```
//*****************************************************************************
//*
//*  DOLPHIN
//*        INTEGRATION
//*
//*****************************************************************************
//*                                 *
//*                 ______          *
//*               /       \ ___     *
//*             /     . _-_-_-      *
//*       _____/          /         *  Dolphin Integration
//*    #_/    /          /     #    *  B.P. 65- ZIRST - 39, avenue du Granier
//*    /                _/          *  38242 MEYLAN CEDEX - FRANCE
//*   |_____           /       #    *
//*         \         /              *
//*    #    |        /          #    *  Tel : (33)/(0)4 76 41 10 96
//*         |       /                *
//*    #    |      /            #    *  Fax : (33)/(0)4 76 90 29 65
//*        /      /                  *
//*  _ # /     /               #    *  Email : gds2@dolphin.fr
//*| \_/    /                       *
//* \ #   /                    #    *
//*  |   / #    #    #    #    #     *
//*  |/                              *
//*                                  *
//*****************************************************************************



/*****************************************************************************
Static RAM generator V4.1

Static RAM VERILOG model V4.2, unidirectional configuration
Author : DOLPHIN Integration
Last modification :  27-07-99
*****************************************************************************/


//`define functional

`define RAM_MESSAGE_X
`define RAM_SIMULT_CHECK


`celldefine
`suppress_faults
`enable_portfaults

`ifdef functional
  `timescale 1ns/1ns
  `delay_mode_distributed
`else
  `timescale 1ns/10ps
  `delay_mode_path
`endif


module sram256x32 (NRST, CS, RD, WR, EN, AD, DI, DO);

   parameter         add_bit  = 8,
                     data_bit = 32,
                     nb_word  = 256;

   input             CS,         /* operation on rising edge */
                     RD,         /* read operation on CS rising edge */
                     WR,         /* write operation on CS rising edge */
                     EN,         /* DO are Z when EN high */
                     NRST;       /* reset, active low */
   input [7:0]       AD;         /* address bus */
   input [31:0]       DI;         /* input data bus */
   output [31:0]      DO;         /* output data bus */

   /* wires to create internal signals for each input
    this has been done for PORT delays in SDF annotator */
   wire              CS_int,
                     RD_int,
```

```
                WR_int,
                EN_int,
                NRST_int;
wire [7:0]      AD_int;
wire [31:0]      DI_int;
reg             prev_CS,        /* to store previous value of CS */
                new_req;        /* variable to detect new request */
wire            INT_EN_DO;      /* internal EN for DO */
reg [1:0]       INT_RDWR;       /* internal value for RD-WR inputs */
reg [7:0]       INT_AD,         /* internal value of address bus */
                PREV_AD;        /* storage of previous READ address */
reg [31:0]      REG_DO,         /* internal register for DO bus */
                INT_DI;         /* internal value for DI bus */
wire [31:0]     INT_DO;         /* internal value for DO bus */

reg [31:0]      PLAN [255:0];  /* memory */

wire [31:0]     RAM_WORD_X='bx; /* to set WORD to all x's */
wire [7:0]      RAM_ADDR_X='bx; /* to set INT_AD to all x's */

/* buffers to create internal signals for each input */
buf (CS_int, CS),
  (RD_int, RD),
  (WR_int, WR),
  (NRST_int, NRST),
  (EN_int, EN);
buf
  (AD_int[0], AD[0]),
  (AD_int[1], AD[1]),
  (AD_int[2], AD[2]),
  (AD_int[3], AD[3]),
  (AD_int[4], AD[4]),
  (AD_int[5], AD[5]),
  (AD_int[6], AD[6]),
  (AD_int[7], AD[7]);
buf
  (DI_int[0], DI[0]),
  (DI_int[1], DI[1]),
  (DI_int[2], DI[2]),
  (DI_int[3], DI[3]),
  (DI_int[4], DI[4]),
  (DI_int[5], DI[5]),
  (DI_int[6], DI[6]),
  (DI_int[7], DI[7]),
  (DI_int[8], DI[8]),
  (DI_int[9], DI[9]),
  (DI_int[10], DI[10]),
  (DI_int[11], DI[11]),
  (DI_int[12], DI[12]),
  (DI_int[13], DI[13]),
  (DI_int[14], DI[14]),
  (DI_int[15], DI[15]),
  (DI_int[16], DI[16]),
  (DI_int[17], DI[17]),
  (DI_int[18], DI[18]),
  (DI_int[19], DI[19]),
  (DI_int[20], DI[20]),
  (DI_int[21], DI[21]),
  (DI_int[22], DI[22]),
  (DI_int[23], DI[23]),
  (DI_int[24], DI[24]),
  (DI_int[25], DI[25]),
  (DI_int[26], DI[26]),
  (DI_int[27], DI[27]),
  (DI_int[28], DI[28]),
  (DI_int[29], DI[29]),
  (DI_int[30], DI[30]),
  (DI_int[31], DI[31]);


/* internal buffer for intermediate EN */
buf     bendo (INT_EN_DO,EN);
/* output buffers for DO
  delay of 1 for delay_mode_unit */
bufif0          #1
  bout0    ( DO[0], INT_DO[0], INT_EN_DO),
  bout1    ( DO[1], INT_DO[1], INT_EN_DO),
```

```
  bout2    ( DO[2], INT_DO[2], INT_EN_DO),
  bout3    ( DO[3], INT_DO[3], INT_EN_DO),
  bout4    ( DO[4], INT_DO[4], INT_EN_DO),
  bout5    ( DO[5], INT_DO[5], INT_EN_DO),
  bout6    ( DO[6], INT_DO[6], INT_EN_DO),
  bout7    ( DO[7], INT_DO[7], INT_EN_DO),
  bout8    ( DO[8], INT_DO[8], INT_EN_DO),
  bout9    ( DO[9], INT_DO[9], INT_EN_DO),
  bout10    ( DO[10], INT_DO[10], INT_EN_DO),
  bout11    ( DO[11], INT_DO[11], INT_EN_DO),
  bout12    ( DO[12], INT_DO[12], INT_EN_DO),
  bout13    ( DO[13], INT_DO[13], INT_EN_DO),
  bout14    ( DO[14], INT_DO[14], INT_EN_DO),
  bout15    ( DO[15], INT_DO[15], INT_EN_DO),
  bout16    ( DO[16], INT_DO[16], INT_EN_DO),
  bout17    ( DO[17], INT_DO[17], INT_EN_DO),
  bout18    ( DO[18], INT_DO[18], INT_EN_DO),
  bout19    ( DO[19], INT_DO[19], INT_EN_DO),
  bout20    ( DO[20], INT_DO[20], INT_EN_DO),
  bout21    ( DO[21], INT_DO[21], INT_EN_DO),
  bout22    ( DO[22], INT_DO[22], INT_EN_DO),
  bout23    ( DO[23], INT_DO[23], INT_EN_DO),
  bout24    ( DO[24], INT_DO[24], INT_EN_DO),
  bout25    ( DO[25], INT_DO[25], INT_EN_DO),
  bout26    ( DO[26], INT_DO[26], INT_EN_DO),
  bout27    ( DO[27], INT_DO[27], INT_EN_DO),
  bout28    ( DO[28], INT_DO[28], INT_EN_DO),
  bout29    ( DO[29], INT_DO[29], INT_EN_DO),
  bout30    ( DO[30], INT_DO[30], INT_EN_DO),
  bout31    ( DO[31], INT_DO[31], INT_EN_DO);

buf
  bouti0   ( INT_DO[0], REG_DO[0]),
  bouti1   ( INT_DO[1], REG_DO[1]),
  bouti2   ( INT_DO[2], REG_DO[2]),
  bouti3   ( INT_DO[3], REG_DO[3]),
  bouti4   ( INT_DO[4], REG_DO[4]),
  bouti5   ( INT_DO[5], REG_DO[5]),
  bouti6   ( INT_DO[6], REG_DO[6]),
  bouti7   ( INT_DO[7], REG_DO[7]),
  bouti8   ( INT_DO[8], REG_DO[8]),
  bouti9   ( INT_DO[9], REG_DO[9]),
  bouti10   ( INT_DO[10], REG_DO[10]),
  bouti11   ( INT_DO[11], REG_DO[11]),
  bouti12   ( INT_DO[12], REG_DO[12]),
  bouti13   ( INT_DO[13], REG_DO[13]),
  bouti14   ( INT_DO[14], REG_DO[14]),
  bouti15   ( INT_DO[15], REG_DO[15]),
  bouti16   ( INT_DO[16], REG_DO[16]),
  bouti17   ( INT_DO[17], REG_DO[17]),
  bouti18   ( INT_DO[18], REG_DO[18]),
  bouti19   ( INT_DO[19], REG_DO[19]),
  bouti20   ( INT_DO[20], REG_DO[20]),
  bouti21   ( INT_DO[21], REG_DO[21]),
  bouti22   ( INT_DO[22], REG_DO[22]),
  bouti23   ( INT_DO[23], REG_DO[23]),
  bouti24   ( INT_DO[24], REG_DO[24]),
  bouti25   ( INT_DO[25], REG_DO[25]),
  bouti26   ( INT_DO[26], REG_DO[26]),
  bouti27   ( INT_DO[27], REG_DO[27]),
  bouti28   ( INT_DO[28], REG_DO[28]),
  bouti29   ( INT_DO[29], REG_DO[29]),
  bouti30   ( INT_DO[30], REG_DO[30]),
  bouti31   ( INT_DO[31], REG_DO[31]);

/* variables to store timing of: */
time              rise_CS,      /* last rising edge of CS */
  set_NR,             /* last change of NRST */
  set_RD,             /* last change of RD */
  set_WR,             /* last change of WR */
  set_AD;             /* last change on AD bus */



/* variables to detect a constraint violation */
`ifdef functional
`else
```

```
   reg              viol_ad, viol_di,
     viol_rd, viol_wr,
     viol_cycle, viol_cs,
     viol_nr;

   specify
      specparam         AD_SETUP_TIME_LEC    =0,   /*address setup - read*/
        AD_SETUP_TIME_ECR       =0,   /*address setup - write*/
        AD_HOLD_TIME_LEC        =0,   /*address hold  - read*/
        AD_HOLD_TIME_ECR        =0,   /*address hold  - write*/
        DATA_SETUP_TIME_ECR     =0,   /*data setup    - write*/
        DATA_HOLD_TIME_ECR      =0,   /*data hold     - write*/
        ACCESS_TIME_LEC         =0,   /*DO valid after CSposedge-rd*/
        RD_SETUP_TIME           =0,   /*RD setup*/
        RD_HOLD_TIME            =0,   /*RD hold*/
        WR_SETUP_TIME           =0,   /*WR setup*/
        WR_HOLD_TIME            =0,   /*WR hold*/
        NRST_SETUP_TIME         =0,   /*end of NRST before CSposedge*/
        NRST_HOLD_TIME          =0,   /*CSposedge before end of NRST*/
        CYCLE_TIME_MIN          =0,   /*CS period*/
        CS_HIGH_TIME            =0,   /*CS high pulse*/
        CS_LOW_TIME             =0,   /*CS low  pulse*/
        HIGH_Z_TIME             =0,   /*high z propagation time*/
        LOW_Z_TIME              =0;   /*low  z propagation time*/

      $setuphold(posedge
CS&&&(RD==1'b1),AD[0],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[1],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[2],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[3],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[4],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[5],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[6],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);
      $setuphold(posedge
CS&&&(RD==1'b1),AD[7],AD_SETUP_TIME_LEC,AD_HOLD_TIME_LEC,viol_ad);

      $setuphold(posedge
CS&&&(WR==1'b1),AD[0],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[1],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[2],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[3],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[4],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[5],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[6],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);
      $setuphold(posedge
CS&&&(WR==1'b1),AD[7],AD_SETUP_TIME_ECR,AD_HOLD_TIME_ECR,viol_ad);

      $setuphold(posedge
CS&&&(WR==1'b1),DI[0],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[1],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[2],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[3],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[4],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[5],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[6],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[7],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[8],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
```

```
      $setuphold(posedge
CS&&&(WR==1'b1),DI[9],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[10],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[11],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[12],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[13],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[14],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[15],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[16],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[17],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[18],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[19],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[20],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[21],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[22],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[23],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[24],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[25],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[26],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[27],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[28],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[29],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[30],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);
      $setuphold(posedge
CS&&&(WR==1'b1),DI[31],DATA_SETUP_TIME_ECR,DATA_HOLD_TIME_ECR,viol_di);

      $setuphold(posedge CS,RD,RD_SETUP_TIME,RD_HOLD_TIME,viol_rd);
      $setuphold(posedge CS,WR,WR_SETUP_TIME,WR_HOLD_TIME,viol_wr);
      $setuphold(posedge CS,NRST,NRST_SETUP_TIME,NRST_HOLD_TIME,viol_nr);
      $period(posedge CS,CYCLE_TIME_MIN,viol_cycle);
      $period(negedge CS,CYCLE_TIME_MIN,viol_cycle);
      $width(posedge CS,CS_HIGH_TIME,0,viol_cs);
      $width(negedge CS,CS_LOW_TIME,0,viol_cs);
      if (!EN) (CS *> DO[0] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[1] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[2] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[3] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[4] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[5] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[6] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[7] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[8] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[9] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[10] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[11] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[12] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[13] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[14] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[15] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[16] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[17] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[18] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[19] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[20] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[21] ) = ( ACCESS_TIME_LEC);
      if (!EN) (CS *> DO[22] ) = ( ACCESS_TIME_LEC);
```

```
    if (!EN) (CS *> DO[23] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[24] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[25] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[26] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[27] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[28] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[29] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[30] ) = ( ACCESS_TIME_LEC);
    if (!EN) (CS *> DO[31] ) = ( ACCESS_TIME_LEC);

    (EN *> DO[0]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[1]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[2]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[3]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[4]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[5]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[6]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[7]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[8]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[9]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[10]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[11]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[12]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[13]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[14]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[15]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[16]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[17]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[18]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[19]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[20]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[21]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[22]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[23]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[24]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[25]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[26]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[27]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[28]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[29]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[30]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);
    (EN *> DO[31]) = ( LOW_Z_TIME,LOW_Z_TIME,HIGH_Z_TIME);


endspecify

always @(viol_ad) INT_AD <= RAM_ADDR_X;

always @(viol_di)
  begin
     INT_DI <= RAM_WORD_X;
     if (WR !== 1'b0) INT_RDWR <= 'bx;
  end // always @ (viol_di)

always @(viol_cycle or viol_cs) INT_RDWR <= 'bx;

always @(viol_nr or viol_wr ) INT_RDWR <= 'bx;

always @(viol_rd)
  begin
     if (WR !== 1'b0 || INT_RDWR === 'bx)
       INT_RDWR <= 'bx;
     else
       INT_RDWR <= 2'bx0;
  end // always @ (viol_rd)

`endif // !ifdef functional



`ifdef RAM_DEBUG
`define RAM_MESSAGE_X
`define RAM_SIMULT_CHECK
`endif // ifdef RAM_DEBUG
```

```
   task allMemorytoX;
      integer i;
      begin
         for (i=0; i< nb_word; i=i+1)
           PLAN[i] = RAM_WORD_X;
         `ifdef RAM_MESSAGE_X
         $display("WARNING! %m: all memory set to X's because of an error at the
inputs at %t",$realtime);
         `endif
         REG_DO <= RAM_WORD_X;
      end
   endtask



   initial
     begin
        $timeformat (-9, 2, " ns", 0);
        new_req = 0;
        `ifdef functional
        `else
        viol_ad = 0;
        viol_di = 0;
        viol_rd = 0;
        viol_wr = 0;
        viol_cs = 0;
        viol_nr = 0;
        viol_cycle = 0;
        `endif
     end



   always @(NRST_int) if (NRST_int==1'b1 || NRST_int==1'b0)
     begin
        `ifdef RAM_SIMULT_CHECK
        set_NR=$realtime;
        if (rise_CS==set_NR)
          begin
             $display("WARNING! %m: NRST and CS are simultaneous at %t",$realtime);
             INT_RDWR <= 'bx;
          end // if (rise_CS==set_NR)
        `endif
     end
   else
     /*illegal value on NRST */
     begin
        `ifdef RAM_MESSAGE_X
        $display("WARNING! %m: NRST=%b at %t",NRST,$realtime);
        `endif
     end



   always @ (CS_int)
     begin
        prev_CS <= CS_int;
        if (CS_int==1'b1 && prev_CS==1'b0)     // real rising edge on CS
          begin
             rise_CS=$realtime;
             `ifdef RAM_DEBUG
             $display("CS rising edge, new_req=%b",new_req);
             `endif
             `ifdef RAM_SIMULT_CHECK
             if (set_NR==rise_CS)
               begin
                  $display("WARNING! %m: CS and NRST are simultaneous at %t",
$realtime);
                  INT_RDWR <= 'bx;
               end // if (set_NR==rise_CS)
             `endif
             if (NRST_int==1'b1)
               begin
                  `ifdef RAM_SIMULT_CHECK
                  if (set_AD==rise_CS)
                    begin
                       $display("WARNING! %m: CS and AD are simultaneous at %t",
```

```
$realtime);
                        INT_RDWR <= 'bx;
                      end
                    if (set_RD==rise_CS)
                      begin
                         $display("WARNING! %m: CS and RD are simultaneous at %t",
$realtime);
                        INT_RDWR <= 'bx;
                      end
                    if (set_WR==rise_CS)
                      begin
                         $display("WARNING! %m: CS and WR are simultaneous at %t",
$realtime);
                        INT_RDWR <= 'bx;
                      end
                    `endif
                    {new_req,INT_RDWR,INT_AD,INT_DI} <=
{~new_req,RD_int,WR_int,AD_int,DI_int};
                end
              else if (NRST_int==1'b0)
                begin
                    {new_req,INT_RDWR,INT_AD,INT_DI} <= {~new_req,
2'b00,AD_int,DI_int};
                end
              else
                begin
                    {new_req,INT_RDWR,INT_AD,INT_DI} <= {~new_req,
2'bxx,AD_int,DI_int};
                end
          end
        else
          begin
             if (CS_int===1'bx)
               begin
                  INT_RDWR <= 'bx;
                  INT_AD <= RAM_ADDR_X;
                  `ifdef RAM_MESSAGE_X
                  $display("WARNING! %m: CS=%b at %t",CS,$realtime);
                  `endif
                end
          end
     end


   /* INT_RDWR and INT_AD are set on CS posedge or asynchrone on NRST or
asynchrone because of a violation */
   always @(new_req or INT_RDWR or INT_AD)
     begin

        case (INT_RDWR)
          2'b00: /* NOP */
            begin
               if (INT_AD!==PREV_AD)
                 REG_DO <= RAM_WORD_X;
               `ifdef RAM_DEBUG
               $display("NOP");
               `endif
            end // case: 2'b00

          2'b01: /* WRITE */
            begin
               if (^INT_AD===1'bx)
                 begin
                    allMemorytoX;
                 end // if (^INT_AD===1'bx)
               else
                 begin
                    if (INT_AD < nb_word)      //To test overflow during write
                      begin
                         `ifdef RAM_DEBUG
                         $display("WRITE");
                         `endif

                         PLAN[INT_AD] <= INT_DI;
                                                `ifdef RAM_MESSAGE_X
                         if(^INT_DI===1'bx) $display("WARNING! %m: write with
```

```
DI=%h at %t",INT_DI,$realtime);
                        `endif
                      end // if (INT_AD < nb_word)
                    else
                      begin
                        `ifdef RAM_MESSAGE_X
                        $display("WARNING! %m: address overflow during write at
%t, nothing to do",$realtime);
                        `endif
                         INT_RDWR <= 'bx;
                      end // else: !if(INT_AD < nb_word)
                  end // else: !if(^INT_AD===1'bx)
               if (INT_AD!==PREV_AD) REG_DO <= RAM_WORD_X;
             end // case: 2'b01

           2'b10: /* READ */
             begin
                PREV_AD <= INT_AD;
                `ifdef RAM_DEBUG
                $display("READ");
                `endif
                if(^INT_AD===1'bx)
                  begin
                     `ifdef RAM_MESSAGE_X
                     $display("WARNING! %m: read with AD=%h at %t, output is set to
X.",INT_AD,$realtime);
                     `endif
                     REG_DO <= RAM_WORD_X;
                  end // if (^INT_AD===1'bx)
                else
                  begin
                     if (INT_AD < nb_word)      // To test overflow during write
                       begin
                          REG_DO <= PLAN[INT_AD];
                       end // else: !if(^INT_AD===1'bx)
                     else
                       begin
                          REG_DO <= RAM_WORD_X;
                          `ifdef RAM_MESSAGE_X
                          $display("WARNING! %m: address overflow during read at %t,
output is set to X",$realtime);
                          `endif
                       end // else: !if(INT_AD < nb_word)
                  end // else: !if(^INT_AD===1'bx)
             end // case: 2'b10

           2'bx0: /* READ operation is not totally accomplished*/
             begin
                `ifdef RAM_DEBUG
                $display("READ, false");
                `endif
                `ifdef RAM_MESSAGE_X
                $display("WARNING! %m: timing violation on READ signal at %t,
output is set to X",$realtime);
                `endif
                REG_DO <= RAM_WORD_X;
             end // case: 2'bx0

           default:
             begin
                allMemorytoX;
             end // case: default

         endcase
      end



   `ifdef RAM_MESSAGE_X
   always @(RD_int) if (RD_int === 1'bx)
     $display("WARNING! %m: RD=%b at %t",RD,$realtime);
   always @(WR_int) if (WR_int === 1'bx)
     $display("WARNING! %m: WR=%b at %t",WR,$realtime);
   always @(EN_int) if (EN_int === 1'bx)
     $display("WARNING! %m: EN=%b at %t",EN,$realtime);
   `endif
```

```
   `ifdef RAM_SIMULT_CHECK

   always @(AD_int)
     begin
        set_AD=$realtime;
        if (rise_CS==set_AD)
          begin
             $display("WARNING! %m: AD and CS are simultaneous at %t",$realtime);
              INT_AD <= RAM_ADDR_X;
          end // if (rise_CS==set_AD)
     end

   always @(WR_int)
     begin
        set_WR=$realtime;
        if (rise_CS==set_WR)
          begin
             $display("WARNING! %m: WR and CS are simultaneous at %t",$realtime);
             INT_RDWR <= 'bx;
          end // if (rise_CS==set_WR)
     end

   always @(RD_int)
     begin
        set_RD=$realtime;
        if (rise_CS==set_RD)
          begin
             $display("WARNING! %m: RD and CS are simultaneous at %t",$realtime);
             if (WR==1'b1)
               INT_RDWR <= 'bx;
             else
               INT_RDWR <= 2'bx0;
          end // if (rise_CS==set_RD)
     end

   `endif


endmodule


`disable_portfaults
`nosuppress_faults
`endcelldefine
```