

УНИВЕРСИТЕТ ИТМО

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ
И КОМПЬЮТЕРНОЙ ТЕХНИКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

СИСТЕМЫ ВВОДА-ВЫВОДА И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА
ЛАБОРАТОРНАЯ РАБОТА №1
«РАЗРАБОТКА КОНТРОЛЛЕРА ВВОДА/ВЫВОДА»

Выполнили:

Милосердов А. О.

Калугин Ф. И.

Группа Р3410

Преподаватель:

Быковский С. В.

Санкт-Петербург

2017 г.

Содержание

1	Описания задания	2
2	Краткая характеристика интерфейса	2
3	Описание разработанного контроллера	2
3.1	Описание режима работы и особенностей	2
3.2	Описание интерфейсов	3
4	Тестирование контроллера	3
5	Исходные тексты	4
5.1	SystemC	4
5.2	Verilog	8
6	Упрощенная схема контроллера	14

1. Описание задания

Для интерфейса SPI разработать модели контроллера ввода-вывода на уровне регистровых передач с использованием языка Verilog и в виде поведенческой модели с использованием языка SystemC.

Разработать тестовое окружение для каждой модели с целью проверки корректности работы контроллера. Провести тестирование обеих моделей, убедиться в корректности реализации и соответствии моделей друг другу.

2. Краткая характеристика интерфейса

SPI (Serial Peripheral Interface) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса. В SPI используются четыре цифровых сигнала:

- MOSI — выход ведущего, вход ведомого (англ. Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.
- MISO — вход ведущего, выход ведомого (англ. Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.
- SCLK — последовательный тактовый сигнал (англ. Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
- CS или SS — выбор микросхемы, выбор ведомого (англ. Chip Select, Slave Select).

Передача осуществляется пакетами. Длина пакета, как правило, составляет 1 байт (8 бит). Ведущее устройство инициирует цикл связи установкой низкого уровня на выводе выбора подчиненного устройства (SS) того устройства, с которым необходимо установить соединение.

Подлежащие передаче данные ведущее и ведомое устройства помещают в сдвиговые регистры. Передача данных осуществляется бит за битом от ведущего по линии MOSI и от ведомого по линии MISO. После передачи каждого пакета данных ведущее устройство, в целях синхронизации ведомого устройства, переводит линию SS в высокое состояние.

3. Описание разработанного контроллера

3.1. Описание режима работы и особенностей

Разработанный контроллер работает только в режиме Master, mode 0 (CPOL = 0, CPHA = 0). Частота синхросигнала в 4 раза меньше основной частоты тактового сигнала. Синхросигнал генерируется всё время активности контроллера (как, например, в ATE SPI). Работает на одном сдвиговом регистре.

3.2. Описание интерфейсов

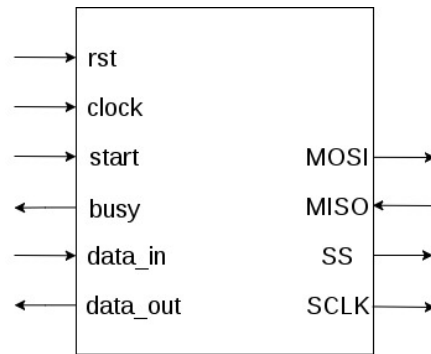


Рис. 1: Интерфейсы контроллера

На данный момент разработанный контроллер работает с одним ведомым устройством. Контроллер предоставляет следующие входы и выходы:

- rst – сигнал сброса состояния контроллера;
- clock – тактовый сигнал для работы контроллера;
- start – сигнал инициализации транзакции;
- busy – сигнал, означающий, что транзакция ещё не закончена;
- data_in – регистр, содержащий пакет на отправку;
- data_out – регистр для хранения полученного пакета от ведомого устройства.

4. Тестирование контроллера

В качестве тестирования было проведено две штатные транзакции и одна, прерванная сигналом сброса. Данные на отправку, сигнал инициализации транзакции и отправка данных по mosi осуществляется тестовым устройством.

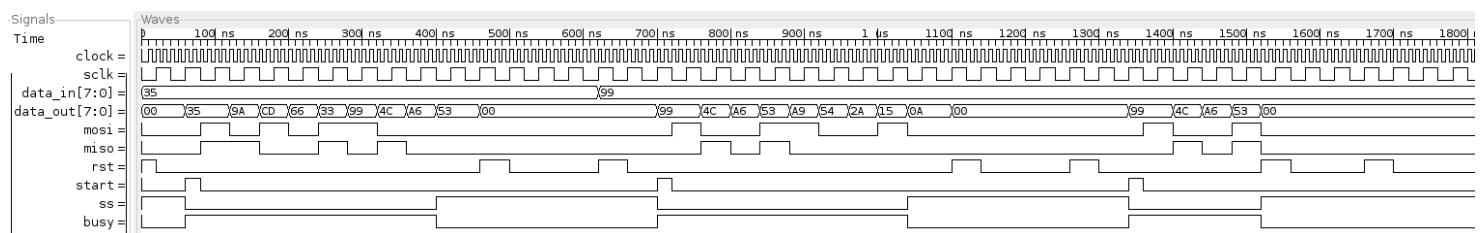


Рис. 2: Временные диаграммы SystemC

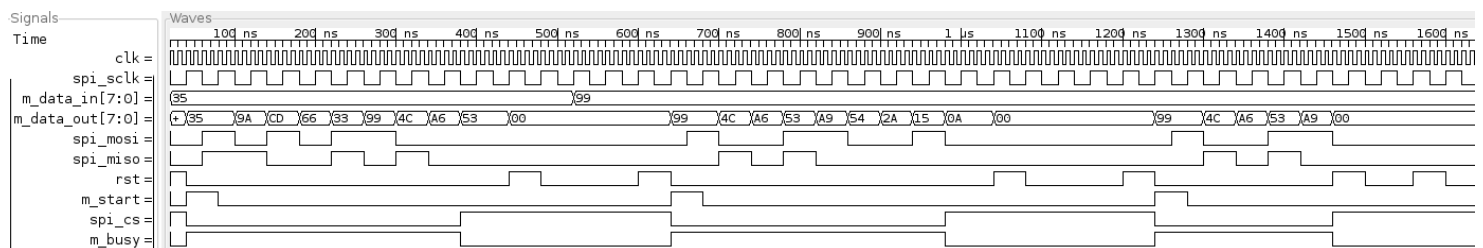


Рис. 3: Временные диаграммы Verilog

5. Исходные тексты

5.1. SystemC

Листинг 1: Объявление контроллера

```
1  /*
2  SPI module, working in mode 0 (CPOL = 0, CPHA = 0), master only, transieves 8 bits in 8 cycles.
3  Ports:
4      clk    -- main input clock
5      sclk   -- synchronous clock for interaction between SPI modules. 4 times slower than clk.
6      rst    -- reset signal, edge sensitive
7      busy   -- indicates transaction in progress.
8      ss     -- slave select. Is low for 8 cycles, set to high on last positive sclk edge of transaction.
9      start  -- signal to start transaction, edge sensitive.
10
11      data_in/data_out -- mosi/miso registers
12      mosi/miso -- master out/master in wires.
13                  Set to high on falling edge of sclk,
14                  read happens on falling edge of sclk,
15                  write to data_in/data_out on rising edge of sclk.
16 */
17
18 #include "systemc.h"
19 #include "clock.h"
20
21 SC_MODULE( spi ) {
22     sc_in<sc_uint<8> > data_in;
23     sc_out<sc_uint<8> > data_out;
24
25     // Shift register
26     sc_uint<8> shiftreg;
27
28     sc_uint<3> ctr;
29
30     sc_in<bool> clk, rst, start, miso;
31     sc_out<bool> sclk, ss, mosi, busy;
32
33     // SPI clock to generate sclk
34     clock_gen clk_gen;
35
36     // Flag for transaction start
37     bool trans_start;
38
39     // Indicate last bit transmission
40     bool last;
41
42     // Flag indicating first tick of transaction
43     bool first;
44
45     bool buf;
46
47     void rx( );
48     void tx( );
49     void reset( );
50     void end_transaction( );
51     void loop( );
```

```

52
53 SC_CT0R( spi ):
54   clk( "CLK" ), rst( "RST" ), miso( "MISO" ), sclk( "SCLK" ),
55   ss( "SS" ), mosi( "MOSI" ), clk_gen( "CLK_GEN" ) {
56
57       // Clock generator for sclk
58       clk_gen.clock( clk );
59       clk_gen.qclk( sclk );
60
61       trans_start = 0;
62
63       SC_METHOD( loop );
64       sensitive << sclk << rst.pos( ) << start.pos( );
65   }
66 };

```

Листинг 2: Логика контроллера

```

1  // SPI, mode 0, only master mode, transieve from 0th bit to 7th
2
3  // Ports are assigned using write and local vars are assigned using `=` just for readability.
4  #include "spi.h"
5
6  // SPI receive
7  void spi::rx( ) {
8      //shiftreg[7] = miso.read( );
9      buf = miso.read( );
10     //cout << "@" << sc_time_stamp( ) << " Receive miso/ctr: " << miso.read( ) << "/" << ctr << endl;
11 }
12
13 // SPI transmit
14 void spi::tx( ) {
15     mosi.write( shiftreg[0] );
16 }
17
18 // On reset end transaction end reset output data
19 void spi::reset( ) {
20     end_transaction( );
21     data_out.write( 0 );
22
23 }
24
25 /* On transaction end:
26     MOSI -- Low
27     SS -- High
28     Busy -- Low
29
30     Reset counter, disable toggler and set last bit flag to 0
31 */
32 void spi::end_transaction( ) {
33     ss.write( 1 );
34     ctr = 0;
35     busy.write( 0 );
36
37     mosi.write( 0 );
38

```

```

39     trans_start = 0;
40     last = 0;
41     first = 1;
42 }
43
44 // Main SPI loop
45 void spi::loop( ) {
46
47     trans_start = trans_start | ( start.read( ) & !busy.read( ) );
48
49     if( rst ) {
50         reset( );
51         return;
52     }
53
54     // Assign shiftreg to input packet
55     if( first && sclk && trans_start ) {
56         shiftreg = data_in.read( );
57     }
58
59     // Main logic on every sclk tick
60     if( sclk ) {
61         if( trans_start ) {
62             if( first ) {
63                 data_out.write( shiftreg );
64                 busy.write( 1 );
65                 ss.write( 0 );
66             }
67             rx( );
68         }
69
70         } else if( busy ) {
71
72
73             if( !first ) {
74                 shiftreg = shiftreg >> 1;
75                 shiftreg[7] = buf;
76             }
77
78             tx( );
79
80             if( !last && !first ) {
81                 ctr++;
82
83             } else if( last ) {
84                 rx( );
85                 end_transaction( );
86             }
87
88             first = 0;
89
90             if( ctr == 7 ) last = 1;
91
92             data_out.write( shiftreg );
93         }
94

```

```
95  
96 }
```

Листинг 3: Тестовое устройство

```
1  #include "test_spi.h"  
2  
3  void wait_fall( ) {  
4      wait( 40, SC_NS );  
5  }  
6  
7  void test_spi::test_send( uint8_t in, uint8_t out, bool reset ) {  
8  
9      data_in.write( in );  
10  
11      rst.write( 1 );  
12  
13      wait( );  
14      rst.write( 0 );  
15      wait( );  
16  
17      start.write( 1 );  
18  
19      wait( 20, SC_NS );  
20  
21      start.write( 0 );  
22  
23      msg = out;  
24  
25      for( counter = 0; counter < 8; counter++ ) {  
26          if( counter >= 4 && reset ) {  
27              miso.write( 0 );  
28              rst.write( 1 );  
29              wait_fall( );  
30              rst.write( 0 );  
31              wait_fall( );  
32              break;  
33          } else {  
34              miso.write( msg << 7 & 0xFF );  
35              msg = msg >> 1;  
36              wait_fall( );  
37          }  
38      }  
39  
40      wait( );  
41      wait( );  
42  
43      // Reset data_out  
44      rst.write( 1 );  
45      wait( );  
46      rst.write( 0 );  
47  
48      for ( int i = 0; i < 3; ++i ) {  
49          wait( );  
50      }  
51
```



```

52 }
53
54 void test_spi::demo_send( ) {
55
56     test_send( 0b00110101, 0b01010011, false );
57     test_send( 0b10011001, 0b00001010, false );
58     test_send( 0b10011001, 0b00001010, true );
59
60     sc_stop( );
61 }

```

5.2. Verilog

Листинг 4: Ведущий контроллера

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Simple SPI master controller with CPOL=0, CPHA=0
4  ///////////////////////////////////////////////////////////////////
5
6  module spi_master_driver(
7      input          clk_i,
8      input          rst_i,
9
10     // system interface
11     input          start_i,    // signal to start transaction
12     input          [7:0] data_in_bi, // data that master will write to slave
13     output reg     busy_o,      // transaction is being processed
14     output reg     [7:0] data_out_bo, // data received from slave in last transaction
15
16     // SPI interface
17     input          spi_miso_i,
18     output reg     spi_mosi_o,
19     output reg     spi_sclk_o = 0,
20     output reg     spi_cs_o
21 );
22
23     reg [3:0] counter;
24     reg clk_div2 = 1;
25     reg mosi_enable;
26     reg [7:0] shiftreg;
27     reg bit_buffer;
28
29     localparam STATE_IDLE          = 0; // wait for transaction begin
30     localparam STATE_WAIT_SCLK_1   = 1; // wait for SCLK to become 1
31     localparam STATE_WAIT_SCLK_0   = 2; // wait for SCLK to become 0
32     localparam STATE_WAIT_SCLK_0_START = 3; // wait for SCLK to become 0, no shift
33
34
35     reg [2:0] state;
36
37     always @(posedge clk_i) begin
38         if (clk_div2) begin
39             spi_sclk_o = !spi_sclk_o;
40         end
41         clk_div2 <= !clk_div2;

```

```

42     if (rst_i) begin
43         counter    <= 0;
44         shiftreg    <= 0;
45         bit_buffer  <= 0;
46         data_out_bo <= 0;
47         spi_mosi_o  <= 0;
48         state       <= STATE_IDLE;
49         mosi_enable <= 0;
50     end
51     else begin
52         case (state)
53             STATE_IDLE: begin
54                 if (start_i) begin
55                     shiftreg <= data_in_bi;
56                     mosi_enable <= 0;
57                     state <= STATE_WAIT_SCLK_0_START;
58                 end
59             end
60             STATE_WAIT_SCLK_1: begin
61                 if (spi_sclk_o == 1) begin
62                     bit_buffer <= spi_miso_i;
63                     state <= STATE_WAIT_SCLK_0;
64                 end
65             end
66             STATE_WAIT_SCLK_0: begin
67                 if (spi_sclk_o == 0) begin
68                     shiftreg <= { bit_buffer, shiftreg[7:1] };
69                     state <= STATE_WAIT_SCLK_1;
70                     if (counter == 7) begin
71                         mosi_enable <= 0;
72                         state <= STATE_IDLE;
73                         counter <= 0;
74                         mosi_enable <= 0;
75                     end else begin
76                         counter <= counter + 1;
77                     end
78                 end
79             end
80             STATE_WAIT_SCLK_0_START: begin
81                 if (spi_sclk_o == 0) begin
82                     mosi_enable <= 1;
83                     state <= STATE_WAIT_SCLK_1;
84                 end
85             end
86             default: begin
87                 state <= STATE_IDLE;
88             end
89         endcase
90     end
91 end
92
93 always @* begin
94     busy_o = (state != STATE_IDLE);
95     spi_cs_o = (state == STATE_IDLE);
96
97     data_out_bo = shiftreg;

```

```

98         spi_mosi_o = shiftreg[0] && mosi_enable;
99     end
100
101
102 endmodule

```

Листинг 5: Ведомый контроллер

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Simple SPI slave controller with CPOL=0 CPHA=0
4  ///////////////////////////////////////////////////////////////////
5
6  module spi_slave_driver(
7      input          clk_i,
8      input          rst_i,
9
10     // system interface
11     input          [7:0] data_in_bi, // data that master can read from slave
12     output reg     ready_o, // transaction is not processed now
13     output reg [7:0] data_out_bo, // data written to slave in last transaction
14
15     // SPI interface
16     output reg     spi_miso_o,
17     input          spi_mosi_i,
18     input          spi_sclk_i,
19     input          spi_cs_i
20 );
21
22     reg [3:0] counter;
23     reg [7:0] shiftreg;
24     reg      bit_buffer;
25     reg      miso_enable;
26     localparam STATE_IDLE = 0; // wait for transaction begin
27     localparam STATE_WAIT_SCLK_1 = 1; // wait for SCLK to become 1
28     localparam STATE_WAIT_SCLK_0 = 2; // wait for SCLK to become 0
29     localparam STATE_WAIT_SCLK_0_START = 3; // wait for SCLK to become 0
30
31     reg [2:0] state;
32
33     always @(posedge rst_i, posedge clk_i) begin
34         if (rst_i == 1) begin
35             shiftreg <= 0;
36             bit_buffer <= 0;
37
38             data_out_bo <= 0;
39             counter <= 0;
40             spi_miso_o <= 0;
41             miso_enable <= 0;
42             state <= STATE_IDLE;
43         end
44         else begin
45             if (spi_cs_i) begin
46                 state <= STATE_IDLE;
47             end
48             else begin

```

```

49         case (state)
50             STATE_IDLE: begin
51                 if (!spi_cs_i) begin
52                     miso_enable <= 0;
53                     counter      <= 0;
54                     state <= STATE_WAIT_SCLK_0_START;
55                 end
56             end
57             STATE_WAIT_SCLK_1: begin
58                 if (spi_sclk_i) begin
59                     bit_buffer <= spi_mosi_i;
60
61                     state <= STATE_WAIT_SCLK_0;
62                 end
63             end
64             STATE_WAIT_SCLK_0: begin
65                 if (spi_sclk_i == 0) begin
66                     shiftreg <= { bit_buffer, shiftreg[7:1] };
67                     miso_enable <= 1;
68                     state <= STATE_WAIT_SCLK_1;
69                     if (counter == 7)
70                         miso_enable <= 0;
71                     counter <= counter + 1;
72                 end
73             end
74             STATE_WAIT_SCLK_0_START: begin
75                 if (spi_sclk_i == 0) begin
76                     shiftreg <= data_in_bi;
77                     miso_enable <= 1;
78                     state <= STATE_WAIT_SCLK_1;
79                 end
80             end
81             default: begin
82                 state <= STATE_IDLE;
83             end
84         endcase
85     end
86 end
87 end
88
89 always @* begin
90     ready_o = (state == STATE_IDLE);
91     data_out_bo = shiftreg;
92
93     spi_miso_o = shiftreg[0] && miso_enable && !spi_cs_i;
94 end
95
96 endmodule

```

Листинг 6: Тестовое окружение

```

1  `timescale 1ns / 1ps
2  //////////////////////////////////////
3  // SPI controllers testbench
4  //////////////////////////////////////
5

```

```

6  module spi_drivers_tb(
7      );
8
9      localparam CLK_HALFPERIOD = 5;
10
11     reg          clk, rst;
12
13     reg          m_start;
14     reg  [7:0] m_data_in;
15     wire         m_busy;
16     wire  [7:0] m_data_out;
17
18     reg  [7:0] s_data_in;
19     wire         s_ready;
20     wire  [7:0] s_data_out;
21
22     wire         spi_miso;
23     wire         spi_mosi;
24     wire         spi_sclk;
25     wire         spi_cs;
26
27     spi_master_driver dut_m(
28         .clk_i(clk),
29         .rst_i(rst),
30
31         .start_i(m_start),
32         .data_in_bi(m_data_in),
33         .busy_o(m_busy),
34         .data_out_bo(m_data_out),
35
36         .spi_miso_i(spi_miso),
37         .spi_mosi_o(spi_mosi),
38         .spi_sclk_o(spi_sclk),
39         .spi_cs_o(spi_cs)
40     );
41
42     spi_slave_driver dut_s(
43         .clk_i(clk),
44         .rst_i(rst),
45
46         .data_in_bi(s_data_in),
47         .ready_o(s_ready),
48         .data_out_bo(s_data_out),
49
50         .spi_miso_o(spi_miso),
51         .spi_mosi_i(spi_mosi),
52         .spi_sclk_i(spi_sclk),
53         .spi_cs_i(spi_cs)
54     );
55
56     always #(CLK_HALFPERIOD) clk = !clk;
57
58     initial begin
59         clk = 1;
60         rst = 1;
61         m_start = 0;

```

```

62
63     m_data_in = 8'b00110101;
64     s_data_in = 8'b01010011;
65
66     #20;
67     $dumpfile ("spi_verilog.vcd");
68     $dumpvars;
69
70     #20;
71
72     rst = 0;
73     m_start = 1;
74     #40;
75     m_start = 0;
76     #360;
77     rst = 1;
78     #40;
79     rst = 0;
80     #40;
81     m_data_in = 8'b10011001;
82     s_data_in = 8'b00001010;
83     #80;
84     rst = 1;
85     #40;
86
87     rst = 0;
88     m_start = 1;
89     #40;
90     m_start = 0;
91     #360;
92     rst = 1;
93     #40;
94     rst = 0;
95     #40;
96     m_data_in = 8'b10011001;
97     s_data_in = 8'b00001010;
98     #80;
99     rst = 1;
100    #40;
101
102    rst = 0;
103    m_start = 1;
104    #40;
105    m_start = 0;
106    #180;
107    rst = 1;
108    #40;
109    rst = 0;
110    #60;
111    rst = 1;
112    #40;
113    rst = 0;
114
115    #1000;
116    $finish;
117 end

```

6. Упрощенная схема контроллера

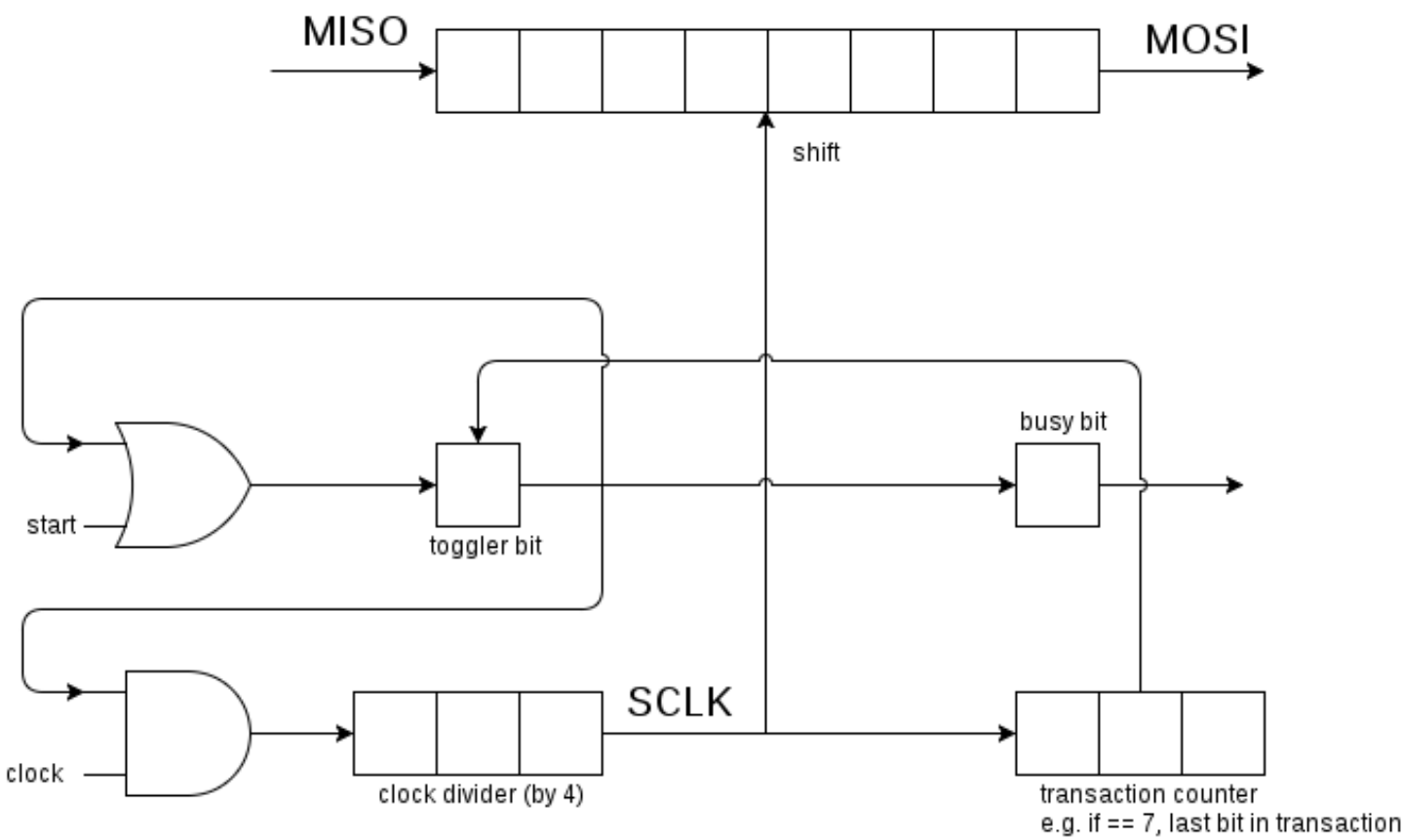


Рис. 4: Упрощенная схема