

УНИВЕРСИТЕТ ИТМО

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ  
И КОМПЬЮТЕРНОЙ ТЕХНИКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

СИСТЕМЫ ВВОДА-ВЫВОДА И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

ЛАБОРАТОРНАЯ РАБОТА №2

«РАЗРАБОТКА ВЫСОКОУРОВНЕВОЙ МОДЕЛИ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ»

Вариант №6

*Выполнили:*

Милосердов А. О.

Калутин Ф. И.

Группа Р3410

*Преподаватель:*

Быковский С. В.

Санкт-Петербург

2017 г.

# Содержание

<b>1</b>	<b>Описания работы</b>	<b>2</b>
<b>2</b>	<b>Описание фрагментов модели</b>	<b>2</b>
2.1	Регистровая карта . . . . .	2
2.2	Контроллер дискретного ввода-вывода . . . . .	2
2.3	Контроллер ввода-вывода периферийного интерфейса . . . . .	3
2.4	Коммутатор шины . . . . .	5
2.5	Периферийное устройство . . . . .	6
2.6	Программное обеспечение . . . . .	8
<b>3</b>	<b>Временные диаграммы</b>	<b>11</b>
<b>4</b>	<b>Отладочный вывод</b>	<b>11</b>

# 1. Описания работы

В данной работе разрабатывается высокоуровневая модель простой микропроцессорной системы с периферийным устройством. В состав модели входят:

- упрощенное описание аппаратного обеспечения микропроцессорной системы;
- упрощенное описание периферийного устройства;
- прототип программного обеспечения для микропроцессора.

В качестве периферийного устройства используется PmodJSTK. Поддерживается запись в устройство и чтение. Чтение происходит 5 байтными кусками:

1. первый байт, отправленный устройством, содержит 8 младших бит координаты X и первый принятый байт интерпретируется как значения светодиодов на плате PmodJSTK;
2. второй байт, отправленный устройством, содержит 2 старших бита координаты X;
3. третий байт, отправленный устройством, содержит 8 младших бит координаты Y;
4. четвертый байт, отправленный устройством, содержит 2 старших бита координаты Y;
5. пятый байт, отправленный устройством, содержит информацию о нажатых кнопках устройства.

## 2. Описание фрагментов модели

### 2.1. Регистровая карта

Таблица 1: Регистровая карта системы

Устройство	Адрес	Название	Биты	Назначение
дискретный контроллер	0x40000004	DD_IN	[15:0]	установка состояния светодиодов
дискретный контроллер	0x40000008	DD_OUT	[15:0]	чтение состояния переключателей
периферийный контроллер	0x40001000	SA_ST	[0]	начало обмена с периферийным устройством
периферийный контроллер	0x40001004	SA_SS	[0]	управление сигналом Slave Select
периферийный контроллер	0x40001008	SA_RE	[0]	сигнал состояния периферийного контроллера
периферийный контроллер	0x40001012	SA_DA	[7:0]	полученные данные с периферийного устройства

### 2.2. Контроллер дискретного ввода-вывода

Листинг 1: Объявление контроллера

```
1  /*
2   Digital controller for switches and leds control for Nexys4 DDR board.
3  */
4
5  #pragma once
6  #include "systemc.h"
7
8  // Register map of bus slave
9  #define DIN_DOUT_IN_REG    0x00000004
10 #define DIN_DOUT_OUT_REG   0x00000008
```

```

11
12 SC_MODULE( din_dout ) {
13     sc_in<bool> hclk_i;
14     sc_in<bool> n_hreset_i;
15     sc_in<sc_uint<32> > haddr_bi;
16     sc_in<sc_uint<32> > hwddata_bi;
17     sc_out<sc_uint<32> > hrdata_bo { "hrdata_bo" };
18     sc_in<bool> hwrite_i { "hwrite_i" };
19     sc_in<bool> hsel_i { "hsel_i" };
20
21     sc_in<sc_uint<16> > switches { "switches" };
22     sc_inout<sc_uint<16> > leds { "leds" };
23
24     SC_CTOR( din_dout ) {
25         SC_METHOD( bus_slave );
26         sensitive << hclk_i.pos( ) << n_hreset_i.neg( );
27     }
28
29     void set_base_address( sc_uint<32> base_addr ) {
30         this->base_addr = base_addr;
31     }
32
33     private:
34         sc_uint<32> base_addr;
35
36         void bus_slave( );
37
38         sc_uint<32> execute_read( sc_uint<16> addr );
39         void execute_write( sc_uint<16> addr, sc_uint<32> data );
40 };

```

## 2.3. Контроллер ввода-вывода периферийного интерфейса

Листинг 2: Объявление контроллера

```

1  /*
2   Controller to connect SPI slave device to AMBA AHB bus
3  */
4
5  #pragma once
6  #include "systemc.h"
7  #include "spi.h"
8
9  // Register map
10 #define SPI_AHB_START      0x00000000 // Write-only
11 #define SPI_AHB_SS         0x00000004
12 #define SPI_AHB_READY      0x00000008 // Read-only
13 #define SPI_AHB_DATA       0x00000012 // Read-only
14
15 SC_MODULE( spi_ahb ) {
16
17     // SPI master to connect to peripheral device
18     spi_m* spi;
19
20     sc_in<bool> clk { "clk" };
21

```

```

22  sc_out<bool> start  { "start" };
23  sc_out<bool> busy   { "busy" };
24
25  // SPI wires (for slave connection)
26  sc_in<bool> miso { "miso" };
27  sc_out<bool> mosi { "mosi" }, sclk { "sclk" }, ss { "ss" }, rst { "rst" };
28
29  sc_out<sc_uint<SPI_BIT_CAP> > data_out { "data_out" };
30  sc_inout<sc_uint<SPI_BIT_CAP> > data_in { "data_in" };
31
32  // AMBA AHB compliant ports to connect to AHB bus
33  sc_in<bool> hwrite { "hwrite" }, hsel { "hsel" };
34  sc_inout<bool> n_hreset { "n_hreset" }; // To send it to SPI rst ports
35
36  sc_in<sc_uint<32> > haddr { "haddr" };
37  sc_in<sc_uint<32> > hwdata { "hwdata" };
38  sc_out<sc_uint<32> > hrdata { "hrdata" };
39
40  sc_uint<1> ready; // Indicates that SPI transaction is finished
41
42  sc_uint<32> buf_data;
43  sc_uint<32> buf_rdata;
44  sc_uint<32> buf_wdata;
45  sc_uint<32> buf_addr;
46
47  enum {
48      SPI_AHB_IDLE,
49      SPI_AHB_READ_START,
50      SPI_AHB_READ_DONE,
51      SPI_AHB_WRITE_START,
52      SPI_AHB_WRITE_DONE
53  } fsm_state;
54
55  void fsm( );
56  void read( sc_uint<12> addr );
57  void write( sc_uint<12> addr );
58
59  SC_CTOR( spi_ahb ) {
60      fsm_state = SPI_AHB_IDLE;
61
62      spi = new spi_m( "SPI_AHB_MASTER" );
63      spi->clk( clk );
64      spi->miso( miso );
65      spi->mosi( mosi );
66      spi->start( start );
67      spi->sclk( sclk );
68      spi->ss( ss );
69      spi->rst( rst );
70      spi->busy( busy );
71      spi->data_out( data_out );
72      spi->data_in( data_in );
73
74      SC_METHOD( fsm );
75      sensitive << clk.pos( )
76                << n_hreset.neg( )
77                << busy.neg( )

```

```

78         << haddr;
79     }
80
81 };

```

## 2.4. Коммутатор шины

Листинг 3: Объявление коммутатора

```

1  /*
2   AMBA AHB bus controller.
3  */
4
5  #pragma once
6  #include "systemc.h"
7
8  #define AMBA_DEV_CNT dev_cnt
9
10 // How many devices are interconnected by AHB bus
11 const char dev_cnt = 3;
12
13 // How many bits are for device address and how many for inner address of device
14 const char dev_inner_addr_size = 12;
15 const char dev_dev_addr_size = 20;
16
17 // Device memory start address and mask for inner address
18 const uint32_t dev_addr_start = 0x40000000;
19 const uint32_t dev_inner_addr_mask = 0xFFFFFFFF >> dev_dev_addr_size;
20
21 // Device memory map
22 // For example
23 // device 0 memory will be 0x40000000 -- 0x40000FFF,
24 // device 1 0x40001000 -- 0x40001FFF and so on
25 struct dev_addr_map_t {
26     uint32_t index; // Device index
27     uint32_t base; // Device memory base address
28     uint32_t end; // Device memory end address
29     uint32_t prefix; // Device memory prefix (e.g. first 20 bits )
30 };
31
32 // Devices on the bus
33 static dev_addr_map_t *devs = new dev_addr_map_t[dev_cnt];
34
35 SC_MODULE( bus_ahb ) {
36     // AHB ports
37     sc_in<bool> hclk, n_hreset;
38     sc_inout<bool> hwrite;
39     sc_out<bool> hsel[ dev_cnt ];
40
41     sc_inout<sc_uint<32>> > haddr { "haddr" };
42     sc_out<sc_uint<32>> > hwdata { "hwdata" };
43     sc_out<sc_uint<32>> > hrdata_out { "hrdata_out" };
44     sc_in<sc_uint<32>> > hrdata_in[ dev_cnt ];
45
46     // Address buffer
47     sc_uint<dev_dev_addr_size+dev_inner_addr_size> buf_haddr;

```

```

48
49 // Address index buffer to select HRDATAx line
50 uint32_t buf_index;
51
52 // Transaction FSM states
53 enum {
54     AHB_IDLE,
55     AHB_READ_ADR,
56     AHB_READ_DATA,
57     AHB_WRITE_ADR,
58     AHB_WRITE_DATA
59 } bus_state;
60
61 SC_CTOR( bus_ahb ): hclk( "hclk" ), hwrite( "hwrite" ) {
62     init_dev( );
63
64     buf_index = 0;
65     buf_haddr = 0;
66
67     bus_state = AHB_IDLE;
68
69     SC_METHOD( hrdata_muxiplexer );
70     for( int i = 0; i < AMBA_DEV_CNT; i++ ) sensitive << hrdata_in[i];
71
72     SC_METHOD( fsm );
73     sensitive << hclk.pos( ) << n_hreset.neg( );
74
75     SC_METHOD( dev_select );
76     sensitive << haddr;
77 }
78
79 // Main transaction loop
80 void fsm( );
81
82 private:
83 void init_dev( ); // Register devices on the bus
84 void dev_select( ); // Select slave device (decoder)
85 void reset_hsel( ); // Reset device select lines
86 void amba_idle( ); // Start read/write transaction from idle state
87 void hrdata_muxiplexer( );
88 };

```

Для расширения шины необходимо переопределить `dev_cnt` и коммутатор шины зарегистрирует новые устройства и создаст необходимые линии `HRDATA` и `HSEL`.

## 2.5. Периферийное устройство

Листинг 4: Объявление устройства

```

1  /*
2     Emulation of PmodJSTK
3     Sends the same 5 bytes constantly:
4     X as 1110100101
5     Y as 1110100101
6     buttons as 00000111
7

```

```

8      X and Y are 10 bit values, buttons info is in a fifth byte
9      First input byte is interpreted as leds statuses
10     */
11     #pragma once
12
13     #include <systemc.h>
14     #include "spi.h"
15
16     SC_MODULE( pmodjstk ) {
17
18         spi_s* spi;
19
20         sc_in<bool> clk, sclk, mosi, rst, ss;
21         sc_out<bool> miso, busy;
22
23         sc_out<sc_uint<SPI_BIT_CAP> > data_in;
24         sc_inout<sc_uint<SPI_BIT_CAP> > data_out { "data_out" };
25
26         sc_uint<8> counter;
27
28         // last 8 bits of X
29         sc_uint<8> x_1;
30
31         // first 2 bits of X
32         sc_uint<8> x_2;
33
34         // last 8 bits of Y
35         sc_uint<8> y_1;
36
37         // first 2 bits of Y
38         sc_uint<8> y_2;
39
40         // buttons state (0b00000abc)
41         sc_uint<8> buttons;
42
43         void emul( );
44
45         SC_CTOR( pmodjstk ) {
46             spi = new spi_s( "PMODJSTK_SPI" );
47             spi->clk( clk );
48             spi->sclk( sclk );
49             spi->rst( rst );
50             spi->ss( ss );
51             spi->busy( busy );
52
53             spi->mosi( mosi );
54             spi->miso( miso );
55
56             spi->data_in( data_in );
57             spi->data_out( data_out );
58
59             x_1 = 0b10101010;
60             x_2 = 0b00000001;
61
62             y_1 = 0b00110011;
63             y_2 = 0b00000010;

```



```

64
65     buttons = 0b00000101;
66
67     counter = 0;
68     SC_METHOD( emul );
69     sensitive << sclk.pos( )
70               << ss.neg( )
71               << ss.pos( )
72               << rst.pos( );
73 }
74
75 ~pmodjstk( ) {
76     delete spi;
77 }
78
79 };

```

Для симуляции устройство отправляет заранее заданные значения X и Y.

## 2.6. Программное обеспечение

```

1  #include "cpu.h"
2  #include "memmap.h"
3
4  // Non-pipeline io
5  uint32_t cpu::write( uint32_t address, uint32_t body ) {
6
7      haddr.write( address );
8      hwrite.write( 1 );
9      wait( );
10     hwrite.write( 0 );
11     hwdata.write( body );
12     wait( );
13
14     #ifdef SW_OUTPUT
15         printf( "CPU write: 0x%08X at 0x%08X\n", (uint32_t) hwdata.read( ), address );
16     #endif
17     return hwdata.read( );
18 }
19
20 uint32_t cpu::read( uint32_t address ) {
21
22     haddr.write( address );
23     hwrite.write( 0 );
24     wait( );
25     haddr.write( 0 );
26     wait( );
27     #ifdef SW_OUTPUT
28         printf( "CPU read: 0x%08X at 0x%08X\n", (uint32_t) hrddata.read( ), address );
29     #endif
30     return hrddata.read( );
31 }
32
33 void cpu::sleep( uint32_t cycles ) {
34     for( uint32_t i = 0; i < cycles; i++ ) wait( );
35 }

```

```

36
37 uint32_t cpu::grab_jstk( ) {
38     write( SA_ST, 0x1 ); // set start
39     while( read( SA_RE ) != 1 );
40     return read( SA_DA );
41 }
42
43 void cpu::software( ) {
44
45     wait( );
46     puts( "-- din_dout testing" );
47
48     write( DD_OUT, 0x0101BEDA );
49     wait( );
50     read( DD_IN );
51     wait( );
52     read( DD_OUT );
53     wait( );
54
55     puts( "-- din_dout testing done" );
56
57     sleep( 20 );
58
59     puts( "-- periph contoller testing" );
60     write( SA_DA, 0x5 ); // set starting byte
61     write( SA_SS, 0x0 ); // set ss
62
63     for( int i = 0; i < 5; i++ ) grab_jstk( );
64
65     write( SA_SS, 0x1 ); // set ss
66     wait( );
67
68     puts( "-- periph contoller testing done" );
69     puts( "-- done" );
70
71     sleep( 10 );
72     sc_stop( );
73
74 }

```

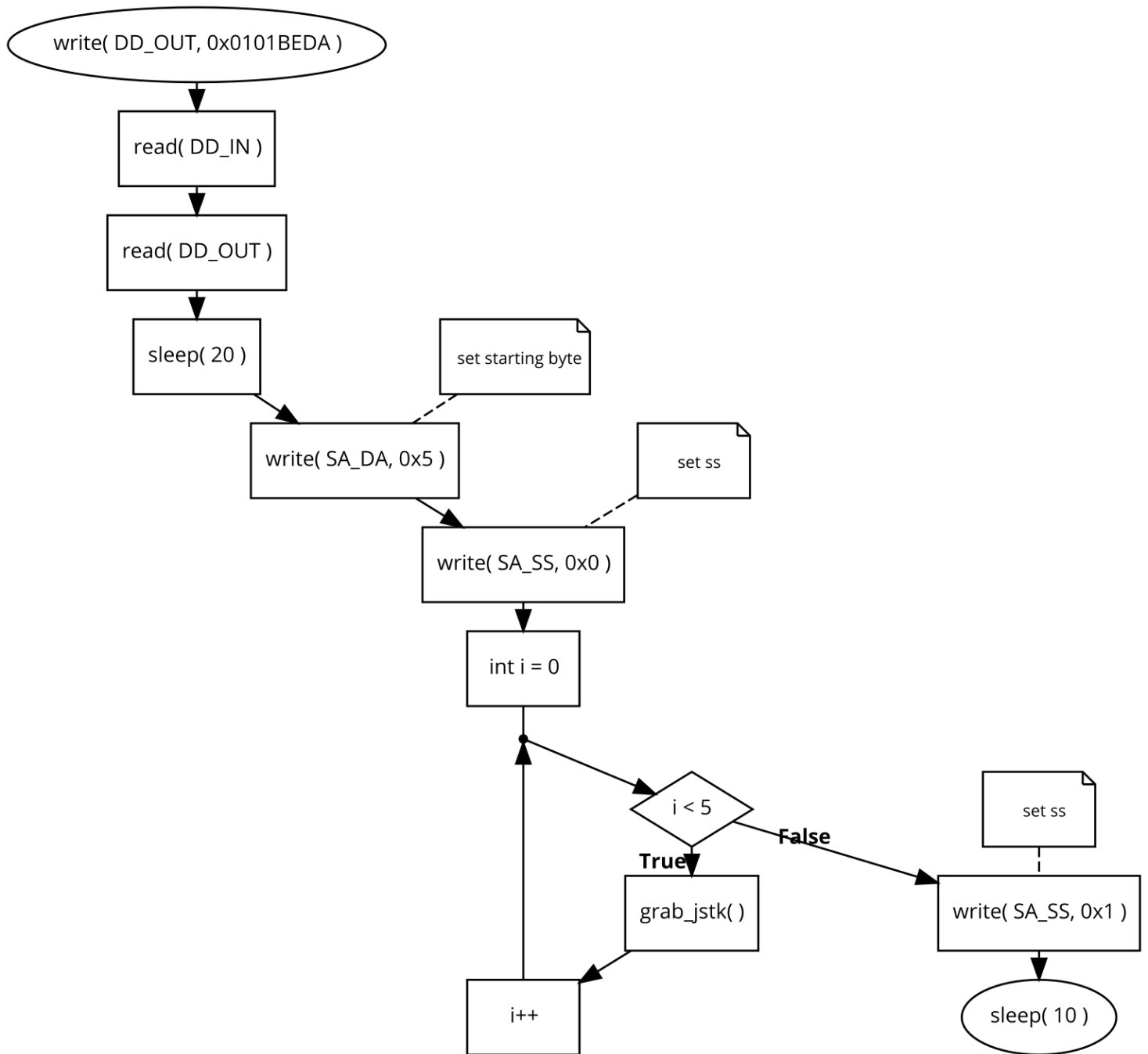


Рис. 1: Блок-схема программного обеспечения

### 3. Временные диаграммы

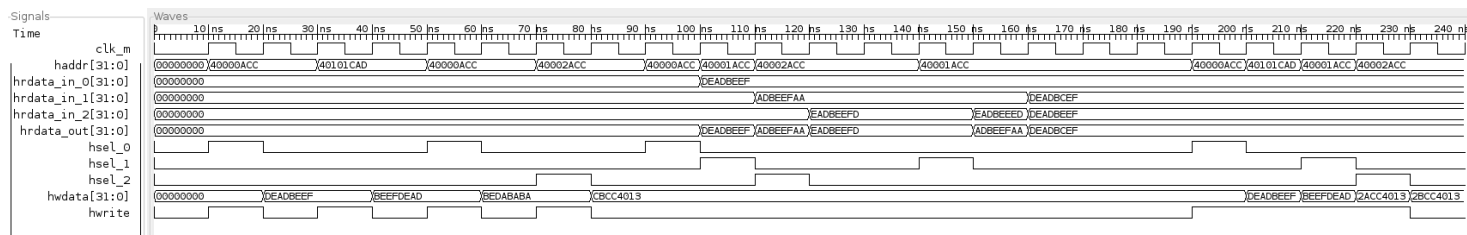


Рис. 2: Временная диаграмма коммутатора шины

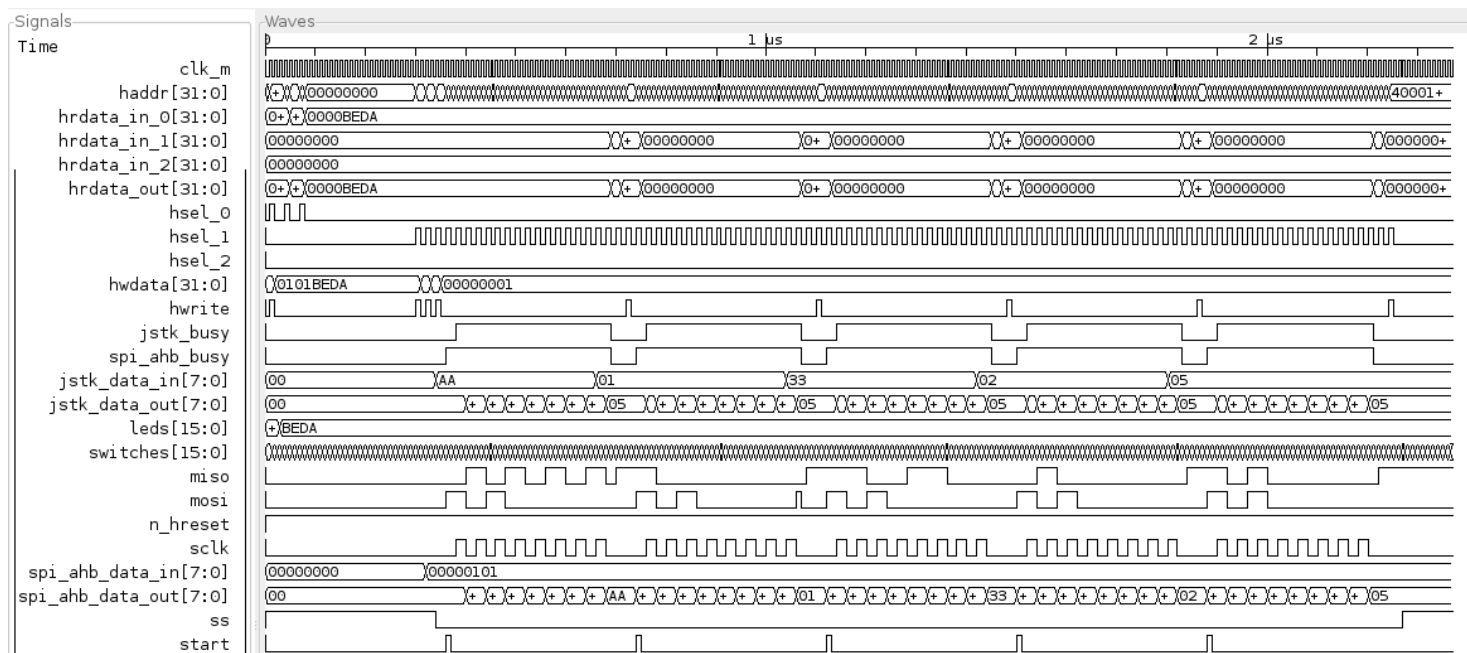


Рис. 3: Временная диаграмма системы в целом

### 4. Отладочный вывод

```
Registered device 0 at 40000000 -- 40000fff prefix 40000
Registered device 1 at 40001000 -- 40001fff prefix 40001
Registered device 2 at 40002000 -- 40002fff prefix 40002
```

```
Info: (I702) default timescale unit used for tracing: 1 ps (system.vcd)
```

```
-- din_dout testing
```

```
CPU write: 0x0101BEDA at 0x40000008
```

```
CPU read: 0x0000D1C6 at 0x40000004
```

```
CPU read: 0x0000BEDA at 0x40000008
```

```
-- din_dout testing done
```

```
-- periph controller testing
```

```
CPU write: 0x00000005 at 0x40001012
```

```
CPU write: 0x00000000 at 0x40001004
```

```
CPU write: 0x00000001 at 0x40001000
```



[illegible]

```
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000001 at 0x40001008
CPU read: 0x00000005 at 0x40001012
CPU write: 0x00000001 at 0x40001004
-- periph controller testing done
-- done
```

```
Info: /OSCI/SystemC: Simulation stopped by user.
```