```
//****************************************************************************
//*
//*  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
//*| \_/   /                     *
//* \ #  /                  #    *
//* |  / #   #   #   #   #       *
//* |/                           *
//*                              *
//***************************************************************************



/***************************************************************************
 Diffusion ROM generator V4.1

 Diffusion ROM VERILOG model V4.1
 Author : DOLPHIN Integration
 Last modification :  03-09-99
 ***************************************************************************/


//`define functional

`celldefine
`suppress_faults
`enable_portfaults

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


module dirom512x32 (NRST, CS, EN, AD, DO);

parameter      add_bit  = 9,              /* number of address bits */
               data_bit = 32,              /* number of data bits */
               nb_word  = 512,            /* number of words */
               prog     = "";               /* programming file */

input          CS,          /* positive edge to start a read */
               EN,          /* DO is Z when EN high */
               NRST;        /* reset, active low */

input   [8:0] AD;          /* address bus */
output [31:0] DO;           /* output data bus */


/* wire for internal signal for each input for SDF annotating */

wire         CS_int,
             EN_int,
             NRST_int;

wire [8:0] AD_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 inputs */

reg  [8:0] INT_AD,         /* internal value of address bus */
            PREV_AD;        /* storage of previous READ address */

reg  [31:0] REG_DO;         /* internal register for DO bus */
wire [31:0] INT_DO;         /* internal value for DO bus */

reg  [31:0] PLAN [511:0];  /* memory */

wire [31:0] ROM_WORD_X='bx; /* to set WORD to all x's */
wire [8:0] ROM_ADDR_X='bx; /* to set INT_AD to all x's */

integer i;

/* internal buffers for each input for SDF annotating */
buf
      (CS_int, CS),
      (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]),
      (AD_int[8], AD[8]);

/* internal buffer for intermediate EN */
buf   bendo (INT_EN_DO, EN);

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 falling edge of NRST */
        set_AD;         /* last change on AD bus     */

`ifdef functional
`else
/* variables to detect a constraint violation */
reg
        viol_ad,
        viol_cycle,
        viol_cs,
        viol_nr_low,
        viol_nr;

specify
  specparam
      AD_SETUP_TIME       =0,     // address setup
      AD_HOLD_TIME        =0,     // address hold
      ACCESS_TIME         =0,     // CS posedge to DO valid
      CS_HIGH_TIME        =0,     // CS high time
      CS_LOW_TIME         =0,     // CS low pulse
      NRST_SETUP_TIME     =0,     // NRST end before CS posedge
      NRST_HOLD_TIME      =0,     // CS posedge before end of NRST
      CYCLE_TIME_MIN      =0,     // period min
      HIGH_Z_TIME         =0,     // EN high to DO = z
      LOW_Z_TIME          =0;     // EN low to DO != z

      $setuphold(posedge CS,AD[0],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[1],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[2],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[3],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[4],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[5],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[6],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[7],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);
      $setuphold(posedge CS,AD[8],AD_SETUP_TIME,AD_HOLD_TIME,viol_ad);

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

        (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 <= ROM_ADDR_X;

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

    `endif
```

```
   task allOutputstoX;
      integer i;
      begin
         #0.01;
         `ifdef ROM_MESSAGE_X
         if ($realtime > 0.01)
           $display("WARNING! %m: output is set to X's because of an error at the
inputs signals at %t",$realtime);
         `endif
         REG_DO <= ROM_WORD_X;
      end
   endtask


   initial
     begin
        new_req=0;
        $timeformat (-9, 2, " ns", 0);
        $readmemb(prog,PLAN);
        for (i=0; i<nb_word; i=i+1)
          if(^PLAN[i]===1'bx)
            begin
               $display("The programming file is not correct : Bad character, bad
line, bad number of lines.");
               $display("Problem at line or at word : ", i+1);
               $finish(0);
               INT_RDWR <= 'bx;
            end
     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 ROM_DEBUG
             $display("CS rising edge, new_req=%b", new_req);
             `endif
             `ifdef ROM_SIMULT_CHECK
             if (set_NR==rise_CS)
               begin
                  $display("WARNING! %m: CS and NRST are simultaneous at %t",
$realtime);
                  INT_RDWR <= 'bx;
               end
             `endif
             if (NRST_int==1'b1)
               begin
                  `ifdef ROM_SIMULT_CHECK
                  if (set_AD==rise_CS)
                    begin
                       $display("WARNING! %m: CS and AD are simultaneous at %t",
$realtime);
                       INT_RDWR <= 'bx;
                    end
                  `endif // ifdef ROM_SIMULT_CHECK
                  {new_req,INT_RDWR,INT_AD} <= {~new_req,2'b10,AD_int};
               end
             else if (NRST_int==1'b0)
               begin
                  {new_req,INT_RDWR,INT_AD} <= {~new_req,2'b00,AD_int};
               end
             else
               begin
                  {new_req,INT_RDWR,INT_AD} <= {~new_req,2'bxx,AD_int};
               end
          end
        else
          begin
             if (CS_int===1'bx)
               begin
                  INT_RDWR <= 'bx;
```

```
                    INT_AD <= ROM_ADDR_X;
                    `ifdef ROM_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 <= ROM_WORD_X;
                `ifdef ROM_DEBUG
                $display("NOP");
                `endif
             end

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

           2'bx0 : /* VIOLATION READ */
             begin
                `ifdef ROM_DEBUG
                $display("READ, false");
                `endif
                `ifdef ROM_MESSAGE_X
                $display("WARNING! %m: timing violation on READ signal at %t,
output is set to X.",$realtime);
                `endif
                REG_DO <= 'bx;
             end // case: 2'bx0

           default : /* forbidden conf or undetermined signals */
             begin
                allOutputstoX;
             end
         endcase
     end
```

```
   `ifdef ROM_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 <= ROM_ADDR_X;
          end
     end
   `endif



   `ifdef ROM_MESSAGE_X
   always @(AD_int) if (^AD_int===1'bx)
     $display("WARNING! %m: AD=%h at %t",AD,$realtime);
   always @(EN_int) if (EN===1'bx)
     $display("WARNING! %m: EN=%b at %t",EN,$realtime);
   `endif


   always @(NRST_int)
     if (NRST_int==1'b1 || NRST_int==1'b0)
       begin
          `ifdef ROM_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
          `endif
       end
     else
       /*illegal value on NRST */
       begin
          `ifdef ROM_MESSAGE_X
          $display("WARNING! %m: NRST=%b at %t",NRST,$realtime);
          `endif
       end



endmodule

`disable_portfaults
`nosuppress_faults
`endcelldefine
```