Serial Peripheral Interface

Final Project

Presented By:
Noran Hany
Hala Hamdy
Yahia Zakaria
Youssef Gamal

Supervised By: Dr. Ihab Talkhan

Cairo University
Faculty of Engineering, Computer Department

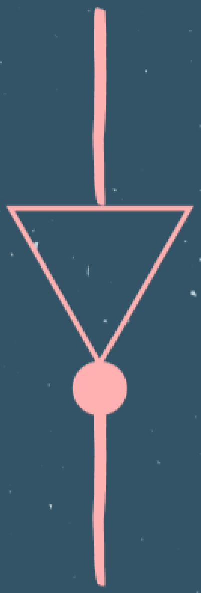


Table of Contents

Introduction

Have you ever wondered how the electrical devices communicate with each other? How a Sensor and a microcontroller send to each other certain instructions? What common language do they share? In our Electronic System, we name the common language they share “the communication protocol”. In this article, we will be discussing 3 basic communication protocols which are: SPI, UART and I2C. All of which are used for small data and low speed communications. Nevertheless, they are simple protocols that use less hardware compared to other high-speed protocols such as WIFI, USB, Ethernet, and Bluetooth. For now, let us only focus on the basic 3 protocols.

SPI

Serial peripheral interface or SPI for short is a synchronized serial communication protocol. The SPI has only one single master where this master could have at least one slave depending on the load capacitance of the system. As figure 1 shows, the SPI has 4 lines in exchange between the master and the slave.

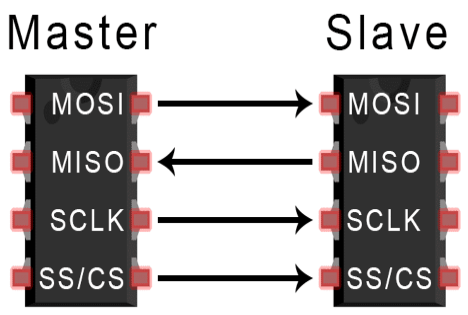


Figure 1 SPI with single slave

MOSI and MISO

MOSI is short for master out slave in; meanwhile, MISO stands for master in slave out. Data from the master to the slave is sent through the MOSI line transferring the most significant bit first whereas data from the slave to the master is transferred through the MISO line sending the least significant bit first. Because there is only one line for data transmission, data is transferred one bit at a time. As a result of this serial communication, it takes 8 clock cycles to transfer 8 bits of data.

Slave Selection

As the name suggests, the SS line is used by the master to identify the slave it wants to initialize the communication with. To start the exchange of the data, the master must send a low signal to the proper slave keeping the SS line for the other slaves on high voltage because the master can communicate with one salve at a time. To connect multiple slaves with the master we have two choices: the Regular mode connection where one SS line for each slave is found in the master or the Daisy chain mode connection with only one line in the master.

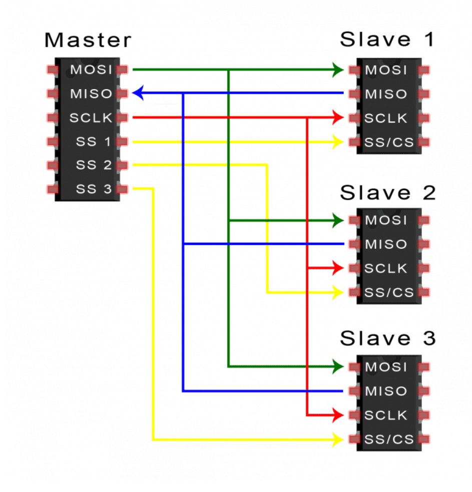


Figure 2 Regular mode connection

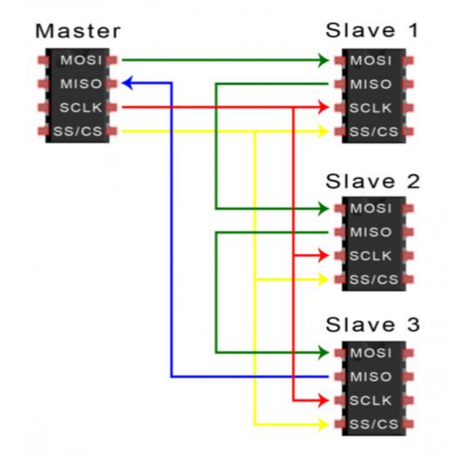


Figure 3 Daisy Chain connection

The difference between figure 2 and 3 is that, in figure 2, the master only takes 8 clock cycles to transfer 8 bits data to slave 3. However, in figure 3, it takes 24 clock cycles to send the same 8 bits to slave 3 because data should pass first to slave 1 in the first 8 cycles followed by another 8 cycles to propagate from slave 1 to slave 2 and at last another 8 cycles to propagate to slave 3.

CLK

Since the communication is synchronous, the clock is essential. The Master is the one responsible for initializing this clock in order to start the communication between the Master and the slave.

Clock Polarity and Clock Phase

Clock polarity indicates which edges the data is sampled and shifted on that is either the falling or rising edge. Meanwhile, Clock phase states whether the rising edge is sampling or shifting

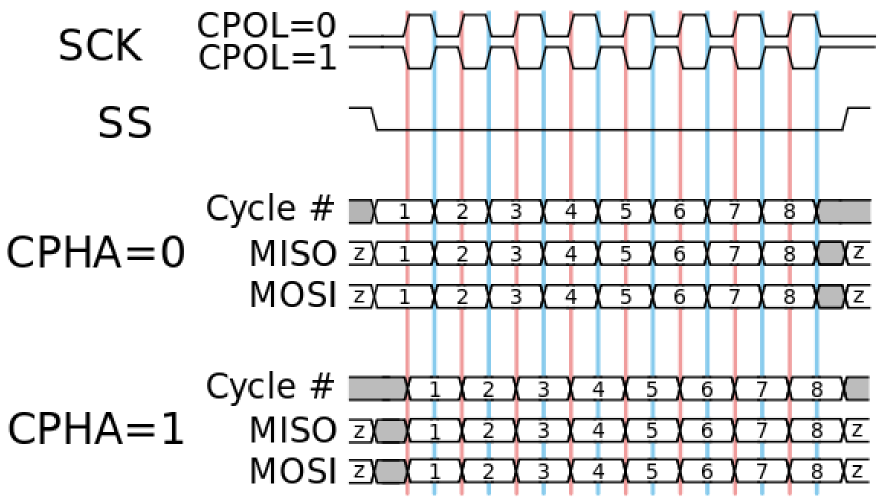


Figure 4 the four modes of the SPI

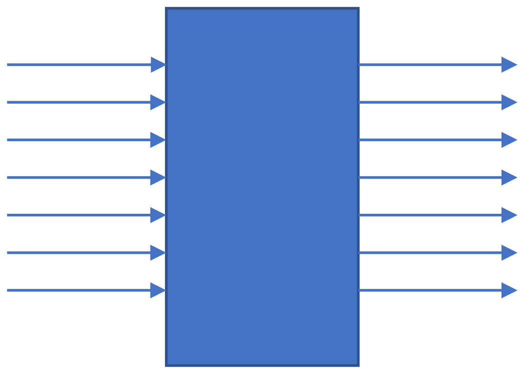
As demonstrated in Fig 4, Clock polarity 0 sets the first edge the rising edge; however, clock polarity 1 makes the first edge the falling edge. Concerning the clock phase, when it is set to low voltage, data will be sampled at the rising edge and shifted at the falling edge and vice versa for high voltage.

Design Process:

1. Master

Inputs:

1. CPHA
2. CPOL
3. MISO
4. SS_IN
5. START
6. READ
7. DATA



Outputs:

1. MOSI
2. SS1_OUT
3. SS2_OUT
4. SS3_OUT
5. CLK
6. SFIFT_STATE
7. MAIN_STATE

Functions:

1. Generate clock so that the communication with the slave is synchronous
2. Choose the mode of communication using the CPOL and the CPHA signals
3. Load data into its memory
4. Sample and shift data according to the mode of communication

Design Process:

1. In our design we have two types of memory in the master. The first one is a shift register that is responsible for sending and receiving the data from the slave. The second one is a register with 8bits. The reason for this choice is to enable the master to send data only without receiving if it is not necessary. This can be achieved through the integration module as it is responsible for deciding whether to copy the data from the shift register to main memory (receive) or not (sending only) at the end of the communication with the slave.

2. For loading data into the main memory our decision was to use the positive edge for the input READ to copy the data from the parallel input DATA to the main memory.

3. For clock generation we are using the negative edge for any of the selectors to start the clock. The clock will still work until the selector returns high again.

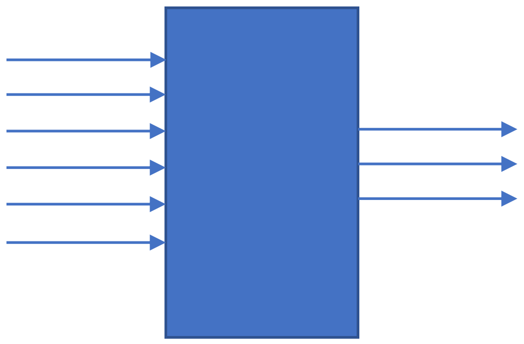
4. To indicate the start of the full duplex communication we used the positive edge for START input. This to reset all the parameters of the master, configure the clock according to the selected mode and choose the slave to communicate with.

5. The process of sampling and shifting is happing throw the positive and the negative edges of the clock according to CPHA. We made some design choices to avoid some issues as follows:
    1. By using the parameter START_CLK we prevent the sampling or shifting process to happen at the edges when initializing the clock value before the communication.
    2. Using IS_VALID parameter to prevent shifting before sampling the data in modes 1 and 3.
    3. Using SAMPLED_COUNT parameter to end the communication after all 8bits is sent and received the new bits from the slave.

2. Slave

Inputs:

1. CPHA
2. CLK
3. SS
4. MOSI
5. READ_MEMORY
6. DATA



Outputs:

MISO
OUT_SHIFT_STATE
OUT_MAIN

Functions:

1. It sends a response back to the master using the port MISO
2. Load Data into its Memory

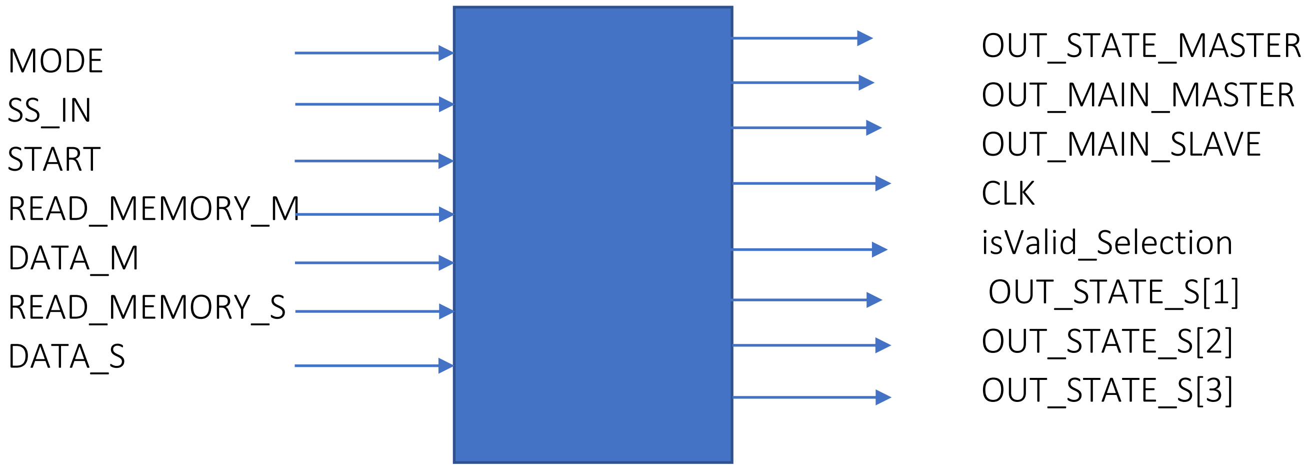
Design Process:

1. The Slave has two types of memories, the first one is its main memory and the second one is a 8 bit shift register, if the memory is accessed to read (assign READ_MEMORY to 1) the data input will be assigned to the Main Memory of the slave, then if the current slave is selected using the input port (SS which is active low), the data of the Main Memory is assigned to the shift register

2. The Slave works only if the SS=0 , then as the SPI has 4 modes, in Slave module these modes is selected from the Clock (CLK) that is generated from the master and the Clock Phase (CPHA) which is an input comes from the integration module, if the clock is positive edge and clock phase is low, the slave will read the input which comes from the master (MOSI), and if the clock phase is high, the data that read from the master, the data of the shift register is shifted right and the input data is assigned to the LSB of the register. If the clock is negative edge the previous logic will be reserved so that if the clock phase is high, the slave will read input from the master and shift and store it in the shift register if the clock phase is low.

3. The output OUT_SHIFT_STATE is the shift register and OUT_MAIN it the Output of the main memory of the slave.

3. Integration



Functions:

1. Connects the master with the 3 slaves.
2. Receives the correct slave the master wants to communicate with and starts sending it the mode of communication along with the parameters needed to start the transmission of the data.
3. The Memory of the slave that was communicating with the master will be send to the SPI test bench module for the self-checking.
4. All the registers were sending to the SPI test bench to assure that only the intended slave was the one that communicates with the master.

Design process:

All connections between the master and the slaves were just made.

Test Cases

1. Master

we designed 4 test cases to test all 4 modes of SPI with different slaves. The master should configure the clock according to the CPHA and shift or sample the data at any edge according to CPOL. In each test case an 8bit vector will be sent on the MISO line from a different slave as shown in table 1.

Test	Data	Mode	Slave
1	01010101	0	1
2	00110011	1	2
3	01101101	2	3
4	10101010	3	3

Table 1 Master test cases

2. Slave

Same as master except that the slave cannot control the clock and we add another test case to make sure that the slave doesn't take any action when its selection line is high. These test cases is shown in table 2

Test	Data	Mode	SS
1	01010101	0	0
2	00110011	1	0
3	01101101	2	0
4	00110011	3	0
5	11111111	3	1

Table 2 Slave test cases

3. Integration

Test	Mode	SS	Data from the master to the activated slave	Data from the activated slave to the master
1	0	1	11111111	00000000
2	1	2	01010101	10101010
3	2	3	01010101	11110111
4	3	2	10010011	01001110

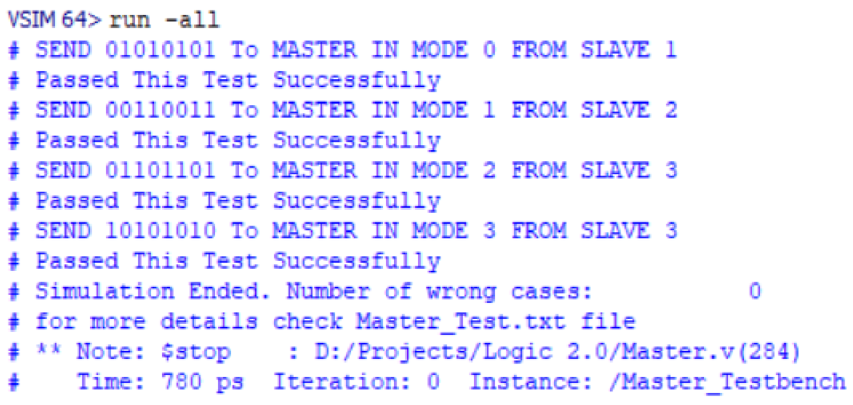
Table 3 Integration test cases

Justification:

- In test case 1, since the initial memory of both the master and the slave's architecture is set to 8'b00000000, and we used the READ_MEMORY flag to over-write the memory of the master with a value of 8'b11111111, i.e. the Master will be sending the slave 8'b11111111, the slave will be sending the master 8'b00000000. The reason behind the choice of this test case was that the interchanging in data is easy to track, and easy to confirm the success of the communication.
- In test case 2,Since in test case 1 the memory of the master was overwritten by a value; meanwhile, the memory of the slave was not changed, in testcase2 we decided to change both the memories with values which is a bit harder than testcase1 but yet easy to detect.
- Moving to testcase3, the memory of the slave was initialized by a value, but the master was left as it is. The approach of sending the memory of the master to another slave is taken to assure that the master can send the same data to different slaves taking a single communication with each slave.
- For the fourth test case, it was made to ensure that Mode3 is working properly. The memories of both the master and slave 2 were changed by the values to be send as seen in Table 3.

Simulation Results

1. Master test bench



```
VSIM 64> run -all
# SEND 01010101 To MASTER IN MODE 0 FROM SLAVE 1
# Passed This Test Successfully
# SEND 00110011 To MASTER IN MODE 1 FROM SLAVE 2
# Passed This Test Successfully
# SEND 01101101 To MASTER IN MODE 2 FROM SLAVE 3
# Passed This Test Successfully
# SEND 10101010 To MASTER IN MODE 3 FROM SLAVE 3
# Passed This Test Successfully
# Simulation Ended. Number of wrong cases:           0
# for more details check Master_Test.txt file
# ** Note: $stop    : D:/Projects/Logic 2.0/Master.v(284)
#    Time: 780 ps  Iteration: 0  Instance: /Master_Testbench
```

Figure 5 Master simulation results

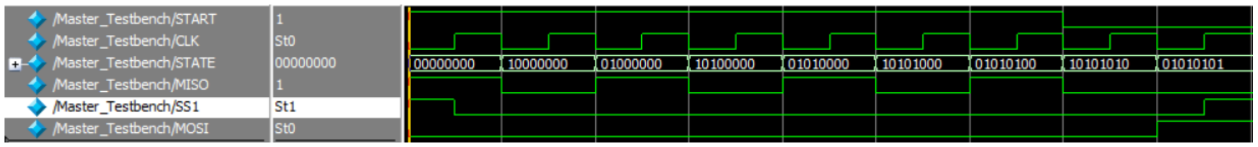


Figure 6 Test 1 of Master

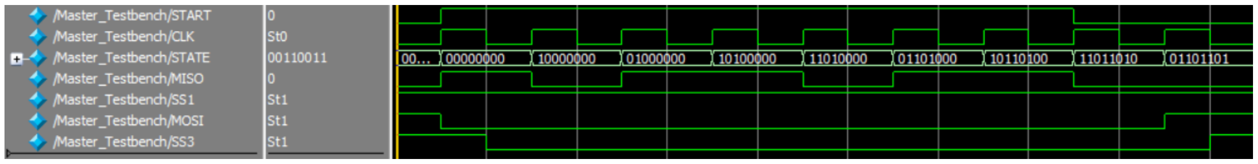


Figure 7 Test 2 of Master

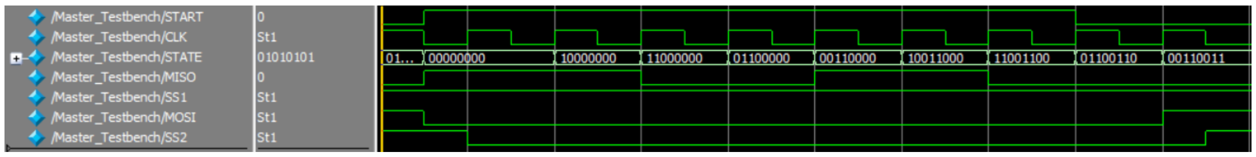


Figure 8 Test 3 of Master

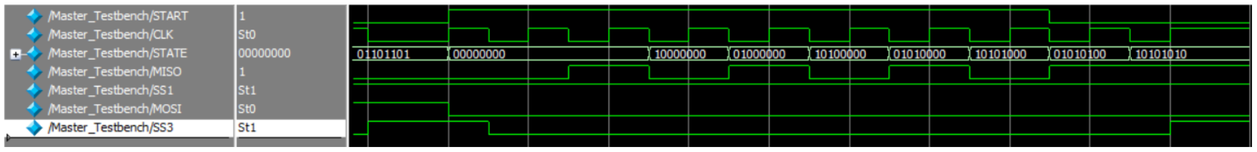
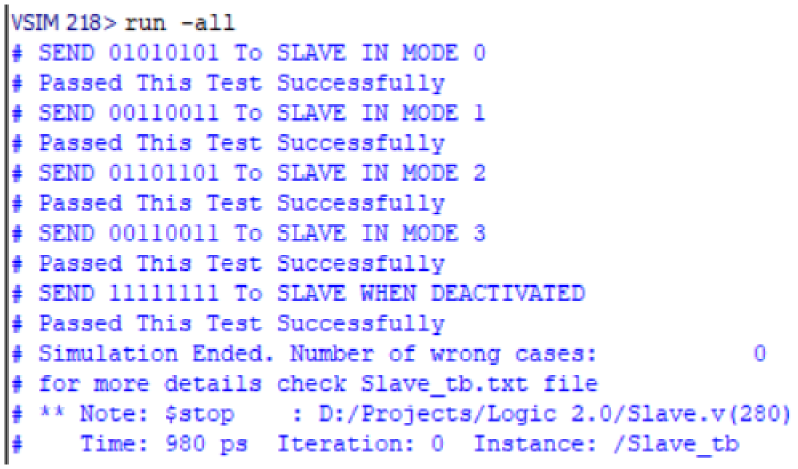


Figure 9 Test 4 of Master

2. Slave test bench



```
VSIM 218> run -all
# SEND 01010101 To SLAVE IN MODE 0
# Passed This Test Successfully
# SEND 00110011 To SLAVE IN MODE 1
# Passed This Test Successfully
# SEND 01101101 To SLAVE IN MODE 2
# Passed This Test Successfully
# SEND 00110011 To SLAVE IN MODE 3
# Passed This Test Successfully
# SEND 11111111 To SLAVE WHEN DEACTIVATED
# Passed This Test Successfully
# Simulation Ended. Number of wrong cases:           0
# for more details check Slave_tb.txt file
# ** Note: $stop    : D:/Projects/Logic 2.0/Slave.v(280)
#    Time: 980 ps  Iteration: 0  Instance: /Slave_tb
```

Figure 10 Slave simulation results

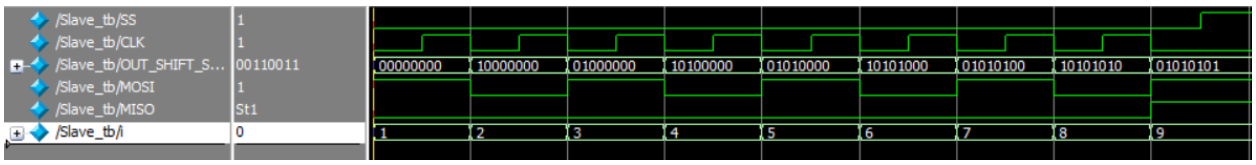


Figure 11 Test 1 of Slave

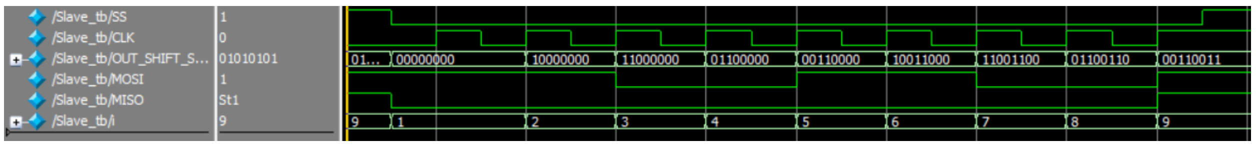


Figure 12 Test 2 of Slave

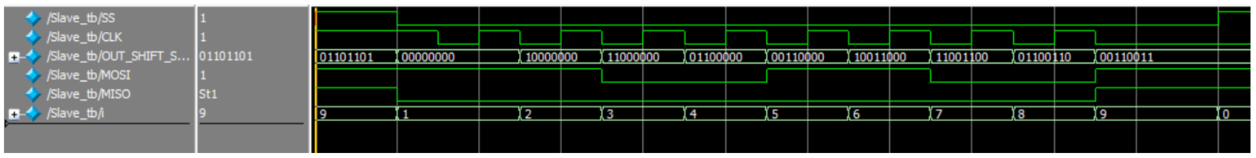


Figure 13 Test 3 of Slave

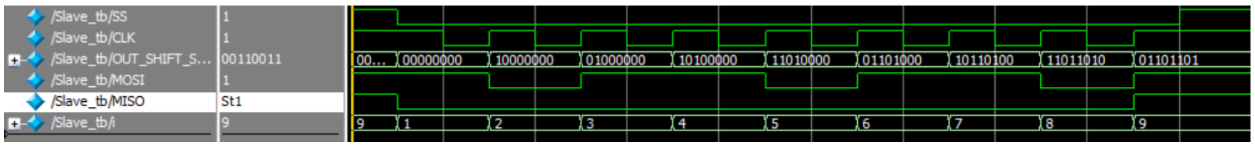


Figure 14 Test 4 of Slave

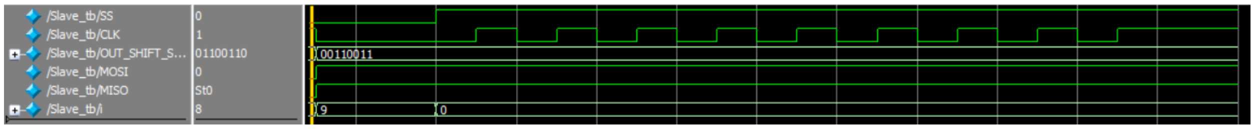


Figure 15 Test 5 of Slave

3. Integration test bench

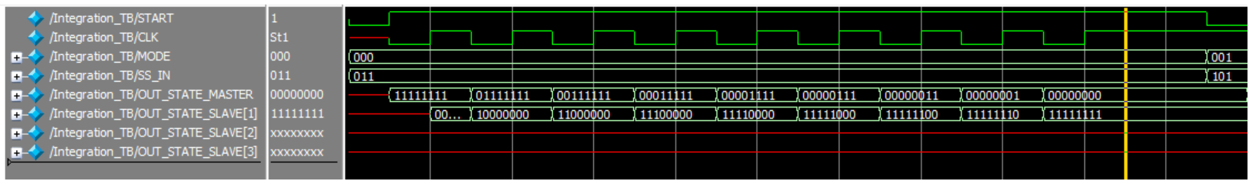


Figure 16 Test 1 of Integration

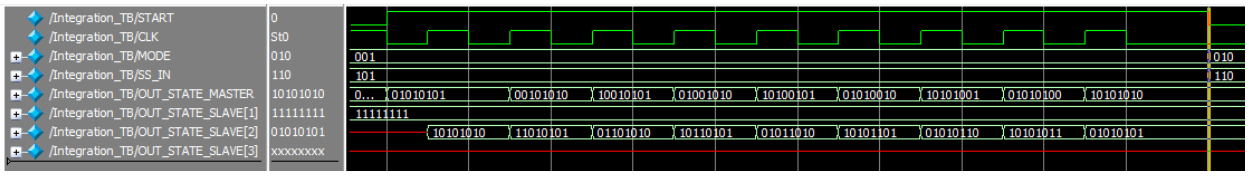


Figure 17 Test 2 of Integration

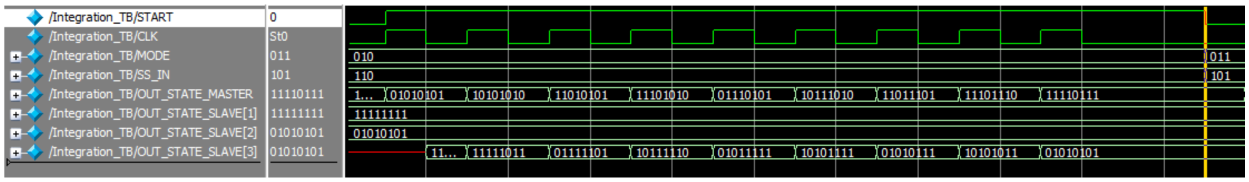


Figure 18 Test 3 of Integration

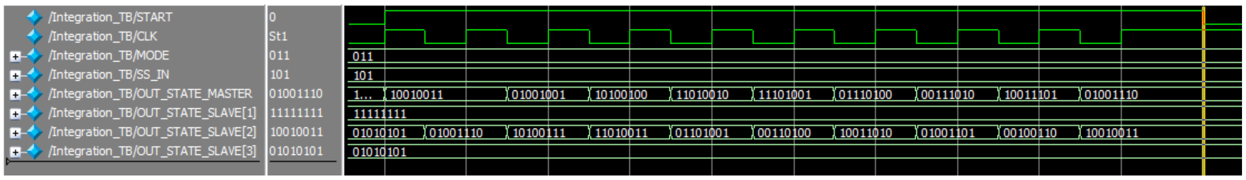


Figure 19 Test 4 of Integration

Other Communication Protocols

1. UART

UART is short for universal asynchronous receiver transmitter, and it is one of the simplest form of communication. UART has only 2 lines for data exchange, one for transmission and the other for receiving. Unlike SPI that needs a master and a slave, UART usually communicates with another UART. For a successful interaction, the two UARTS should be settled up for the same baud rate. Baud rate is simply the rate of data transfer between the 2 devices expressed in bits per second.

Like SPI, UART transfers data serially; nevertheless, it is an asynchronous communication which does not rely on clock pulses to coordinate the data transmission. To signify the start and end of the communication, data is sent as a packet. As illustrated in Figure 15, a packet is classified into 4 sections, one of which is not compulsory that is the parity bit.
Start Bit:
The start bit is sent to inform the receiving UART that communication will start. Normally, unless data is transmitted, the start bit is high; thus, to begin the transmission, the start bit is set to zero.

Data Frame:
The data frame contains the actual data needed to be delivered to the receiving UART. Typically, this data frame ranges from 5 to 8 bits when there is a parity bit included, but when there is no parity bit, the maximum it could reach is 9 bits.

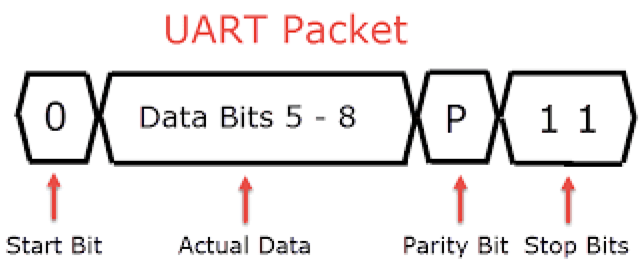


Figure 20 Data in the packet form

Parity Bit:

It is one of the methods used to check for the data sent against any error that could occur during transmission. Errors arise due to long distance communication, mismatch baud rate or any other reason. It simply checks the count of the 1's in the data being sent and compares it with the parity bit. To clarify, 0 parity bit means that the number of ones in the data frame should be even; also, a parity bit of 1 signifies an odd count of ones. If the parity check is incorrect, an action will be taken.

Stop Bit:

Identifying the end of the transmission, the stop bits, which is either a single bit or 2 bits, sets the transmission line back to high.

A simple interaction between the UARTS follows the following sequence. Firstly, the data needed to be delivered is sent to the transmitting UART in parallel form using a data bus. Then, the transmitting UART converts this data to a packet. Afterwards, the packet is being sent one bit at a time until all the data is being sent. Finally, the receiving UART takes only the data frame excluding the other sections and passes them through the data bus to where they are required. Figure 21 simplifies the above-mentioned process.

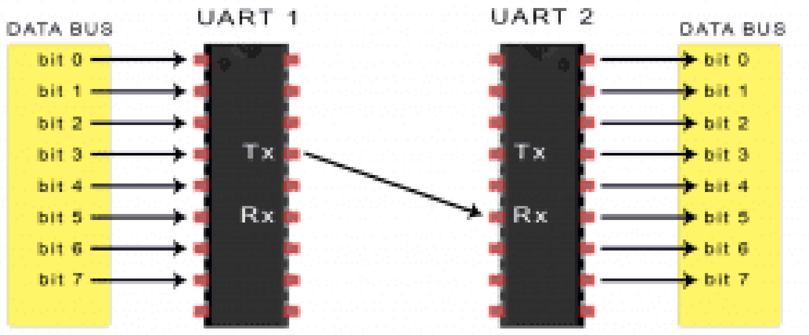


Figure 21 Communication between the UARTs

1. I2C

2C stands for Inter Integrated Circuit. It is a synchronous serial communication protocol developed by Philips Semiconductor. Like SPI, this protocol uses a clock to synchronize the transmission of data, where the frequency of the clock pulses is adjusted to match the frequency of the data being sent. It has 2 wires; a bidirectional data line (SDA), and a clock line (SCL). I2C combines between SPI and UART by being capable of having several slaves, like SPI, and multiple masters, as UART. Let us focus first on how “the single master multiple slaves” communication occurs. In fact, data is sent in the form of messages where it is classified into several frames as seen in Fig 22.

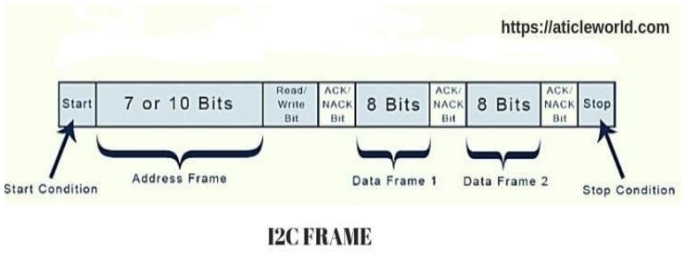


Figure 22 I2C frame

Start Condition:

Initially, When the master wants to start the communication, it sends a start bit altering the SDA line from high to low voltage. After that, the SCL line should change from low to a high voltage. We just need to keep in mind that the order of these steps is vital, so the SDA line should become low before the SCL line becomes high.

Address Frame:

These set of bits are the one responsible for identifying which slave the master will initialize the communication with. It ranges from 7 to 10 bits that represents an address of a specific slave. All the slaves receive this address, and they start comparing this address with their own address. If the addresses match, the slave returns back a low voltage bit (ACK/NACK); however, if they do not match, the slave SDA line remains high. For instance, Figure 8 shows that the Slave 2 is the one the master intended to communicate with; nonetheless, Slave 1 and 3 will not continue the communication and will be grounded. All this could be done using pull up resistors as seen from Fig 8. Since each slave should have a unique address, the master could have up-till 128 slaves for 7 bits address and 1024 slaves for 10 bits address.

Read/Write Bit:

This bit is sometimes considered part of the Address Frame that stated whether the master will receive or send. For the slave to send, the read/write bit should be 0, and it will be 1 in case the master is transmitting.

Stop Condition:

Finally, to get the communication to a halt, the SCL returns from low to high voltage; then, the SDA changes back from high to low. Again, the order is important, so the serial clock should retrieve its state before the serial data does.

Multiple Master Multiple slaves:

The problem that could be encountered in this connection is the request of 2 masters to communicate with the same slave at the exact time. To overcome this point, a master checks first if the SDA line of the needed slave is low, which states that this slave is in use by another master, it waits till the communication is ended to start another one. Nevertheless, if the SDA line is high, it is safe to request a transmission.

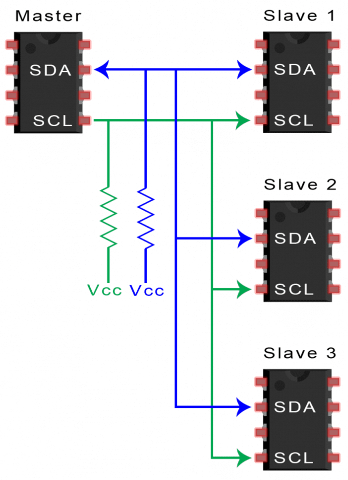


Figure 23 Master communicating with Slave 3

Comparison Between the SPI, I2C and UART.

Points of comparison	SPI	UART	I2C
Number of devices that can be connected	Theoretically Unlimited	Only 2 devices	Up to 128 devices for 7 address bits
Max transmission speed (Megabytes Per Second)	4 Mbps	115.2 Kbps	1 Mbps
Max transmission distance (meter)	0.1 m	15 m	0.5 m
Full duplex mode	Operates in full duplex mode i.e. the master and the slave can send data at the same time	Doesn't operate in full duplex mode i.e. only one device can send data at a time	Doesn't operate in full duplex mode i.e. only one device can send data at a time
Advantages	1) data is send uninterrupted as there is no start and stop bit 2) Easier identifying the slave than I2C 3) Higher data transmission rate than I2C. 4) Data could be sent from master and slave	1) Check any error in transmission using the parity bit. 2) Does not rely on any clock signal 3) uses only 2 wires.	1) Checks successful interaction using ACK/NACK bit 2) Multiple slaves and multiple masters could be used. 3) only 2 wires are used

	simultaneously		
Drawbacks	1) uses 4 wires 2) No check for successful data transmission 3) Cannot connect more than one master	1) data frame is limited to 9 bits 2) The baud rate between the UARTs must range in 10%. 3) cannot connect more than one master and one slave.	1) data frame is limited to only 8 bits 2) slower data transmission rate than SPI

Work Load

Master & Master test bench: Yahia Zakaria

Slave & Slave test bench: Youssef Gamal

Module design report: Yahia Zakaria & Youssef Gamal

Simulation results: Yahia Zakaria & Youssef Gamal

Integration & Integration test bench : Noran Hany & Hala Hamdy

Search and comparison with I2C and UART: Noran Hany & Hala Hamdy

Testcases :   Noran Hany & Hala Hamdy