) of a 12 bit sample has been read, see Figure 5.1

.

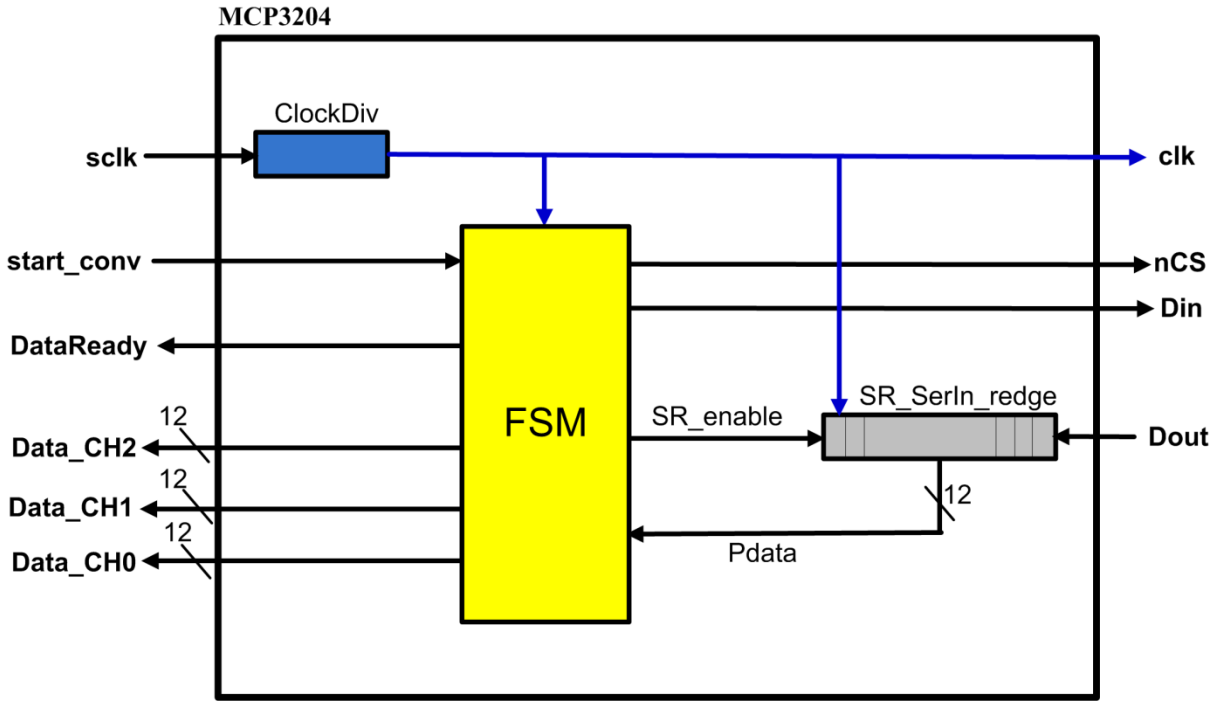

Figure 3

Perform the following steps:

1. Open the provided files **SR_SerIn_redge (**a generic shift register reading data in on the rising clock edge, when enabled). The shift register uses the rising edge of the clock signal, such that the serial data from the ADC are read (shifted in) ”in the middle of the converted data bit”. Make sure that you understand how the shift register works, based on the VHDL code.
2. Create a clock divider component, and name the file **ClockDiv.vhd**. The input clock is assumed to be 50 MHz, and the output clock should be a 1 MHz clock with a duty cycle of close to 50 %.
3. Perform a functional simulation to verify the design of the clock divider.
4. Read the data sheet for the ADC (mcp3204.pdf)
5. Design the state machine (by drawing the ASM chart with pen and paper) for the ADC controller, according to the specification in the lab text (Figure 2 and the explanation of the signals) and the Data sheet for the ADC. Specify the output for each state e.g. in the ASM chart or in a table. (Hint: should the process in the FSM

trigger on the rising edge or falling edge, based on Figure 5.1 and the datasheet for the ADC?).

6. Create a VHDL file called **MCP3204.vhd** and write the code for the ADC controller, based on your ASM chart and the block diagram in Figure 3. Include the serial shift register **SR_SerIn_redge.vhd** and the clock divider **ClockDiv.vhd** as components in your design.
7. Create a new Quartus II project called Lab3, and add your design files.
8. Check your design (MCP3204.vhd**)** for errors (***Start Compiling*** or ***Start Analysis & Elaboration***), and correct any errors.
9. Make a testbench to test your ADC controller; name the file **tb_MCP3204.vhd**
10. Perform a functional (RTL level) simulation using the Modelsim-Altera simulator, in order to verify the correct behaviour of the circuit (according to Figure 5.1/Figure 2 and the specification given in the lab).
11. Timing analysis:
    a. Q1: What is the maximum clock frequency of your design?

Required hand-in:

- **ClockDiv.vhd**
- A Print screen image of the clock divider simulation
- ASM chart for the state machine (the yellow component in the block diagram of Figure 3)
- **MCP3204.vhd**
- **tb_MCP3204.vhd**
- A screen shot of the functional simulation of the ADC controller (MCP3204.vhd)
- Answer to Question Q1