



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：LAB3 综合技巧实验

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

Spartan 3E 开发板上有 8 个并排防止的发光二极管 LED7~LED0。实验要求使用通过向板子载入不同的比特流文件，观察 LED 灯的变化。比特流文件将使用已有工程 snyth_lab。

三、 实验目标

在工程 snyth_lab 中

使用 Xilinx Constraints Editor 输入全局时钟约束;

查看 Post-Map Static Timing Report 报告,检查时钟约束是否可实现;

查看 Post-Place & Route Static Timing Report 报告,确定每个时钟约束的最长路径

同时使用该工程，通过向板子载入不同的比特流文件（不同的综合条件下产生的不同文件），观察 LED 灯的变化。

练习综合技巧，并了解 LED 灯亮灯情况不同的背后的原理。

四、 实验过程以及步骤

本实验将利用已有工程 snyth_lab 其中有一个简单的嵌入式系统并输入位置和全局时钟约束，通过向板子载入该项目不同综合条件下产生不同的比特流文件，来观察 LED 灯的变化。当没更改综合选项时，LED7~LED4 常亮，而剩下 4 个灯可以通过旁边四个开关 进行控制。当更改完综合选项后再载入比特流时，LED7~LED4 只有两个灯亮， 而其余四个等还是可以通过四个开关进行控制

1. 启动 ISE 打开已有的工程

- ✓ 打开 ISE 集成环境
- ✓ 选择 File->Open Project, 在以下路径中打开。在 ISE14.7 默认打开 xise 工程文件, 需要在打开文件时, 更改文件类型才能找到工程文件
- ✓ 打开工程之后, 会需求 PROGRAM.VHD 文件, 更改文件路径位于 Assembler 文件夹中

2. 综合与实现

- ✓ 双击 GenerateProgramming File 生成比特流文件

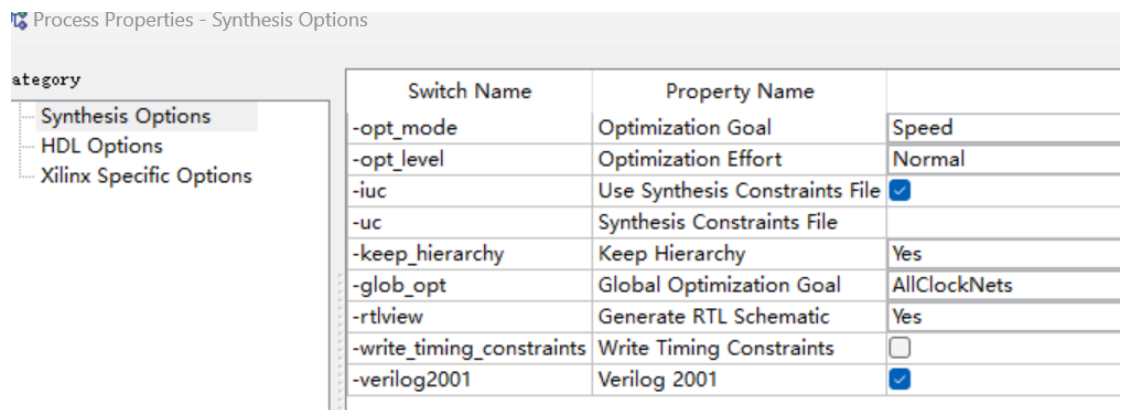
3. 器件配置

- ✓ 连接并启动开发板
- ✓ 在工程管理区域内选择 ConfigureTargetDevice
- ✓ 当出现对话框时, 选择 OK, 启动 iMPACTGUI
- ✓ 双击 BoundaryScan
- ✓ 在弹出的对话框中选择 yes, 并指定之前生成的比特流文件 loopback.bit
- ✓ 之后在弹出的对话框中询问是否添加 FlashPROM 配置文件, 选择 no
- ✓ 随后弹出两个对话框, 询问是否向开发板上的 CPLD 芯片和 Flash 芯片添加配置文件, 不需要添加, 选择 Bypass
- ✓ 在芯片上点击邮件, 选择 Program, 配置完成后, 画面上会出现

Program Succeeded 字样

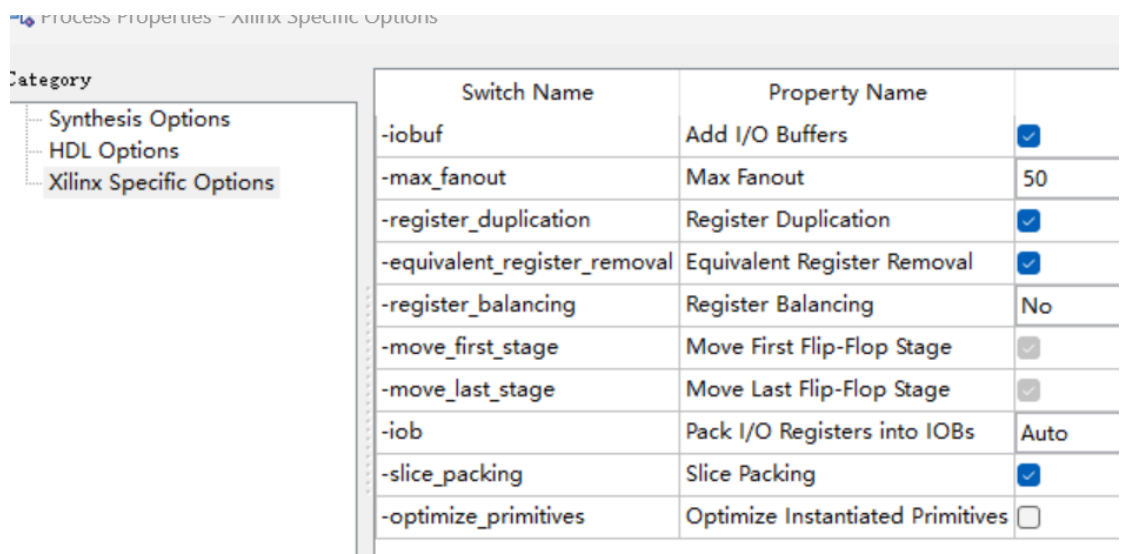
- ✓ 此时可以观察到前四个 LED 灯常亮，后四个 LED 等可以通过四个开关进行控制
- ✓ 更改综合选项

右键 Synthesize-XST->Process Properties->Keep Hierarchy，选择 Yes



Switch Name	Property Name	
-opt_mode	Optimization Goal	Speed
-opt_level	Optimization Effort	Normal
-iuc	Use Synthesis Constraints File	<input checked="" type="checkbox"/>
-uc	Synthesis Constraints File	
-keep_hierarchy	Keep Hierarchy	Yes
-glob_opt	Global Optimization Goal	AllClockNets
-rtlview	Generate RTL Schematic	Yes
-write_timing_constraints	Write Timing Constraints	<input type="checkbox"/>
-verilog2001	Verilog 2001	<input checked="" type="checkbox"/>

右键 Synthesize-XST->Process Properties->Xilinx Specific Options->Max Fanout 参数改为 50



Switch Name	Property Name	
-iobuf	Add I/O Buffers	<input checked="" type="checkbox"/>
-max_fanout	Max Fanout	50
-register_duplication	Register Duplication	<input checked="" type="checkbox"/>
-equivalent_register_removal	Equivalent Register Removal	<input checked="" type="checkbox"/>
-register_balancing	Register Balancing	No
-move_first_stage	Move First Flip-Flop Stage	<input checked="" type="checkbox"/>
-move_last_stage	Move Last Flip-Flop Stage	<input checked="" type="checkbox"/>
-iob	Pack I/O Registers into IOBs	Auto
-slice_packing	Slice Packing	<input checked="" type="checkbox"/>
-optimize_primitives	Optimize Instantiated Primitives	<input type="checkbox"/>

- ✓ 重新双击 Generate Programming File 生成比特流文件

- ✓ 按照之前的步骤点击 ConfigureTargetDevice, 并且指定新生成的比特流文件。
- ✓ 观察此时板子上的 LED 灯, 会发现前四个灯中只有两个常亮, 后四个灯依然可以通过四个开关进行控制。

五、 实验总结

通过本次实验, 观察到开发板在导入初始生成的比特流文件时, LED7~LED4 四个灯全亮, 其余四个灯可以通过四个开关控制。而在进行修改综合选项之后, 再将新生成的比特流文件导入, LED7~LED4 中只有两个灯常量, 而其余四个灯可以通过四个开关进行控制。

根据我查阅的资料内容, 出现上面这样修改了综合选项后出现的现象, 是因为修改综合选项 Keep Hierarchy = Yes 会禁止跨模块优化, 导致 LED 控制路径延迟增加, 再加上 LED 控制信号通常具有高扇出特性 (需驱动多个物理 LED)。原设置 Max Fanout=500 允许单信号直接驱动所有 LED, 而修改为 50 后, 综合器会自动插入缓冲器分担负载, 这里插入的缓冲器就会引入额外延迟, 可能使信号无法在时钟有效沿到达 LED, 此时信号会因时序违例被综合器强制插入锁存器, 锁存了复位后的初始值, 出现了 LED7~LED4 中只有两个灯常量的现象。

六、 实验代码

- 文件名: loopback.vhd
- 功能描述: 基于 PicoBlaze 处理器的 UART 回环测试系统
- 主要功能:
 - 1. 通过 UART 接收数据并回传 (Loopback)
 - 2. 通过开关控制 LED 显示
 - 3. 使用 DCM 模块生成 55MHz 系统时钟
- 硬件接口:
 - - RS232 串口 (接收 rx 和发送 tx)
 - - 8 位拨码开关输入
 - - 8 位 LED 输出

```

-- 关键组件:
--   - PicoBlaze 软核处理器 (kcpsm3)
--   - 指令存储器 (program)
--   - UART 收发模块 (uart_rx/uart_tx)
--   - 数字时钟管理模块 (my_dcm)

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- 实体声明
entity loopback is
    Port (
        clk      : in std_logic;          -- 主时钟输入 (例如 50MHz)
        rst      : in std_logic;          -- 复位信号 (高有效)
        lock     : out std_logic;          -- DCM 时钟锁定指示
        leds     : out std_logic_vector(7 downto 0); -- LED 输出
        switches : in std_logic_vector(7 downto 0); -- 拨码开关
        rs232_rx : in std_logic;          -- RS232 接收数据线
        rs232_tx : out std_logic;         -- RS232 发送数据线
    );
end loopback;

architecture Behavioral of loopback is

    -- 指令存储器组件声明
    COMPONENT program
    PORT(
        address      : IN std_logic_vector(9 downto 0);
        clk          : IN std_logic;
        instruction   : OUT std_logic_vector(17 downto 0)
    );
    END COMPONENT;

    -- PicoBlaze 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);
        port_id      : OUT std_logic_vector(7 downto 0);
        write_strobe  : OUT std_logic;
        out_port      : OUT std_logic_vector(7 downto 0);
        read_strobe   : OUT std_logic;
        interrupt_ack : OUT std_logic
    );
END COMPONENT;

```

-- UART 接收模块

```

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
);
END COMPONENT;

```

-- UART 发送模块

```

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
);
END COMPONENT;

```

-- 数字时钟管理模块 (生成 55MHz 时钟)

```

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
    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); -- 程序
地址总线
signal instruction   : std_logic_vector(17 downto 0); -- 指令总
线
signal port_id       : std_logic_vector(7 downto 0); -- I/O 端
口地址
signal write_strobe  : std_logic;                    -- 写使
能信号
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
能信号
signal clk55MHz      : std_logic;                    -- 生成
的 55MHz 时钟

-- 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 rx_data       : std_logic_vector(7 downto 0); -- 接收数
据寄存器
signal data_present  : std_logic;                    -- 接收
数据有效标志

begin

-- 实例化 PicoBlaze 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
);

```



```

-- 实例化指令存储器
my_program: program PORT MAP(
    address      => address,
    instruction => instruction,
    clk          => clk55MHz
);

-- 实例化时钟管理模块
Inst_my_dcm: my_dcm PORT MAP(
    CLKIN_IN  => clk,          -- 输入原始时钟
    RST_IN    => rst,
    CLKFX_OUT => clk55MHz,    -- 输出 55MHz 时钟
    LOCKED_OUT => lock        -- 时钟锁定状态
);

-- 波特率生成进程 (55MHz -> 9600bps)
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 -- 55MHz/(9600 * 16) ≈ 358
            baud_count <= (others => '0');
            en_16_x_baud <= '1'; -- 每 358 个周期产生一个使能
        else
            baud_count <= baud_count + 1;
            en_16_x_baud <= '0';
        end if;
    end if;
end process;

-- LED 输出控制
process (clk55MHz, rst)
begin
    if rising_edge(clk55MHz) then
        if rst = '1' then
            leds <= (others => '0');
        elsif write_strobe = '1' and port_id(0) = '1' then --
            leds <= out_port; -- 将处理器的输出数据送至
        end if;
    end if;
end process;

```

脉冲

写端口 0x01

LED

```

        end if;
    end if;
end process;

-- UART 发送模块实例化
transmit: uart_tx PORT MAP(
    data_in      => out_port,      -- 待发送数据
    write_buffer => write_strobe and port_id(1) and port_id(0),
-- 写端口 0x03
    reset_buffer => rst,
    en_16_x_baud => en_16_x_baud,
    serial_out   => rs232_tx,      -- 串行输出
    clk          => clk55MHz
);

-- 输入数据多路选择器
process (clk55MHz, rst)
begin
    if rising_edge(clk55MHz) then
        if rst = '1' then
            in_port <= (others => '0');
        else
            case port_id is
                when X"00" => in_port <= switches;      -- 读开
-- 关状态
                when X"02" => in_port <= rx_data;      -- 读接
-- 收数据
                when X"04" => in_port <= "0000000" &
data_present; -- 数据接收标志
                when others => in_port <= (others => '0');
            end case;
        end if;
    end if;
end process;

-- UART 接收模块实例化
receive: uart_rx PORT MAP(
    serial_in     => rs232_rx,      -- 串行输入
    data_out      => rx_data,      -- 并行输出
    read_buffer   => read_strobe and port_id(2), -- 读端口 0x04
    reset_buffer  => rst,
    en_16_x_baud  => en_16_x_baud,
    buffer_data_present => data_present, -- 数据接收有效
    clk           => clk55MHz

```

```
);
```

```
end Behavioral;
```

七、 实验中的问题

由于板子上的 LED 器件可能已被损坏, 在进行测试时, 本来在导入初始生成的比特流文件的时候, LED7~LED4 四个灯应该全亮, 但是实际上只有 LED6, LED4 亮起, 这是没有办法修改的问题, 但是我们可以通过对代码的分析了解到本来应该出现的实验现象就是四个灯全部亮起。