12.2.1.4 Timing

As mentioned previously, `SCK` is generated by the leader and fed to the follower. Here, `SCK` depends on the maximum data rate of the transmitter and receiver. Hence, the device with the lowest rate defines its limit. Besides frequency, the leader also adjusts the polarity and phase of clock denoted by `CPOL` and `CPHA`, respectively. Four possible combinations of `CPOL` and `CPHA` are presented in Fig. 12.3. These combinations are called modes of the SPI.

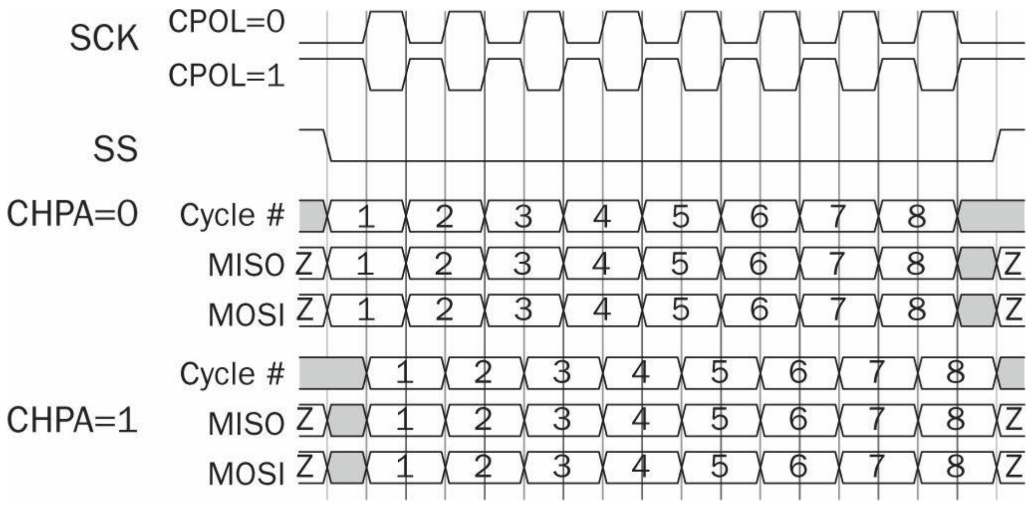

FIGURE 12.3 SPI communication timing diagram.

Modes		Clock	Data	Bits on `MISO` and `MOSI` lines		
`CPOL`	`CPHA`	base	availability	Placed	Stay	Captured
0	0	0	0 to 1 transition	Clock goes to 0	Clock at 1	0 to 1 transition
0	1	1	0 to 1 transition	Clock goes to 1	Clock at 0	1 to 0 transition
1	0	0	1 to 0 transition	Clock goes to 1	Clock at 0	1 to 0 transition
1	1	1	1 to 0 transition	Clock goes to 0	Clock at 1	0 to 1 transition

TABLE 12.1 SPI Modes in Tabular Form Based on Clock Operation

We can briefly summarize the SPI modes presented in Fig. 12.3 and in Table 12.1. Here, each operation in the corresponding mode is summarized based on the clock. For more information on the SPI modes, please see [47].

12.2.2 SPI in Verilog

We can describe the transmit and receive operations (for the leader and follower) as separate modules in Verilog. Let's start with transmitter modules.

12.2.2.1 Transmitter Modules

As explained previously, either the leader or follower can be a transmitter. Therefore, we should form a different description for each option. The Verilog description of the leader-transmitter setup is presented in Listing 12.9. Inputs to this module are clock (`clk`), data to be sent (`data`), and send event trigger (`send`). The outputs of this module are `sck`, `mosi`, `ss`, and `busy`. The first three of these are directly associated with the SPI lines. The fourth output shows if the module is busy while transmitting data. Within the module, the data length to be sent is set as a parameter. The frequency division is applied to the input clock so that `sck` is set at 2 MHz. `CPOL` and `CPHA` are set to zero. Thus, all changes are performed on the falling edge of `sck`.

The working principle of the leader-transmitter module can be explained as follows. The module is a state machine triggered in every falling edge of `sck`. In other words, the data on the `mosi` line does not change while `sck` is at logic level 1. The state machine is initially at `RDY` state. The transmission of data starts when `send` goes to logic level 1. The state changes to `START` while `index` is set to the first bit of data and `busy` goes to logic level 1. In `START` state, the leader module sets `ss` to logic level 0 to wake up the follower. Then, the first bit of the data is loaded to `mosi` line and `index` is decreased by one. The next state is `TRANSMIT` in which `index` is decreased by one and the corresponding bit of data vector is sent to the output via `mosi` step by step. When `index` equals to zero, iteration ends. The state machine switches to `STOP` state. Then, `busy` goes to logic level 0 and `ss` goes to logic level 1. Next, the state machine turns back to `RDY` state and waits for another `send` trigger.

Listing 12.9 Verilog Description of the SPI Leader-Transmitter Module

```
module SPI_leader_transmitter(clk,data,send,sck,ss,mosi,busy);

parameter data_length=8;

input clk;
input [data_length-1:0] data;
input send;
output reg sck=0;
output reg ss=1;
output reg mosi;
output reg busy=0;

localparam RDY=2'b00, START=2'b01, TRANSMIT=2'b10, STOP=2'b11;

reg [1:0] state=RDY;
reg [7:0] clkdiv=0;
reg [7:0] index=0;

always @ (posedge clk)
//sck is set to 2 MHz
if (clkdiv == 8'd24)
        begin
        clkdiv <= 0;
        sck <= ~sck;
        end
else clkdiv <= clkdiv + 1;

always @ (negedge sck)

case(state)
RDY:
        if (send)
        begin
        state <= START;
        busy <= 1;
        index <= data_length-1;
        end
START:
        begin
        ss <= 0;
        mosi <= data[index];
        index <= index - 1;
        state <= TRANSMIT;
        end
TRANSMIT:
        begin
        if (index == 0)
        state <= STOP;
        mosi <= data[index];
        index <= index - 1;
        end
STOP:
        begin
        busy <= 0;
        ss <= 1;
        state <= RDY;
        end
endcase
endmodule
```

Listing 12.10 Verilog Description of the SPI Follower-Transmitter Module

```
module SPI_follower_transmitter(sck,ss,data,miso,busy);

parameter data_length=8;

input sck;
input ss;
input [data_length-1:0] data;
output reg miso;
output reg busy;

localparam RDY=2'b00, TRANSMIT=2'b01, STOP=2'b10;

reg [1:0] state=RDY;
reg [7:0] index=data_length-1;

always @ (negedge sck)

case(state)
RDY:
        if (!ss)
        begin
        miso <= data[index];
        state <= TRANSMIT;
        busy <= 1;
        index <= index - 1;
        end
TRANSMIT:
        begin
        if (index == 0)
        state <= STOP;
        miso <= data[index];
        index <= index - 1;
        end
STOP:
        begin
        index <= data_length-1;
        busy <= 0;
        state <= RDY;
        end
endcase
endmodule
```

The Verilog description of the follower-transmitter module is presented in Listing 12.10. Since this is the follower module, it has inputs `sck`, `ss`, and `data`. The outputs of the module are `miso` and `busy`. Timing modes `CPOL` and `CPHA` are selected as zero. Thus, all changes are performed on the falling edge of `sck`.

The working principle of the follower-transmitter module (as a state machine) will be similar to the leader-transmitter module. However, there are major differences as follows. The state machine is initially at `RDY` state. It is triggered by the falling edge of `sck` which is generated by the leader-receiver. If `ss` is at logic level 0, then `busy` goes to logic level 1. The first bit of the data vector is loaded to `mosi`. Then, the state machine goes to the state `TRANSMIT`. Afterward, the module starts sending data. When `index` equals to 0, it indicates that all data bits have been transmitted. Next, the state machine goes to `STOP` state. It resets `index` and sets `busy` to logic level 0. Afterward, the state machine turns back to `RDY` state and waits for `ss` to go logic level 0 for the next transmission cycle.

12.2.2.2 Receiver Modules

As in the transmitter, either the leader or follower can act as a receiver. Therefore, we will analyze both scenarios in this section. Let's start with the follower-receiver module in Listing 12.11. Inputs to this module are `sck`, `ss`, and `mosi` which are directly related to the corresponding SPI signals. The outputs of the module are `data`, `busy`, and `ready`. Here, `busy` is at logic level 1 when data is being received. When `ready` goes to logic level 1, the received data will be available in `data` vector. Here, the data length is defined as a parameter so that it can be changed depending on the application.

The working principles of the follower-receiver module (as a state machine) are as follows. Initially, `data_temp` is set to the logic level 0 and `index` is set to the address of the first bit in data vector (`data_length`-1). The state machine has three states: `RDY`, `RECEIVE`, and `STOP`. `RDY` is the initial state in which module checks for `ss` to become logic level 0 at every rising edge of `sck`. Once `ss` goes to logic level 0, `data_temp` is set to receive the first data bit from `mosi`. Afterward, `index` is decreased by one; `busy` goes to logic level 1; `ready` goes to logic level 0; and the state machine goes to `RECEIVE` state. Then, the module receives data bits from `mosi` in every clock cycle like a shift register. When `index` reaches zero, the state machine goes to `STOP` state. In this state, `busy` goes to logic level 0; the received data is written to the `data` vector; `data_temp` is set to logic level 0; `index` is set to the address of the first bit of `data`. In the next cycle, the state machine turns back to `RDY` state and waits for another falling edge on `ss` to receive the next data package. As a reminder, the follower-receiver module should work together with the leader-transmitter module.

The leader-receiver module is presented in Listing 12.12. Here, the clock (`clk`), MISO line (`miso`), and receive trigger (`get`) are inputs of the module which are directly related to the corresponding SPI signals. The outputs of the module are `data`, `sck`, `ss`, `busy`, and `ready`. The data length is defined as a parameter of flexibility. The leader-receiver module starts listening the follower-transmitter when `get` goes to logic level 1. While receiving data, `busy` stays at logic level 1. Once all bits are received, `ready` goes to logic level 1. Then, the received data can be obtained from the `data` vector.

The working principles of the leader-receiver module (as a state machine) are as follows. Within the module, the frequency division is applied to the main clock to have 2-MHz `sck`. The state machine has three states: `RDY`, `RECEIVE`, and `STOP`. In `RDY` state, the module checks for a receive trigger in every rising edge of `sck`. Once the module is triggered, the state machine goes to `RECEIVE` state. Then, `ss` goes to logic level 0; `busy` goes to logic level 1; `ready` goes to logic level 0; all bits of `data_temp` are set to logic level 0; and (`data_length`-1) is loaded to `index`. In `RECEIVE` state, `data` bits are received from the follower in every rising edge of `sck`. Meanwhile, `index` is decreased until it reaches zero. Afterward, the state machine goes to `STOP` state. Here, `busy` goes to logic level 0; `ready` goes to logic level 1; `ss` goes to logic level 1; and the received data is written to the `data` vector. In the next clock cycle, state machine turns back to `RDY` state and waits for another `get` trigger to receive the next incoming data. As a reminder, the leader-receiver module should work together with the follower-transmitter module.

Listing 12.11 Verilog Description of the SPI Follower-Receiver Module

```
module SPI_follower_receiver(sck,ss,mosi,data,busy,ready);

parameter data_length=8;

input sck;
input ss;
input mosi;
output reg [data_length-1:0] data;
output reg busy=0;
output reg ready=0;

localparam RDY=2'b00, RECEIVE=2'b01, STOP=2'b10;
reg [1:0] state=RDY;

reg [data_length-1:0] data_temp=0;
reg [7:0] index=data_length-1;

always @ (posedge sck)

case(state)
RDY:
        if (!ss)
        begin
        data_temp[index] <= mosi;
        index <= index - 1;
        busy <= 1;
        ready <= 0;
        state <= RECEIVE;
        end
RECEIVE:
        begin
        if (index == 0)
        state <= STOP;
        else index <= index - 1;
        data_temp[index] <= mosi;
        end
STOP:
        begin
        busy <= 0;
        ready <= 1;
        data_temp <= 0;
        data <= data_temp;
        index <= data_length-1;
        state <= RDY;
        end
endcase
endmodule
```

Listing 12.12 Verilog Description of the SPI Leader-Receiver Module

```
module SPI_leader_receiver(clk,miso,get,data,sck,ss,busy,ready);

parameter data_length=8;

input clk;
input miso;
input get;
output reg [data_length-1:0] data;
output reg sck=0;
output reg ss=1;
output reg busy=0;
output reg ready=0;

localparam RDY=2'b00, RECEIVE=2'b01, STOP=2'b10;

reg [1:0] state=RDY;
reg [data_length-1:0] data_temp=0;
reg [7:0] clkdiv=0;
reg [7:0] index=0;

always @ (posedge clk)
//sck is set to 2 MHz
        if (clkdiv == 8'd24)
        begin
        clkdiv <= 0;
        sck <= ~sck;
        end
        else
        clkdiv <= clkdiv + 1;

always @ (posedge sck)

case(state)
RDY:
        if (get)
        begin
        ss <= 0;
        state <= RECEIVE;
        busy <= 1;
        ready <= 0;
        data_temp <= 0;
        index <= data_length-1;
        end
RECEIVE:
        begin
        if (index == 0) state <= STOP;
        data_temp[index] <= miso;
        index <= index - 1;
        end
STOP:
        begin
        busy <= 0;
        ready <= 1;
        ss <= 1;
        data <= data_temp;
        state <= RDY;
        end
endcase
endmodule
```