# 10.SRAM 读写实验例程说明

## 10.1 PGX-Nano 开发板简介

PGX-Nano 开发板搭载了一颗 2Mbit 的 16 位宽 SRAM，型号为 IS61WV12816DBLL-10TLI。此型号 SRAM 不需要时钟，只需使信号满足相应时序要求与状态保持时间，即可完成数据的存取操作。

## 10.2 实验目的

实验在 SRAM 的 17'h00000 地址位置的低字节写入 8bit 数据，按下 S1 按键为写入；再按下 S2 按键读出 17'h00000 地址位置低字节的 8bit 数据，led 灯显示读出的 8bit 数据。按下 S0 进行复位操作。

## 10.3 实验原理

此型号 SRAM 共有 **5** 个控制信号，**16** 位数据信号，**17** 位地址信号（开发板原理图对 SRAM 预留了 19 位地址信号，但最高位与次高位两位地址信号无效）。

关于 IS61WV12816DBLL10TLI 的详细描述请参考 IS61WV12816DBLL 数据手册（-10 表示访问时间 10ns）。

控制信号共有 5 个，均为**低电平**有效入下表所示：

| 信号名称 | 信号描述 |
| --- | --- |
| **CE** | 片选信号 |
| **WE** | 写使能信号 |
| **OE** | 输出使能信号 |
| **LB** | 低字节控制信号 |
| **UB** | 高字节控制信号 |

有关控制信号与数据信号的真值表如下图所示：

**TRUTH TABLE**

| Mode | $\overline{WE}$ | $\overline{CE}$ | $\overline{OE}$ | $\overline{LB}$ | $\overline{UB}$ | I/O PIN I/O0-I/O7 | I/O PIN I/O8-I/O15 | $V_{DD}$ Current |
|---|---|---|---|---|---|---|---|---|
| Not Selected | X | H | X | X | X | High-Z | High-Z | $I_{SB1}$, $I_{SB2}$ |
| Output Disabled | H | L | H | X | X | High-Z | High-Z | $I_{CC}$ |
| | X | L | X | H | H | High-Z | High-Z | |
| Read | H | L | L | L | H | $D_{OUT}$ | High-Z | $I_{CC}$ |
| | H | L | L | H | L | High-Z | $D_{OUT}$ | |
| | H | L | L | L | L | $D_{OUT}$ | $D_{OUT}$ | |
| Write | L | L | X | L | H | $D_{IN}$ | High-Z | $I_{CC}$ |
| | L | L | X | H | L | High-Z | $D_{IN}$ | |
| | L | L | X | L | L | $D_{IN}$ | $D_{IN}$ | |

分析上表所知：

SRAM **写状态**对应控制信号电平如下表所示（0 表示低电平，1 表示高电平）：

| 信号 | WE | CE | OE |
|---|---|---|---|
| 电平 | 0 | 0 | 1 |

此时，SRAM 将数据存入地址信号对应位置。

SRAM **读状态**对应控制信号电平如下表所示：

| 信号 | WE | CE | OE |
|---|---|---|---|
| 电平 | 1 | 0 | 0 |

此时，SRAM 将地址信号对应位置数据取出。

如果需要控制低字节\高字节输出、低字节\高字节输入，则就需要使用 LB、HB 两个控制信号，LB 为低电平时，数据的低字节可以存入 SRAM，或者地址信号对应位置数据的低字节可以取出，同理，HB 为低电平时，数据的高字节可以可以存入 SRAM，或者地址信号对应位置数据的高字节可以取出。

取出\存入 **16bit 完整数据**时：

| 信号 | LB | HB |
|---|---|---|
| 电平 | 0 | 0 |

仅取出\存入**低 8bit 数据**时：

| 信号 | LB | HB |
|---|---|---|
| 电平 | 0 | 1 |

仅取出\存入**高 8bit 数据**时：

| 信号 | LB | HB |
|---|---|---|
| 电平 | 1 | 0 |

在对 SRAM 进行读写操作时，因为此型号 SRAM 不需要时钟，所以相关信号需要满足指定的时间条件才能对 SRAM 进行读写操作。详细描述请参考 IS61WV12816DBLL 数据手册。

板卡晶振为 50MHz，周期为 20ns，此型号 SRAM 的访问时间时 10ns，因此通过查阅 SRAM 数据手册可知，控制 SRAM 控制信号在一个时钟周期内保持相应电平，即可满足时间条件与时序要求；例如在一个时钟周期内将 CE、WE 信号拉低（LB、HB 依据情况拉高或拉低，OE 信号拉高），即可将数据写入 SRAM 对应地址位置；在一个时钟周期内将 CE、OE 信号拉低（LB、HB 依据情况拉高或拉低，WE 信号拉高），即可读出 SRAM 对应地址位置的数据。

## 10.4 实验源码设计

**顶层：**在按键 S1 按下时，向 SRAM 写入数据，当 S2 按键按下时，向 SRAM 读出数据。

```
module sram_test(
     input        sys_clk    ,
     input        rst_n      ,
     input   [1:0]key        ,
     output   [7:0]led       /* synthesis PAP_MARK_DEBUG="true" */,
     output        sram_oe    /* synthesis PAP_MARK_DEBUG="true" */,
     output        sram_ce    /* synthesis PAP_MARK_DEBUG="true" */,
     output        sram_we    /* synthesis PAP_MARK_DEBUG="true" */,
     output        sram_ub    /* synthesis PAP_MARK_DEBUG="true" */,
     output        sram_lb    /* synthesis PAP_MARK_DEBUG="true" */,
     output [16:0]sram_addr  /* synthesis PAP_MARK_DEBUG="true" */,
     inout  [15:0]sram_data  /* synthesis PAP_MARK_DEBUG="true" */

   );

wire  [16:0]data_dbyte ;
reg         wr_en      ;
wire  [16:0]wr_addr    ;
wire  [15:0]wr_data    ;
reg         rd_en      /* synthesis PAP_MARK_DEBUG="true" */;
wire  [16:0]rd_addr    /* synthesis PAP_MARK_DEBUG="true" */;
wire  [15:0]rd_data    /* synthesis PAP_MARK_DEBUG="true" */;
wire        rd_data_en /* synthesis PAP_MARK_DEBUG="true" */;
wire        busy       ;
```

```
wire        busy_end   ;
wire [1:0]key_sig    /* synthesis PAP_MARK_DEBUG="true" */;
reg  [1:0]key_sig_reg     /* synthesis PAP_MARK_DEBUG="true" */;
reg  [7:0]led_reg ;

btn_deb_fix#(
    .BTN_WIDTH (4'd2        )    ,
    .BTN_DELAY (20'h8_3F7C  )
)u_btn_deb_fix
(
    .clk          (sys_clk) ,
    .btn_in       (key) ,
    .btn_deb      (key_sig)
);

always @(posedge sys_clk)begin
    if (~rst_n) begin
        key_sig_reg <= 2'b0 ;
    end else begin
        key_sig_reg <= key_sig ;
    end end

always @(posedge sys_clk) begin
    if (~rst_n) begin
        wr_en <= 1'b0 ;
        rd_en <= 1'b0 ;
    end else if ((~key_sig[0]) & (key_sig_reg[0])) begin
        wr_en <= 1'b1 ;
        rd_en <= 1'b0 ;
    end else if ((~key_sig[1]) & (key_sig_reg[1])) begin
        wr_en <= 1'b0 ;
        rd_en <= 1'b1 ;
    end else begin
        wr_en <= 1'b0 ;
        rd_en <= 1'b0 ;
    end end

assign wr_data = {8'b0 ,8'b1010_1010} ;

sram_dri #(
    .clk_frequency (27'd27_000_000)    //sram max 100MHz
//    parameter       data_dbyte  = 12'd1280
)my_sram_dri(
   .clk                (sys_clk       ) ,
```

```
    .rst_n              (rst_n      ) ,
    .data_dbyte         (17'd1      ),
    .wr_en              (wr_en      ),
    .wr_addr            (17'b0      ),
    .wr_data            (wr_data      ),
    .wr_lub             (2'b10      ),
    .rd_en              (rd_en      ),
    .rd_addr            (17'b0      ),
    .rd_data            (rd_data      ),
    .rd_data_en         (rd_data_en       ),
    .rd_lub             (2'b10     ),
    .busy               (busy   ),
    .busy_end           (busy_end ),
    .sram_oe            (sram_oe  ),
    .sram_ce            (sram_ce  ),
    .sram_we            (sram_we  ),
    .sram_ub            (sram_ub  ),
    .sram_lb            (sram_lb  ),
    .sram_addr          (sram_addr),
    .sram_data          (sram_data)
);
always @(posedge sys_clk) begin
    if (~rst_n)
        led_reg <= 7'd0 ;
    else if (rd_data_en )
        led_reg <= rd_data[7:0] ;
end
assign led = led_reg ;
endmodule
```

**SRAM** 驱动模块：对 SRAM 进行写操作时，拉低 WE、CE 信号。

对 SRAM 进行读操作时，拉低 OE、CE 信号。

```
`timescale 1ns / 1ps
`define UD #1

module sram_dri #(
    parameter       clk_frequency = 27'd100_000_000    //sram max 100MHz
//    parameter       data_dbyte  = 12'd1280
)(
    input             clk        ,
    input             rst_n      ,
    input       [16:0]data_dbyte ,
    input             wr_en      ,
```

```
    input        [16:0]wr_addr    ,
    input        [15:0]wr_data    ,
    input         [1:0]wr_lub     ,//write data active low
    input              rd_en      /* synthesis PAP_MARK_DEBUG="true" */,
    input        [16:0]rd_addr    /* synthesis PAP_MARK_DEBUG="true" */,
    output   reg [15:0]rd_data    /* synthesis PAP_MARK_DEBUG="true" */,
    output        reg  rd_data_en /* synthesis PAP_MARK_DEBUG="true" */,
    input         [1:0]rd_lub     ,//read data active low
    output             busy       ,
    output             busy_end   ,
    output        reg  sram_oe    /* synthesis PAP_MARK_DEBUG="true" */,
    output        reg  sram_ce    ,
    output        reg  sram_we    /* synthesis PAP_MARK_DEBUG="true" */,
    output        reg  sram_ub    ,
    output        reg  sram_lb    ,
    output       [16:0]sram_addr  /* synthesis PAP_MARK_DEBUG="true" */,
    inout        [15:0]sram_data  /* synthesis PAP_MARK_DEBUG="true" */
);

reg wr_en_reg1 /* synthesis PAP_MARK_DEBUG="true" */;
reg wr_en_reg2 /* synthesis PAP_MARK_DEBUG="true" */;

wire wr_rise ;

reg [15:0] wr_data_reg1 /* synthesis PAP_MARK_DEBUG="true" */;
reg [15:0] wr_data_reg2 /* synthesis PAP_MARK_DEBUG="true" */;
reg [15:0] wr_data_reg3 /* synthesis PAP_MARK_DEBUG="true" */;
reg [15:0] wr_data_reg4 /* synthesis PAP_MARK_DEBUG="true" */;

reg [16:0] wr_addr_reg1 /* synthesis PAP_MARK_DEBUG="true" */;
reg [16:0] wr_addr_reg2 /* synthesis PAP_MARK_DEBUG="true" */;
reg [16:0] wr_addr_reg3 /* synthesis PAP_MARK_DEBUG="true" */;
reg [16:0] wr_addr_reg4 /* synthesis PAP_MARK_DEBUG="true" */;

reg [1:0] wr_lub_reg1 ;
reg [1:0] wr_lub_reg2 ;
reg [1:0] wr_lub_reg3 ;
reg [1:0] wr_lub_reg4 ;
reg [1:0] rd_lub_reg1 ;
reg [1:0] rd_lub_reg2 ;
reg [1:0] rd_lub_reg3 ;
reg [1:0] rd_lub_reg4 ;

reg rd_en_reg1 ;
```

```
reg rd_en_reg2 ;
reg rd_en_reg3 ;
reg rd_en_reg4 ;

wire rd_rise /* synthesis PAP_MARK_DEBUG="true" */;

reg [16:0] rd_addr_reg1 ;
reg [16:0] rd_addr_reg2 ;
reg [16:0] rd_addr_reg3 ;
reg [16:0] rd_addr_reg4 ;
reg rd_data_en_reg ;

wire [15:0] sram_data_out /* synthesis PAP_MARK_DEBUG="true" */;

reg [16:0] cnt_dbyte /* synthesis PAP_MARK_DEBUG="true" */;
reg flag_end /* synthesis PAP_MARK_DEBUG="true" */;

assign sram_data_out = sram_data ;

assign sram_data = (sram_we == 1'b0  )? wr_data_reg3 : 16'hz ;

assign sram_addr = (sram_we == 1'b0  )? wr_addr_reg3 : rd_addr_reg3 ;

always @(posedge clk ) begin
    if (~rst_n) begin
        wr_en_reg1 <= 1'b0 ;
        wr_en_reg2 <= 1'b0 ;
    end
    else begin
        wr_en_reg1 <= wr_en ;
        wr_en_reg2 <= wr_en_reg1 ;
    end
end

assign wr_rise = ((wr_en_reg1)&&(~wr_en_reg2)) ;

always @(posedge clk ) begin
    if(~rst_n) begin
        wr_data_reg1 <= 16'd0 ;
        wr_data_reg2 <= 16'd0 ;
        wr_data_reg3 <= 16'd0 ;
        wr_data_reg4 <= 16'd0 ;
    end
    else begin
```

```
        wr_data_reg1 <= wr_data ;
        wr_data_reg2 <= wr_data_reg1 ;
        wr_data_reg3 <= wr_data_reg2 ;
        wr_data_reg4 <= wr_data_reg3 ;
    end
end

always @(posedge clk ) begin
    if(~rst_n) begin
        wr_addr_reg1 <= 17'd0 ;
        wr_addr_reg2 <= 17'd0 ;
        wr_addr_reg3 <= 17'd0 ;
        wr_addr_reg4 <= 17'd0 ;
    end
    else begin
        wr_addr_reg1 <= wr_addr ;
        wr_addr_reg2 <= wr_addr_reg1 ;
        wr_addr_reg3 <= wr_addr_reg2 ;
        wr_addr_reg4 <= wr_addr_reg3 ;
    end
end

always @(posedge clk ) begin
    if(~rst_n) begin
        wr_lub_reg1 <= 2'd0 ;
        wr_lub_reg2 <= 2'd0 ;
        wr_lub_reg3 <= 2'd0 ;
        wr_lub_reg4 <= 2'd0 ;
        rd_lub_reg1 <= 2'd0 ;
        rd_lub_reg2 <= 2'd0 ;
        rd_lub_reg3 <= 2'd0 ;
        rd_lub_reg4 <= 2'd0 ;
    end
    else begin
        wr_lub_reg1 <= wr_lub ;
        wr_lub_reg2 <= wr_lub_reg1 ;
        wr_lub_reg3 <= wr_lub_reg2 ;
        wr_lub_reg4 <= wr_lub_reg3 ;
        rd_lub_reg1 <= rd_lub ;
        rd_lub_reg2 <= rd_lub_reg1 ;
        rd_lub_reg3 <= rd_lub_reg2 ;
        rd_lub_reg4 <= rd_lub_reg3 ;
    end
end
```

```
always @(posedge clk ) begin
    if (~rst_n) begin
        rd_en_reg1 <= 1'b0 ;
        rd_en_reg2 <= 1'b0 ;
        rd_en_reg3 <= 1'b0 ;
        rd_en_reg4 <= 1'b0 ;
        rd_data_en <= 1'b0 ;
    end
    else begin
        rd_en_reg1 <= rd_en ;
        rd_en_reg2 <= rd_en_reg1 ;
        rd_en_reg3 <= rd_en_reg2 ;
        rd_en_reg4 <= rd_en_reg3 ;
        rd_data_en_reg <= ~sram_oe ;
        rd_data_en <= rd_data_en_reg ;
    end
end

assign rd_rise = ((rd_en_reg1)&&(~rd_en_reg2)) ;

always @(posedge clk ) begin
    if(~rst_n) begin
        rd_addr_reg1 <= 17'd0 ;
        rd_addr_reg2 <= 17'd0 ;
        rd_addr_reg3 <= 17'd0 ;
        rd_addr_reg4 <= 17'd0 ;
    end
    else begin
        rd_addr_reg1 <= rd_addr ;
        rd_addr_reg2 <= rd_addr_reg1 ;
        rd_addr_reg3 <= rd_addr_reg2 ;
        rd_addr_reg4 <= rd_addr_reg3 ;
    end
end

always @(posedge clk ) begin
    if (~rst_n)
        rd_data <= 16'd0 ;
    else if (rd_data_en_reg)
        rd_data <= sram_data_out ;
    else
        rd_data <= rd_data ;
end
```

```
//--------------------------state-----------------------------------//

reg [3:0]state /* synthesis PAP_MARK_DEBUG="true" */;

parameter idle = 4'b0001 ;
parameter write = 4'b0010 ;
parameter read = 4'b0100 ;
parameter state_end = 4'b1000 ;

always @(posedge clk ) begin
    if (~rst_n)
        state <= idle ;
    else begin case (state)
        idle    : begin
            if (wr_rise)
                state <= write ;
            else if (rd_rise)
                state <= read ;
            else
                state <= idle ;
        end
        write   : begin
            if (flag_end)
                state <= state_end ;
            else
                state <= write ;
        end
        read    : begin
            if (flag_end)
                state <= state_end ;
            else
                state <= read ;
        end
        state_end:begin
            state <= idle ;
        end
        default: state <= idle ;
    endcase end
end

always @(posedge clk ) begin
   if (~rst_n)begin
     sram_oe <= 1'b1 ;
```

```
      sram_ce <= 1'b1 ;
      sram_we <= 1'b1 ;
   end
   else if ((state == write)&&(!flag_end))begin
      sram_oe <= 1'b1 ;
      sram_ce <= 1'b0 ;
      sram_we <= 1'b0 ;
   end
   else if ((state == read)&&(!flag_end))begin
      sram_oe <= 1'b0 ;
      sram_ce <= 1'b0 ;
      sram_we <= 1'b1 ;
   end
   else begin
      sram_oe <= 1'b1 ;
      sram_ce <= 1'b1 ;
      sram_we <= 1'b1 ;
   end
end

always @(posedge clk ) begin
   if (~rst_n)begin
      sram_lb <= 1'b1 ;
      sram_ub <= 1'b1 ;
   end
   else if ((state == write)&&(!flag_end))begin
      sram_lb <= wr_lub_reg4[0] ;
      sram_ub <= wr_lub_reg4[1] ;
   end
   else if ((state == read)&&(!flag_end))begin
      sram_lb <= rd_lub_reg4[0] ;
      sram_ub <= rd_lub_reg4[1] ;
   end
   else begin
      sram_lb <= 1'b1 ;
      sram_ub <= 1'b1 ;
   end
end

always @(posedge clk ) begin
    if (~rst_n)
        cnt_dbyte <= 17'd0 ;
    else if (flag_end)
        cnt_dbyte <= 17'd0 ;
```

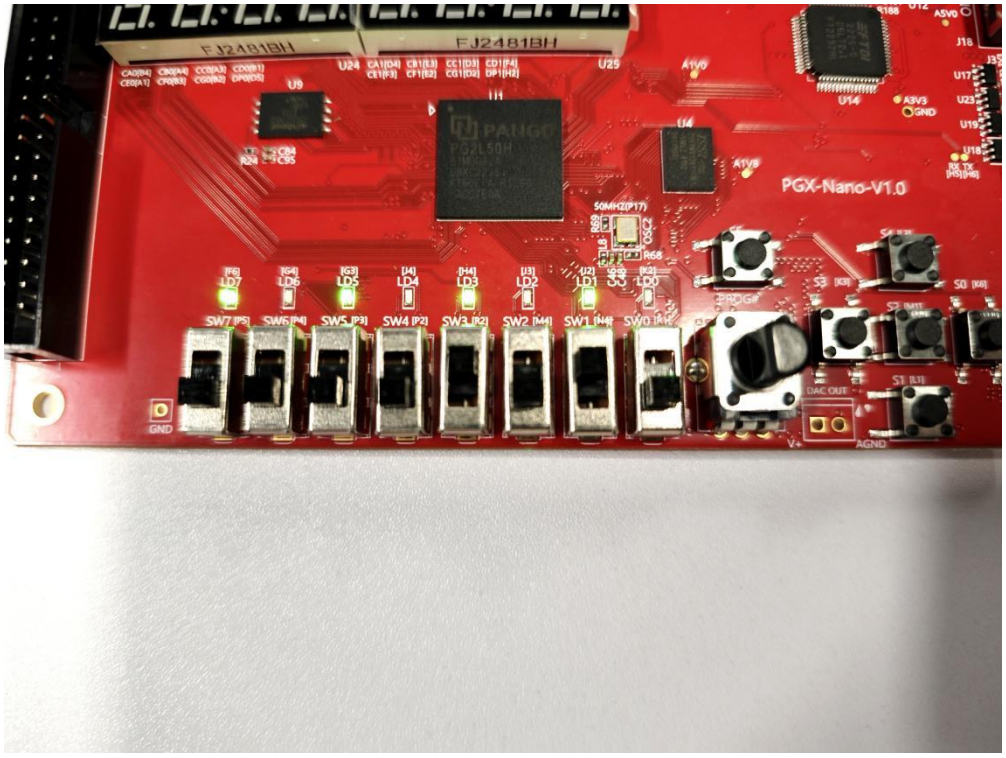
```
    else if (((state == write)||(state == read))&&(~sram_ce))
        cnt_dbyte <= cnt_dbyte + 17'd1 ;
    else
        cnt_dbyte <= 17'd0 ;
end

always @(posedge clk ) begin
    if (~rst_n)
        flag_end <= 1'b0 ;
    else if (cnt_dbyte == data_dbyte - 17'd2)
        flag_end <= 1'b1 ;
    else
        flag_end <= 1'b0 ;
end

assign busy = (state != idle)? 1'b1 : 1'b0 ;
assign busy_end = (state == state_end)? 1'b1 : 1'b0 ;
endmodule
```

按键消抖模块：不再过多赘述。

## 10.5 实验现象

按下 S1 按键在 SRAM 的 17’h00000 地址位置写入 16bit 数据 0000_0000_1010_1010；再按下 S1 按键读出 17’h00000 地址位置的 16bit 数据 0000_0000_1010_1010；LED0、LED2、LED4、LED6 灯灭、LED1、LED3、LED5、LED7 灯亮。

按下 S3 使 LED 恢复为全灭状态。


FJ2481BH
FJ2481BH
PGX-Nano-V1.0
LD7
LD6
LD5
LD4
LD3
LD2
LD1
LD0
SW7
SW6
SW5
SW4
SW3
SW2
SW1
SW0
GND
DAC OUT
V+
AGND