用 Verilog 语言实现把 lena 图像逆时针旋转 90 度

何天阳　　熊荣宇

2024 年 6 月 28 日

1　引言

本项目旨在通过 Verilog 语言实现对图像的路径描述，具体任务是将 lena 图像（或其他图像，尺寸为 256*256 或 512*512 像素）进行 90 度逆时针旋转。

报告中包括：

1. 部分原代码、仿真结果、工作报告与工作总结。

2. 对非 256*256*24bit 格式的 BMP 图像进行格式转换，并确保输出图像尺寸为 256*256 像素。

3. 详细记录设计中的输入输出信号、像素数据处理以及同步方式，确保输出图像的正确性。

2　模块设计

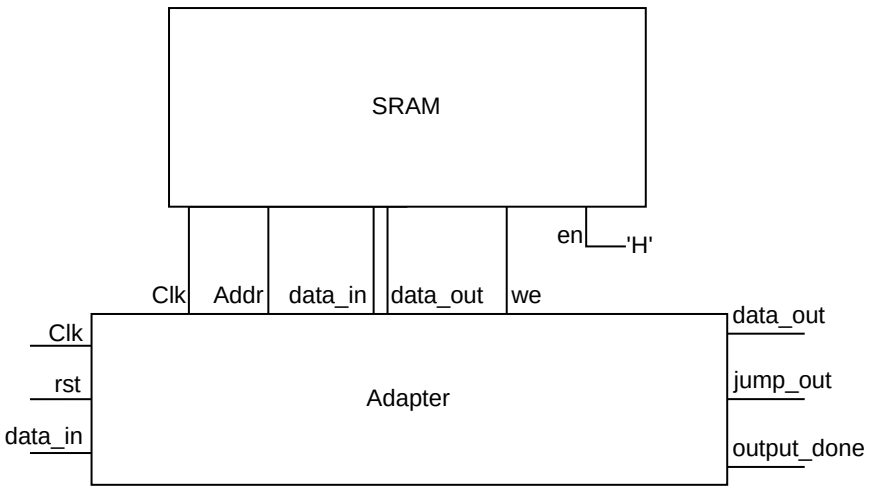

图 1: adapter 模块设计

如图. 1所示，整体模块设计分为 SRAM 与 Adapter 两个部分。SRAM 负责输入图像数据的存储，Adapter 负责输入流程控制与数据存储的控制，控制整个流程。

3 工作内容

3.1 环境搭建

构建运行流程

由于需要读取与保存 BMP 图片格式，考虑到 c++ 对于文件操作的便利性以及其生态的丰富，本次实验选择了 Verilator 编译器，该编译器可以将 Verilog 模块编译为 C++ 以供测试代码调用仿真。

为了简化编译与运行的流程，这里使用了 GNUMake，在 Makefile 中指定了项目构建的流程与步骤的依赖关系。

```
TOP_MODULE=adapter
BUILD_DIR=obj_dir
# Scan All Source Code
VERILOG_SRC=$(wildcard ./src/*.v)
CPP_SRC=$(wildcard *.cpp *.hpp)
# Verilog Compile Parameters
VERILOG_FLAGS= --cc --exe -Wall --top-module $(TOP_MODULE)

all: run

# Verify Verilog Module
verify: $(VERILOG_SRC)
    verilator $(VERILOG_FLAGS) $(VERILOG_SRC) $(CPP_SRC)
# Compile Project
compile: verify
    make -j -C $(BUILD_DIR) -f V$(TOP_MODULE).mk V$(TOP_MODULE)

# Run Simulation
run: compile png2bmp
    ./$(BUILD_DIR)/V$(TOP_MODULE)

# Convert png to bmp with 24bit (RGB channels)
png2bmp: res/lenna.png
    python converter.py ./res/lenna.png


```

```
clean:
    rm -rf $(BUILD_DIR)
    rm ./output_lenna.bmp
    rm ./res/lenna.bmp
    rm ./output.bmp
```

图片转换脚本

由于网络下载的 Lenna 图片多为 png, 且格式可能不满足要求，使用 python 脚本与 Pillow 库完成图片格式的标准化转换。脚本如下:

```
import sys
from PIL import Image
import os

def convert_png_to_bmp(input_path, size=(256, 256)):
    output_path = os.path.splitext(input_path)[0] + ".bmp"
    with Image.open(input_path) as img:
        img = img.resize(size, Image.LANCZOS)
        img.save(output_path, format="BMP")

    print(f"Converted image saved to {output_path}")

if len(sys.argv) != 2:
    print("Usage: python convert.py <input_path>")
    sys.exit(1)

input_path = sys.argv[1]
convert_png_to_bmp(input_path)
```

图片读入

为了方便快速的读取 BMP 文件，使用第三方 STBI 库完成读取操作。使用 git submodule 来管理第三方库

项目结构

综上所述项目结构大致如下文件树所示

```
converter.py
external
Makefile
obj_dir
output_lenna.bmp
res
src
test.cpp
```

3.2 Verilog 实现

如图 1 所示的架构，SRAM 负责输入图像数据的存储，而 Adapter 负责输入流程控制和数据存储控制。

对于 RAM 模块，首先定义输入的时钟周期信号 Clk、使能信号 en 和写使能信号 we，以及像素对应的地址 addr 和像素参数 date_in，输出的像素参数 date_out。

在时钟周期的上升沿，如果使能信号 en 为高电平，若输入输出信号 we 为高电平则给 mem[addr] 赋值 data_in，若输入输出信号 en 为低电平则将 mem[addr] 赋值给 data_out。

代码使用给出的参考代码，具体实现如下：

```
`define RAM_WIDTH 32 // Width of RAM
`define RAM_DEPTH 1024*1024 // Depth of RAM
`define ADDR_SZ 20 // Size of RAM address

module sram(
    clk,
    en, we,
    addr,
    data_in, data_out
);
    input clk, en, we;
    input [`ADDR_SZ-1:0] addr;
    input [`RAM_WIDTH-1:0] data_in;
    output reg [`RAM_WIDTH-1:0] data_out;

    reg [`RAM_WIDTH-1:0] mem [`RAM_DEPTH-1:0];

    always @(posedge clk) begin
        if (en) begin
            if (we) begin
                mem[addr] <= data_in;
            end
            else begin
                data_out <= mem[addr];
            end
        end
    end

endmodule : sram
```

adapter 模块主要实现了图像的 90 度逆时针旋转，包括输入信号的存储，信号的处理和信号输出，功能分解如下。

3.2.1 输入输出端口

输入设计

1. 复位信号 rst
2. 输入输出模式切换 mode
3. 时钟信号 clk
4. 数据信号线 data_in

输出共设计

1. 数据输出信号 data_out
2. 输出换行信号 jump_out
3. 输出完成信号 output_done

将 Adapter 设计为一个状态机，设计两状态。1. State 代表当前读入读出状态；2. 、Coord（实际使用 x,y 两寄存器）代表当前的坐标状态，使用两个 always 块实现状态机。

在第一个 always 块中，控制状态的切换和输出换行信号与输出完成信号的设置。

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        x <= 0;
        y <= 0;
        jump_out <= 0;
        output_done <= 0; // Initialize output_done
    end else begin
        x <= next_x;
        y <= next_y;
        jump_out <= (next_x == 8'd255) ? 1'b1:1'b0;
        output_done <= (next_x == 8'd255 && next_y == 8'd255 && mode == 1'b1) ? 1'b1:0;
    end
end
```

第二个 always 块根据当前输入引脚的状态，设置 next_state。xy 逐行输入，当到达行尾进行换行。此处完成了经典的两段式状态机的实现。

```
always @(*) begin
    if (mode == 1'b0) begin
        if (x == 8'd255) begin
```

```
            next_x = 0;
            next_y = y+1;
        end else begin
            next_x = x+1;
            next_y = y;
        end
    end else begin
        if (x == 8'd255) begin
            next_x = 0;
            next_y = y+1;
        end else begin
            next_x = x+1;
            next_y = y;
        end
    end
end
```

SRAM 连接部分，将 en 保持置高，we 与 mode 连接，共用时钟信号。

```
sram ram(
    .clk(clk),
    .en(1'b1),
    .we(~mode),
    .addr(addr),
    .data_in(mem_in),
    .data_out(mem_out)
);
```

其余线路与图像旋转部分直接使用 assign 连接

```
    assign addr = (mode == 1'b0) ? {4'b0, x, y}:{4'b0, 8'd255-y, x};
    assign mem_in = {8'b0, data_in};
    assign data_out = mem_out[23:0];
```

上述代码中，当 mode 处于读入模式时，正常读入坐标，当 mode 处于输出模式时，进行图片旋转的坐标映射：

$$(x', y') = (\text{size} - y, x)$$

综上，全部代码如下：

```
module adapter(
    input rst,
    input mode,
    input clk,
    input [23:0] data_in, // 8bit-rgb image
```

```
6
7       output reg [23:0] data_out,
8       output reg jump_out,
9       output reg output_done // New output signal
10  );
11
12      reg [7:0] x, next_x;
13      reg [7:0] y, next_y;
14      reg [19:0] addr;
15      reg [31:0] mem_in;
16
17      /* verilator lint_off UNUSEDSIGNAL */
18      reg [31:0] mem_out;
19
20      sram ram(
21          .clk(clk),
22          .en(1'b1),
23          .we(~mode),
24          .addr(addr),
25          .data_in(mem_in),
26          .data_out(mem_out)
27      );
28
29      always @(posedge clk) begin
30          $display("x: %d, y: %d, addr: %h, data_in: %h, data_out: %h,
        jump_out: %h, output_done: %b", x, y, addr, mem_in, mem_out,
        jump_out, output_done);
31          $display("mode %b, next_x: %d, next_y: %d", mode, next_x,
        next_y);
32      end
33
34      always @(posedge clk or posedge rst) begin
35          if (rst) begin
36              x <= 0;
37              y <= 0;
38              jump_out <= 0;
39              output_done <= 0; // Initialize output_done
40          end else begin
41              x <= next_x;
42              y <= next_y;
43              jump_out <= (next_x == 8'd255) ? 1'b1:1'b0;
44              output_done <= (next_x == 8'd255 && next_y == 8'd255 &&
        mode == 1'b1) ? 1'b1:0;
```

```
45          end
46      end
47
48      always @(*) begin
49          if (mode == 1'b0) begin
50              if (x == 8'd255) begin
51                  next_x = 0;
52                  next_y = y+1;
53              end else begin
54                  next_x = x+1;
55                  next_y = y;
56              end
57          end else begin
58              if (x == 8'd255) begin
59                  next_x = 0;
60                  next_y = y+1;
61              end else begin
62                  next_x = x+1;
63                  next_y = y;
64              end
65          end
66      end
67
68      // Transform the address for 90-degree rotation
69      assign addr = (mode == 1'b0) ? {4'b0, x, y}:{4'b0, 8'd255-y, x};
70      assign mem_in = {8'b0, data_in};
71      assign data_out = mem_out[23:0];
72  endmodule
73
```

3.3 TestBench 实现

TestBench 模块主要功能是验证 Verilog 模块的正确性。为了简便性，这里使用了 Verilator 编译器，该编译器可以将 Verilog 编译为 C++ 以供仿真验证。

C++ 部分，使用了外部 STBI 库对图片进行读入和写入的操作。实例化 Verilated 对象，获取 Top 的 Verilog 模块，直接使用 C++ 设置引脚数值，完成仿真流程的验证。

该部分较为简单，具体流程如下：

1. 实例化 Verilog 模块。
2. 读入 BMP 图片。

3. 设置引脚为输入模式，循环时钟输入图片。

4. 设置引脚为读取模式，循环时钟获取图片。

5. 保存图片到文件。

代码如下：

```
#define STB_IMAGE_IMPLEMENTATION
#include "../external/stb/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "../external/stb/stb_image_write.h"

#include "Vadapter.h"
#include "verilated.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <cstdint>

using namespace std;

bool readBMP(const char* filename, vector<uint8_t>& image, int& width, int& height, int& channels) {
    unsigned char* data = stbi_load(filename, &width, &height, &channels, 3);
    if (!data) {
        cerr << "Error: Could not open the file." << endl;
        return false;
    }

    image.assign(data, data + width * height * 3);
    stbi_image_free(data);
    return true;
}

bool writeBMP(const char* filename, const vector<uint8_t>& image, int width, int height) {
    if (stbi_write_bmp(filename, width, height, 3, image.data()) == 0) {
        cerr << "Error: Could not create the output file." << endl;
        return false;
    }
```

```
33      return true;
34  }
35
36  int main(int argc, char **argv) {
37      VerilatedContext *contextp = new VerilatedContext;
38      contextp->commandArgs(argc, argv);
39      Vadapter *top = new Vadapter{contextp};
40
41      vector<uint8_t> image;
42      int width, height, channels;
43
44      if (!readBMP("./res/lenna.bmp", image, width, height, channels)) {
45          return -1;
46      }
47
48      vector<uint8_t> output_data;
49      output_data.resize(image.size());
50
51      top->rst = 1;
52      top->clk = 0;
53      top->eval();
54      top->rst = 0;
55      top->eval();
56
57
58      // Write image data into the Verilog module
59      for (int y = 0; y < height; ++y) {
60          for (int x = 0; x < width; ++x) {
61              int index = (y * width + x) * 3;
62              uint8_t r = image[index + 0];
63              uint8_t g = image[index + 1];
64              uint8_t b = image[index + 2];
65              top->data_in = (r << 16) | (g << 8) | b;
66              std::cout << std::hex << top->data_in << std::endl;
67              top->eval();
68              top->clk = 1;
69              top->eval();
70              top->clk = 0;
71              top->eval();
72
73          }
74      }
75
```

```
76
77      // Switch to read mode
78      top->rst = 1;
79      top->mode = 1;
80      top->eval();
81      top->rst = 0;
82      top->mode = 1;
83      top->eval();
84
85      int index = 0;
86      // Read image data from the Verilog module
87      // Read image data from the Verilog module
88      for (int y = 0; y < height; ++y) {
89          for (int x = 0; x < width; ++x,index+=3) {
90              top->eval();
91              top->clk = 1;
92              top->eval();
93              output_data[index + 0] = (top->data_out >> 16) & 0xFF; // R
94              output_data[index + 1] = (top->data_out >> 8) & 0xFF;  // G
95              output_data[index + 2] = top->data_out & 0xFF;         // B
96              top->clk = 0;
97              top->eval();
98          }
99      }
100
101     // Save output data to BMP file
102     if (!writeBMP("output_lenna.bmp", output_data, width, height)) {
103         return -1;
104     }
105
106     delete top;
107     delete contextp;
108     return 0;
109 }
```

4 结果与分析

经过运行，*lena* 的原图和逆时针选转 90 度后 *lena* 的图像如下：

图 2: lenna.bmp

图 3: lenna.bmp

5 个人工作总结

本次小组工作共有两名组员参与：何天阳、熊荣宇，以下为各自的个人工作总结。

5.1 何天阳个人工作总结

在本组项目中，我主要完成了 *Verilog* 模块部分编写，用于将 *lena* 图像逆时针旋转 90 度；以及分析了 *testbench* 的测试结果，撰写了项目报告。

在本项目中，我通过参与 *Verilog* 模块的编写和测试，我深入理解了图像处理算法在硬件层面的实现方法，学会了如何使用 Verilog 进行仿真与验证与其他编程语言的交互和引用方法。通过团队合作，提升了沟通协调能力，提高了在团队中分工合作，共同解决技术问题和项目挑战的能力。

5.2 熊荣宇个人工作总结

在本次项目中，我主要完成了 *Verilog* 模块的部分编写，主要是数据传输和逻辑控制的编写。我还完成了 *testbench* 关键组件的编写，以及报告相应内容的撰写。

在本项目中，我通过深入参与 *Verilog* 模块的开发和测试，我提升了对硬件设计和验证流程的理解。提升了有效协作和沟通，尤其是在技术讨论和解决问题时的团队合作能力。我还收获了宝贵的实践经验，对未来的硬件开发项目有了更加自信的态度和准备。