EIE330 FPGA Based System Design

Report of Laboratory 5

FPGA Lab5: Key Debounce, VGA Display with ROM

Source Code of this report can be found at: EIE330-FPGA-Laboratory

by

PENGRUI K. TANG
Student No: 1220031811
from EIE330 D1,
Faculty of Innovation Engineering,
Universidade de Ciência e Tecnologia de Macau

XIAOYANG S. ZHEN
Student No: 1220034170
from EIE330 D1,
Faculty of Innovation Engineering,
Universidade de Ciência e Tecnologia de Macau

Nov, 2024
(Coloane, Macao SAR)

This report is a part of the result for EIE330 FPGA Based System Design in U.C.T.M

This page is intentionally left blank.

Conteúdo

Capítulo I

Abstract

I Result location

The Source Code of this report can be found in the git-hub repository: EIE330-FPGA-Laboratory. And the demonstration video of certain codes can be found at the Github release Page

II Contribution

All relevant output in this report is shown in the following directory.

- **FPGA Lab5**
  1. 5-0. Report [T][Z]
     (a) Text [T][Z]
  2. 5-3. Key Debounce
     (a) Design [T][Z]
     (b) Test [T]
  3. 5-4. VGA Display with ROM
     (a) Design [T][Z]
     (b) Test [T]

And the contribution of each member is labeled with their last name:

1. Pengrui Tang: [T]
2. Xiaoyang Zhen: [Z]

Capítulo II

Key Debounce

I Introduction

I: i Background

- **What is the key debounce:**
  When the key is pressed and released in a mechanical spring switch, the mechanical contact inside the device is not ideal. It will not change instantly from signal to signal while a key switch is closed or opened, signal will not be immediately and stably conversed. However, the key switch might transfer the signal that has a series of jitters at the moment of closing and disconnecting, which causes multiple rapid on-off signals before stabilizing in the "pressed"or "released"state.
  Key debounce is a mechanism used in electronics to eliminate the effects of jitteriness or bouncing. Debouncing ensures that only one clean signal is interpreted by the system for a single press or release. By applying key debounce, devices will be more reliable and accurate inputs handling, improving the experience and stability of system performance.
- **Risk:**
  The key debounce is a solution to the problem of jitteriness in mechanical switches, yet there is the possibility that it is implemented improperly or not accounting for its trade-offs. The risks type:
  - **Over-Debouncing**
    Delayed responsiveness, Missing inputs
  - **Under-Debouncing**
    Failed to eliminate noise, Erratic behaviour
  - **Increased Complexity**
    Timing issues, Resource consumption
  - **Hardware Debouncing Challenges**
    Component reliability, Inflexibility, Space and cost
  - **Missing Genuine Multi-Key or Multi-Switch Inputs**
- **Possible solutions:**
  There are several possible solutions that can solve the key debounce:
  - Construct low-pass filters with RC-circuit to smooth the signal.
  - Ignore signals for a short period after the key-press.

- Signal processing with advanced algorithms for highly noisy environments.
- Combine hardware with software to balance reliability and flexibility.

In this experiment, we are going to use the second solution that ignores signals for a short period to filter the jitter signal.

I: ii Purpose

The purpose of this experiment is to design a project that avoids the signal-jitter situation, allowing the system to receive clean and accurate signals.

I: iii Design and Key-results

The designed circuit should be able to filter the unideal signal while the key is pressed or released, making the circuit provide expected signals to the system.

II Materials and Methods

II: i Structure Block Diagram

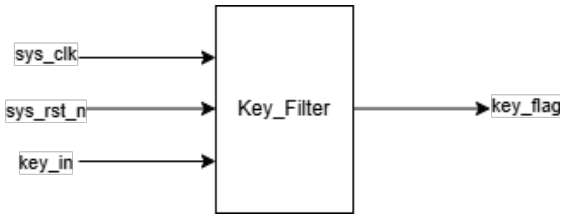

Figura 1: Structure Block Diagram of Key Debounce

As figure 1 shows, this circuit contains three inputs sys_clk, sys_rst_n, and key_in; one output key_flag.

key_flag is the filtered key_in, it ignores 20 ms signal after the last input. sys_clk and sys_rst_n control the system clock and reset the system respectively.

II: ii Designed Signal Waveforms

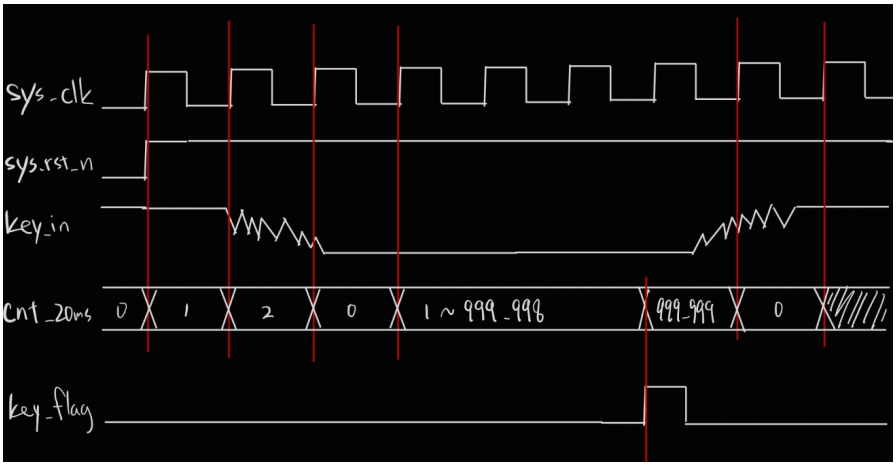

Figura 2: Designed Waveform of Key Debounce

As figure 2 shows, when the key is pressed or released, there is jitteriness happened. To avoid mistaken inputs, we use a 20ms counter to count the 20ms period of ignoring inputs. When the cnt_20ms reaches its maximum, key_flag generates a one-clock-cycle high-signal. When the signal of key_in is received, cnt_20ms starts counting from 0 to its maximum 999_999. Signal of Key_flag represents the true input that will be sent to the system.

III Results

III: i Testbench

For the convenience of testing, we set a virtual bouncing situation to simulate the jitter of the input signal in the testbench, and we also observed signals of the counter (cnt_20ms) and key_flag to verify the filter is functional. The simulation result is in the following subsection.

III: ii Simulated Result

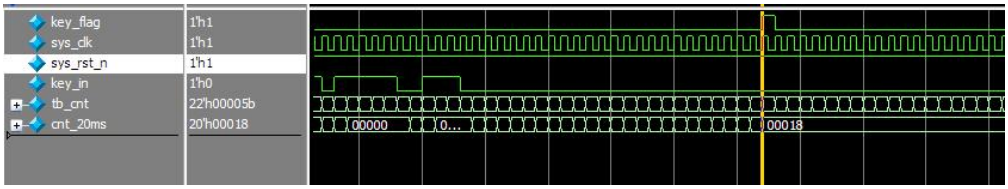

Figura 3: Waveform of Key Debounce

As figure 3 shows, there are several signals are sent after the first press of key_in, we use cnt_20ms to count the input from the last high-signal sent. After the counter reaches its maximum, key_flag

turns to 1 for a clock cycle to send the real signal of key_in.

IV Discussions and Conclusion

IV: i Conclusion

In this lab, we learned how the signal jitter is formed and how to avoid it with software methods. In this experiment, we successfully filter the unideal signal by ignoring the short time of signals after the key is triggered.

V Appendices

The code in this experiment referenced the following sources (IEEE Style):

Referências

[1] pikipity, Aug, 2024, "FPGA-Laboratory/../2_Key_Debounce," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/tree/main/Lab5/2_Key_Debounce

Capítulo III

VGA Display with ROM

I Introduction

I: i Background

- **What is the ROM:**
  ROM, Read-Only Memory, is a type of non-volatile memory, which means memory retaining data without power, used in computers and electronic devices. The data stored in ROM will not be modified during normal use.
  The content in ROM usually is preprogrammed, it typically contains firmware or software that is permanently programmed during manufacturing. As it was named, it should be read-only access under normal operations, data in ROM cannot be modified or erased.
- **Differences between ROM and RAM:**
  ROM provides permanent storage for essential data and instruction for the system to operate, and it can retain data without power. Moreover, ROM is mostly read-only.
  RAM, on the other hand, serves as a temporary workspace for data and applications that need quick access by the processor during operation. It will lose data when the power is off. Yet, RAM is faster than ROM, and it usually has high-speed read/write operations.

I: ii Purpose

This experiment requires us to build a VGA driver and use the FPGA development board to drive the VGA display to the MUST logo that is stored in ROM with ten-colour equal-width colour bars as the background.

To fulfill the aim, we should be able to understand the mechanism of ROM, and how the VGA works in Verilog codes.

I: iii Design and Key-results

The designed circuit should be able to display the Macau University of Science and Technology logo in the middle of the screen, and ten-colour equal-width bars should be also shown on the monitor as background.

II Materials and Methods

II: i Structure Block Diagram

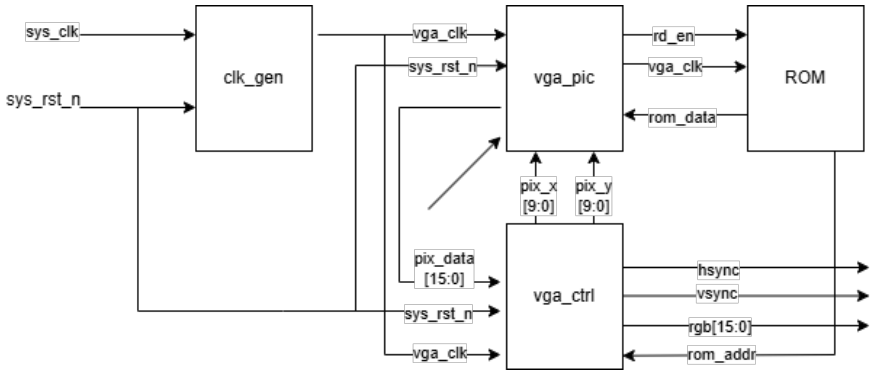

Figura 4: VGA with ROM Block Diagram

As the figure 4 shows, the VGA block diagram consists of three blocks:

- **clk_gen**
  This block generates the clock signal for VGA scan from the system clock. The pixel clock generated should have a frequency of 25.175MHz according to the VGA standard for 640x480@60Hz resolution. [**?**].
- **vga_ctrl**
  This block will generate the control signals for the VGA port according to the VGA standard. When set to work, it'll continuously scan the screen by sending the vertical and horizontal synchronization signals to the display. When it scans to the valid area, it will request the pixel value by outputting the pixel coordinate in the valid area.
- **vga_pic**
  This block generates the pixel's color value based on the pixel position input. If it receives a valid pixel coordinate, it will generate the pixel value of the screen.
  This module works like a multiplexer when the pixel coordinate is in the LOGO area, it will display a signal from ROM, else it will display the background color bar.
  To request the pixel from ROM, it sends the rd_en signal to the ROM to enable the ROM to read the pixel value. One important notice is that the ROM has one clock delay in reading data, so we need to advance one clock cycle to eliminate the effect of delay.
- **ROM**
  This block reads the pixel value from the ROM. The ROM stores the pixel value of the MUST logo. When the VGA_PIC module requests the pixel value, it will read the pixel value from the ROM and output it to the VGA_PIC module.

II: ii Designed Signal Waveforms

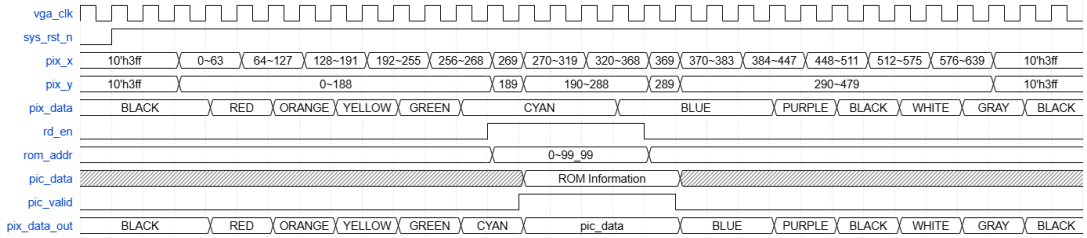

Figura 5: Design Wave of VGA Display with ROM

As figure 8 shows, there are five different signal channels.

sys_rst_n represents the state of the $RESET$ button. Low means the key is pressed, and high means the key is released.

vga_clk controls the clock of the system, it synchronizes the clock of vertical pixel scan and horizontal pixel scan. It ensures the synchronization of ROM, vertical and horizontal signals.

pix_x and pix_y represent the pixel number in the row and the pixel number in the column respectively.

Be reminded that there is an empty period for synchronizing vertical signals and horizontal signals before the project starts. Furthermore, in this experiment, we are going to use ROM to store picture information, and reading information from ROM will be delayed with one clock cycle being a feature of it.

rd_en is the signal to enable the ROM to read the pixel value. When the pixel coordinate is one clock ahead of the valid area, the rd_en signal will be high, and the ROM will read the pixel value from the ROM and output it to the vga_pic module. pic_valid is the signal to indicate the pixel coordinate is in the valid area of the LOGO. It is one clock cycle behind the rd_en signal, and when it's high, the vga_pic module will display the pixel value from the ROM instead of the background color bar. pix_data is the contents information read from the ROM, the corresponding range of pix_data to the pix_x and pix_y is calculated as equations below:

$$pix\_x_{min} = \frac{640}{2} - \frac{100}{2} - 1 = 269 \tag{1}$$

$$pix\_x_{max} = \frac{640}{2} + \frac{100}{2} - 1 = 369 \tag{2}$$

$$pix\_y_{min} = \frac{480}{2} - \frac{100}{2} - 1 = 189 \tag{3}$$

$$pix\_y_{max} = \frac{480}{2} + \frac{100}{2} - 1 = 289 \tag{4}$$

So the valid range of pix_data is pix_x from 269 to 369, pix_y from 189 to 289. Due to the scan logic, each column should be fully scanned when the row number is changing, so pix_y has more clocks than pix_x, it is scanned from pix_x equals 0 to pix_x equals 639. Moreover, this time we are using ROM as the picture source, so we need to advance one clock cycle to eliminate the effect of delay.

II: iii Image processing

In order to store an image in ROM, we use a Matlab script to convert a jpg file to an array of colors in HEX color format, and automatically write it into an imx file.

III Results

III: i Testbench

For the convenience of testing, we wire up vga_control and vga_pic to the testbench, and we also monitor the v_sync and h_sync signals to verify the timing is correct. The simulation result is in the following subsection.

III: ii Simulated Result

1. **rd_en**

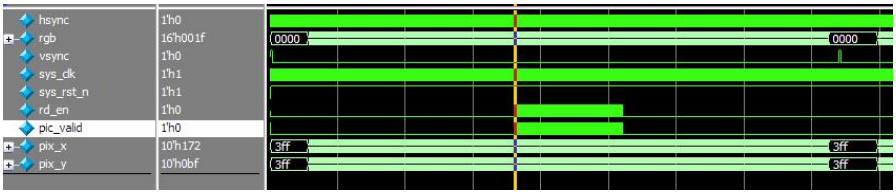

Figura 6: Simulated Result of rd_en

Figure 6 shows the overview of the timing of rd_en and pic_valid. According to the label on the left, you can locate the rd_en signal, the pic_valid signal, and the pix_x signal. And one may notice the rd_en is set to high in the middle of the pix_x range. This means the ROM is being read in the middle of the screen.

2. **ROM Read**

Figura 7: Reading ROM

Figure 7 shows the ROM reading process. One may reference the label on the left to find which line represents which signal. At the yellow line, the re_en signal is raised to high, and after a clock, the pix_data signal is raised to the value of the ROM. At the same time, the RGB signal is displayed 0, which means the data from the logo, which is inaccessible in the simulation, is displayed.

III: iii On Board Result

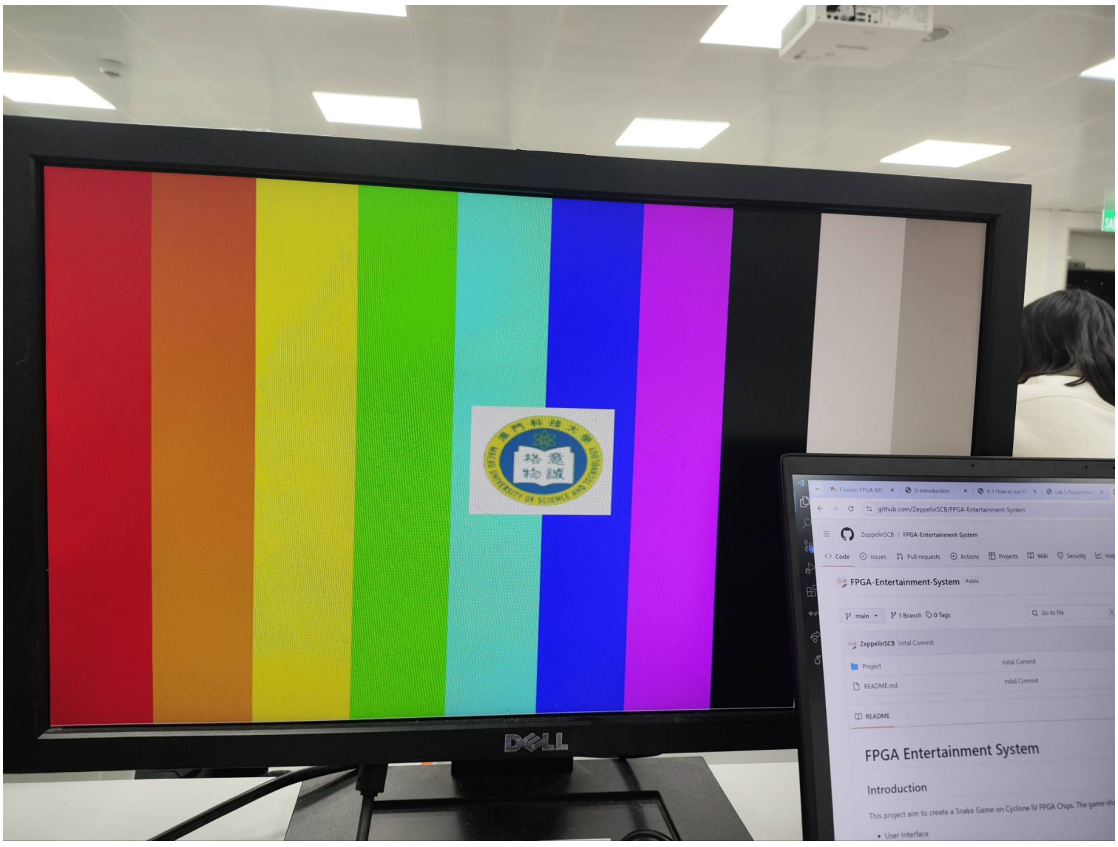

Figura 8: On Board Result of VGA Display with ROM

Burn the code into the development board, we have the display display correctly, with the color bar background and the MUST logo in the middle of the screen.

IV Discussions and Conclusion

IV: i Discussions

1. At first, we create the ROM file just following the instructions of creating ROM in the IP core. But this makes the display show a blue block instead of MUST logo. Later, we discovered the problem was that we didn't set the correct data width and address depth. It should be 16*100000, but we set 8*256. So it only loads the first 256 items=and only the first 8 bits. We discuss the reason for it to be blue. The reason is the first 256 items happen to be white, so the ROM is filled with FF. However, our design reads 16 bits from it, and it fills the first 8 bits with 0s. So the color becomes 00FF, which is blue. After we correct the setting, the display shows the MUST logo correctly.
2. In addition, when converting the image

IV: ii Conclusion

In this lab, we learned how to construct a ROM file and use it to store the MUST logo and display it on the screen using VGA protocol. We now learned how to set up ROM to store data of any width and depth within the ROM size.

V Appendices

The code in this experiment referenced the following sources (IEEE Style):

Referências

[1] pikipity, Aug, 2024, "FPGA-Laboratory/../4_VGA_Display," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/tree/main/Lab5/4_ROM_VGA