す（**リスト2.35**の③）．

15 FIFO（同期バス）

- **作成者名：田原迫仁治**
- **サンプル記述：リスト2.37（fifo_sync.vhd），2.38（fifo_sync.v）**
- **モデルの種類：RTL モデル**
- **検証に使用したシミュレータ：Actel Desktop VeriBest（VHDL），ModelSim（Verilog HDL）**
- **端子表**

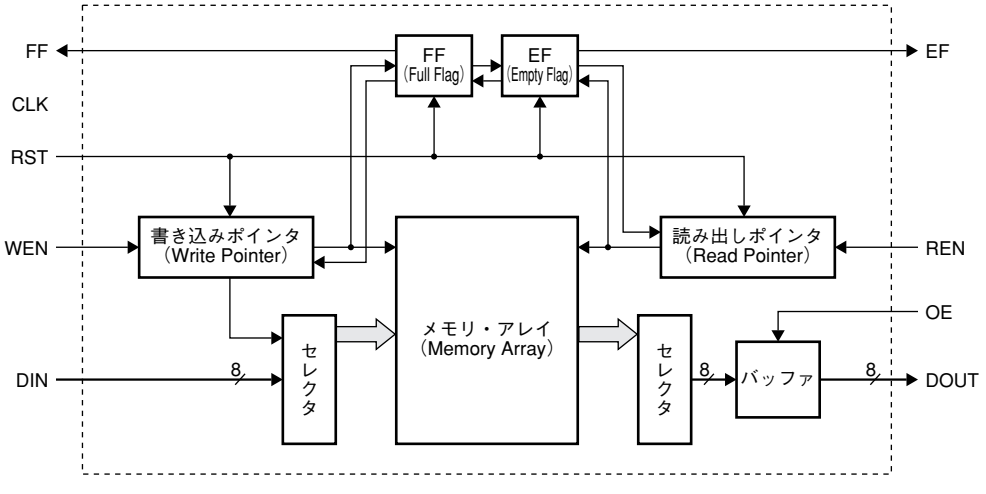
  入力：RST，CLK，DIN，WEN，REN，OE

  出力：DOUT，EF，FF

FIFO（first-in first-out）は，LSI 設計において使用頻度が非常に高い回路の一つです．LSI設計者であれば，FIFOのHDL記述テンプレートを持っていても損はないでしょう．FIFOは，書き込んだデータをその順番に読み出します．アドレス入力がない代わりに，内部に書き込み（ライト）ポインタと読み出し（リード）ポインタを持っています．ここでは，同期バスを持ったFIFOの内部動作の記述を同期設計で作成してみます．ただし，書き込む側と読み出す側のクロック周波数が異なり，FIFOを使ってクロックの乗り換えを行うようなケースでは，非同期バスを使って単一クロックで動作させたほうが設計（論理合成）が容易です．

〔図2.29〕FIFOのブロック図

FIFOでは，書き込んだデータをその順番に読み出す．アドレス入力がない代わりに，内部に書き込み（ライト）ポインタと読み出し（リード）ポインタを持っている．



〔リスト2.37〕FIFOのVHDL記述(fifo_sync.vhd)

```
library IEEE;
use     IEEE.std_logic_1164.all ;
use     IEEE.std_logic_unsigned.all ;

entity FIFO_SYNC is
  port (
        RST :     in        std_logic;
        CLK :     in        std_logic;
        DIN :     in        std_logic_vector(7
                                        downto 0);
        DOUT :    out       std_logic_vector(7
                                        downto 0);
        WEN :     in        std_logic;
        REN :     in        std_logic;
        OE :      in        std_logic;
        EF :      out       std_logic;
        FF :      out       std_logic
        );
end FIFO_SYNC;

architecture RTL of FIFO_SYNC is

subtype SFIFOWORD is std_logic_vector(7 downto 0);
type SFIFOARRAY is array (0 to 15) of SFIFOWORD;

signal SF_RAM :             SFIFOARRAY;
signal WPTR :               integer range 0 to 15;
signal RPTR :               integer range 0 to 15;
signal EF_TMP :             std_logic;
signal FF_REG, EF_REG :     std_logic;

begin
  -- 出力アサインメント
  DOUT <= SF_RAM(RPTR) when OE='1' else (others
                                        => 'Z');
  FF <= FF_REG;
  EF <= EF_REG;
  -- 出力アサインメント記述の終了
  -- FIFO
  process(CLK) begin
    if(CLK'event and CLK = '1') then
          if(WEN ='1' and FF_REG='0') then
            SF_RAM(WPTR) <= DIN;
      end if;
    end if;
  end process;
  -- 書き込みポインタ
  process(CLK, RST) begin
    if(RST='1') then
           WPTR <= 0;
    elsif(CLK'event and CLK = '1') then
          if(WEN = '1' and FF_REG='0') then -- ②
            if(WPTR=15) then
                   WPTR <= 0;
                 else
                   WPTR <= WPTR+1;
            end if;
      end if;
    end if;
  end process;
  -- 読み出しポインタ
  process(CLK, RST) begin
    if(RST='1') then
           RPTR <= 15;
    elsif(CLK'event and CLK = '1') then
          if(REN = '1' and EF_REG='0') then  --①
            if(RPTR=15) then
                   RPTR <= 0;
                 else
                   RPTR <= RPTR+1;
            end if;
      end if;
    end if;
  end process;
  -- Full Flag
  process(CLK, RST) begin
    if(RST='1') then
           FF_REG <= '0';
    elsif(CLK'event and CLK = '1') then
          if(WPTR=RPTR and WEN='1' and REN='0')
                                          then
                   FF_REG <= '1';
          elsif(FF_REG='1' and REN='1') then
                   FF_REG <= '0';
      end if;
    end if;
  end process;
  -- Almost Empty 1
  EF_TMP <= '1' when (RPTR=WPTR-1 or (RPTR=15 and
     WPTR=1) or (RPTR=15-1 and WPTR=0)) else '0';
  -- Empty Flag
  process(CLK, RST) begin
    if(RST='1') then
           EF_REG <= '1';
    elsif(CLK'event and CLK = '1') then
          if(EF_TMP='1' and REN='1'and WEN='0')
                                              then
                     EF_REG <= '1';
          elsif(EF_REG='1' and WEN='1') then
                     EF_REG <= '0';
          end if;
    end if;
  end process;
end RTL;
configuration CFG_FIFO_SYNC of FIFO_SYNC is
  for RTL
  end for;
end CFG_FIFO_SYNC;
```

〔リスト2.38〕FIFOのVerilog HDL記述（fifo_sync.v）

```
module FIFO_SYNC(RST, CLK, DIN, DOUT, WEN, REN,
                                        OE, EF, FF);
parameter DATA_WIDTH = 8;
parameter FIFO_DEPTH = 4;
`define D_WIDTH 8
`define F_DEPTH 4

input RST, CLK, WEN, REN, OE;
input [DATA_WIDTH-1:0] DIN;
inout [DATA_WIDTH-1:0] DOUT;
output EF, FF;

reg [DATA_WIDTH-1:0] SF_RAM [0:(1<<FIFO_DEPTH)-1];
                        // (1<<FIFO_DEPTH)-1 = 15
reg [0:(1<<FIFO_DEPTH)-1] WPTR;
reg [0:(1<<FIFO_DEPTH)-1] RPTR;
reg FF_REG, EF_REG;
wire EF_TMP;

// 出力アサインメント
assign  DOUT = (OE==1'b1)? SF_RAM[RPTR]:
                                        `D_WIDTH'bz;
assign  FF = FF_REG;
assign  EF = EF_REG;
// 出力アサインメント記述の終了

// FIFO
always @(posedge CLK)
  begin
    if(WEN==1'b1 & FF_REG==1'b0)
                SF_RAM[WPTR] <= DIN;
  end
//書き込みポインタ
always @(posedge CLK or posedge RST)
begin
  if(RST==1'b1)
        WPTR <= `F_DEPTH'b0;
  else
    begin
      if(WEN == 1'b1 & FF_REG == 1'b0)
        if(WPTR == `F_DEPTH'b1111)
                WPTR <= `F_DEPTH'b0;
        else
                WPTR <= WPTR + 1'b1;
   end
end
//読み出しポインタ
always @(posedge CLK or posedge RST)
begin
  if(RST==1'b1)
        RPTR <= `F_DEPTH'b1111;
  else
    begin
      if(REN == 1'b1 & EF_REG == 1'b0)
        if(RPTR == `F_DEPTH'b1111)
                RPTR <= `F_DEPTH'b0;
        else
                RPTR <= RPTR + 1'b1;
    end
end
//Full Flag
always @(posedge CLK or posedge RST)
begin
  if(RST==1'b1)
        FF_REG <= 1'b0;
  else
    begin
      if(WPTR == RPTR & WEN == 1'b1 & REN ==
                                            1'b0)
                FF_REG <= 1'b1;
      else if(FF_REG == 1'b1 & REN == 1'b1)
                FF_REG <= 1'b0;
    end
end
//Almost Empty
assign EF_TMP = ((RPTR == WPTR-1'b1)|
                 (RPTR == `F_DEPTH'd15 & WPTR ==
                                `F_DEPTH'd1)|
                 (RPTR == `F_DEPTH'd14 & WPTR ==
                      `F_DEPTH'd0))? 1'b1 : 1'b0;
//Empty Flag
always @(posedge CLK or posedge RST)
begin
  if(RST==1'b1)
        EF_REG <= 1'b1;
  else
    begin
      if(EF_TMP == 1'b1 & WEN == 1'b0 & REN == 1'b1)
                EF_REG <= 1'b1;
      else if(EF_REG == 1'b1 & WEN == 1'b1)
                EF_REG <= 1'b0;
    end
end
endmodule
```

二つのフラグを見ながら制御

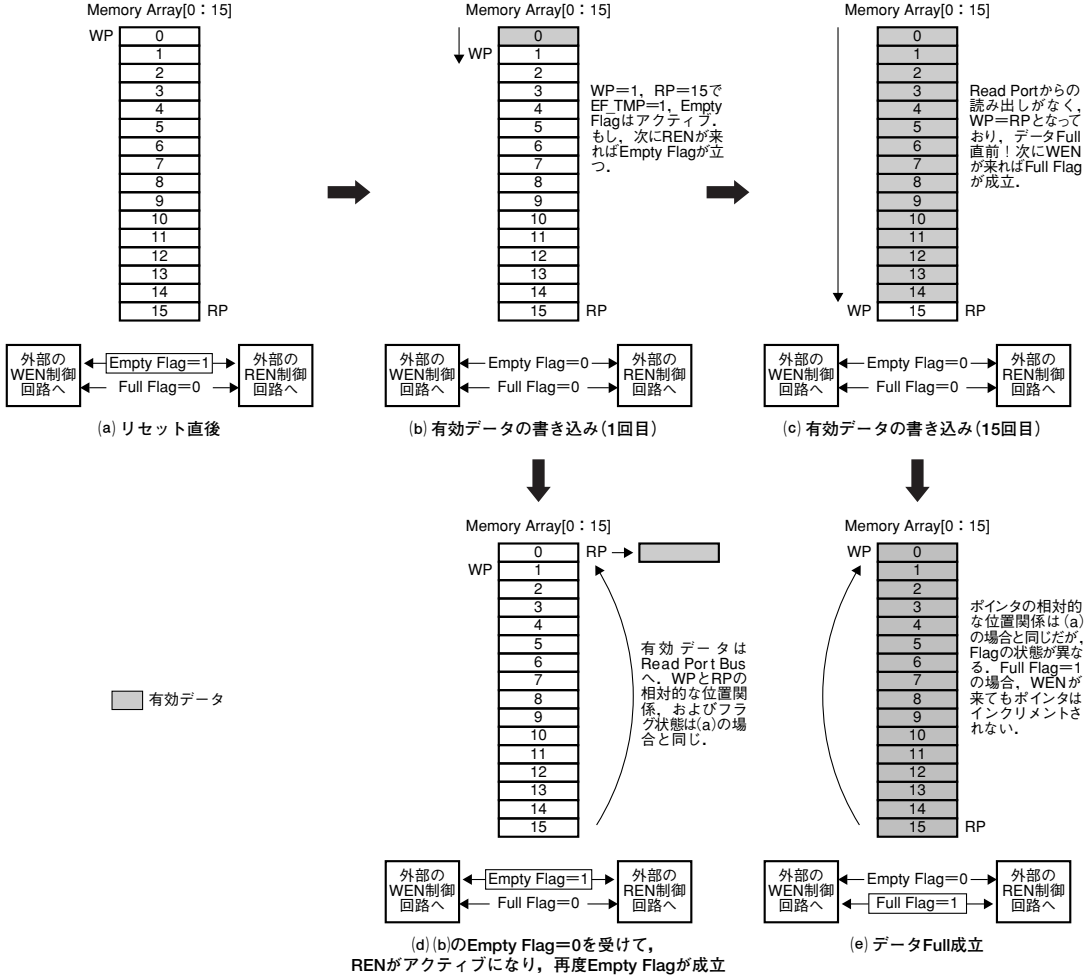
さて，同期バスとはすなわち，クロックに同期して動作するバスです．非同期バスのように書き込み信号の立ち上がりエッジや読み出し信号の立ち下がりエッジに合わせて状態が決まるのではなく，クロック信号

の立ち上がり時のバス制御信号の状態によって動作が決まります．

図2.29にFIFOのブロック図を示します．入力信号OEは，不必要な場合には取り除いてください．では，FIFOの動作を**図2.30**を使って説明します．**図2.30**(a)はリセット直後の状態です．そして，**図2.30**(b)は，データが一つ書き込まれた状態です．書き込みポインタ（WP：Write Pointer）は一つ進みますが，読み出しポインタ（RP：Read Pointer）はそのままです．また，通常は，書き込み側の外部制御回路が

〔図2.30〕FIFOの動作

(a)は，リセット直後のFIFOのデータとフラグのようす．読み出しポインタ（Read Pointer）が“15”，書き込みポインタ（Write Pointer）が“0”の状態でEmpty Flagが立っている．(b)は，データが一つ書き込まれたときのデータとフラグのようす．Empty Flagは消える．読み出し側は，通常，このEmpty Flagを見て読み出しを開始する．(c)は，データが読み出されず，書きためられていく場合のデータとフラグのようす．なんらかの理由で読み出し側がFIFOからの読み出しを行えない場合，FIFOにデータがたまっていく．次のWENが来た瞬間にFull Flagが成立する．(d)は，(b)からデータが読み出された直後のデータとフラグのようす．読み出し側から読み出された場合，ポインタはリセット直後と同じくRP＝WP－1の関係を保ってEmpty Flagを立てる．ポインタの絶対位置以外は，まったくリセット直後と同じ状態になる．(e)は，データがFullになったときのデータとフラグのようす．ポインタの位置関係は(a)のリセット直後と同じだが，フラグの状態が異なることに注意する．この場合，書き込み側は書き込みを中断し，Full Flagが消えるのを待つ．



Empty Flag（EF）を見て書き込み開始，Full Flag（FF）を見て書き込み中断となるはずです．**図2.30**(d)の状態になるとデータは読み取られて，**図2.30**(a)と同じ状態に戻ります．この状態で読み出しを行っても，EFでマスクされているため，読み出しポインタは進みません（**リスト2.37**の①）．

また，**図2.30**(b)の状態から**図2.30**(c)のようにデータがたまっていくと，最後にはFull Flagが成立し，**図2.30**(e)の状態となります．この状態で次のデータを書き込んでもFFでマスクされているため書き込みポインタは進みませんし，データも書き込まれません（**リスト2.37**の②）．

図2.30(e)のポインタの相対的な位置関係は，**図2.30**(a)や**図2.30**(d)と同じですが，フラグの状態は異なります．そのため，どちらかのポインタのインクリメントがマスクされます（Empty Flagのときは読み出しポインタ，Full Flagのときは書き込みポインタがマスクされる）．これにより，読み出しポインタが書き込みポインタを追い越すことはなく，読み込みポインタが追い越されて周回遅れになるようなこともありません．

Almost FullやAlmost Emptyのフラグが有効

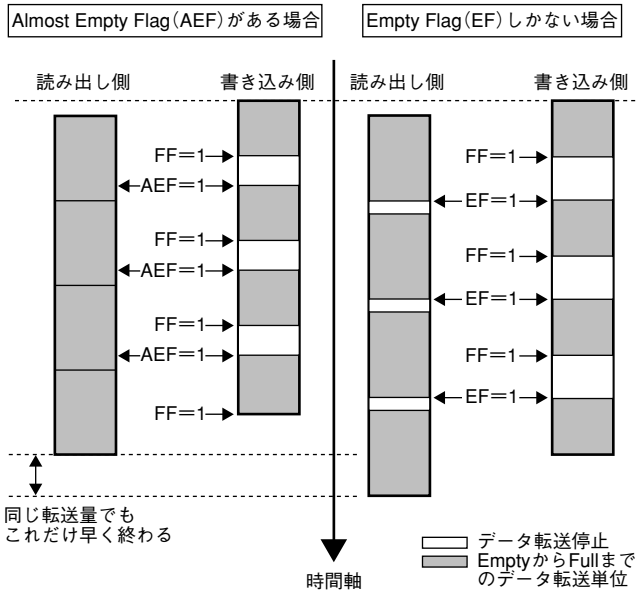
また，もっと早い段階でEmptyやFullの情報を知りたい場合があります．これは，フラグ情報を早めに知ることで，効率よくFIFOを制御できるからです．

たとえばEmpty Flagは，書き込み側がFullになってからEmptyになるまでなにも書き込まないという制御を行ってしまうと，読み出し側はEmptyまで読んだ後，次のEmpty Flagがクリアされるまで休ませてしまうことになり，読み出し側の性能が下がります．Empty Flagをなるべく成立させないように書き込み側で制御すれば，読み出し側はEmptyによるペナルティなしに読み続けます．

そのようすを**図2.31**に示します．右側がEmpty Flagしかもたないfifo，左側がAlmost Empty Flag（Empty + 1）をもつFIFOです．この例では，書き込み側の速度が読み出し側より速くなっています．Almost Empty Flagで書き込みを再開している左側のほうが，転送が速くなることがわかります．この

〔図2.31〕Almost Empty Flagによる性能向上

左側はAlmost Empty Flag（Emptyの一つ前の情報）がある場合の，右側はEmpty Flagしかない場合のデータ転送のようすである．ここでは書き込み速度が読み出し速度に比べて速いものとしている（データ転送中のボックスの縦方向の長さが短いほど，転送が速い）．右のようにEmpty Flagしかない場合，Empty Flagが立つまで次の転送を待っていると，読み出し側にもなにも転送しない時間が発生する．一方，Almost Empty Flagを使った場合，読み出し側にむだな時間がない．つねにFIFOにデータが存在する状態を保つことで，読み出し側の性能を最大にしている．



ように，Emptyの前段階の情報があると書き込み側は便利です．ポインタを先読みしてAlmost FullやAlmost Emptyなどの信号を追加してみてください．

16 デュアル・ポートSRAM(非同期バス)

- 作成者名：田原迫仁治
- サンプル記述：リスト2.39(dpram_async.vhd)，リスト2.40（dpram_async.v）
- モデルの種類：ビヘイビア・モデル
- 検証に使用したシミュレータ：Actel Desktop VeriBest(VHDL)，ModelSim(Verilog HDL)
- 端子表

  入力：AL，AR，CEL_L，CER_L，OEL_L，OER_L，RHWLL，RHWLR

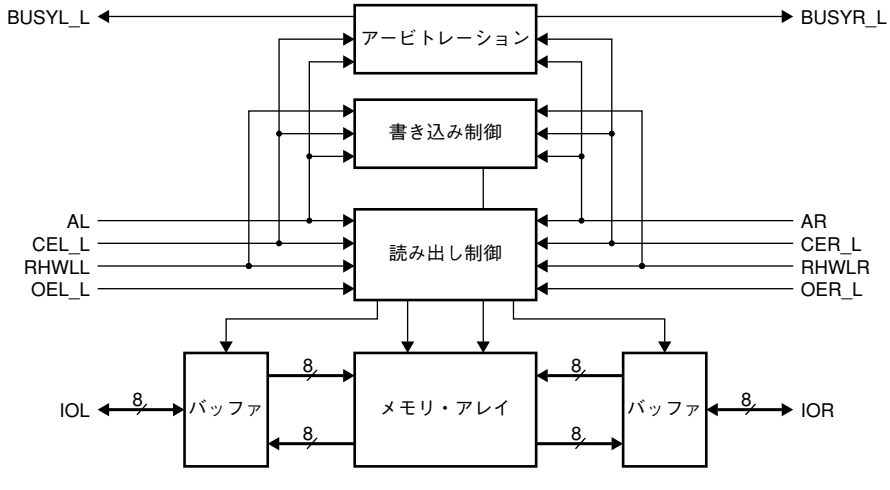
  入出力：IOL，IOR

  出力：BUSYL_L，BUSYR_L

デュアル・ポートSRAMは，2方向から読み出し/書き込み可能なSRAMです．細部の仕様はメーカごとに異なりますが，共通している点は，同一アドレスに対する同時書き込みと書き込み中の読み出し，読み出し中の書き込みを監視していることです．これは，入力されてくる2ポート双方からのアドレスとCEL_LまたはCER_L(Chip Enable)がマッチした際，後からアドレスを出力してきたほうのポート側にBUSY信号を出力し，「いま，このアドレスには書き込み/読み出しできない」ということを伝えます．

〔図2.32〕デュアル・ポートSRAMのブロック図

同一アドレスの読み出し/書き込みを監視・調停するアービトレーション・ブロックがある．両サイドの制御回路は，それぞれのブロックのBUSY信号にだけ注意を払えば，不ぐあいなく読み出し/書き込みを行える．



〔リスト2.39〕デュアル・ポートSRAMのVHDL記述（dpram_async.vhd）

```
library IEEE;
use     IEEE.std_logic_1164.all ;
use     IEEE.std_logic_unsigned.all ;

entity DPRAM_ASYNC is
  port (
    AL :        in        std_logic_vector(3
                                      downto 0);
    AR :        in        std_logic_vector(3
                                      downto 0);
    IOL :       inout     std_logic_vector(7
                                      downto 0);
    IOR :       inout     std_logic_vector(7
                                      downto 0);
    CEL_L :     in        std_logic;
    CER_L :     in        std_logic;
    OEL_L :     in        std_logic;
    OER_L :     in        std_logic;
    RHWLL :     in        std_logic;
    RHWLR :     in        std_logic;
    BUSYL_L :   out       std_logic;
    BUSYR_L :   out       std_logic
        );
end DPRAM_ASYNC;

architecture RTL of DPRAM_ASYNC is

subtype DPRAMWORD is std_logic_vector(7 downto 0);
type DPRAMARRAY is array (0 to 2**3-1) of DPRAMWORD;

signal DPRAM_D : DPRAMARRAY;
signal AL_IN :   integer range 0 to 2**3-1;
signal AR_IN :   integer range 0 to 2**3-1;
signal ALHLD :   std_logic_vector(3 downto 0);
signal ARHLD:    std_logic_vector(3 downto 0);
signal RHWL :    std_logic;
signal CEL_L_D : std_logic;
signal CER_L_D : std_logic;
signal CEDIR :   std_logic;

begin
  -- 出力アサインメント

  -- 出力アサインメント記述の終了
  AL_IN <= CONV_INTEGER(AL);        -- ①
  AR_IN <= CONV_INTEGER(AR);
  -- 書き込みフェーズ
  CEL_L_D <= CEL_L after 1ns;       -- ③
  CER_L_D <= CER_L after 1ns;
  RHWL <= '0' when ((RHWLL='0'and CEL_L='0') or
             (RHWLR='0' and CER_L='0')) else '1';
  process(RHWL) begin
    if(RHWL'event and RHWL = '1') then
          if(CEL_L_D ='0') then
            DPRAM_D(AL_IN) <= IOL;  -- ②
          elsif(CER_L_D='0') then
            DPRAM_D(AR_IN) <= IOR;
      end if;
    end if;
  end process;
  -- 読み出しフェーズ ④
  IOL <= DPRAM_D(AL_IN) when (CEL_L='0' and
     OEL_L='0' and RHWLL='1') else (others=>'Z');
  IOR <= DPRAM_D(AR_IN) when (CER_L='0' and
     OER_L='0' and RHWLR='1') else (others=>'Z');
  -- ⑤
  -- BUSY
  process(CEL_L) begin
    if(CEL_L'event and CEL_L = '0') then
            ALHLD <= AL;
    end if;
  end process;
  process(CER_L) begin
    if(CER_L'event and CER_L = '0') then
            ARHLD <= AR;
    end if;
  end process;
  process(CER_L, CEL_L) begin
    if(CER_L'event and CER_L = '0') then
            CEDIR <= '1';
        elsif(CEL_L'event and CEL_L = '0') then
            CEDIR <= '0';
    end if;
  end process;
  BUSYL_L <= '0' when (CEL_L='0' and CER_L='0'
             and ARHLD=AL and CEDIR='0') else '1';
  BUSYR_L <= '0' when (CER_L='0' and CEL_L='0'
             and ALHLD=AR and CEDIR='1') else '1';
end RTL;
configuration CFG_DPRAM_ASYNC of DPRAM_ASYNC is
  for RTL
  end for;
end CFG_DPRAM_ASYNC;
```

〔リスト2.40〕デュアル・ポートSRAMのVerilog HDL記述(dpram_async.v)

```
module DPRAM_ASYNC(AL, AR, IOL, IOR,
                   CEL_L, CER_L, OEL_L, OER_L,
                   RHWLL, RHWLR, BUSYL_L, BUSYR_L);
parameter ADDRESS = 4;
parameter DATA_WIDTH = 8;
parameter DELAY = 1 ;
`define WIDTH 8
input [ADDRESS-1:0] AL, AR;
inout [DATA_WIDTH-1:0] IOL, IOR;
input CEL_L, CER_L, OEL_L, OER_L, RHWLL, RHWLR;
output BUSYL_L, BUSYR_L;

//reg [DATA_WIDTH-1:0] DPRAM_D [0:7];
reg [DATA_WIDTH-1:0] DPRAM_D [0:(1<<(ADDRESS-1))-1];
//wire [(2**(ADDRESS-1))-1:0] AL_IN, AR_IN;
reg [ADDRESS-1:0] ALHLD, ARHLD;
wire RHWL, CEL_L_D, CER_L_D;

// 出力アサインメント
assign RHWL = ((RHWLL== 1'b0 & CEL_L==1'b0) |
               (RHWLR==1'b0 & CER_L==1'b0))?
                                    1'b0: 1'b1;
// 出力アサインメント記述の終了

// 書き込みフェーズ
assign #DELAY CEL_L_D = CEL_L;
assign #DELAY CER_L_D = CER_L;

always @(posedge RHWL)
   begin
      if(CEL_L_D==1'b0)
         DPRAM_D[AL] <= IOL;
      else if(CER_L_D==1'b0)
         DPRAM_D[AR] <= IOR;
  end
// 読み出しフェーズ
assign IOL = (CEL_L==1'b0 & OEL_L==1'b0 &
                                    RHWLL==1'b1)?
              DPRAM_D[AL] : `WIDTH'bz;
assign IOR = (CER_L==1'b0 & OER_L==1'b0 &
                                    RHWLR==1'b1)?
              DPRAM_D[AR] : `WIDTH'bz;
// BUSY

always @(negedge CEL_L)
begin
   ALHLD <= AL;
end

always @(negedge CER_L)
begin
   ARHLD <= AR;
end

assign BUSYL_L = (CEL_L==1'b0 & CER_L==1'b0 &
                      (ARHLD == AL))? 1'b0 : 1'b1;
assign BUSYR_L = (CER_L==1'b0 & CEL_L==1'b0 &
                      (ALHLD == AR))? 1'b0 : 1'b1;
endmodule
```

ラッチを使ってビヘイビア・モデルを記述

ここではシミュレーション・モデルとして，内部を同期設計せずに，ラッチを使った短い記述で書いてみましょう(この記述から回路を合成することはできない)．**図2.32**にブロック図を示します．**リスト2.39**の①でアドレス入力AL，ARは，INTEGERへの型変換を行います．また，AL_IN，AR_INは②でRAMのアドレスを直接記述できるようにします．③では，RAMへの書き込みイネーブル信号RHWLの立ち上がりに対するホールドを確保するため，1ns遅らせます．これにより，②において選択されたアドレスにデータが書き込まれます．

④は，読み出し側でRHWLL= '1'，CEL_L= '0' のときに，該当アドレスのデータを出力する回路です．ここでは，ビヘイビア記述であることもあって，OEL_L= '0' またはOEL_R= '0' でない場合，この階層で直接ハイ・インピーダンスになるように記述しています．⑤以下ではBUSY信号のアービトレーションを行います．CEL_LまたはCER_Lの立ち下がりごとのデータを保持し，バス上のデータと比較します．

そして，両ポートのCEL_L，CER_Lがアクティブの状態で，こちらのポートと他方のアドレスを比較し，かつこちらのポートのBUSYがアクティブでなければ，こちらのポートに使用権を与え，他方のポートのBUSYをアクティブにします．

17 ISAバス・インターフェース・コントローラ+スクラッチパッド・レジスタ

- 作成者名：田原迫仁治
- サンプル記述：リスト2.41（isa.vhd），リスト2.42（isa.v）
- モデルの種類：RTLモデル
- 検証に使用したシミュレータ：Actel Desktop VeriBest（VHDL），ModelSim（Verilog HDL）
- 端子表

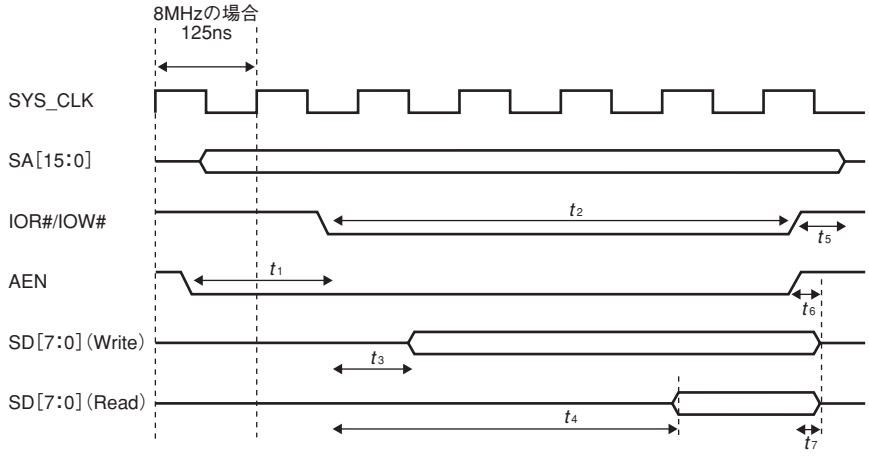
  入力：RST_N，SYS_CLK，SA，IOR_N，IOW_N，AEN

  入出力：SD

図2.33にISAバスのタイミング・チャートを示します．細かい数字が並んでいますが，要は読み出しデータをt_4以内に出力し，IOW#の立ち上がりでデータを取り込むということです．

〔図2.33〕ISAバス（8ビットI/O）の読み出し/書き込みサイクル

細かいタイミングの数字が示されているが，IOR#の立ち下がりでt_4区間以内にデータを出力し，IOW#の立ち上がりでデータを取り込めばよい．



t_1：AEN解放から読み出し/書き込みアクティブまで	最小	150ns
t_2：読み出し/書き込み信号幅	最小	415ns
t_3：書き込み信号からデータ確定まで	最大	55ns
t_4：読み出し信号からデータ確定まで	最大	372ns
t_5：SAホールド時間	最小	20ns
t_6：書き込みデータのホールド時間	最小	15ns
t_7：読み出しデータのホールド時間	最小	0ns

〔リスト2.41〕ISAバス・インターフェース・コントローラのVHDL記述(isa.vhd)

```
library IEEE;
use     IEEE.std_logic_1164.all ;
use     IEEE.std_logic_misc.all ;
use     IEEE.std_logic_unsigned.all ;
use     IEEE.std_logic_arith.all ;

entity ISA is
  generic( IOADDR_B  : bit_vector := X"300"); --300h
  port (
        RST_N :  in       std_logic;
        SYS_CLK :in       std_logic;
        SD :     inout    std_logic_vector(7
                                    downto 0);
        SA :     in       std_logic_vector(15
                                    downto 0);
        IOR_N :  in       std_logic;
        IOW_N :  in       std_logic;
        AEN :    in       std_logic
        );
end ISA;

architecture RTL of ISA is
signal IOCS,IOCS1,IOCS2 : std_logic;
signal IOW1,IOW2,IOW_F : std_logic;
signal IOADDR : std_logic_vector(11 downto 0);
signal SD1, SD2, SREG : std_logic_vector(7 downto 0);
begin
  -- 出力アサインメント
  SD <= SREG when (IOCS='1' and IOR_N='0' and
                    AEN='0') else "ZZZZZZZZ";
  -- 出力アサインメント記述の終了
  IOADDR <= TO_STDLOGICVECTOR(IOADDR_B);
  IOCS <= '1' when (SA = "0000" & IOADDR) and
                  (AEN='0') else '0'; --300h ①
  process(SYS_CLK) begin
        if(SYS_CLK'event and SYS_CLK='1') then
          IOCS1 <= IOCS;
          IOCS2 <= IOCS1;
        end if;
  end process;

  process(SYS_CLK) begin
        if(SYS_CLK'event and SYS_CLK='1') then
          IOW1 <= not IOW_N;
          IOW2 <= IOW1;
        end if;
  end process;

  IOW_F <= '1' when IOW2='1' and IOW1='0' and
                      IOW_N ='1' else '0'; -- ②

  process(SYS_CLK) begin
        if(SYS_CLK'event and SYS_CLK='1') then
          SD1 <= SD;
          SD2 <= SD1;
        end if;
  end process;

  process(SYS_CLK, RST_N) begin
    if(RST_N='0') then
          SREG <= "00000000";
        elsif(SYS_CLK'event and SYS_CLK='1') then
          if(IOW_F ='1' and IOCS2='1') then -- ③
            SREG <= SD2;
          end if;
        end if;
  end process;

end RTL;
configuration CFG_ISA of ISA is
  for RTL
  end for;
end CFG_ISA;
```

〔リスト2.42〕ISAバス・インターフェース・コントローラのVerilog HDL記述(isa.v)

```
module ISA(RST_N, SYS_CLK, SD, SA, IOR_N, IOW_N,
                                          AEN);
parameter IOADDR = 16'h0300;
input RST_N, SYS_CLK, IOR_N, IOW_N, AEN;
inout [7:0] SD;
input [15:0] SA;

wire IOCS;
reg IOCS1,IOCS2;
reg IOW1,IOW2;
wire IOW_F;

reg [7:0] SD1, SD2, SREG;

// 出力アサインメント
assign SD = (IOCS==1'b1 & IOR_N==1'b0 &
                        AEN==1'b0)? SREG : 8'bz;
// 出力アサインメント記述の終了

assign IOCS = ((SA==IOADDR) & (AEN==1'b0))? 1'b1
                                    : 1'b0; //300h

always @(posedge SYS_CLK)
   begin
        IOCS1 <= IOCS;
        IOCS2 <= IOCS1;
   end
```

〔リスト2.42〕ISAバス・インターフェース・コントローラのVerilog HDL記述(isa.v)(つづき)

```
always @(posedge SYS_CLK)
begin
        IOW1 <= ~IOW_N;
        IOW2 <= IOW1;
end

assign IOW_F = (IOW2==1'b1 & IOW1==1'b0 &
                        IOW_N==1'b1)? 1'b1 : 1'b0;

always @(posedge SYS_CLK)
begin
        SD1 <= SD;
        SD2 <= SD1;
end

always @(posedge SYS_CLK or negedge RST_N)
begin
   if(RST_N==1'b0)
        SREG <= 8'b00000000;
   else
        begin
           if(IOW_F==1'b1)
                SREG <= SD2;
        end
end

endmodule
```

I/Oアドレスにはbit_vector指定を利用

リスト2.41の①でI/Oチップ・セレクトを生成しています．ISAのI/Oアドレス空間は16ビット(64Kバイト)あり，フル・デコードします．

ここでは，仮に300hをI/Oアドレスと仮定します．IOADDR_Bというgenericポート名により300hをbit_vector指定しています．std_logic_vectorでは2進数の指定のみで，_で区切ることもできませんが，bit_vectorは8進，16進，_の挿入などが可能です．次に，AENはDMAコントローラがISAバスを占有している間，“H”となる信号で，チップ・セレクトを生成する際はこの信号が“L”であることを確認する必要があります．

②ではIOW_N信号の立ち上がりを判定するため，IOW，IOCS，SDをそれぞれ2クロック分保持し，IOWより2クロック遅延した各データを用いて，立ち上がりの動きがあったかどうかを判定します．この回路によりIOW#のグリッジによる誤動作も防止しています．③では，この②のIOW_FとIOCSの条件がそろったときに，SD2に保持されていたデータが，スクラッチ・パッド・レジスタに書き込まれます．

読み出しのほうは，ここではたんなるレジスタなので，IOCS='1'，IOR_N='0'，AEN='0'を条件にSREGの値をバス上に出力します．これが，内部のアプリケーションから書き換えられる可能性のあるレジスタ(たとえばステータス・レジスタ)だった場合，読み出し期間中にアプリケーション側から値が書き換えられないようにする必要があります．