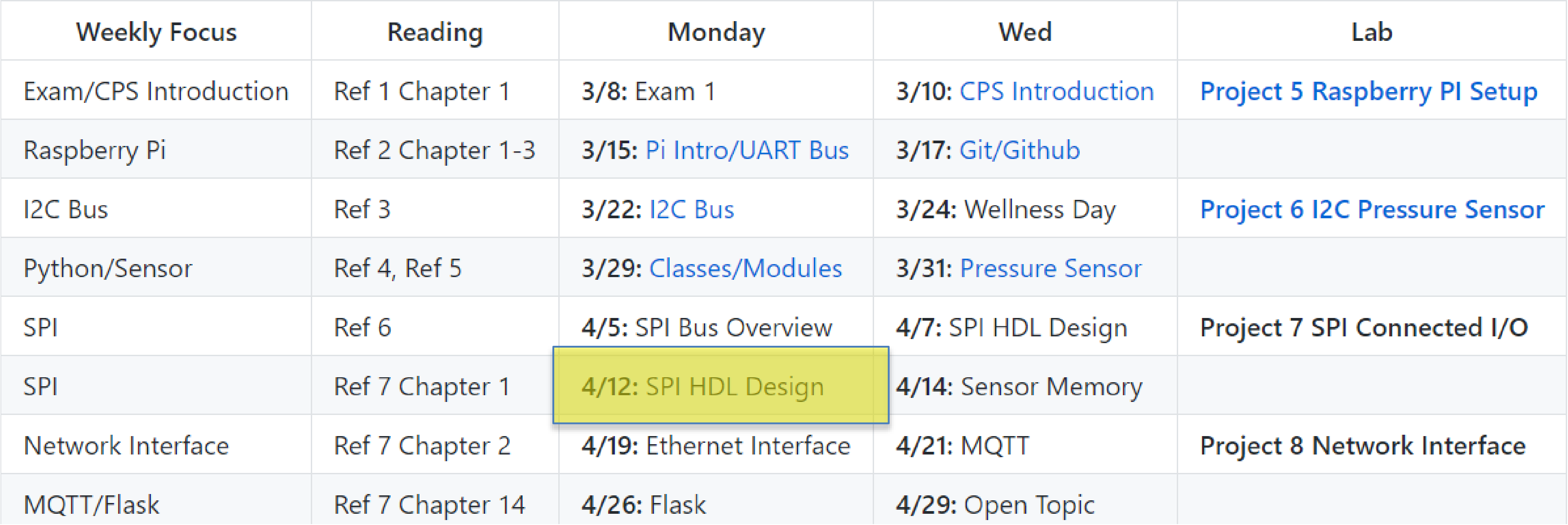
E210 Engineering Cyber-Physical Systems (Spring 2021)

SPI Peripheral (Basys3)

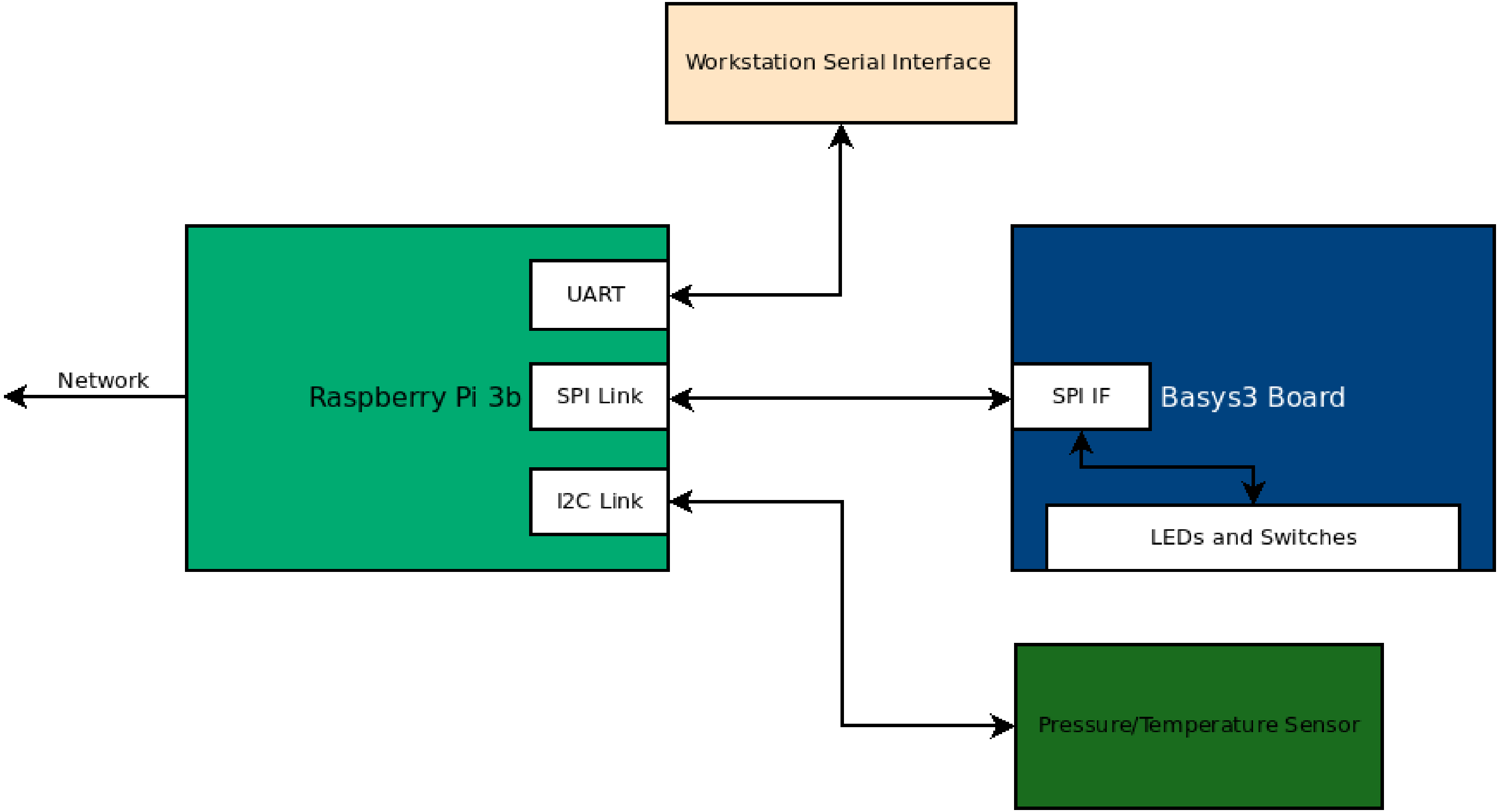
Bryce Himebaugh



Weekly Focus	Reading	Monday	Wed	Lab
Exam/CPS Introduction	Ref 1 Chapter 1	**3/8:** Exam 1	**3/10:** CPS Introduction	**Project 5 Raspberry PI Setup**
Raspberry Pi	Ref 2 Chapter 1-3	**3/15:** Pi Intro/UART Bus	**3/17:** Git/Github	
I2C Bus	Ref 3	**3/22:** I2C Bus	**3/24:** Wellness Day	**Project 6 I2C Pressure Sensor**
Python/Sensor	Ref 4, Ref 5	**3/29:** Classes/Modules	**3/31:** Pressure Sensor	
SPI	Ref 6	**4/5:** SPI Bus Overview	**4/7:** SPI HDL Design	**Project 7 SPI Connected I/O**
SPI	Ref 7 Chapter 1	**4/12:** SPI HDL Design	**4/14:** Sensor Memory	
Network Interface	Ref 7 Chapter 2	**4/19:** Ethernet Interface	**4/21:** MQTT	**Project 8 Network Interface**
MQTT/Flask	Ref 7 Chapter 14	**4/26:** Flask	**4/29:** Open Topic	

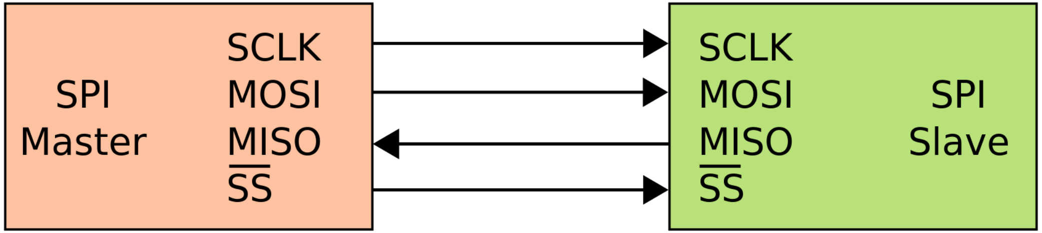
https://engr210.github.io/

Raspberry SPI Link



SPI Bus

Single Peripheral

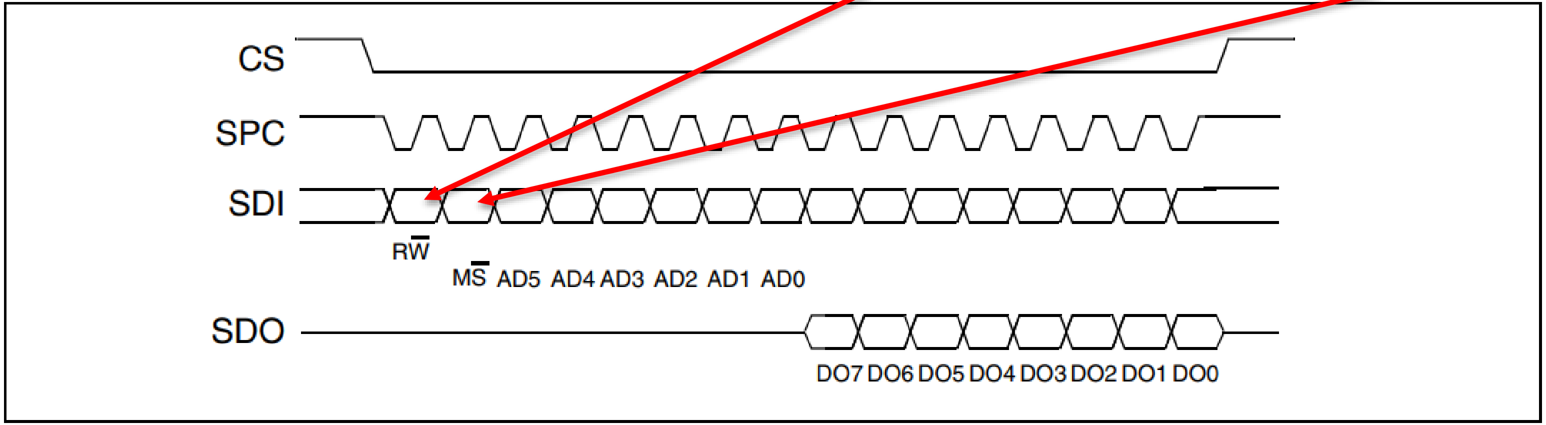


Reading

Set to 1, for reading

Auto-Increment Address

Figure 5. SPI read protocol


CS
SPC
SDI
$R\overline{W}$
$M\overline{S}$ AD5 AD4 AD3 AD2 AD1 AD0
SDO
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0

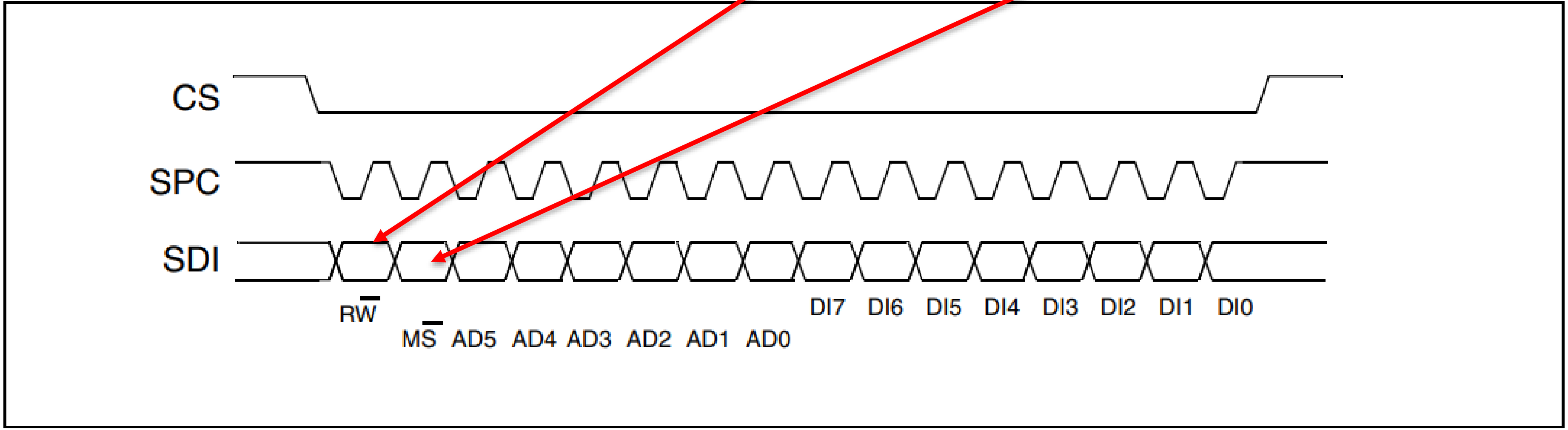
The SPI Read command is performed with 16 clock pulses. The multiple byte read command is performed adding blocks of 8 clock pulses at the previous one.

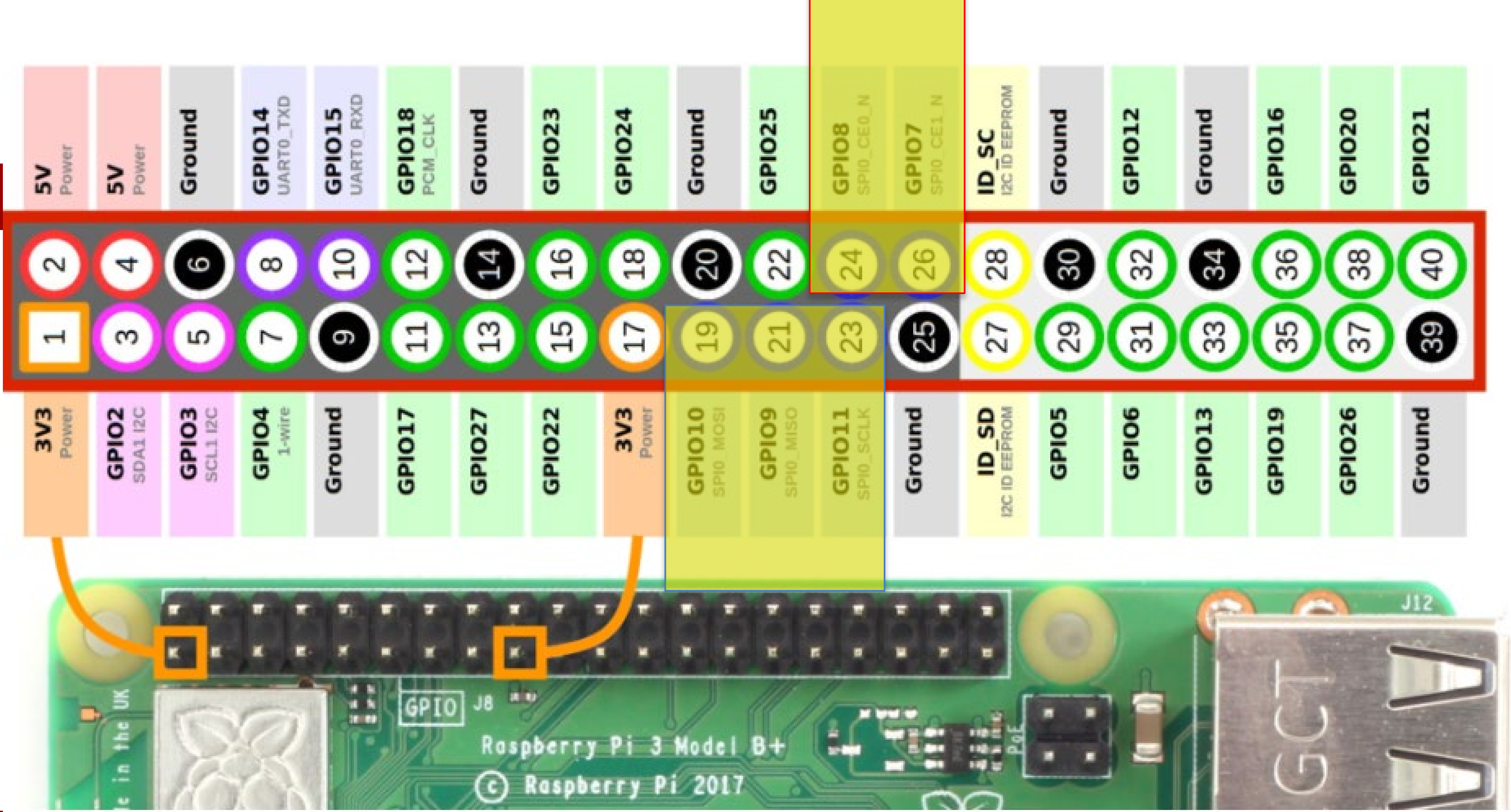
Writing

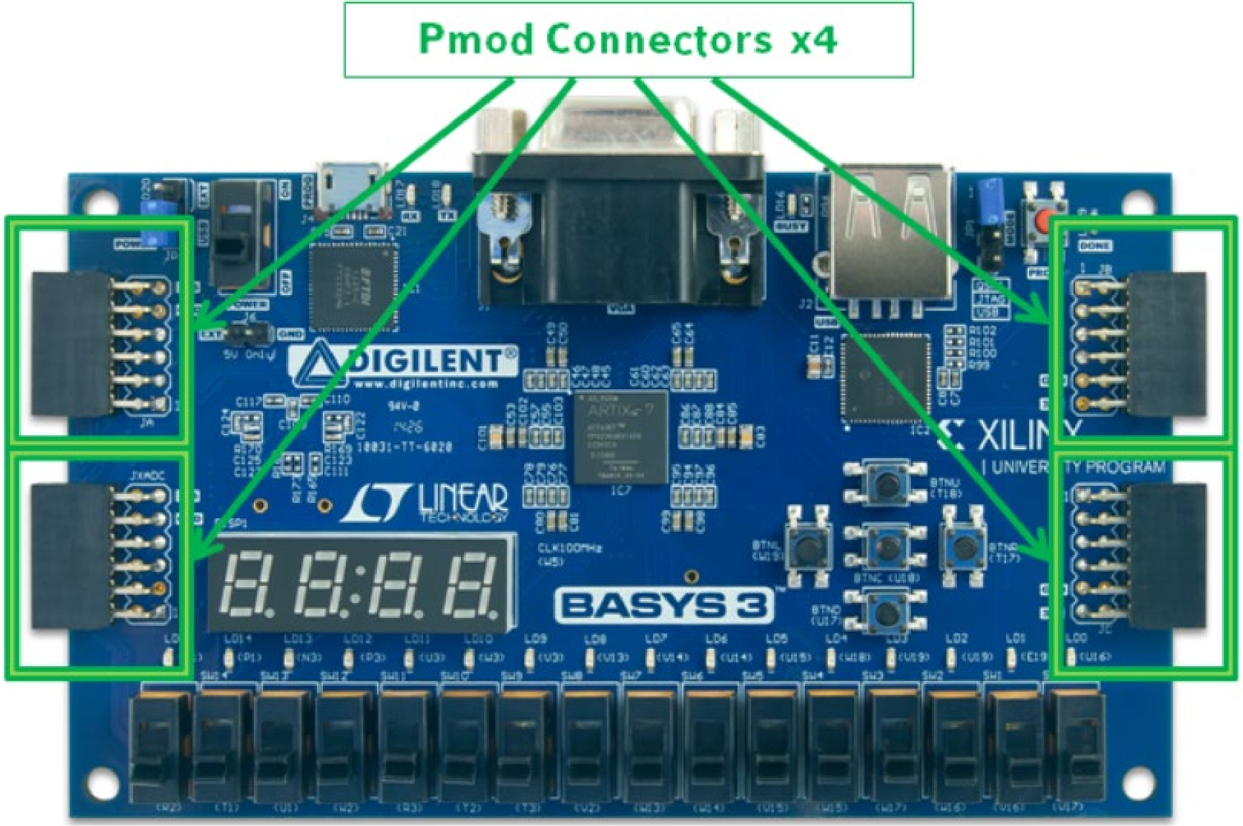
Set to 0, for writing

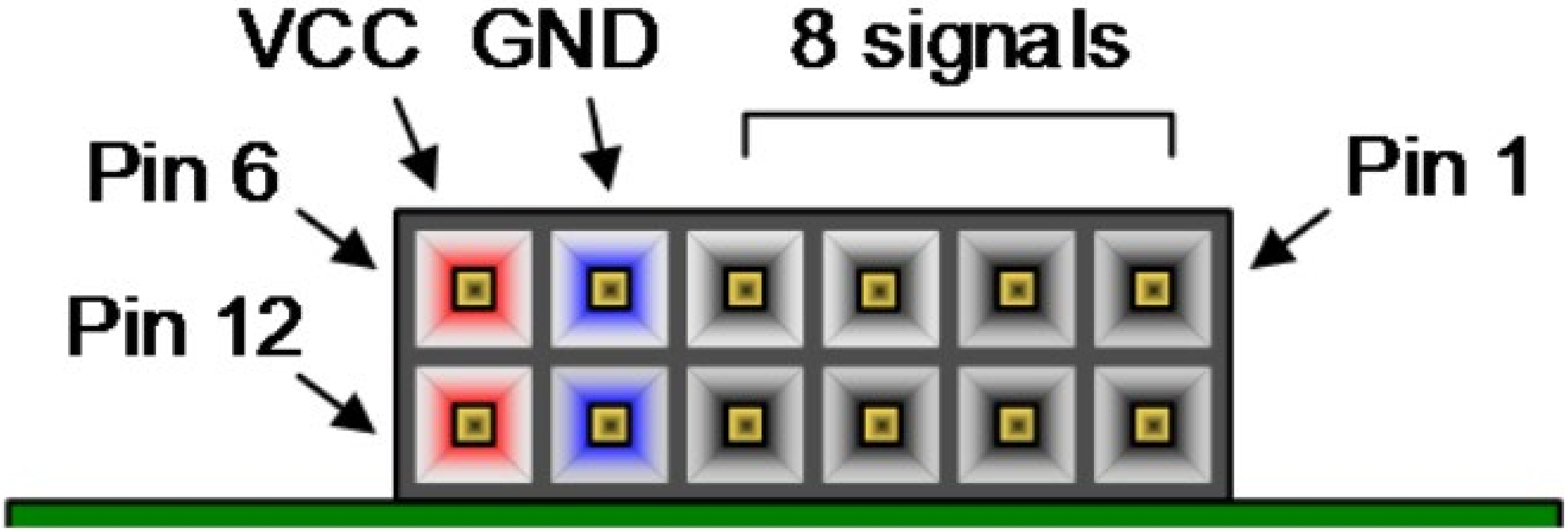
Multiple Register Write …

Figure 7. SPI write protocol




5V Power
5V Power
Ground
GPIO14 UART0_TXD
GPIO15 UART0_RXD
GPIO18 PCM_CLK
Ground
GPIO23
GPIO24
Ground
GPIO25
GPIO8 SPI0_CE0_N
GPIO7 SPI0_CE1_N
ID_SC I2C ID EEPROM
Ground
GPIO12
Ground
GPIO16
GPIO20
GPIO21
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
3V3 Power
GPIO2 SDA1 I2C
GPIO3 SCL1 I2C
GPIO4 1-wire
Ground
GPIO17
GPIO27
GPIO22
3V3 Power
GPIO10 SPI0_MOSI
GPIO9 SPI0_MISO
GPIO11 SPI0_SCLK
Ground
ID_SD I2C ID EEPROM
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
Ground
J12
GPIO
J8
Raspberry Pi 3 Model B+
© Raspberry Pi 2017
PoE
GCT
Made in the UK


Pmod Connectors x4
DIGILENT
www.digilentinc.com
LINEAR TECHNOLOGY
BASYS 3
XILINX
UNIVERSITY PROGRAM
ARTIX-7
JA
JXADC
JB
JC
BTNU
BTNL
BTNC
BTNR
BTND
CLK100MHz


VCC
GND
8 signals
Pin 6
Pin 1
Pin 12

SPI SCK, MOSI, and CE Signals

Asynchronous Input

Solution: Add a flip flop …

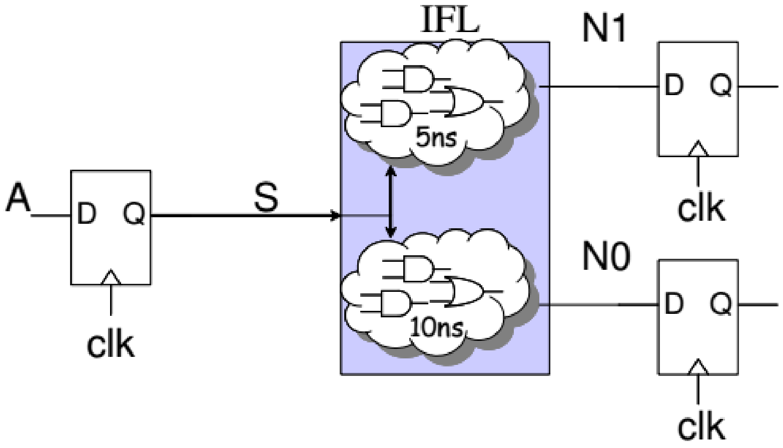
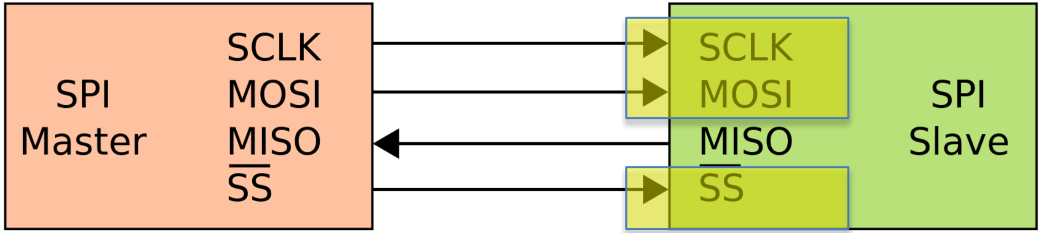


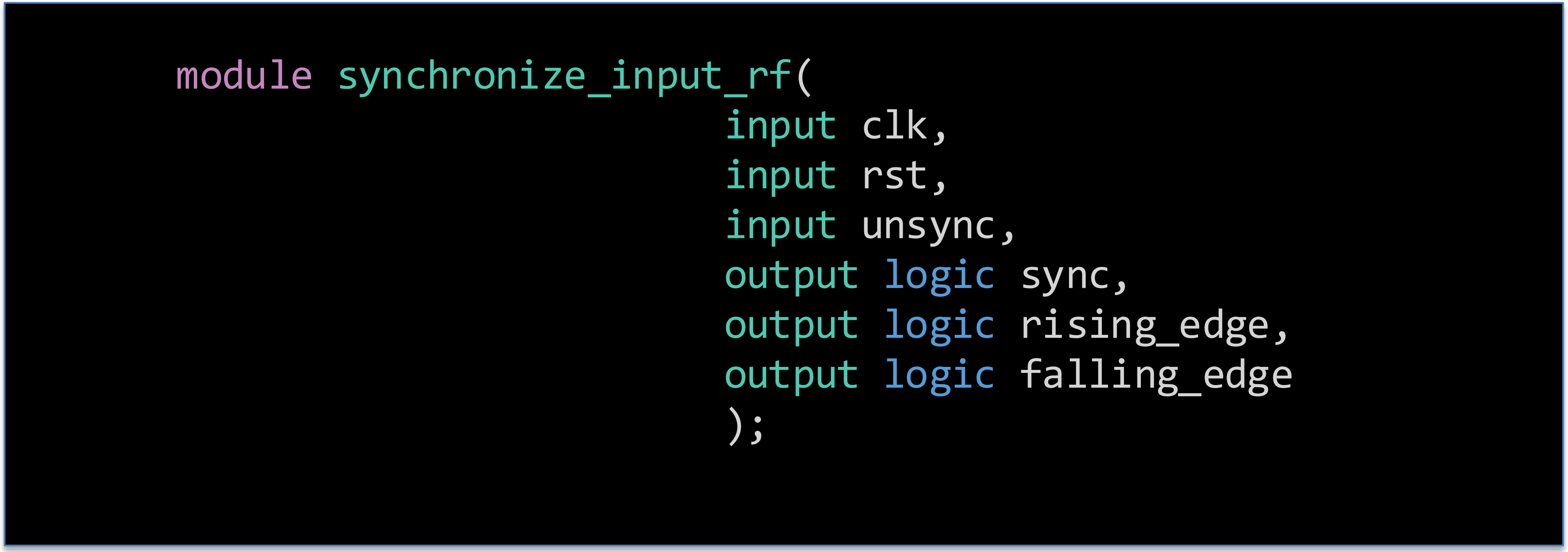
Figure 24.3: Synchronizing an Asynchronous Input

No free lunch … A is now stable but delayed by 1 clock.

General Synchronization

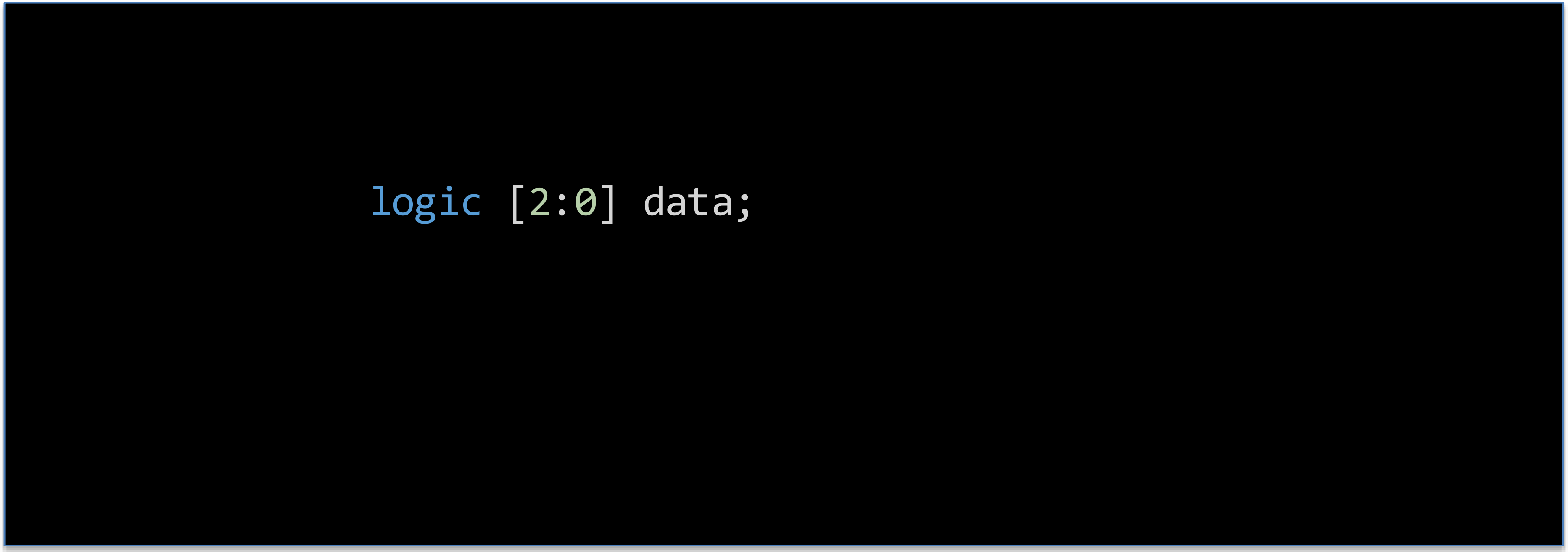


Module I/O



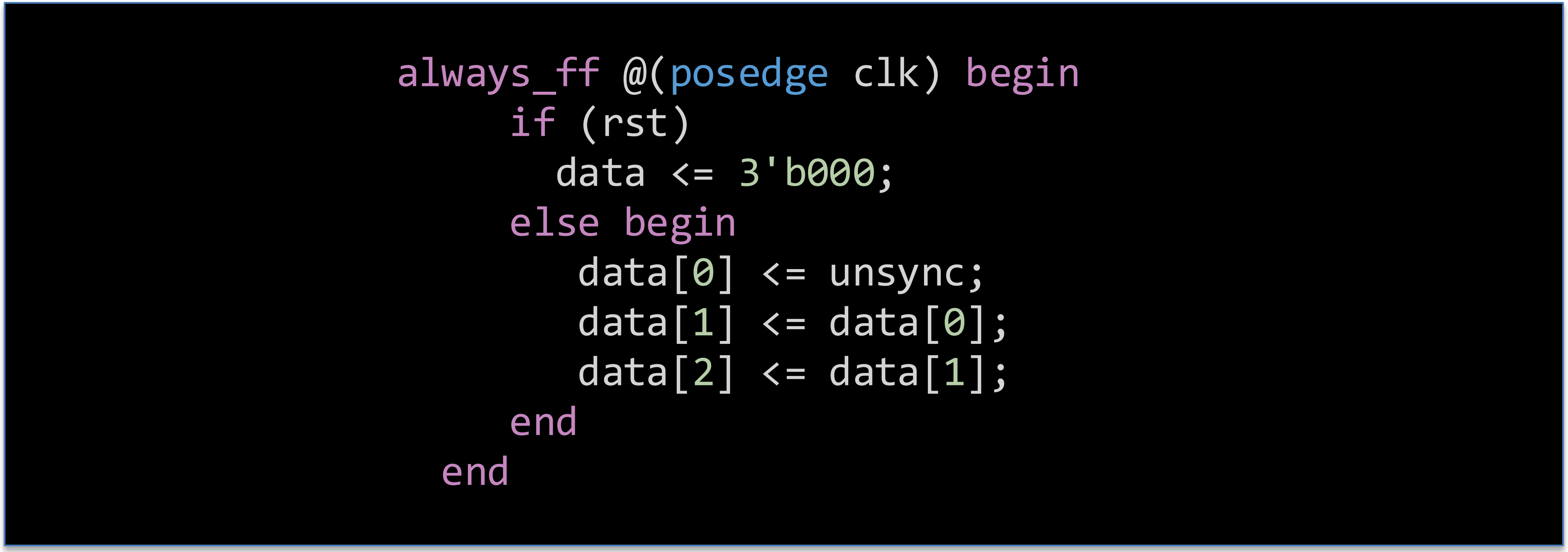
```
module synchronize_input_rf(
                        input clk,
                        input rst,
                        input unsync,
                        output logic sync,
                        output logic rising_edge,
                        output logic falling_edge
                        );
```

Internal Signals



```
logic [2:0] data;
```

Sequential Logic



```
always_ff @(posedge clk) begin
     if (rst)
       data <= 3'b000;
     else begin
        data[0] <= unsync;
        data[1] <= data[0];
        data[2] <= data[1];
     end
  end
```

Combinational Logic

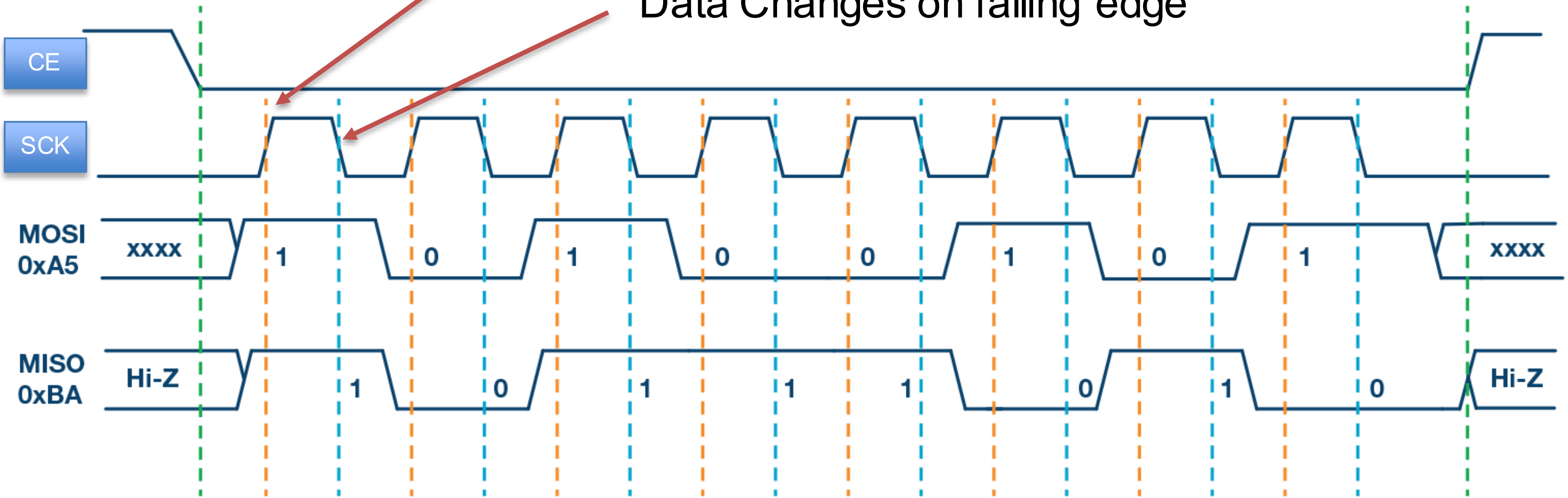
```
always_comb begin
    rising_edge = 0;
    falling_edge = 0;
    sync = data[2];
    if (data[2:1]==2'b01) begin
      rising_edge = 1;
    end
    if (data[2:1]==2'b10) begin
      falling_edge = 1;
    end
 end
```

SPI Peripheral

SPI Signal

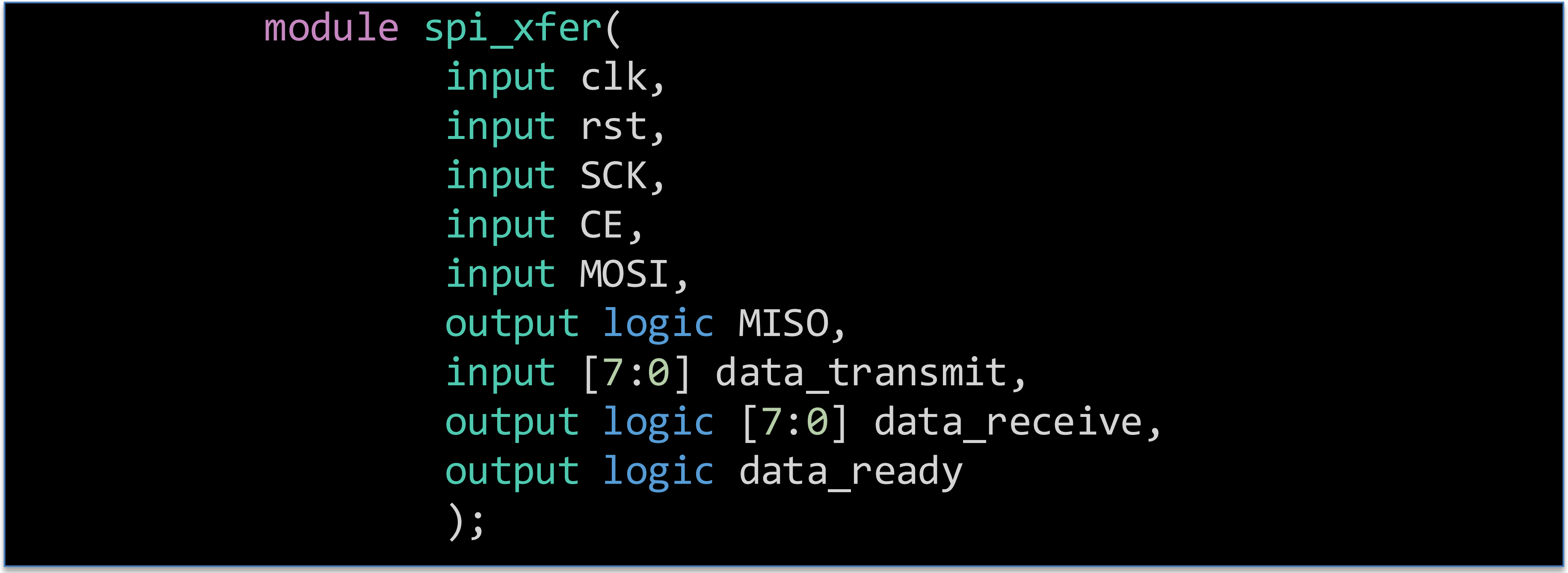
Data sampled on rising edge
Must be stable just before and after edge.

Data Changes on falling edge



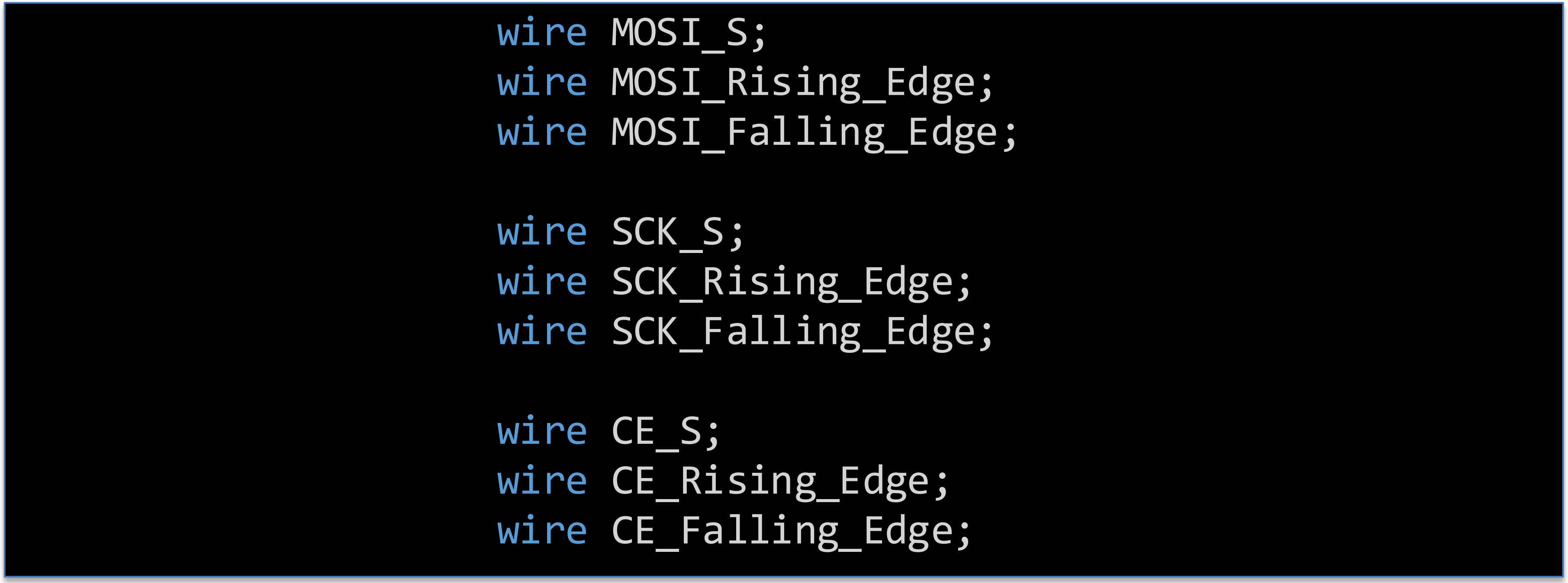
https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

Module I/O



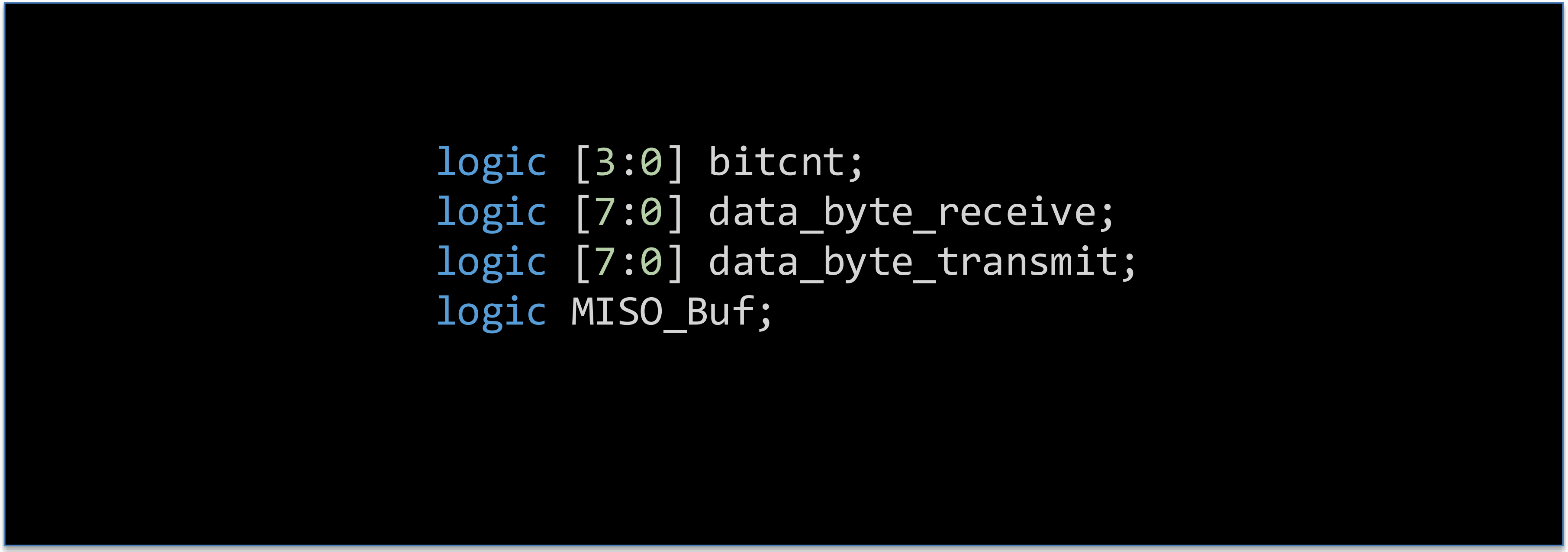
```
module spi_xfer(
        input clk,
        input rst,
        input SCK,
        input CE,
        input MOSI,
        output logic MISO,
        input [7:0] data_transmit,
        output logic [7:0] data_receive,
        output logic data_ready
        );
```

Internal Signals



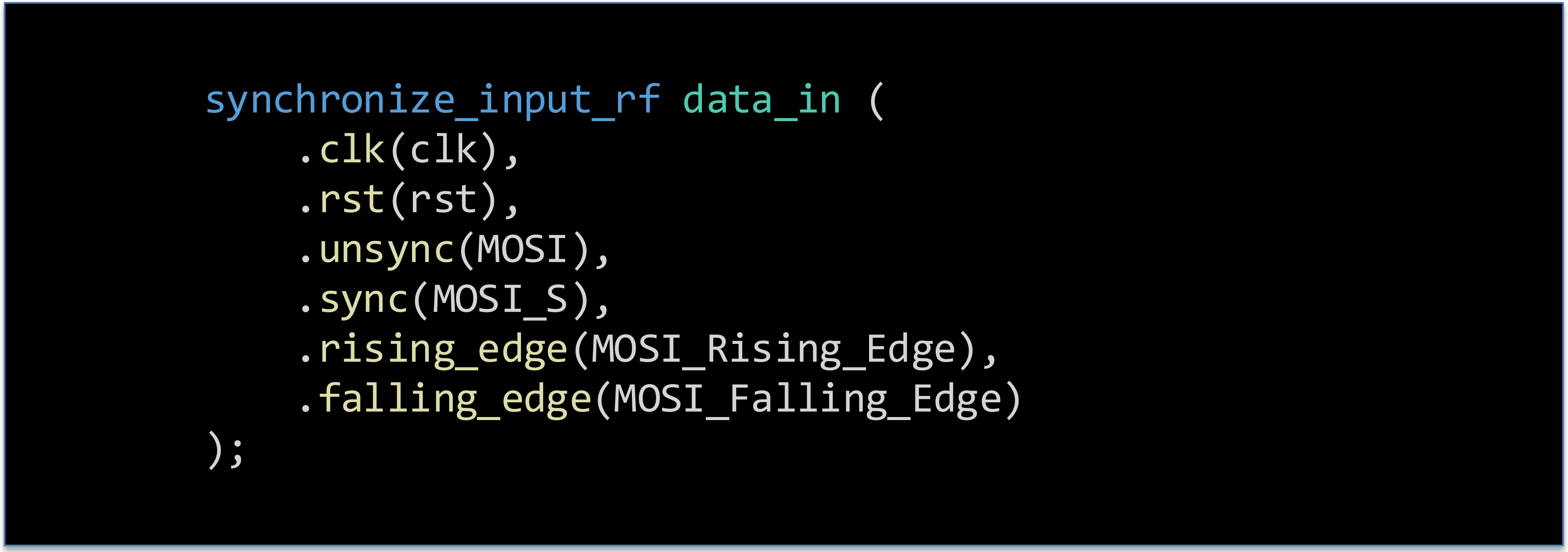
```
wire MOSI_S;
wire MOSI_Rising_Edge;
wire MOSI_Falling_Edge;

wire SCK_S;
wire SCK_Rising_Edge;
wire SCK_Falling_Edge;

wire CE_S;
wire CE_Rising_Edge;
wire CE_Falling_Edge;
```

Internal Signals (Cont.)



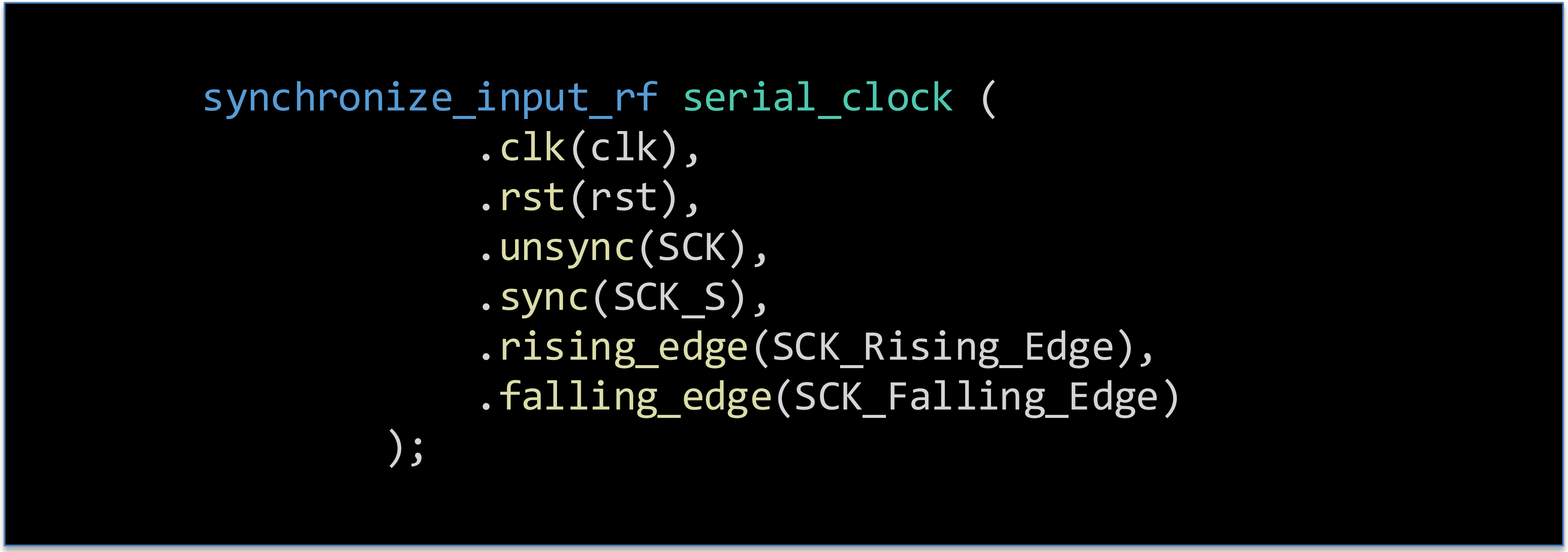
```
logic [3:0] bitcnt;
logic [7:0] data_byte_receive;
logic [7:0] data_byte_transmit;
logic MISO_Buf;
```

Component Instantiations



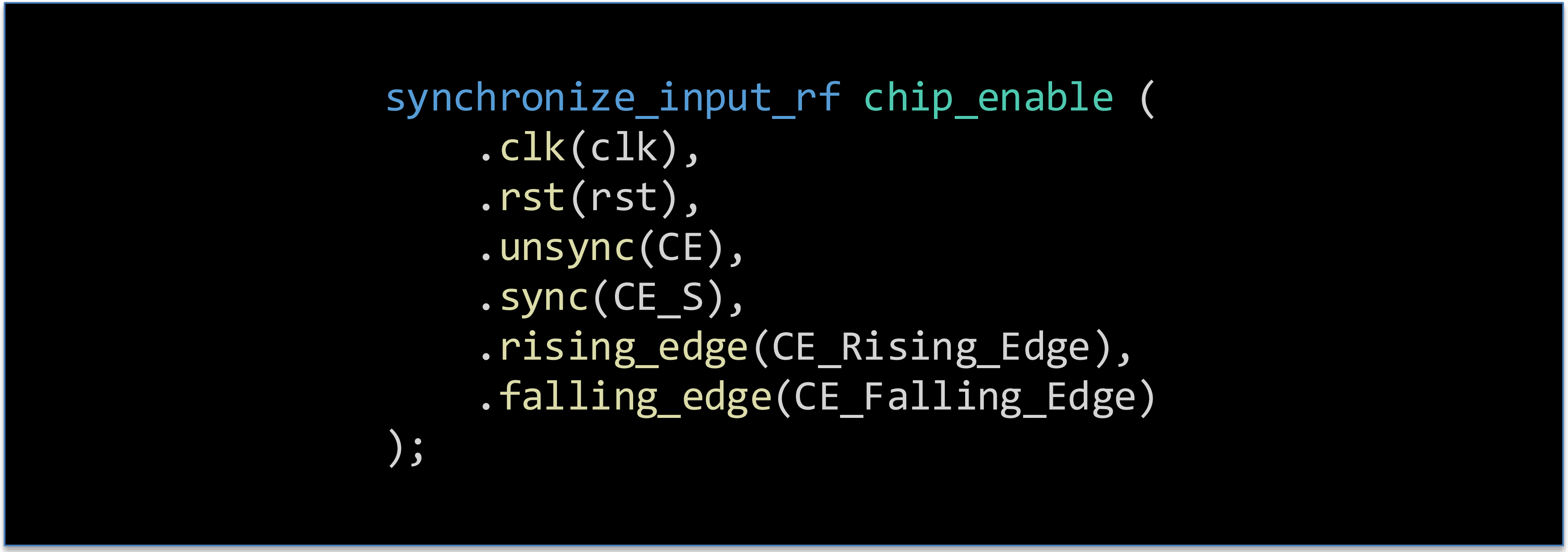
```
synchronize_input_rf data_in (
    .clk(clk),
    .rst(rst),
    .unsync(MOSI),
    .sync(MOSI_S),
    .rising_edge(MOSI_Rising_Edge),
    .falling_edge(MOSI_Falling_Edge)
);
```

Component Instantiations (cont)



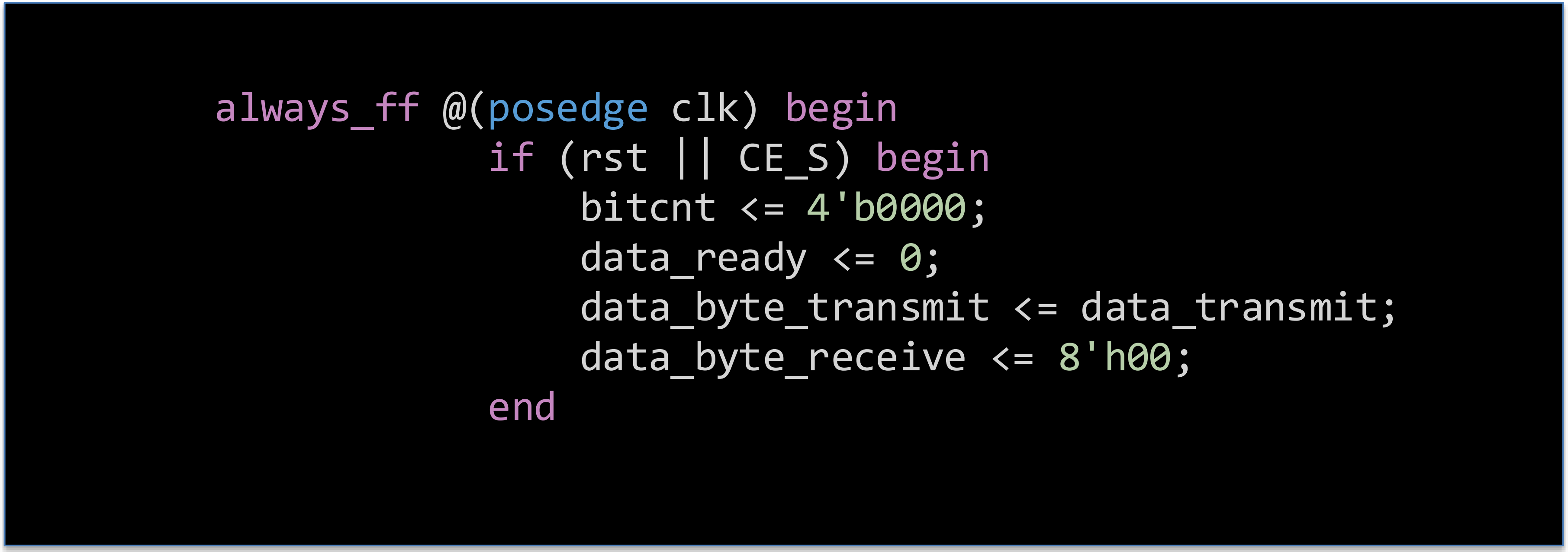
```
synchronize_input_rf serial_clock (
            .clk(clk),
            .rst(rst),
            .unsync(SCK),
            .sync(SCK_S),
            .rising_edge(SCK_Rising_Edge),
            .falling_edge(SCK_Falling_Edge)
        );
```

Component Instantiations



```
synchronize_input_rf chip_enable (
    .clk(clk),
    .rst(rst),
    .unsync(CE),
    .sync(CE_S),
    .rising_edge(CE_Rising_Edge),
    .falling_edge(CE_Falling_Edge)
);
```

Sequential Logic

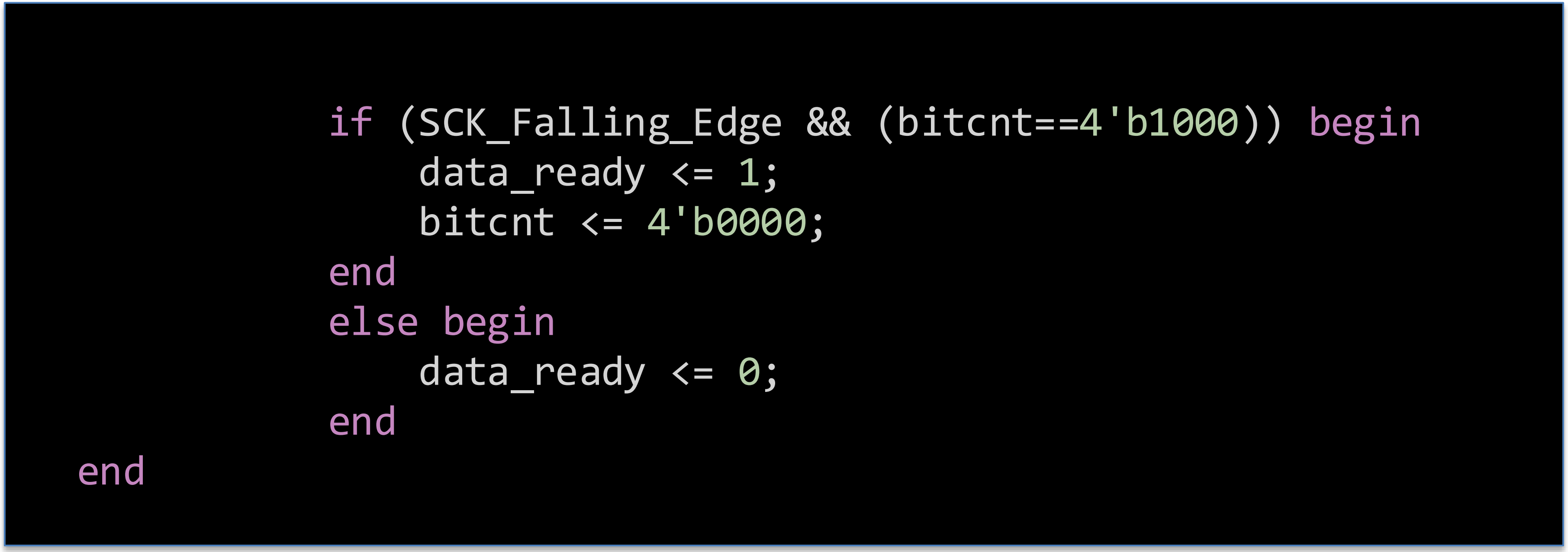


```
always_ff @(posedge clk) begin
            if (rst || CE_S) begin
                bitcnt <= 4'b0000;
                data_ready <= 0;
                data_byte_transmit <= data_transmit;
                data_byte_receive <= 8'h00;
            end
```

Sequential Logic (Cont.)

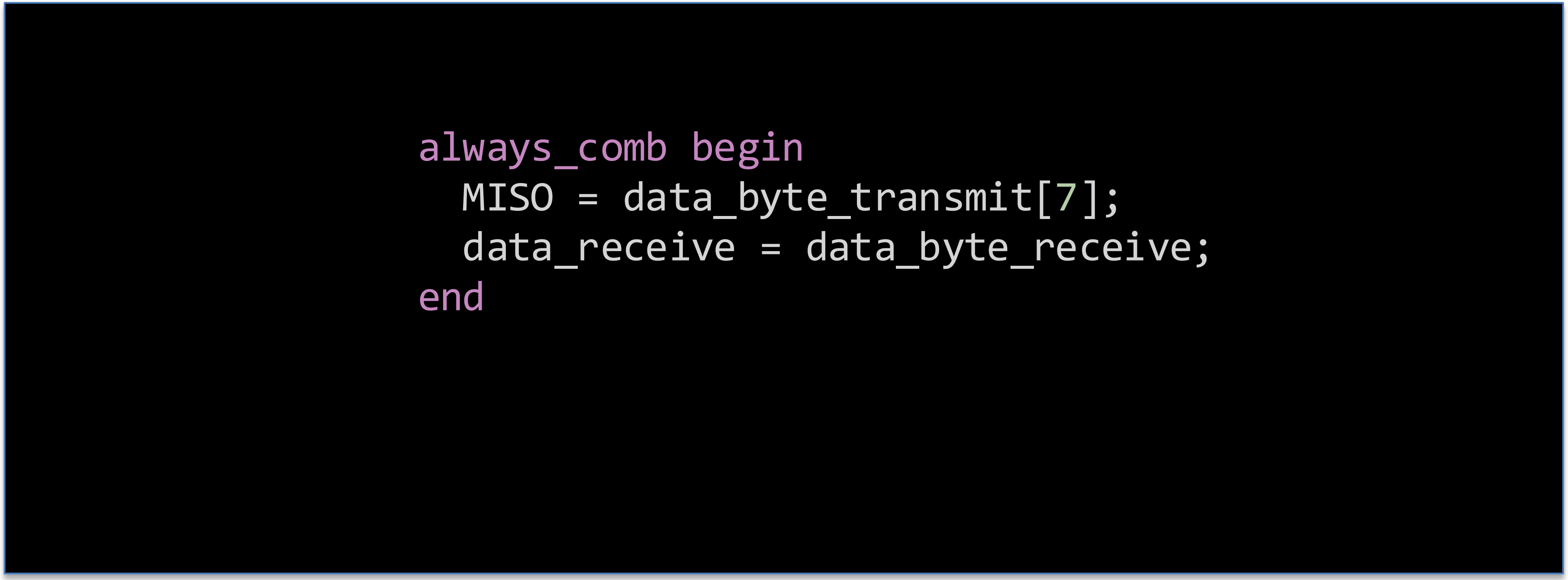
```
else begin
                if (SCK_Rising_Edge) begin
                    bitcnt <= bitcnt + 4'b0001;
                    data_byte_receive <= {data_byte_receive[6:0],MOSI_S};
                end
                if (SCK_Falling_Edge) begin
                    data_byte_transmit <= {data_byte_transmit[6:0],1'b0};

                end
```

Sequential Logic (Cont.)



```
            if (SCK_Falling_Edge && (bitcnt==4'b1000)) begin
                data_ready <= 1;
                bitcnt <= 4'b0000;
            end
            else begin
                data_ready <= 0;
            end
end
```

Combinational Logic



```
always_comb begin
  MISO = data_byte_transmit[7];
  data_receive = data_byte_receive;
end
```