# Project 3: 存储器与串口实验

## 实验目的

1. 熟悉THINPAD硬件环境，特别是基本总线
2. 学习使用SRAM，测试SRAM功能是否正确
3. 学习使用THINPAD的UART接口，测试UART功能是否正确
4. 进一步锻炼硬件调试和测试的能力

## 实验任务

本次实验要求将数据从串口输入，存储到基本内存，然后存储到扩展内存，最终发送到串口显示。

我们的具体实现为一个状态机，它从串口读入数据，加一后写入基本内存，再从基本内存读出，再加一后写入扩展内存，再从扩展内存读出，再加一后写入串口，上位机可以用PuTTY等串口终端软件接收数据并显示。

## 实验结果

经过实际硬件测试，我们的状态机可以配合SRAM以及UART正常工作。

同时，我们确认了SRAM以及UART的功能的正确性。

## 代码注解

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

entity sram_uart is
    port
    (
        CLK: in std_logic;
        KEY0: in std_logic;    -- 锁存输入地址的按钮
        nRST: in std_logic;
        nInputSW: in std_logic_vector(WORD_WIDTH - 1 downto 0);    -- 输入的地址
        LEDOutput: out std_logic_vector(WORD_WIDTH - 1 downto 0);  -- 将data显示到LED

        -- 基本总线（系统总线）
        SYSBUS_ADDR: out std_logic_vector(ADDR_WIDTH - 1 downto 0);
        SYSBUS_DQ: inout std_logic_vector(WORD_WIDTH - 1 downto 0);
        -- SRAM1的控制信号
```

```
        RAM1_nWE: out std_logic;
        RAM1_nOE: out std_logic;
        RAM1_nCE: out std_logic;

        -- 扩展总线
        EXTBUS_ADDR: out std_logic_vector(ADDR_WIDTH - 1 downto 0);
        EXTBUS_DQ: inout std_logic_vector(WORD_WIDTH - 1 downto 0);
        -- SRAM2的控制信号
        RAM2_nWE: out std_logic;
        RAM2_nOE: out std_logic;
        RAM2_nCE: out std_logic;

        -- UART的控制信号（读端）
        UART_nRE: out std_logic;
        UART_READY: in std_logic;
        -- UART的控制信号（写端）
        UART_nWE: out std_logic;
        UART_TBRE: in std_logic;
        UART_TSRE: in std_logic
    );
end;

architecture behavioral of sram_uart is
    type state_t is (st_init, st_read_init, st_read_wait, st_read,
                     st_write_sram1, st_clear_bus_11, st_clear_bus_12,
st_read_sram1, st_read_sram_wait1,
                     st_write_sram2, st_clear_bus_21, st_clear_bus_22,
st_read_sram2, st_read_sram_wait2,
                     st_write_init, st_write, st_write_wait);
    signal current_state: state_t;

    signal InputSW: std_logic_vector(WORD_WIDTH - 1 downto 0);
    signal RST: std_logic;
    signal SYSBUS_DIN, SYSBUS_DOUT: std_logic_vector(WORD_WIDTH - 1 downto
0);
    signal SYSBUS_DEN: std_logic;
    signal EXTBUS_DIN, EXTBUS_DOUT: std_logic_vector(WORD_WIDTH - 1 downto
0);
    signal EXTBUS_DEN: std_logic;
    signal addr: std_logic_vector(ADDR_WIDTH - 1 downto 0);
    signal data: std_logic_vector(WORD_WIDTH - 1 downto 0);
begin
    RST <= not nRST;
    InputSW <= not nInputSW;
    SYSBUS_DQ <= SYSBUS_DOUT when SYSBUS_DEN = '1' else (others => 'Z');
    SYSBUS_DIN <= SYSBUS_DQ;
    EXTBUS_DQ <= EXTBUS_DOUT when EXTBUS_DEN = '1' else (others => 'Z');
    EXTBUS_DIN <= EXTBUS_DQ;
    RAM1_nCE <= '0'; -- 将SRAM的片选都选中，下面用/WE、/OE控制
```

```
    RAM2_nCE <= '0';
    LEDOutput <= data;

    -- 地址触发器
    process(KEY0, RST)
    begin
        if RST = '1' then
            addr <= (others => '0');
        elsif rising_edge(KEY0) then
            addr <= "00" & InputSW;
        end if;
    end process;

    process(CLK, RST)
    begin
        if RST = '1' then
            RAM1_nWE <= '1'; -- 关闭SRAM
            RAM1_nOE <= '1';
            UART_nRE <= '1'; -- 关闭UART
            SYSBUS_ADDR <= (others =>'0');
            SYSBUS_DEN <= '0'; -- 不驱动系统总线
            SYSBUS_DOUT <= (others => '0');
            EXTBUS_ADDR <= (others =>'0');
            EXTBUS_DEN <= '0'; -- 不驱动扩展总线
            EXTBUS_DOUT <= (others => '0');
            current_state <= st_init;
        elsif rising_edge(CLK) then
            case current_state is
                when st_init =>
                    current_state <= st_read_init;
                when st_read_init =>
                    RAM1_nWE <= '1';
                    RAM1_nOE <= '1';
                    UART_nRE <= '1';
                    SYSBUS_DEN <= '0';
                    current_state <= st_read_wait;
                when st_read_wait => -- 等待，直到UART有数据可读取
                    if UART_READY = '1' then
                        UART_nRE <= '0';
                        current_state <= st_read;
                    end if;
                when st_read => -- 从UART读取数据
                    data <= SYSBUS_DIN;
                    UART_nRE <= '1';
                    current_state <= st_write_sram1;
                when st_write_sram1 => -- 写入SRAM1
                    RAM1_nWE <= '0';
                    RAM1_nOE <= '1';
                    SYSBUS_ADDR <= addr;
```

```
            SYSBUS_DEN <= '1';
            SYSBUS_DOUT <= data + 1;
            current_state <= st_clear_bus_11;
        when st_clear_bus_11 => -- 该状态解释见下
            RAM1_nWE <= '1';
            SYSBUS_DOUT <= x"05AF";
            current_state <= st_clear_bus_12;
        when st_clear_bus_12 => -- 该状态解释见下
            SYSBUS_DEN <= '0';
            current_state <= st_read_sram1;
        when st_read_sram1 => -- 发送地址给SRAM1
            RAM1_nWE <= '1';
            RAM1_nOE <= '0';
            SYSBUS_ADDR <= addr;
            SYSBUS_DEN <= '0';
            current_state <= st_read_sram_wait1;
        when st_read_sram_wait1 => -- 读取SRAM1
            RAM1_nOE <= '1';
            SYSBUS_DEN <= '0';
            data <= SYSBUS_DIN;
            current_state <= st_write_sram2;
        when st_write_sram2 => -- 写入SRAM2
            RAM2_nWE <= '0';
            RAM2_nOE <= '1';
            EXTBUS_ADDR <= addr;
            EXTBUS_DEN <= '1';
            EXTBUS_DOUT <= data + 1;
            current_state <= st_clear_bus_21;
        when st_clear_bus_21 => -- 该状态解释见下
            RAM2_nWE <= '1';
            EXTBUS_DOUT <= x"FA50";
            current_state <= st_clear_bus_22;
        when st_clear_bus_22 => -- 该状态解释见下
            EXTBUS_DEN <= '0';
            current_state <= st_read_sram2;
        when st_read_sram2 => -- 发送地址给SRAM1
            RAM2_nWE <= '1';
            RAM2_nOE <= '0';
            EXTBUS_ADDR <= addr;
            EXTBUS_DEN <= '0';
            current_state <= st_read_sram_wait2;
        when st_read_sram_wait2 => -- 读取SRAM2
            RAM2_nOE <= '1';
            EXTBUS_DEN <= '0';
            data <= EXTBUS_DIN;
            current_state <= st_write_init;
        when st_write_init => -- 写入串口
            SYSBUS_DEN <= '1';
            UART_nWE <= '1';
```

```
                    SYSBUS_DOUT <= data + 1;
                    current_state <= st_write;
                when st_write =>
                    UART_nWE <= '0';
                    current_state <= st_write_wait;
                when st_write_wait => -- 等待直到串口发送完成
                    UART_nWE <= '1';
                    if UART_TBRE = '1' and UART_TSRE = '1' then
                        current_state <= st_read_init;
                    end if;
                when others =>
                    current_state <= st_init;
            end case;
        end if;
    end process;
end;
```

# SRAM测试问题

在我们的测试中，我们发现FPGA驱动完总线后，再将FPGA设置为高阻态，FPGA刚刚向总线写的值会留在总线上，如果此时没有其他设备驱动总线，FPGA再尝试读取总线的话，会读出相同的值。因此，如果SRAM损坏或者`/CE`或`/OE`信号设置错误而导致SRAM根本没有驱动总线，是检测不出错误的。为了解决这个问题，在FPGA向总线写入正确的值后，需要将SRAM设置为不写入（`/WE = '1'`），然后让FPGA向总线写入一些错误的值，之后再读取。

这直接反映在几个“奇怪”的状态
——`st_clear_bus_11`、`st_clear_bus_12`、`st_clear_bus_21`和`st_clear_bus_22`中。

# 跨时钟域问题

我们注意到，UART的三个输入到FPGA的信号，`UART_READY`、`UART_TBRE`以及`UART_TSRE`，所在时钟域和FPGA的时钟不同，直接输入到FPGA的触发器中可能会产生跨时钟域相关问题，导致触发器进入亚稳态，从而导致状态机异常甚至卡死。为了解决这个问题，可以将这三个输入信号用FPGA的时钟延迟两个周期再传给需要的地方。不过，由于本次实验我们考虑到演示的方便性，将FPGA的时钟设置为手动控制的微动开关，这个问题不是十分明显，所以代码中也就没有处理。

# 思考题

**SRAM读写特点**

SRAM读取的时序要求更加严格，因为读取操作需要FPGA先将地址发送到总线，等待SRAM内部逻辑稳定之后，再从总线中把数据取出。假设FPGA向总线发送地址的时间为$t_{addr}$，SRAM最小读访问时间为$t_{read}$，FPGA读取总线的时间为$t_{in}$，实际留给SRAM进行读访问的时间为$t_{real}$，则一次操作的时间$t$满足：

$$t = t_{addr} + t_{real} + t_{in} \geq t_{addr} + t_{read} + t_{in}$$

写操作稍好一些，因为FPGA只需将地址和数据同时发送到SRAM，等待SRAM稳定后即可。假设FPGA向总线发送地址的时间为$t_{addr}$，FPGA将数据发送到总线的时间为$t_{out}$，SRAM最小写访问时间为$t_{write}$，实际留给SRAM进行写访问的时间为$t_{real}$，则一次操作的时间$t$满足：

$$t = \max\{t_{addr}, t_{out}\} + t_{real} \geq \max\{t_{addr}, t_{out}\} + t_{write}$$

顺便一提，THINPAD使用的SRAM IS61LV25616-10TI，$t_{read} = t_{write} = 10\text{ns}$。

**什么是高阻态?**

将IO端口设置为高阻态使得IO端口不驱动总线，就相当于在总线上连了一个阻值很大的电阻。

这对于FPGA、SRAM和UART都成立。

**如何将SRAM1和SRAM2作为32位的存储器使用?**

只需要将控制信号`/CE`、`/OE`、`/WE`以及地址连在一起，两条数据总线并置、两套`/UB`与`/LB`信号并置即可。