中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：FPGA 实验平台及 IP 核使用

学生姓名：傅申

学生学号：PB20000051

完成日期：2021 年 12 月 8 日

计算机实验教学中心制

2020 年 09 月

【实验题目】

FPGA 实验平台及 IP 核使用

【实验目的】

- 熟悉 FPGAOL 在线实验平台结构及使用
- 掌握 FPGA 开发各关键环节
- 学会使用 IP 核（知识产权核）

【实验环境】

- Windows PC
- Microsoft Visual Studio Code
- Xilinx Design Tools Vivado HL Design Edition 2019.1

【实验练习】

题目 1: 选择 IP 核目录 Vivado Repository/Memories & Storage Elements/RAMs & ROMs 中的 Distributed Memory Generator. 选择存储器类型为 ROM. 将其名称设置为 `dist_mem_gen_0`, 存储器深度设置为 $16 = 2^4$, 数据位宽设置为 8 位. 然后在 "RST & Initialization" 页面对存储器数据进行初始化, 对应的 `coe` 文件如下. 例化过程如图 1.

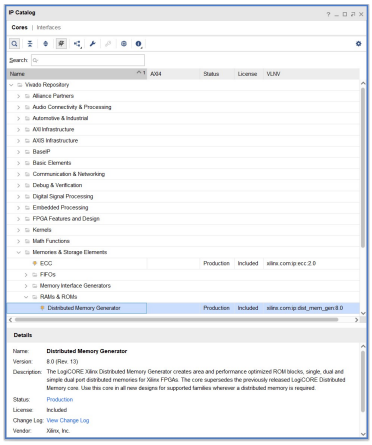
代码 1: ROM 初始化 `coe` 文件内容

```
memory_initialization_radix=16;
memory_initialization_vector=3F 06 5B 4F 66 6D 7D 07
                             7F 6F 77 7C 39 5E 79 71;
```

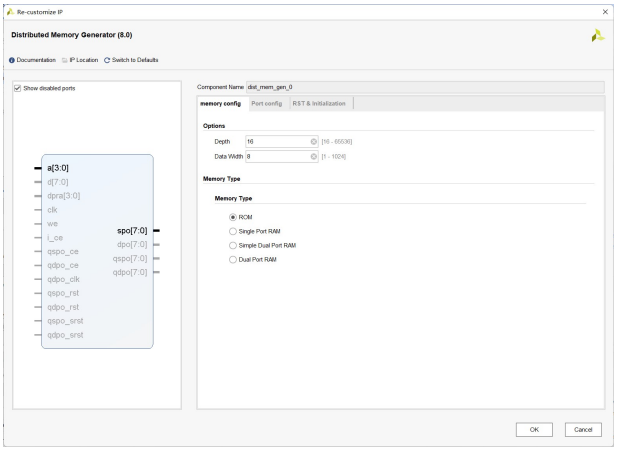
其中, `memory_initialization_vector` 对应的是每个数字在七段数码管上表示时的输入值. 比如 `memory_initialization_vector[2]=5B=01011011`, 对应数码管 G, E, D, B, A 亮起, 构成数字 '2'. 有了这样的存储器, 只需要将 `sw[3:0]` 信号输入到 ROM 中的 `a[3:0]` 端口, 再将 ROM 中的数据通过 `spo[7:0]` 端口输出到 `led[7:0]` 中即可. 对应的 Verilog 模块如下.

代码 2: 控制数码管显示的模块

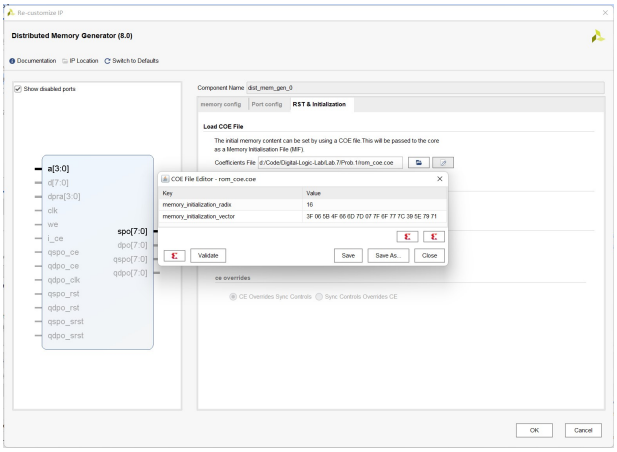
```
module display(
    input  [3:0] sw,
    output [7:0] led
    );
dist_mem_gen_0 rom(.a(sw), .spo(led));
endmodule
```



(a) IP 核目录



(b) 存储器设置



(c) 存储器数据初始化

图 1: 例化 ROM 过程

xdc 约束文件如下.

代码 3: xdc 约束文件

```
## FPGAOL LED (single-digit-SEGPLAY)
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];

## FPGAOL SWITCH
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
```

最后, 生成比特流并烧写到 FPGAOL 上, 效果如图 2.

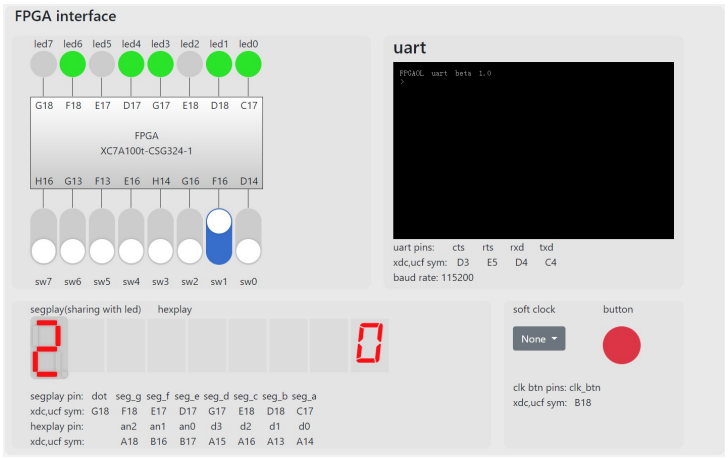


图 2: 题目 1 在 FPGAOL 上的效果

题目 2: 阅读 FPGAOL 的手册, 可以知道数码管建议的扫描频率为 50Hz, 在本题中我们需要驱动两个数码管, 因此需要一个 100Hz 的时钟. 显然 IP 核无法做到这一点, 所以我们需要使用一个计数器来产生 100Hz 的时钟. 同时, 我们需要采用时分复用的方式显示数字, 在这里我们在 100Hz 时钟信号高电平时显示高四位, 低电平时显示低四位. 同步复位信号高电平有效, 此时显示数字为 00. 对应的 Verilog 模块如下.

代码 4: 分时复用显示数字

```
module display(
    input  clk, rst,
    input  [7:0] sw,
    output reg an,
    output reg [3:0] d
    );
reg [19:0] cnt;
wire clk_100hz;
// 生成 100Hz 时钟信号
assign clk_100hz = (cnt >= 500000);
always @(posedge clk)
begin
    if (rst) cnt <= 0;
    else if (cnt >= 999999) cnt <= 0;
    else cnt <= cnt + 1;
end
// 分时复用显示数字
always @(posedge clk)
begin
    if (clk_100hz) an <= 1'b1;
    else an <= 1'b0;
end
always @(posedge clk)
```

```
begin
    if (rst) d <= 4'b0;
    else if (clk_100hz) d <= sw[7:4];
    else d <= sw[3:0];
end
endmodule
```

xdc 约束文件如下.

代码 5: xdc 约束文件

```
## Clock signal

#IO_L12P_T1_MRCC_35 Sch=clk100mhz
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];

##Switches

set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];


##7-Segment Display

set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33     } [get_ports { d[0] }];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33     } [get_ports { d[1] }];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33     } [get_ports { d[2] }];
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33     } [get_ports { d[3] }];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33     } [get_ports { an }];
```

最后, 生成比特流并烧写到 FPGAOL 上, 效果如图 3.

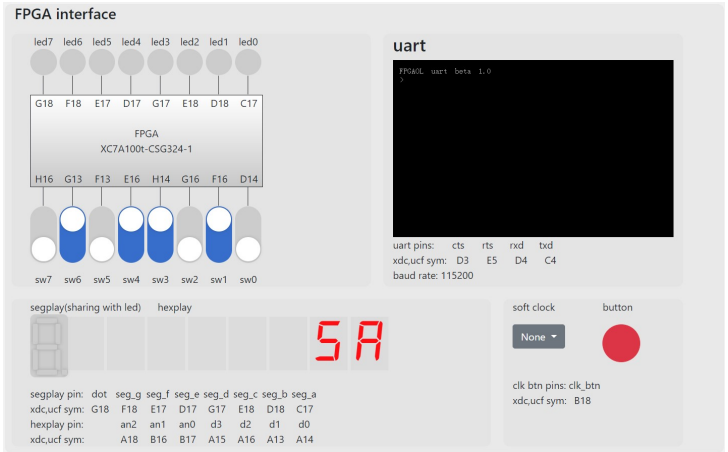


图 3: 题目 2 在 FPGAOL 上的效果

题目 3: 同样地, 我们需要一个 200Hz 的脉冲信号来驱动数码管, 每脉冲一次 `hexplay_an` 就增加 1 (模 4 意义下的), 同时切换对应的信号到 `hexplay_data` 中. 同时我们需要一个 10Hz 的脉冲信号来驱动计时器, 并且处理好可能的进位情况. 对应的 Verilog 模块如下.

代码 6: 计时器模块

```
module timer(
    input  clk_100MHz, rst,
    output reg [3:0] hexplay_data,
    output reg [1:0] hexplay_an
    );

wire pulse_10Hz, pulse_200Hz;
reg  [23:0] cnt_1;
reg  [18:0] cnt_2;
assign pulse_10Hz = (cnt_1 == 24'h1);
assign pulse_200Hz = (cnt_2 == 19'h1);

always @(posedge clk_100MHz)
begin
    if (rst) cnt_1 <= 0;
    else if (cnt_1 >= 9999999) cnt_1 <= 0;
    else cnt_1 <= cnt_1 + 1;
end

always @(posedge clk_100MHz)
begin
    if (cnt_2 >= 499999) cnt_2 <= 0;
    else cnt_2 <= cnt_2 + 1;
end

reg [3:0] deca_sec, sec, ten_sec, min;
always @(posedge clk_100MHz)
begin
    if (rst)
    begin
        min      <= 4'h1;
        ten_sec  <= 4'h2;
        sec      <= 4'h3;
        deca_sec <= 4'h4;
    end
    else if (pulse_10Hz)
    begin
```

```
        if (deca_sec >= 4'h9)
        begin
            deca_sec <= 4'h0;
            if (sec >= 4'h9)
            begin
                sec <= 4'h0;
                if (ten_sec >= 4'h5)
                begin
                    ten_sec <= 4'h0;
                    if (min >= 4'h9) min <= 4'h0;
                    else  min <= min + 1;
                end
                else ten_sec <= ten_sec + 1;
            end
            else sec <= sec + 1;
        end
        else deca_sec <= deca_sec + 1;
    end
end

always @(posedge clk_100MHz)
begin
    if (pulse_200Hz)
    begin
        if (hexplay_an >= 2'h3) hexplay_an <= 2'h0;
        else hexplay_an <= hexplay_an + 1;
    end
end

always @(clk_100MHz)
begin
    case(hexplay_an)
    2'h0: hexplay_data <= deca_sec;
    2'h1: hexplay_data <= sec;
    2'h2: hexplay_data <= ten_sec;
    2'h3: hexplay_data <= min;
    endcase
end
endmodule
```

xdc 约束文件如下.

代码 7: xdc 约束文件

```
## Clock signal

#IO_L12P_T1_MRCC_35 Sch=clk100mhz
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];

## FPGAOL HEXPLAY

set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[0] }];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[1] }];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[2] }];
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[3] }];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[0] }];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[1] }];

## FPGAOL BUTTON & SOFT_CLOCK

set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { rst }];
```

最后, 生成比特流并烧写到 FPGAOL 上, 效果如图 4.

图 4: 题目 3 在 FPGAOL 上的效果

【总结与思考】

收获 学习了如何通过固定频率的时钟信号生成指定频率的时钟信号, 了解了 IP 核的使用.

难易程度 中等

任务量 中等

建议 暂无