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.
Калугин Ф. И.
Группа P3410

Преподаватель:
Быковский С. В.

Санкт-Петербург
2017 г.

Содержание

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

```
/*
  Digital controller for switches and leds control for Nexys4 DDR board.
*/

#pragma once
#include "systemc.h"

// Register map of bus slave
#define DIN_DOUT_IN_REG   0x00000004
#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> >  hwdata_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( )
```

```
              << haddr;
  }

};
```

2.4. Коммутатор шины

Листинг 3: Объявление коммутатора

```
/*
  AMBA AHB bus controller.
*/

#pragma once
#include "systemc.h"

#define AMBA_DEV_CNT dev_cnt

// How many devices are interconnected by AHB bus
const char dev_cnt = 3;

// How many bits are for device address and how many for inner address of device
const char dev_inner_addr_size  = 12;
const char dev_dev_addr_size    = 20;

// Device memory start address and mask for inner address
const uint32_t dev_addr_start       = 0x40000000;
const uint32_t dev_inner_addr_mask  = 0xFFFFFFFF >> dev_dev_addr_size;

// Device memory map
// For example
// device 0 memory will be 0x40000000 -- 0x40000FFF,
// device 1 0x40001000 -- 0x40001FFF and so on
struct dev_addr_map_t {
  uint32_t index;   // Device index
  uint32_t base;    // Device memory base address
  uint32_t end;     // Device memory end address
  uint32_t prefix;  // Device memory prefix (e.g. first 20 bits )
};

// Devices on the bus
static dev_addr_map_t *devs = new dev_addr_map_t[dev_cnt];

SC_MODULE( bus_ahb ) {
  // AHB ports
  sc_in<bool>     hclk, n_hreset;
  sc_inout<bool>  hwrite;
  sc_out<bool>    hsel[ dev_cnt ];

  sc_inout<sc_uint<32> >  haddr       { "haddr" };
  sc_out<sc_uint<32> >    hwdata      { "hwdata" };
  sc_out<sc_uint<32> >    hrdata_out  { "hrdata_out" };
  sc_in<sc_uint<32> >     hrdata_in[ dev_cnt ];

  // Address buffer
  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_multiplexer );
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_multiplexer( );
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
```

```
    X and Y are 10 bit values, buttons info is in a fifth byte
    First input byte is interpreted as leds statuses
*/
#pragma once

#include <systemc.h>
#include "spi.h"

SC_MODULE( pmodjstk ) {

  spi_s* spi;

  sc_in<bool> clk, sclk, mosi, rst, ss;
  sc_out<bool> miso, busy;

  sc_out<sc_uint<SPI_BIT_CAP> > data_in;
  sc_inout<sc_uint<SPI_BIT_CAP> > data_out { "data_out" };

  sc_uint<8> counter;

  // last 8 bits of X
  sc_uint<8> x_1;

  // first 2 bits of X
  sc_uint<8> x_2;

  // last 8 bits of Y
  sc_uint<8> y_1;

  // first 2 bits of Y
  sc_uint<8> y_2;

  // buttons state (0b00000abc)
  sc_uint<8> buttons;

  void emul( );

  SC_CTOR( pmodjstk ) {
    spi = new spi_s( "PMODJSTK_SPI" );
    spi->clk( clk );
    spi->sclk( sclk );
    spi->rst( rst );
    spi->ss( ss );
    spi->busy( busy );

    spi->mosi( mosi );
    spi->miso( miso );

    spi->data_in( data_in );
    spi->data_out( data_out );

    x_1 = 0b10101010;
    x_2 = 0b00000001;

    y_1 = 0b00110011;
    y_2 = 0b00000010;
```

```
    buttons = 0b00000101;

    counter = 0;
    SC_METHOD( emul );
    sensitive << sclk.pos( )
              << ss.neg( )
              << ss.pos( )
              << rst.pos( );
  }

  ~pmodjstk( ) {
    delete spi;
  }

};
```

Для симуляции устройство отсылает заранее заданные значения X и Y.

2.6. Программное обеспечение

```
#include "cpu.h"
#include "memmap.h"

// Non-pipeline io
uint32_t cpu::write( uint32_t address, uint32_t body ) {

  haddr.write( address );
  hwrite.write( 1 );
  wait( );
  hwrite.write( 0 );
  hwdata.write( body );
  wait( );

#ifdef SW_OUTPUT
  printf( "CPU write: 0x%08X at 0x%08X\n", (uint32_t) hwdata.read( ), address );
#endif
  return hwdata.read( );
}

uint32_t cpu::read( uint32_t address ) {

  haddr.write( address );
  hwrite.write( 0 );
  wait( );
  haddr.write( 0 );
  wait( );
#ifdef SW_OUTPUT
  printf( "CPU read: 0x%08X at 0x%08X\n", (uint32_t) hrdata.read( ), address );
#endif
  return hrdata.read( );
}

void cpu::sleep( uint32_t cycles ) {
  for( uint32_t i = 0; i < cycles; i++ ) wait( );
}
```

```
36
37  uint32_t cpu::grab_jstk_byte( ) {
38    write( SA_ST, 0x1 ); // set start
39    while( read( SA_RE ) != 1 );
40    return read( SA_DA );
41  }
42
43  void cpu::set_leds( sc_uint<16> data ) {
44    write( DD_OUT, data );
45  }
46
47  sc_uint<16> cpu::get_switches( ) {
48    return (sc_uint<16>) read( DD_IN );
49  }
50
51  sc_uint<16> cpu::get_leds( ) {
52    return (sc_uint<16>) read( DD_OUT );
53  }
54
55  void cpu::start_jstk_tr( sc_uint<16> data ) {
56    write( SA_DA, data ); // set starting byte
57    write( SA_SS, 0x0 ); // set ss
58  }
59
60  void cpu::end_jstk_tr( ) {
61    write( SA_SS, 0x1 ); // set ss
62  }
63
64  void cpu::software( ) {
65
66    while( 1 ) {
67      wait( );
68      puts( "-- din_dout testing" );
69
70      set_leds( 0xBABA );
71      wait( );
72      get_switches( );
73      wait( );
74      get_leds( );
75      wait( );
76
77      puts( "-- din_dout testing done" );
78
79      sleep( 20 );
80
81      puts( "-- periph contoller testing" );
82
83      start_jstk_tr( 0x5 );
84      for( int i = 0; i < 5; i++ ) grab_jstk_byte( );
85      end_jstk_tr( );
86
87      puts( "-- periph contoller testing done" );
88      puts( "-- done" );
89
90      sleep( 10 );
91    }
```

```
92    sc_stop( );
93
94  }
```

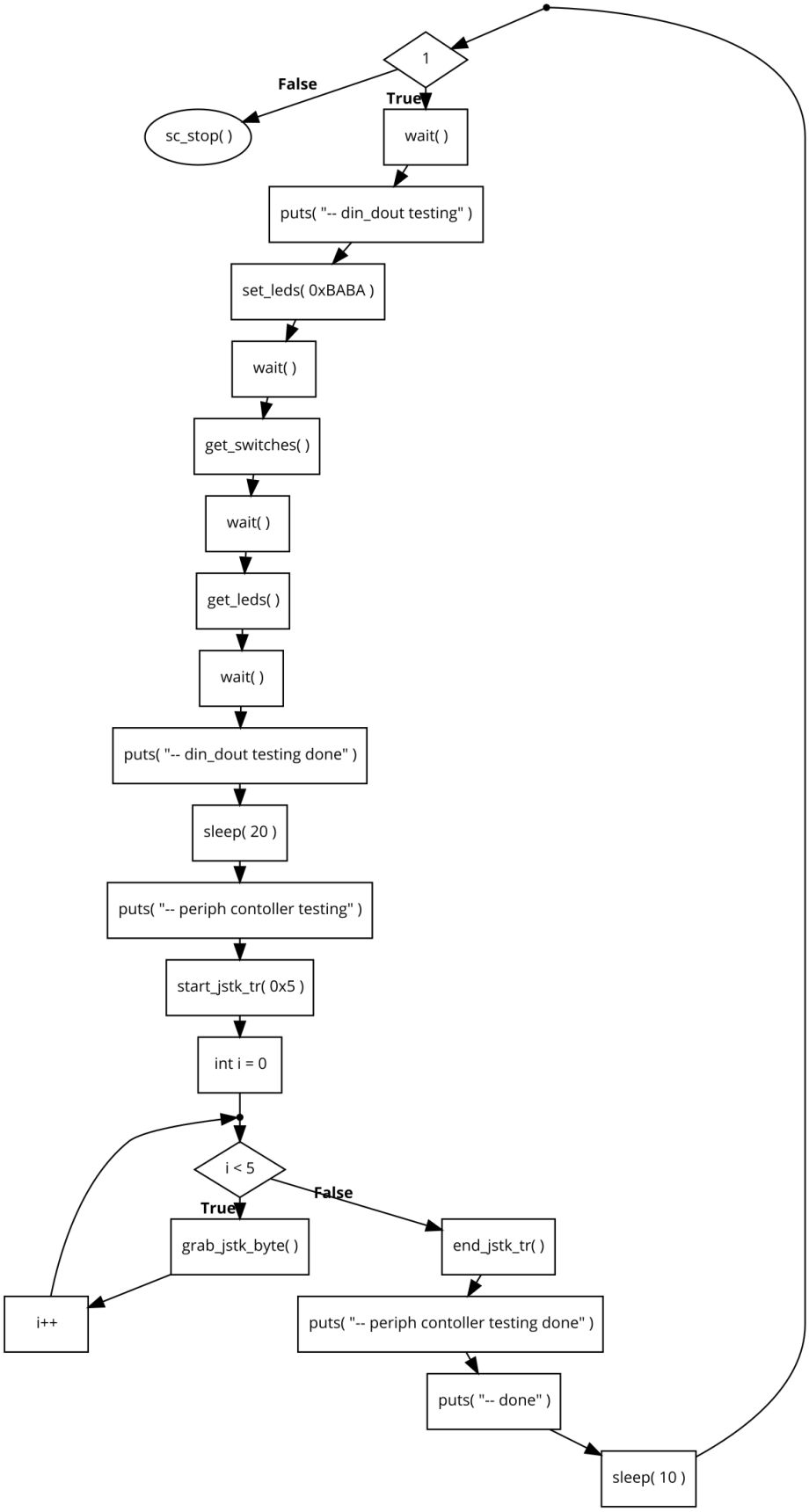

Рис. 1: Блок-схема программного обеспечения

3. Временные диаграммы

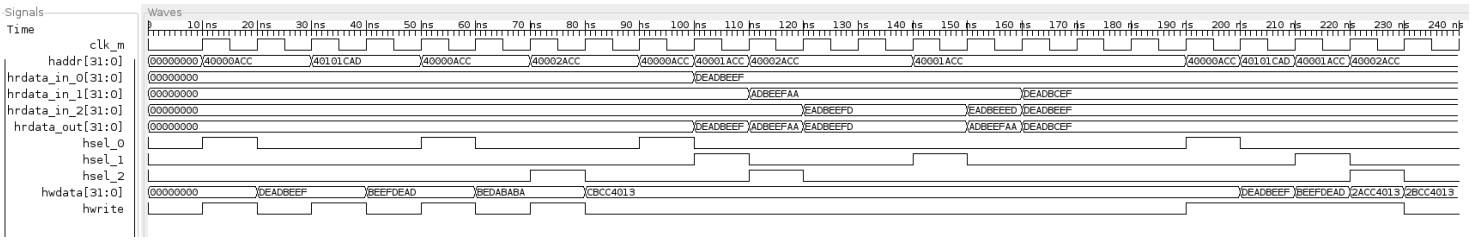

Рис. 2: Временная диаграмма коммутатора шины

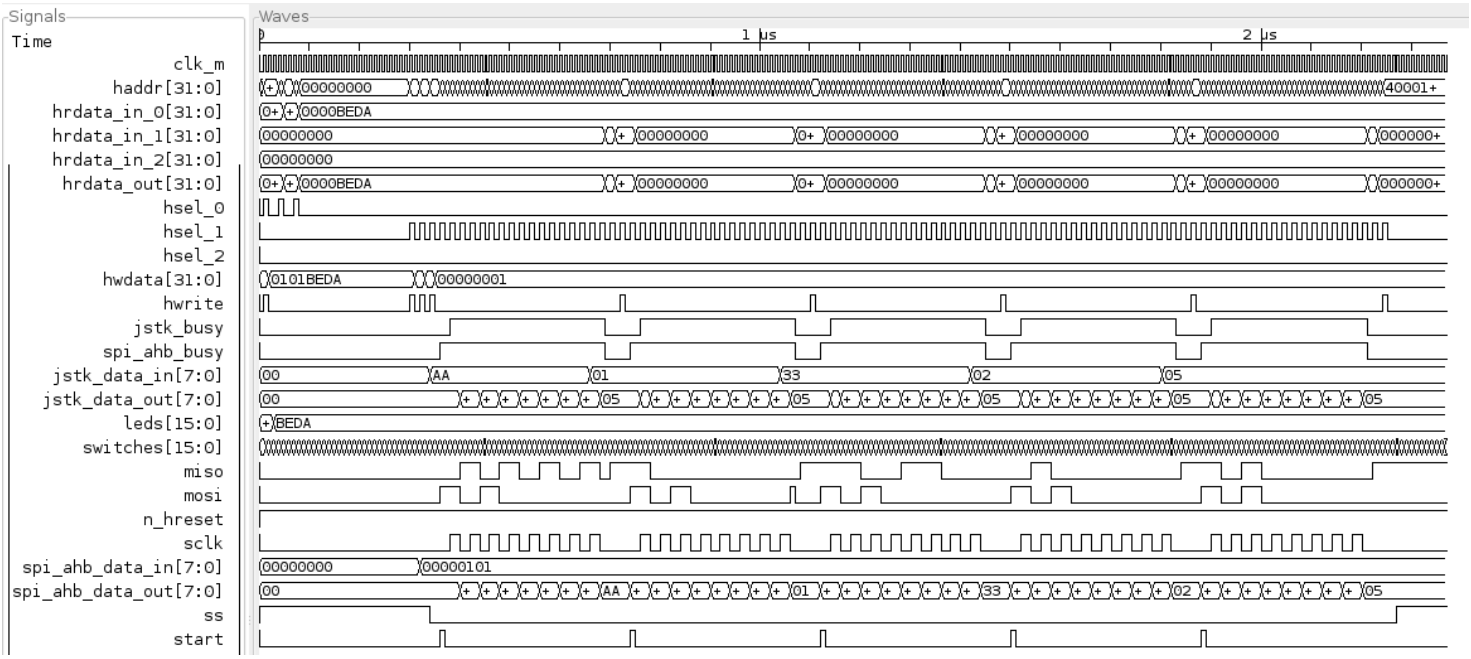

Рис. 3: Временная диаграмма системы в целом

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 contoller testing
CPU write: 0x00000005 at 0x40001012
CPU write: 0x00000000 at 0x40001004
CPU write: 0x00000001 at 0x40001000
```

```
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x0000BEDA at 0x40001008
CPU read: 0x00000001 at 0x40001008
CPU read: 0x000000AA at 0x40001012
CPU write: 0x00000001 at 0x40001000
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
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: 0x00000001 at 0x40001012
CPU write: 0x00000001 at 0x40001000
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
```

```
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
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: 0x00000033 at 0x40001012
CPU write: 0x00000001 at 0x40001000
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
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: 0x00000002 at 0x40001012
CPU write: 0x00000001 at 0x40001000
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
CPU read: 0x00000000 at 0x40001008
```

```
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 contoller testing done
-- done

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