01.1. Пакет AMBA_AHB_p.

У наступному пакеті для шини AMBA AHB описані константи, відповідні допустимих значень керуючих сигналів, константи, що використовуються для визначення розрядності шини адреси і шин даних, а також група типів, які використовуються для опису вхідних сигналів для модуля комунікаційної системи.

** У розглянутих далі прикладах передбачається, що всі пристрої мають 32-розрядні виходи на шини даних, і значення сигналу HSIZE завжди "010", що відповідає обміну 4-байтовими словами даних. Тому в моделях провідних і ведених пристроїв значення цього сигналу не аналізується. Чи не розглядаються також можливості використання сигналу HPROT.*

01.2. Пакет AMBA_AHB_p.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
package AMBA_AHB_p is
--HTRANS
constant HTRANS_IDLE: std_logic_vector (1 downto 0):="00";
constant HTRANS_BUSY: std_logic_vector (1 downto 0):="01";
constant HTRANS_NONSEQ: std_logic_vector (1 downto 0):="10";
constant HTRANS_SEQ: std_logic_vector (1 downto 0):="11";
```

01.3. Пакет AMBA_AHB_p.

```
--HBURST
constant HBURST_SINGLE: std_logic_vector (2 downto 0):="000";
constant HBURST_INCR: std_logic_vector (2 downto 0):="001";
constant HBURST_WRAP4:std_logic_vector (2 downto 0):="010";
constant HBURST_INCR4:std_logic_vector (2 downto 0):="011";
constant HBURST_WRAP8:std_logic_vector (2 downto 0):="100";
constant HBURST_INCR8:std_logic_vector (2 downto 0):="101";
constant HBURST_WRAP16:std_logic_vector (2 downto 0):="110";
constant HBURST_INCR16:std_logic_vector (2 downto 0):="111";
```

01.4. Пакет AMBA_AHB_p.

```
--HRESP
constant HRESP_OKAY: std_logic_vector (1 downto 0):="00";
constant HRESP_ERROR: std_logic_vector (1 downto 0):="01";
constant HRESP_RETRY: std_logic_vector (1 downto 0):="10";
constant HRESP_SPLIT: std_logic_vector (1 downto 0):="11";
--
constant H_ADDR: natural:=32;
constant H_DATA: natural:=32;
```

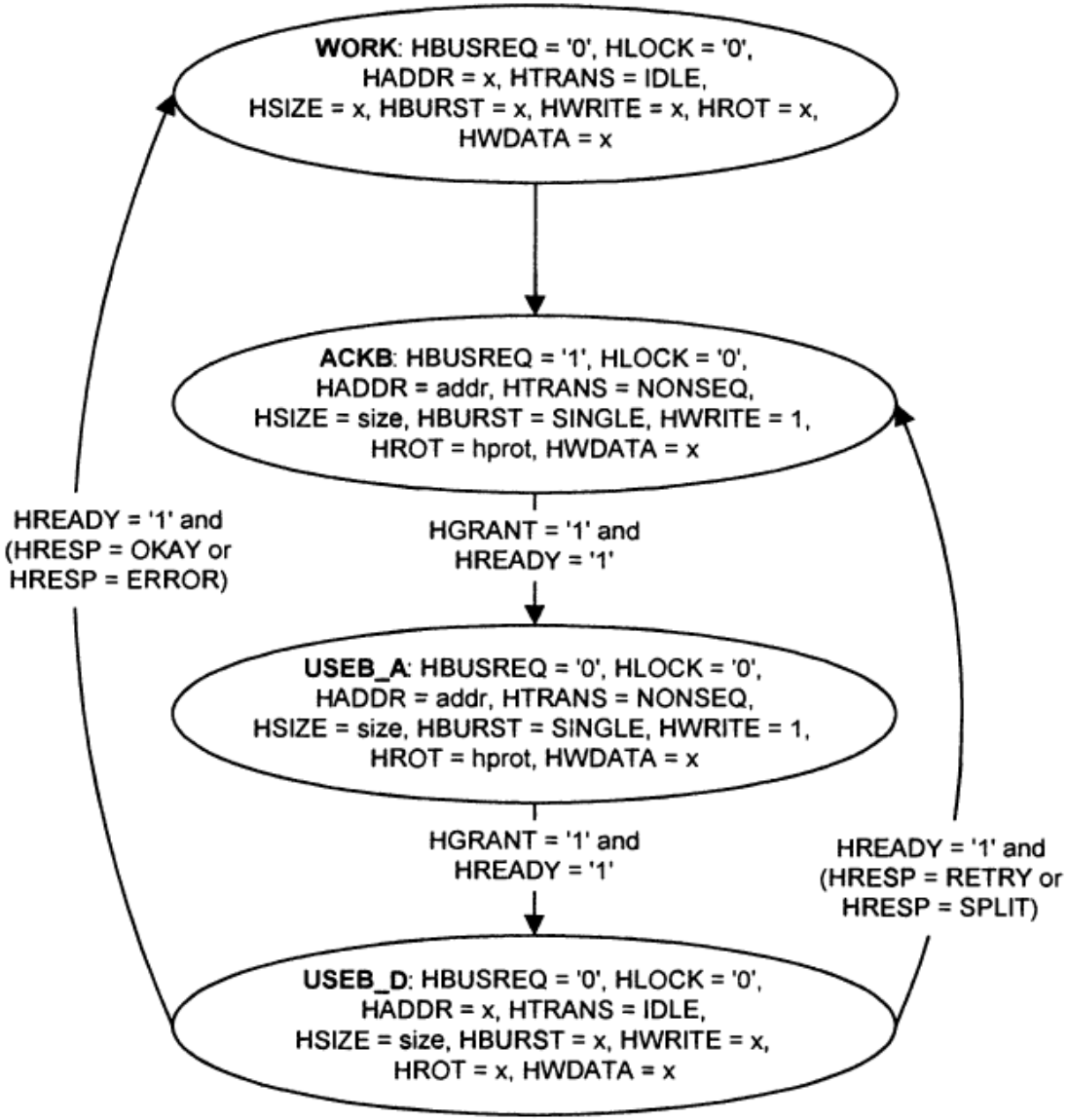
01.5. Пакет AMBA_AHB_p.

```
type H1 is array(natural range <>)of std_logic;
type H2 is array(natural range <>)of std_logic_vector(1 downto 0);
type H3 is array(natural range <>)of std_logic_vector(2 downto 0);
type H4 is array(natural range <>)of std_logic_vector(3 downto 0);
type HA is array(natural range <>)of std_logic_vector(H_ADDR-1 downto 0);
type HD is array(natural range <>)of std_logic_vector(H_DATA-1 downto 0);
--
end package AMBA_AHB_p;
```

02.1. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

Розглядається модель найпростішого інтерфейсу MASTER-пристрою. Цей інтерфейс дозволяє MASTER-пристрою записувати в SLAVE-пристрій окремі слова даних. Граф кінцевого автомату для функціонування цього MASTER-пристрою наведено далі.

02.2. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.



02.3. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

Цей автомат містить чотири стани. У стані WORK MASTER-пристрій виконує дії, які не пов'язані з використанням шини. У цьому стані HBUSREQ='0'. Вихід HTRANS=IDLE, що вказує SLAVE-пристрою не виконувати будь-яких дій, якщо даний MASTER-пристрій в системі виявиться MASTER-пристроєм за замовчуванням. При цьому на інших виходах MASTER-пристрою на шині AMBA AHB можуть бути будь-які значення. При необхідності використовувати шину MASTER-пристрій переходить зі стану WORK в стан ACKB. У цьому стані HBUSREQ='1'. У наведеному прикладі передбачається, що даний пристрій не виконує запитів з блокуванням, тому HLOCK='0'. У цьому стані MASTER-пристрій може встановити на лініях адреси і управління значення, які відповідають фазі адреси першого обміну. Після того як MASTER-пристрій отримує сигнал надання шини(сигнал HGRANT), він повинний дочекатися того моменту, коли MASTER-пристрій, який брав участь в попередньому обміні, звільнить лінії адреси і управління. Цей момент визначається по HREADY='1'. Після цього MASTER-пристрій переходить в стан адреси - USEB_A. У цьому стані він має утримувати актуальні значення на лінії адреси і управління. При приході чергового сигналу HREADY='1', даний пристрій зі стану USEB_A переходить в стан запису даних - USEB_D.

02.4. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

Така модель поведінки MASTER-пристрою є спрощеною. У ній не враховується, що якщо попередній запит завершується підтвердженням, відмінним від OKAY, то арбітр може змінити MASTER-пристрій, якому буде надана шина для чергового обміну. Для врахування цього необхідно було б додати перехід зі стану USEB_A в стан ACKB, коли HGRANT='0'.

02.5. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

У стані USEB_D пристрій знаходиться до тих пір, поки не отримає від SLAVE-пристрою сигнал завершення обміну HREADY='1'. Залежно від значення сигналу HRESP, MASTER-пристрій переходить або в стан WORK, або в стан ACKB. При переході в стан WORK, коли HRESP=OKAY або HRESP=ERROR, помилка не обробляється спеціальним чином. При переході в стан ACKB, коли HRESP=RETRY або HRESP=SPLIT, MASTER-пристрій в обох випадках має знову запросити шину і, після того як вона буде надана, повторити запит ще раз.

02.6. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

Лістинг моделі інтерфейсу MASTER-пристрою наведено далі. Назви вхідних і вихідних портів в тексті програми утворені в такий спосіб: назва починається з m1, далі йде назва сигналу відповідно до стандарту, але з опущеною першою літерою H (наприклад, порт сигналу HADDR називається mladdr, HTRANS - mltrans).

02.7. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use AMBA_AHB_p.AMBA_AHB_p.all;
entity master1 is
port(m1reset:in std_logic;
     m1clk: in std_logic;
     m1busreq: out std_logic;
     m1lock: out std_logic;
     m1grant: in std_logic;
     m1addr: out std_logic_vector((H_ADDR-1) downto 0);
     m1trans: out std_logic_vector(1 downto 0);
     m1size: out std_logic_vector(2 downto 0);
     m1burst: out std_logic_vector(2 downto 0);
     m1write: out std_logic;
     m1prot: out std_logic_vector(3 downto 0);
     m1wdata: out std_logic_vector((H_DATA-1) downto 0);
     m1rdata: in std_logic_vector((H_DATA-1) downto 0);
     m1ready: in std_logic;
     m1resp: in std_logic_vector(1 downto 0)
    );
end entity master1;
```

02.8. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

```
architecture rtl of master1 is
type m1_state_type is (WORK, ACKB, USEB_A, USEB_D);
signal m1c_state,m1c_nextstate: m1_state_type;
signal cou: natural range 0 to 2;
begin
p_lockstate: process (m1reset,m1clk)
begin
if m1reset='0' then m1c_state<=WORK; cou<=2;
else
  if rising_edge(m1clk) then
        if cou>0 then cou<=cou-1;
        else if m1c_nextstate=WORK then cou<=2; end if;
        end if;
        if m1ready='1' or m1c_state=WORK then m1c_state<=m1c_nextstate;
        end if;
  end if;
end if;
end process p_lockstate;
```

02.9. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

```
p_outs: process (m1c_state,cou,m1grant,m1ready)
begin
case m1c_state is
when WORK => m1busreq<='0'; m1lock<='0';m1addr<=(others =>'0');
             m1trans<=HTRANS_IDLE; m1size<="010"; m1burst<=HBURST_SINGLE;
             m1write<='0'; m1prot<="0001"; m1wdata<=(others=>'0');
             if cou>0 then m1c_nextstate<=WORK;
                else m1c_nextstate<=ACKB; end if;
when ACKB =>m1busreq<='1'; m1lock<='0';
             m1addr <="00000000000000000000000000000001";
             m1trans<=HTRANS_NONSEQ; m1size<="010";
             m1burst<=HBURST_SINGLE; m1write<='1'; m1prot<="0001";
             m1wdata<=(others=>'0');
             if m1grant='1' and m1ready='1' then m1c_nextstate<=USEB_A;
               else m1c_nextstate<=ACKB; end if;
```

02.10. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

```
when USEB_A => m1busreq<='0'; m1lock<='0';
               m1addr <="00000000000000000000000000000001";
               m1trans<=HTRANS_NONSEQ; m1size<="010";
               m1burst<=HBURST_SINGLE; m1write<='1'; m1prot<="0001";
               m1wdata<=(others=>'0');
               if m1grant='1' and m1ready='1' then m1c_nextstate<=USEB_D;
                 else m1c_nextstate<=USEB_A; end if;
when USEB_D => m1busreq<='0'; m1lock<='0'; m1addr <=(others=>'0');
               m1trans<=HTRANS_IDLE; m1size<="010"; m1burst<=HBURST_SINGLE;
               m1write<='0'; m1prot<="0001";
               m1wdata<="01010101010101010101010101010101";
               if m1ready='0' then m1c_nextstate<=USEB_D;
                 else
               if (m1resp=HRESP_OKAY) or (m1resp=HRESP_ERROR) then
                   m1c_nextstate<=WORK;
                 else m1c_nextstate<=ACKB;
               end if;
             end if;
```

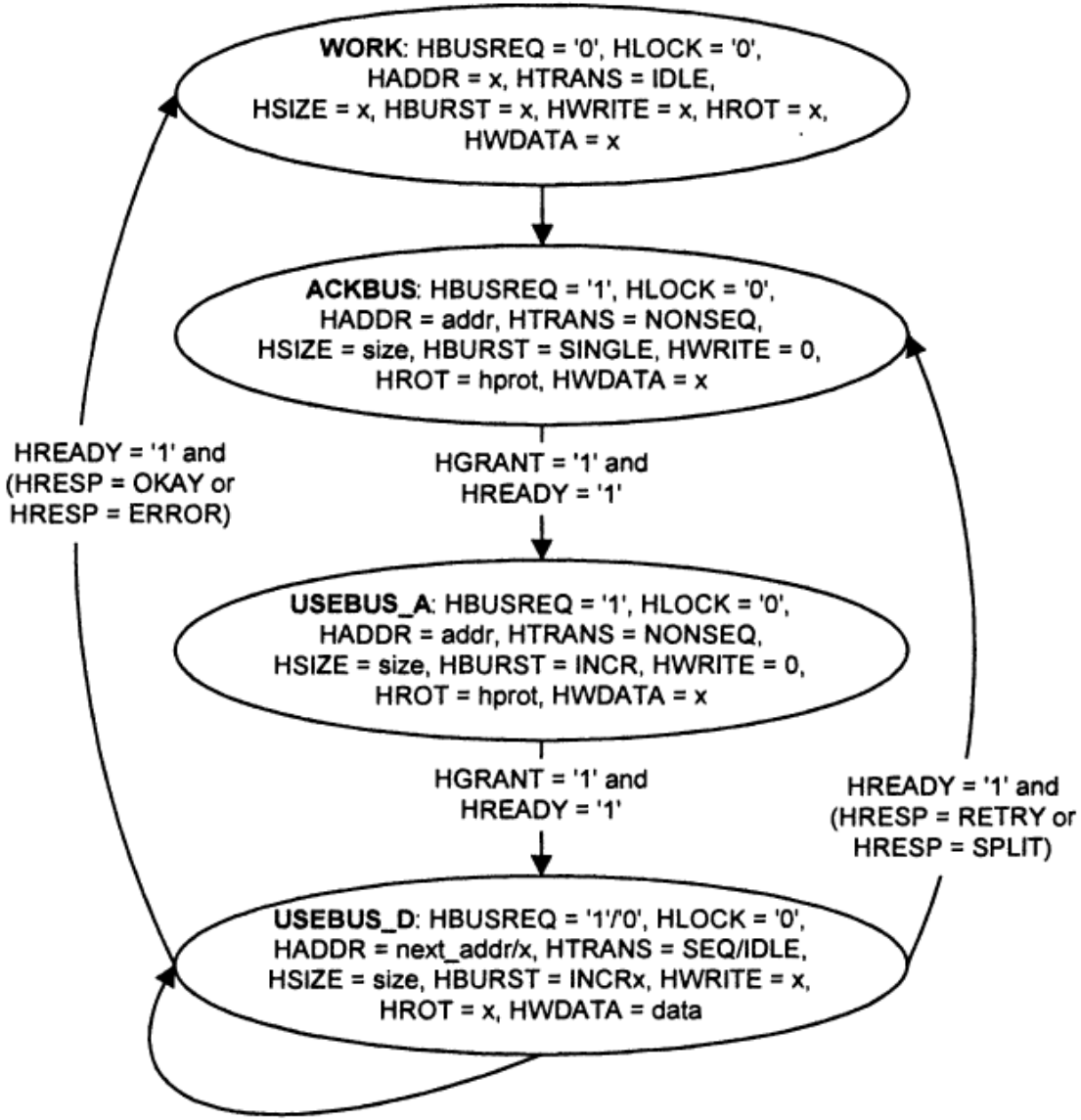
02.11. Інтерфейс MASTER-пристрою на шину AMBA AHB для запису одного слова.

```
when others =>m1busreq<='0'; m1lock<='0';m1addr<=(others =>'0');
              m1trans<=HTRANS_IDLE;
              m1size<="010"; m1burst<=HBURST_SINGLE; m1write<='0';
              m1prot<="0001"; m1wdata<=(others=>'0');
              m1c_nextstate<=WORK;
end case;
end process p_outs;
end architecture rtl;
```

03.1. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

Розглядається модель інтерфейсу MASTER-пристрою, яка дозволяє йому читати з SLAVE-пристрою послідовності слів. Граф станів кінцевого автомата, що відповідає цьому інтерфейсу, наведено далі.

03.2. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.



03.3. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

Граф подібний до попереднього, однак в стані USEBUS_D можливий додатковий перехід в самого себе, якщо не всі слова даних ще прочитані.
Лістинг поведінкового опису моделі наведено далі. Ця модель має такий ж опис портів, як і попередня модель, але назви всіх сигналів починаються не з m1, а з m2.

03.4. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

```
architecture rtl of master2 is
type m2_state_type is (WORK, ACKBUS, USEBUS_A, USEBUS_D);
signal m2c_state,m2c_nextstate: m2_state_type;
signal cou1:natural range 0 to 2;
signal cou2:natural range 0 to 3;
signal readed_data: std_logic_vector((H_DATA-1) downto 0);
signal n_addr: natural;
begin
```

03.5. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

```
p_lockstate: process(m2reset,m2clk,m2ready)
begin
if m2reset='0' then  m2c_state<=WORK; cou1<=1; cou2<=0; n_addr<=0;
else
  if rising_edge(m2clk) then
        if m2c_state=USEBUS_D and m2ready='1' and m2resp=HRESP_OKAY then
        readed_data<=m2rdata;  end if;
        if cou1>0 then cou1<=cou1-1;
          else if m2c_nextstate=WORK then cou1<=1; end if; end if;
        if cou2>0  then
           if m2c_state=USEBUS_D and m2ready='1' and m2resp=HRESP_OKAY
              then cou2<=cou2-1; n_addr<=n_addr+4;
              else if m2c_nextstate=WORK then cou2<=2; n_addr<=0; end if;
           end if;
        else
           if m2c_nextstate=WORK then cou2<=2; n_addr<=0; end if;
        end if;
        if m2ready='1' or m2c_state=WORK then
           m2c_state<=m2c_nextstate; end if;
  end if;
end if;
end process p_lockstate;
```

03.6. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

```
p_outs: process  (m2c_state,cou1,cou2,m2grant,m2ready)
begin
case m2c_state is
when WORK => m2busreq<='0'; m2lock<='0';m2addr<=(others =>'0');
             m2trans<=HTRANS_IDLE; m2size<="010"; m2burst<=HBURST_SINGLE;
             m2write<='0'; m2prot<="0001"; m2wdata<=(others=>'0');
             if cou1>0 then m2c_nextstate<=WORK;
               else m2c_nextstate<=ackbus; end if;
```

03.6. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

```
when ACKBUS =>m2busreq<='1'; m2lock<='0';
              m2addr<= conv_std_logic_vector(conv_unsigned(n_addr,32),32);
              m2trans<=HTRANS_NONSEQ; m2size<="010"; m2burst<=HBURST_INCR;
              m2write<='0'; m2prot<="0001"; m2wdata<=(others=>'0');
              if m2grant='1' and m2ready='1' then m2c_nextstate<=USEBUS_A;
                else m2c_nextstate<=ACKBUS; end if;
when USEBUS_A => m2busreq<='1'; m2lock<='0';
              m2addr<= conv_std_logic_vector(conv_unsigned(n_addr,32),32);
              m2trans<=HTRANS_NONSEQ; m2size<="010"; m2burst<=HBURST_INCR;
              m2write<='0'; m2prot<="0001"; m2wdata<=(others=>'0');
              if m2grant='1' and m2ready='1' then m2c_nextstate<=USEBUS_D;
                else m2c_nextstate<=USEBUS_A; end if;
```

03.6. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

```
when USEBUS_D =>
                if cou2>1 then m2busreq<='1'; else m2busreq<='0'; end if;
                m2lock<='0';
                m2addr<= conv_std_logic_vector(conv_unsigned(n_addr,32),32);
                if cou2>0 then m2trans<=HTRANS_SEQ;
                  else m2trans<=HTRANS_IDLE; end if;
                m2size<="010"; m2burst<=HBURST_INCR; m2write<='0';
                m2prot<="0001"; m2wdata<=(others=>'0');
                if m2ready='0' then m2c_nextstate<=USEBUS_D;
                  else
                  if m2resp=HRESP_OKAY then
                       if cou2=0 then m2c_nextstate<=WORK;
                           else m2c_nextstate<=USEBUS_D; end if;
                  else if m2resp=HRESP_ERROR then m2c_nextstate<=WORK;
                          else m2c_nextstate<=ACKBUS;
                       end if;
                  end if;
                end if;
```

03.6. Інтерфейс MASTER-пристрою на шину AMBA AHB для читання послідовності слів.

```
when others =>m2busreq<='0'; m2lock<='0';m2addr<=(others=>'0');
             m2trans<=HTRANS_IDLE; m2size<="010"; m2burst<=HBURST_SINGLE;
             m2write<='0'; m2prot<="0001"; m2wdata<=(others=>'0');
             m2c_nextstate<=WORK;
end case;
end process p_outs;
end architecture rtl;
```

04.1. Інтерфейс SLAVE-пристрою на шину AMBA AHB для блоку пам’яті.

Розглядається інтерфейс SLAVE-пристрою на прикладі найпростішого блоку пам'яті.

04.2. Інтерфейс SLAVE-пристрою на шину AMBA AHB для блоку пам'яті.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use AMBA_AHB_p.AMBA_AHB_p.all;
entity c_mem1 is
port(sreset: in std_logic;
     sclk: in std_logic;
     ssel: in std_logic;
     saddr: in std_logic_vector((H_ADDR-1) downto 0);
     swrite: in std_logic;
     strans: in std_logic_vector(1 downto 0);
     ssize: in std_logic_vector(2 downto 0);
     sburst: in std_logic_vector(2 downto 0);
     swdata: in std_logic_vector((H_DATA-1) downto 0);
     smaster: in std_logic_vector(3 downto 0);
     smastlock: in std_logic;
     sready: out std_logic;
     smready: in std_logic;
     sresp: out std_logic_vector(1 downto 0);
     srdata: out std_logic_vector((H_DATA-1) downto 0);
     ssplit: out std_logic_vector(15 downto 0)
    );
end entity c_mem1;
```

04.3. Інтерфейс SLAVE-пристрою на шину AMBA AHB для блоку пам'яті.

```
architecture rtl of c_mem1 is
type mem_cont is array(1 to 32) of std_logic_vector((H_DATA-1) downto 0);
signal memory_contents: mem_cont;
signal addr: std_logic_vector((H_ADDR-1) downto 0);
signal write: std_logic;
signal trans: std_logic_vector(1 downto 0);
signal size: std_logic_vector(2 downto 0);
signal burst: std_logic_vector(2 downto 0);
signal sel: std_logic;
begin
```

04.4. Інтерфейс SLAVE-пристрою на шину AMBA AHB для блоку пам'яті.

```
p_ac_lock: process (sclk)
begin
if rising_edge(sclk) then
  if ssel='1' then
  addr<=saddr ;
  write<=swrite;
  trans<=strans;
  size<=ssize ;
  burst<=sburst;
  end if;
end if;
end process p_ac_lock;
p_sel: process  (sclk)
begin
if rising_edge(sclk) and smready='1' then sel<=ssel; end if;
end process p_sel;
p_resp: process  (sclk)
begin
if rising_edge(sclk) then sready<='1'; sresp<=HRESP_OKAY; end if;
end process p_resp;
```

04.5. Інтерфейс SLAVE-пристрою на шину AMBA AHB для блоку пам'яті.

```
p_write: process  (sclk,sel,addr)
begin
if sel='1' then
   if rising_edge(sclk) then
    if write='1' and trans/=HTRANS_BUSY and trans/=HTRANS_IDLE then
      memory_contents(conv_integer(addr,32))<=swdata;
     end if;
   end if;
end if;
end process p_write;

p_read: process (sel,write)
begin
if sel='1' then
  if write='0' then srdata<=memory_contents(conv_integer(addr)); end if;
end if;
end process p_read;

end architecture rtl;
```