西安电子科技大学
XIDIAN UNIVERSITY

基于 FPGA 的数字系统设计

实验报告

实验名称：LAB4 Chipscope 调试实验

任课教师：沈沛意老师

学号姓名：

提交日期：

一、 实验环境

win11

ISE14.7, Modelsim SE-64 10.4

二、 实验介绍

本实验介绍如何加入 ILA/ICON 核到设计,并实现片上验证功能。

三、 实验目标

- 在 ISE 中创建 Chipscope-Pro 源;
- 使用 Chipscope-Pro 创建 ILA 和 ICON 核,并将其加入到 PicoBlaze 设计中;
- 在 Chipscope Analyzer 中定义触发条件;
- 下载 bitstream,硬件运行设计;
- 查看 Chipscope Analyzer 的波形显示,实现片上验证。

四、 实验过程以及步骤

本实验主要包含两个部分:更改针对 PicoBlaze 的程序应用;使用 Chipscope-Pro 实现

1. 创建一个新的 Chipscope-Pro 源
    - ✓ 打开 ISE 集成环境
    - ✓ 在 ISE 中,选择菜单栏中的 File→Open Project,打开已有工程;:chipscope.ise,
    - ✓ 选择 Project→New Source,打开新建源对话框,单击 Chipscope Definition and Connection,输入名称 loopback_cs,再单击 Next 按钮.

- ✓ 选择 loopback 作为源,单击 Next 按钮,然后单击 Finish 按钮。一个 Chipscope-Pro 源将加入到 Sources 窗口中

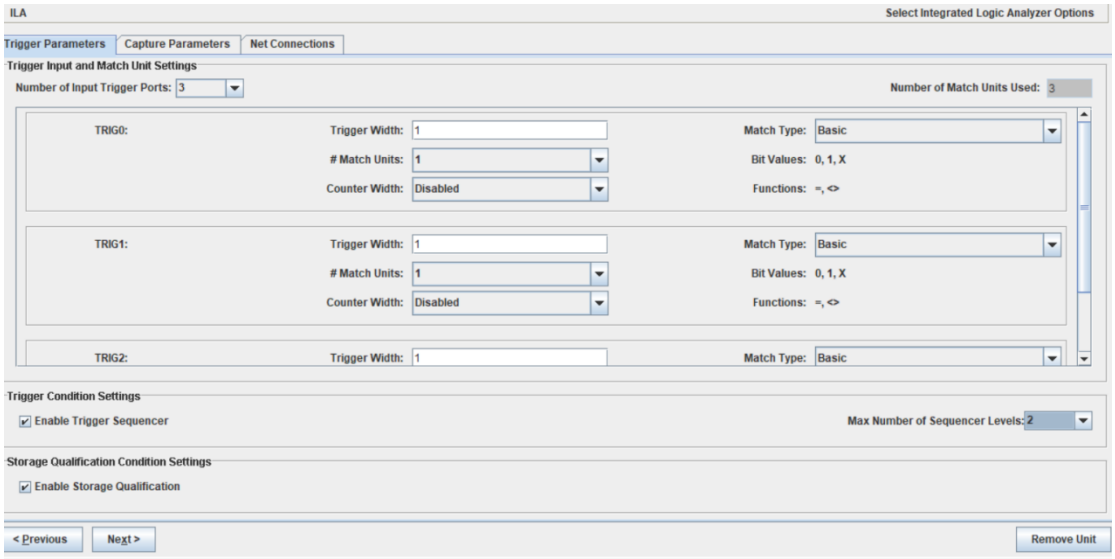
2. 更改 ILA 核的参数和连接

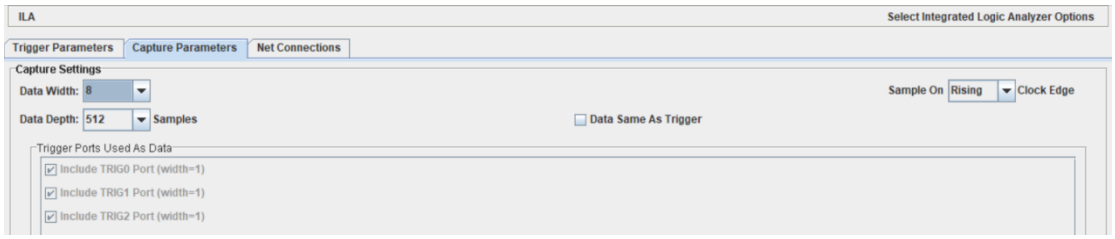
使用 Chipscope-Pro Core Inserter 加入一个 ICON 和 ILA 核到设计网表,将累加器的输出与 ILA 核的触发和输入数据端口连接。

- ✓ 在 Sources 窗口中,双击 loopback_cs.cdc.
- ✓ 单击 Next 按钮，进行 LIA 的设置：

  Number of input Trigger Ports：3——每一个 Trigger 的 Trigger Width 为 1，Match Basic 为 Basic

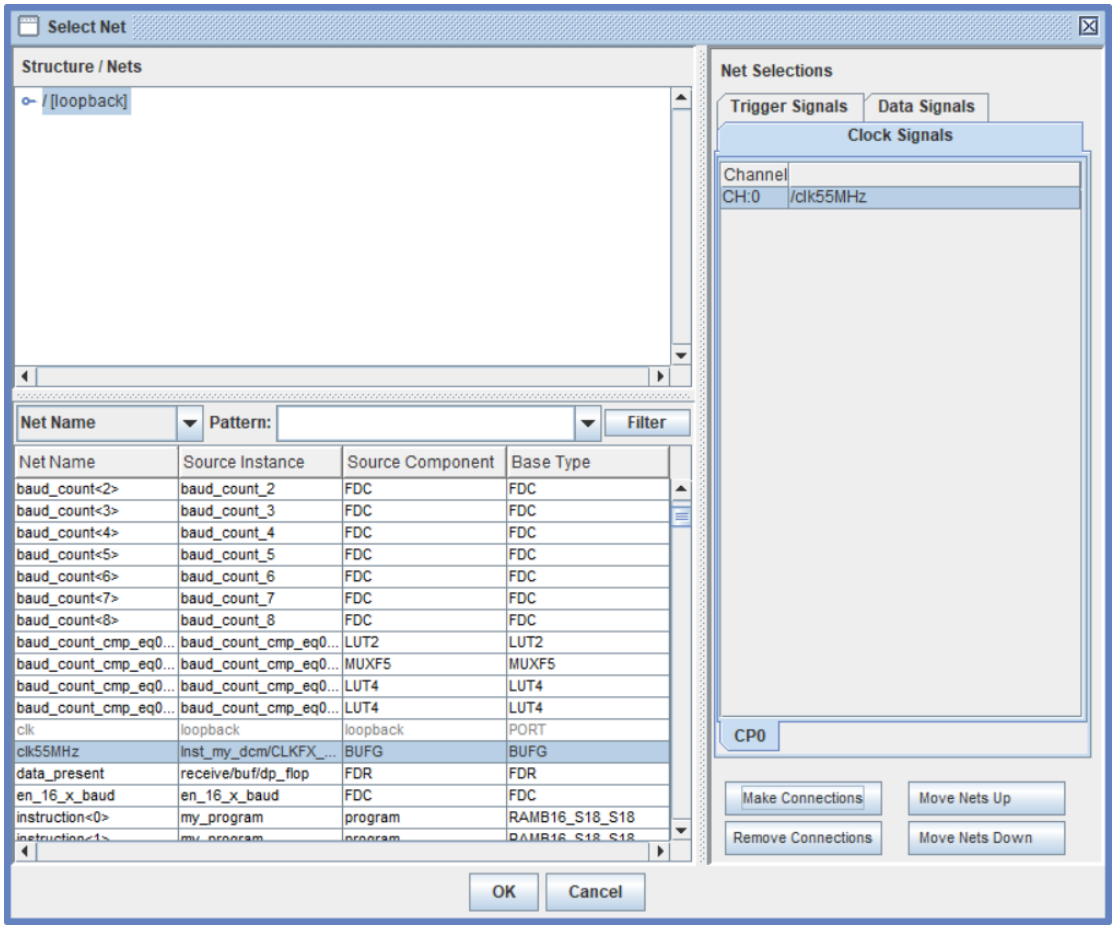
  Data Width 为 8



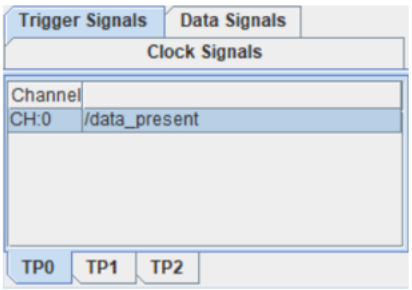


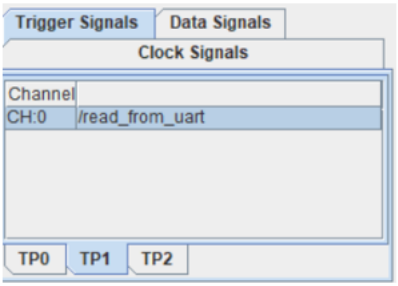
- ✓ 点击 Net Connections 标签下的 Modify Connections，选中右上的 Clock Signals 标签，选中左下角的 clk55MHz 时钟信号然后单

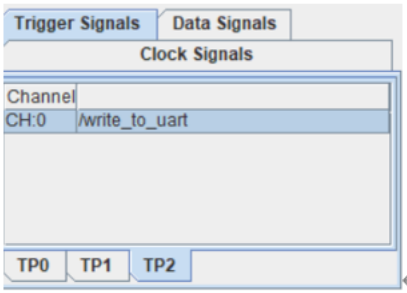
击 Make Connections，如下图所示；



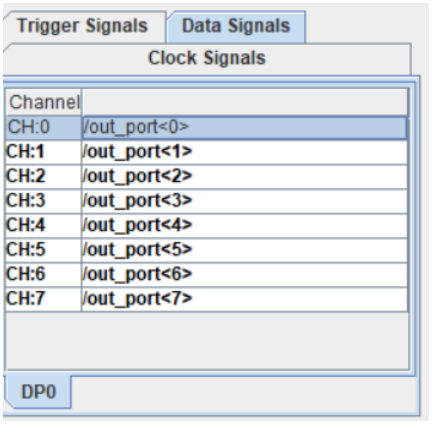
✓ 选择 Trigger Signals 标签，如下图所示设置





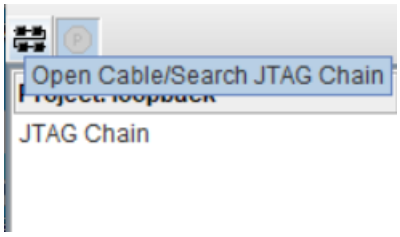


- ✓ 选中 Data Signals 标签，如下图设置，然后点击 OK，返回之前的界面后单击 Return to Project Navigator 并保存文件;



3. 定义 Chipscope 分析器选项

- ✓ 选中顶层文件 loopback.vhd, 双击 Analyze Design Using Chipscope。
- ✓ 连接 JTAG 下载线,给开发板上电。
- ✓ 单击 Open Cable/Search JTAG Chain 按钮,如图所示。



- ✓ 点击左上角按钮连接开发板，选择 xc3s500e 进行配置，单击 Select New File，导入 loopback.bit 文件;
- ✓ 选择 File->Import，导入 loopback_cs.cdc 文件
- ✓ 定义 Match Units

  单击 Trigger Condition Equation 的下方，在 Sequencer 标签下选择以下创建表达式 M0->M1，然后点击 OK

  查看 Storage Qualification 的旁边，选择 AND Equation，然后点

击 OK

设置缓冲深度为 16

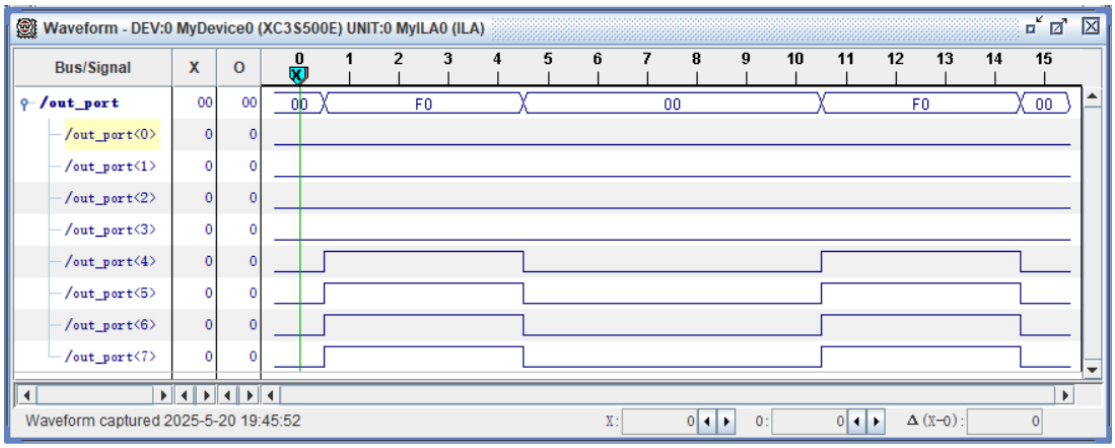
✓ 点击 T！按键出结果

五、 实验总结

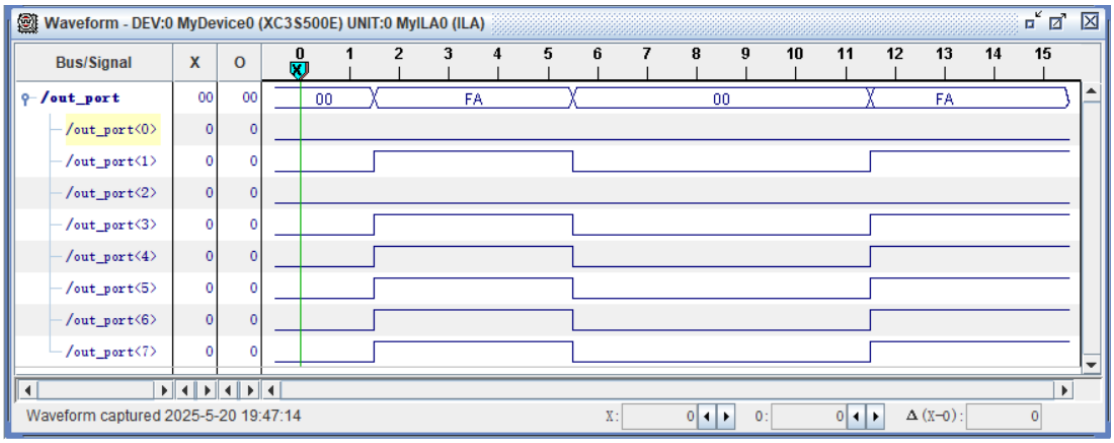
在本实验中,将一个 ICON 和 ILA 核加入到 PicoBlaze 设计,在 Chipscope Analyzer 内设置触发条件,完成片上验证,并在 Chipscope-Pro Analyzer 内分析波形。PicoBlaze（kcpsm3）从 program ROM 读取指令执行，通过端口与外部设备交互，在 I/O 控制中；通过端口 0x00 读取 8 位开关状态

实验中的波形图:

当开关全部关闭时，可以看见，输出的 0-3 均为低电平



第二个、第四个开关打开时，可以看到输出的 1,3 端口出现信号高电平



六、 实验代码

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;          -- 标准逻辑类型
use IEEE.STD_LOGIC_ARITH.ALL;         -- 算术运算
use IEEE.STD_LOGIC_UNSIGNED.ALL; -- 无符号数处理


-- 实体声明：定义模块端口
entity loopback is
    Port (
        clk         : in    std_logic;    -- 主时钟输入
        rst         : in    std_logic;    -- 复位信号（高有效）
        lock        : out std_logic;    -- 时钟锁定信号（来自 DCM）
        leds        : out std_logic_vector(7 downto 0);    -- LED 输出
        switches : in    std_logic_vector(7 downto 0);     -- 开关输入
        rs232_rx : in    std_logic;    -- UART 接收数据线
        rs232_tx : out std_logic      -- UART 发送数据线
    );
end loopback;


architecture Behavioral of loopback is


    -- 程序存储器（双端口 ROM）组件声明
```

```
        component program
        port (
            clka    : in    std_logic;
            addra : in    std_logic_vector(9 downto 0);
            douta : out std_logic_vector(17 downto 0);   -- 指令总线（18
位）
            clkb    : in    std_logic;
            addrb : in    std_logic_vector(9 downto 0);
            doutb : out std_logic_vector(17 downto 0)
        );
        end component;

        -- PicoBlaze 微处理器组件声明
        component kcpsm3
        port(
            instruction     : in    std_logic_vector(17 downto 0);
            in_port           : in    std_logic_vector(7 downto 0);
            interrupt        : in    std_logic;
            reset              : in    std_logic;
            clk                : in    std_logic;
            address          : out std_logic_vector(9 downto 0);   -- 指令
地址
```

```
        port_id          : out std_logic_vector(7 downto 0);   -- I/O 端口 ID
        write_strobe     : out std_logic;   -- 写使能
        out_port         : out std_logic_vector(7 downto 0); -- 输出数据
        read_strobe      : out std_logic;   -- 读使能
        interrupt_ack : out std_logic
    );
    end component;

    -- UART 接收器组件
    component uart_rx
    port(
        serial_in       : in    std_logic;
        read_buffer     : in    std_logic;
        reset_buffer : in    std_logic;
        en_16_x_baud : in    std_logic;
        clk             : in    std_logic;
        data_out        : out std_logic_vector(7 downto 0);
        buffer_data_present : out std_logic;   -- 数据存在标志
        buffer_full     : out std_logic;            -- 缓冲区满
        buffer_half_full : out std_logic
```

```
);
end component;

-- UART 发送器组件
component uart_tx
port(
    data_in          : in    std_logic_vector(7 downto 0);
    write_buffer : in    std_logic;
    reset_buffer : in    std_logic;
    en_16_x_baud : in    std_logic;
    clk                  : in    std_logic;
    serial_out     : out std_logic;
    buffer_full    : out std_logic;              -- 发送缓冲区满
    buffer_half_full : out std_logic
);
end component;

-- 时钟管理模块（DCM）组件
component my_dcm
port(
    CLKIN_IN             : in    std_logic;
    RST_IN               : in    std_logic;
```

```
            CLKFX_OUT           : out std_logic;        -- 生成的 55MHz 时
钟
            CLKIN_IBUFG_OUT : out std_logic;
            CLK0_OUT            : out std_logic;
            LOCKED_OUT          : out std_logic          -- 时钟锁定信号
        );
        end component;

        -- 内部信号定义
        signal address          : std_logic_vector(9 downto 0);     -- 指令地
址
        signal instruction    : std_logic_vector(17 downto 0);   -- 指令内容
        signal port_id          : std_logic_vector(7 downto 0);     -- 端口 ID
        signal write_strobe : std_logic;                  -- 写使能
        signal in_port          : std_logic_vector(7 downto 0);     -- 输入数据
总线
        signal out_port         : std_logic_vector(7 downto 0);     -- 输出数据
总线
        signal read_strobe    : std_logic;                -- 读使能

        -- UART 相关信号
        signal baud_count       : std_logic_vector(8 downto 0) := (others =>
```

```
'0');   -- 波特率计数器
        signal en_16_x_baud   : std_logic;              -- 16 倍波特率使能
        signal read_from_uart: std_logic;               -- UART 读触发
        signal rx_data          : std_logic_vector(7 downto 0); -- 接收数据
        signal data_present   : std_logic;              -- 接收数据存在标志
        signal write_to_uart : std_logic;               -- UART 写触发
        signal write_to_leds : std_logic;               -- LED 写触发
        signal buffer_full    : std_logic;              -- 发送缓冲区满标志

        -- 时钟信号
        signal clk55MHz : std_logic;   -- DCM 生成的 55MHz 时钟

    begin

        -- 实例化 PicoBlaze 微处理器
        my_kcpsm3: kcpsm3
        port map (
            address          => address,
            instruction     => instruction,
            port_id          => port_id,
            write_strobe    => write_strobe,
            out_port         => out_port,
```

```
        read_strobe     => read_strobe,
        in_port         => in_port,
        interrupt       => '0',         -- 未使用中断
        interrupt_ack => open,
        reset           => rst,
        clk             => clk55MHz   -- 使用 55MHz 时钟
);

-- 实例化程序存储器（指令 ROM）
my_program : program
port map (
        clka   => clk55MHz,
        addra => address,
        douta => instruction,
        clkb   => '0',                  -- 端口 B 未使用
        addrb => (others => '0'),
        doutb => open
);

-- 实例化时钟管理模块（DCM）
Inst_my_dcm: my_dcm
port map (
```

```
        CLKIN_IN        => clk,         -- 输入时钟（外部）
        RST_IN          => rst,
        CLKFX_OUT       => clk55MHz,    -- 输出 55MHz 时钟
        CLKIN_IBUFG_OUT => open,
        CLK0_OUT        => open,
        LOCKED_OUT      => lock         -- 时钟锁定状态输出
    );

    -- 波特率生成进程（16 倍波特率）
    -- 55MHz 时钟下，9600 波特率对应的计数值：55e6 / (16 * 9600) ≈ 358
    baudgen: process (clk55MHz, rst)
    begin
        if rst = '1' then
            baud_count <= (others => '0');
            en_16_x_baud <= '0';
        elsif rising_edge(clk55MHz) then
            if baud_count = 358 then    -- 0x166 = 358
                baud_count <= (others => '0');
                en_16_x_baud <= '1';    -- 每 358 周期产生一个脉冲
            else
                baud_count <= baud_count + 1;
```

```
                    en_16_x_baud <= '0';
                end if;
            end if;
        end process;

        -- LED 输出控制进程
        process (clk55MHz, rst)
        begin
            if rising_edge(clk55MHz) then
                if rst = '1' then
                    leds <= (others => '0');    -- 复位时 LED 全灭
                elsif write_to_leds = '1' then
                    leds <= out_port;              -- 将输出端口数据写入
LED
                end if;
            end if;
        end process;

        -- UART 发送控制逻辑
        write_to_uart <= write_strobe and port_id(0) and port_id(1); -- 端
口 0x03 触发发送
        transmit: uart_tx
```

```
port map (
        data_in          => out_port,
        write_buffer => write_to_uart,
        reset_buffer => rst,
        en_16_x_baud => en_16_x_baud,
        serial_out     => rs232_tx,
        buffer_full    => open,
        buffer_half_full => open,
        clk                 => clk55MHz
);

-- 输入端口多路选择器
process (clk55MHz, rst)
begin
        if rising_edge(clk55MHz) then
                if rst = '1' then
                        in_port <= (others => '0');
                        read_from_uart <= '0';
                else
                        case port_id is
                                when X"00" => in_port <= switches;          -- 读
```

取开关状态

```
                    when X"02" => in_port <= rx_data;           -- 读
取 UART 接收数据
                    when X"04" => in_port <= "0000000" &
data_present;  -- 数据存在标志
                    when X"05" => in_port <= "0000000" &
buffer_full;   -- 发送缓冲区满
                    when others => in_port <= (others => '0');
                end case;
            end if;
            read_from_uart <= read_strobe and port_id(2);  -- 端口
0x04/0x05 读触发
        end if;
    end process;

    -- UART 接收器实例化
    receive: uart_rx
    port map (
        serial_in       => rs232_rx,
        data_out        => rx_data,
        read_buffer     => read_from_uart,
        reset_buffer => rst,
        en_16_x_baud => en_16_x_baud,
```

```
            buffer_data_present => data_present,
            buffer_full    => open,
            buffer_half_full => open,
            clk                => clk55MHz
        );
end Behavioral;
```