EL203 Project Report

Lab Group: Group1

Question No: 2

Subject: Synchronous Serial Peripheral Interface (SPI)

-Assigned by Prof. Biswajit Mishra

Members

Student ID	Name
201701006	Swarnaditya Maitra
201701007	Sanyam Desai
201701010	Milan Nagariya
201701011	Anish Deshpande

Introduction to the project

This document describes our effort to simulate the design of the Serial Peripheral Interface (SPI). SPI provides synchronized communication between a microcontroller and peripheral devices or other microcontrollers.

This report will proceed by first describing the assumptions made on the design, followed by an elaborate block diagram of SPI. The logic flow for VHDL is described by the design of a State-Machine Chart, and then we finally look at the behavioral design (VHDL code snippets) of the SPI implementation.

Our Decisions and Simplifications

We have made a few simplifications and modifications to the circuitry, which will later also be schematically seen in the block diagrams and SM-charts.

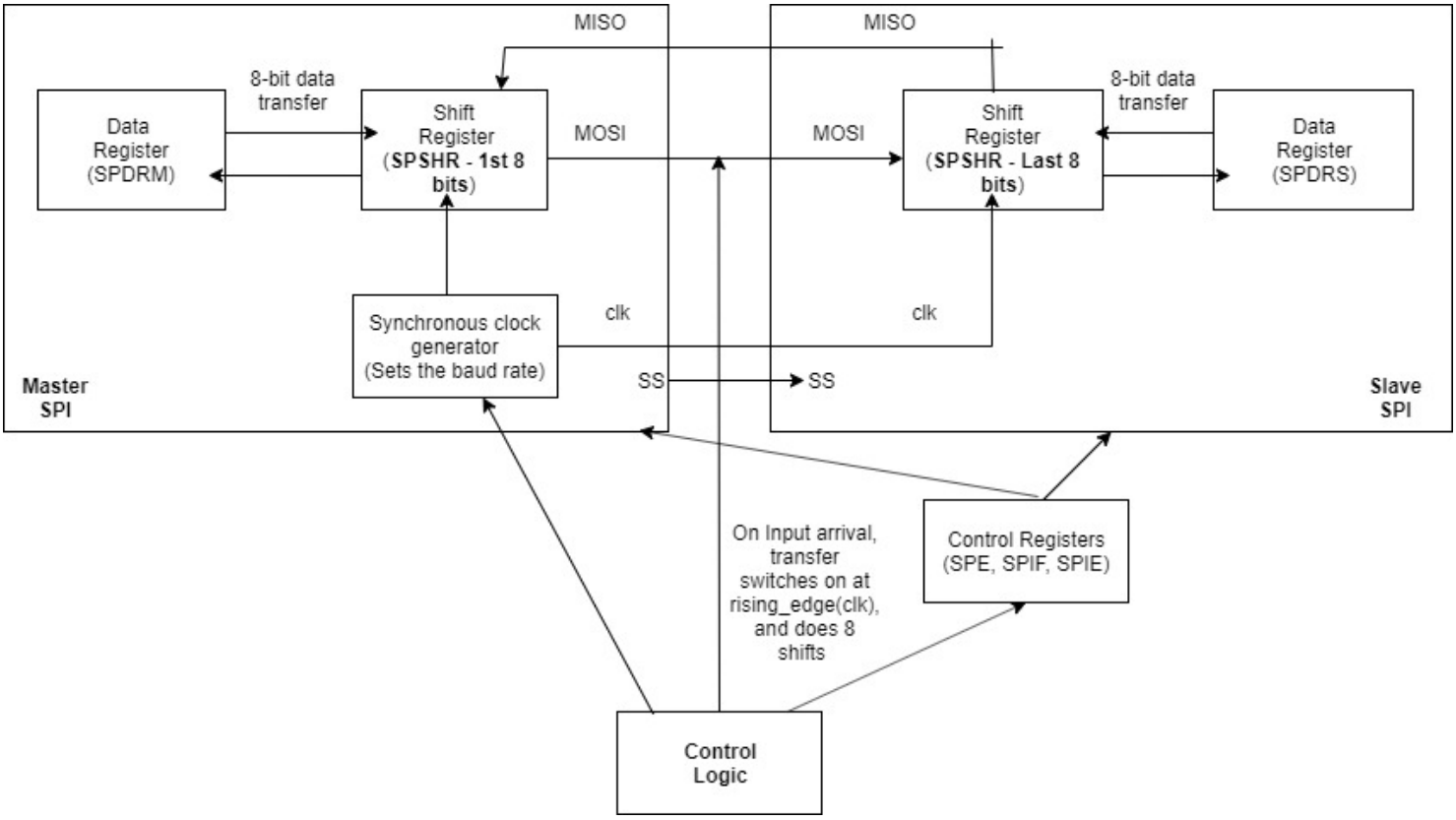
They are as follows:

1) We have assumed a single clock signal, that will be chosen by the control block, in the very beginning and will decide the clock rate of data transfer, between the two SPI terminals. In Actual SPI, the serial clock used for transfer is derived from an internal clock of the SPI subsystem. This clock is divided by 2,4,16, or 32 depending on the value of the two SPI Bit Rate control bits (SPR1 and SPR2). The Clock Logic block then selects one of these clocks (in master mode) or the SCK input pin (in slave mode) depending on the value of SPR0, SPR1, and MSTR. This simplification was mainly done to reduce complexity.

2) We have not specifically incorporated the logic for CPOL and CPHA, as that can be practically implemented by exchanging the roles of Master and Slave without much hassle. Hence, we left that out in the VHDL code. Similarly, we left out a few control and status registers for simplicity.

3) To simulate the basic operation, we have selected one master and one slave. As a consequence of this, we chose not to implement the MSTR flags and SS connection lines.

Block Diagram

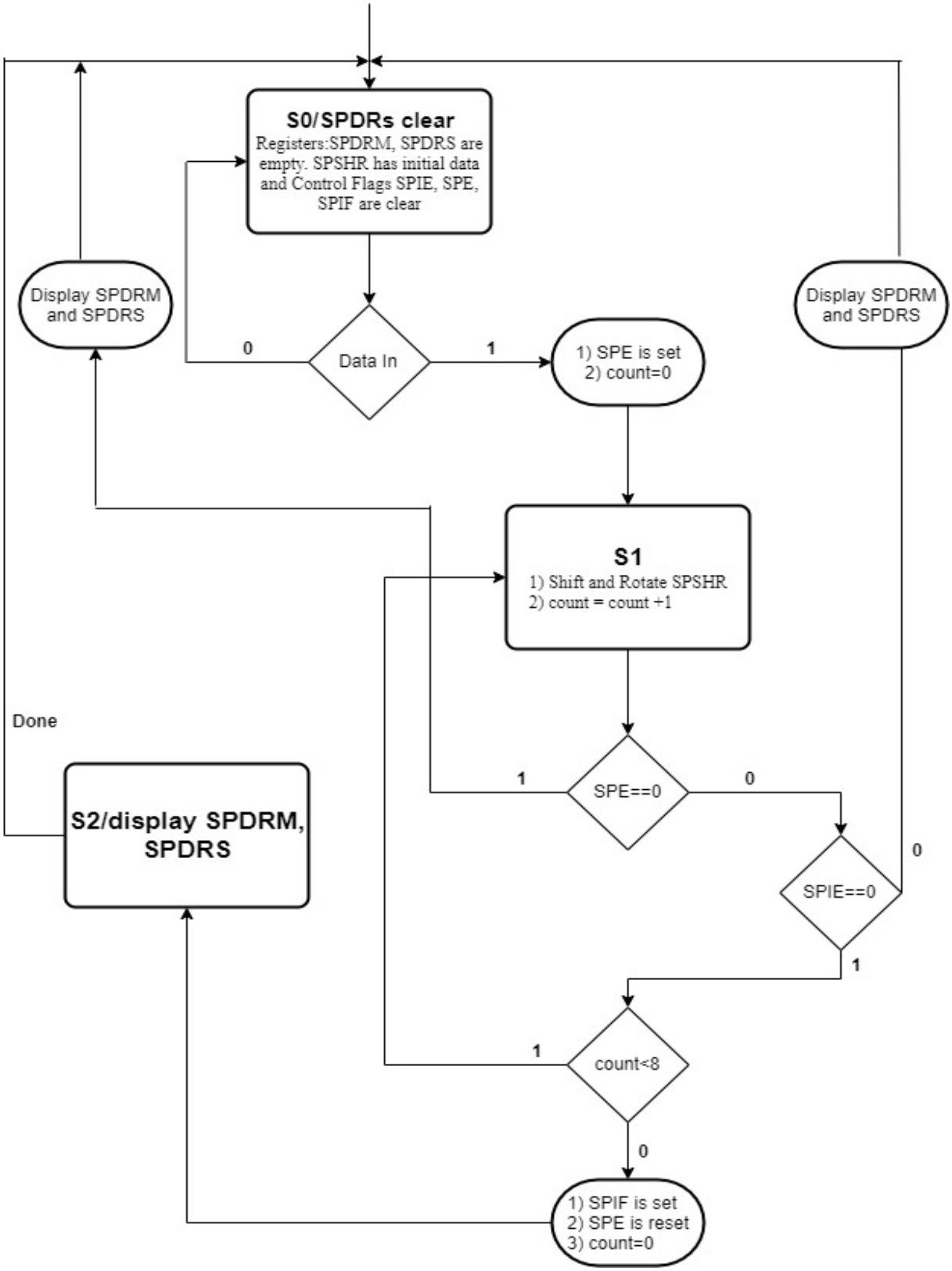


- **Made using the tool 'draw.io'**

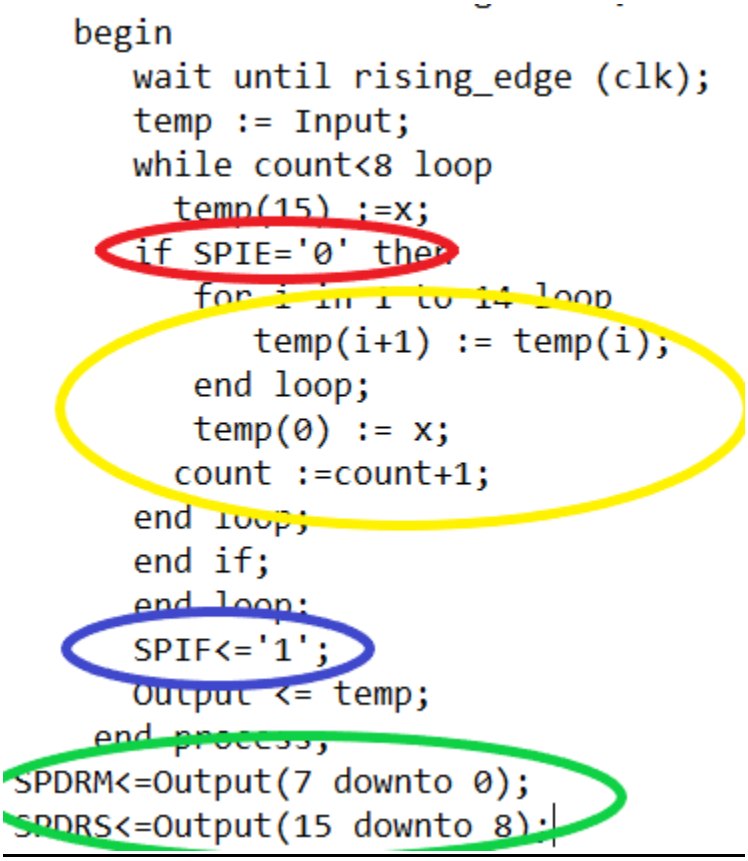
Explanation of our circuit design

Our SPI subsystem performs synchronous master/slave type exchange of two data bytes each, that is, both the master and the slave change states simultaneously with the clock. Both master and slave units contain an 8-bit shift register. When input becomes available and SPE is set, data transfer occurs. During the transfer, incoming bits are shifted into the master's 8-bit register, while the old values are shifted out to the slave. When connected together the master and slave shift registers form a circular 16-bit shift register. In the process, the control logic ensures that the contents of the 16-bit shift register are shifted by 8 bits, exchanging the two values in master's and slave's registers, at the end of the process of 8 clock cycles. The circuit is rising_edge triggered. The shift register is controlled by a single clock provided at the master, i.e., no division of clock occurs. When data transfer finishes after 8 cycles, SPIF is set and the loop terminates, followed by which, the new data is transferred to Master's data register. Finally, count is reset to zero, and all registers go back to the initial state.

SM-Chart


S0/SPDRs clear
Registers:SPDRM, SPDRS are empty. SPSHR has initial data and Control Flags SPIE, SPE, SPIF are clear
Display SPDRM and SPDRS
Display SPDRM and SPDRS
0
Data In
1
1) SPE is set
2) count=0
S1
1) Shift and Rotate SPSHR
2) count = count +1
Done
1
SPE==0
0
S2/display SPDRM, SPDRS
0
SPIE==0
1
1
count<8
0
1) SPIF is set
2) SPE is reset
3) count=0

VHDL Code Implementation Snippet:



The logical behavioral part of the code is shown above. We have converted the transfer of 8 bits of data from the master to slave and 8 bits from slave to master in SPI as a rotating shifter *[Shown in Yellow]*.

Considering that the master already has some 8 bits of data fed into it(Old data) and in the slave, 8 bits are entered(New data). Since in 8 clock cycles, new data and old data are to be swapped from slave and master respectively, we have considered SPIE which is INTERRUPT ENABLE. When value of SPIE=0, the loop will run till the data is swapped otherwise the loop will exit*[Shown in Red]*.

We have also taken one more output as SPIF which depicts Flag which shows us if the transfer of data to master from slave is completed or not. If SPIF=1, then the transfer is complete and the SPSHR i.e Shift register has now the new data. So the next step will be to give that data to SPDR i.e. Data register so we can see the output and the Flag will turn to zero*[Shown in Blue]*.

Since we took 8 bits from master and 8 bits from slave and appended them together to form a 16 bit Input, the 16-bit Output now needs to be bifurcated to master and slave similarly[Shown in Green].

Scope of Improvement

We can have multiple slaves (SS lines), to model real-world applications. Also, we can synchronize different slaves and the master, using clock-rate adjustments (clock division). Several other flags, control and status registers, which take care of more complex situations, can be included in the design. We can also add pull-up control to tie unused pins to either a high or a low logic level.

Member Contribution Details

Student ID	Name	Contribution percentage
201701006	Swarnaditya Maitra	35
201701007	Sanyam Desai	35
201701010	Milan Nagariya	20
201701011	Anish Deshpande	10

References and tools

1) **https://www.rpi.edu/dept/ecse/mps/SPI.pdf**
2) **https://www.draw.io/**