EIE330 FPGA Based System Design

Report of Laboratory 2

FPGA Lab1: Full adder, Decoder, and Comparator

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

Oct, 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

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

II Contribution

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

- **FPGA Lab2**
  1. 2-0. Report [T][Z]
     (a) Text [T][Z]
     (b) Video [Z]
  2. 2-2. Avoid Latch
     (a) Test [T]
  3. 2-3. Flip-flop
     (a) Test [Z]
  4. 2-4. Counter
     (a) Design [Z]
     (b) Test [T] [Z]
  5. 2-5. RunningLight
     (a) Design [T][Z]
     (b) Test [T][Z]

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

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

Capítulo II

Avoid Latch

I Introduction

I: i Latch

A latch is a component that stores value until it's changed. But sometimes we may accidentally introduce unexpected latches in our design.

I: ii Drawback of Latch

1. Latch may cause unnecessary resources to cost. Most FPGA chips don't have a latch component and will require many components to implement it, which is unnecessary.
2. It might result in unstable output. When the input signal is unstable, the output of the latch will be unstable as well.
3. The analysis of the circuit might be difficult because the latch makes the circuit no longer a pure combinational circuit.

I: iii Purpose of experiment

In this experiment, we will demonstrate how to introduce and avoid latches in three kinds of condition logic.

II Materials and Methods

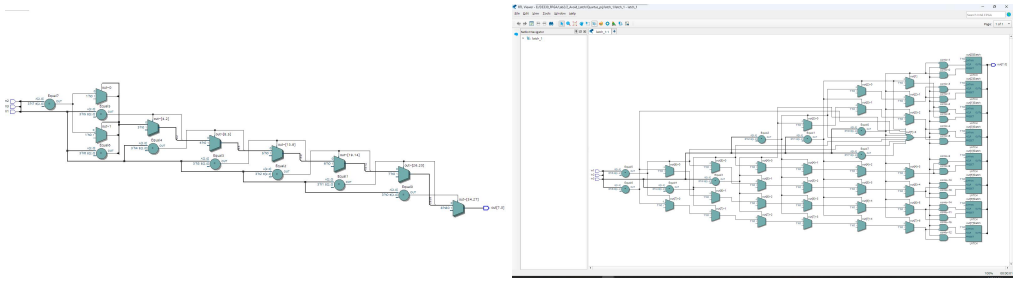
II: i To latch or not to latch

In the design, we introduce a latch by not giving the circuit a default output for not listed input. That is, if the input for any change don't meet the condition, the output will maintain it's previous input, and causing the latch.

To avoid it, we simply add the default case back in the code.

III Results

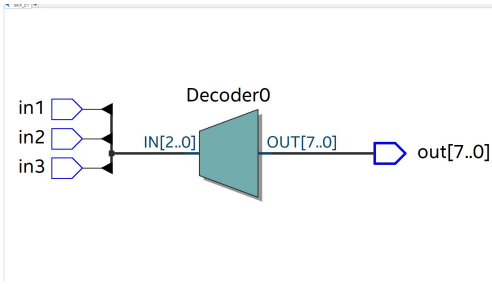
III: i Difference between Latch and no latch



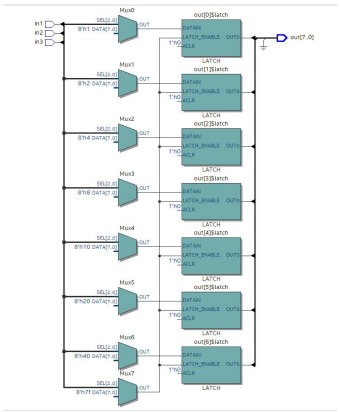
(a) With out Latch

(b) With out Latch

Figura 1: Latch 1

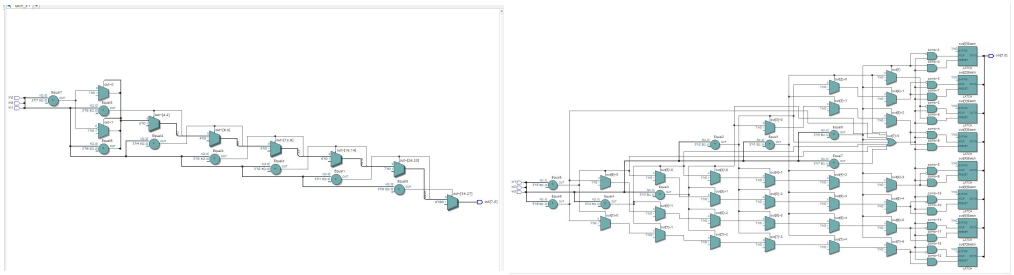


(a) With out Latch



(b) With out Latch

Figura 2: Latch 2



(a) With out Latch

(b) With out Latch

Figura 3: Latch 3

III: ii Discussion

According to the figure, the circuit compiled with a latchs is much more complicated than the circuit without latches. And overcomplecate the circuit that was meant to be very simple.

The latch makes the circuit use unnecessary resources, due to the fact that most FPGA chips don't have a latch component and will implement it with many basic components, which is unnecessary.

Latch might result in unstable output. When the input signal is unstable, the output of the latch will be unstable as well. The analysis of the circuit might be difficult because the latch makes the circuit no longer a pure combinational circuit.

IV Conclusion

Summary

In conclusion, when a latch happens, it will use much more component compares to a design without latch. Through this experiment, we have seen latches occur under different conditions.

Avoid Latch

To avoid latch, we basically need to make sure the circuit makes an output when no preset condition is satisfied. For example, in if-else condition, there must be an else for other possibilities, such as don't care. In case of condition, it should always have a default for other possible inputs. As said above, the output of the latch will be unstable, so it is important to avoid latch.

V Appendices

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

Referências

[1] pikipity, Aug, 2024, "FPGA-Laboratory/../2_Avoid_Latch," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/tree/main/Lab2/2_Avoid_Latch/RTL

[2] pikipity, Aug, 2024, "FPGA-Laboratory/../2_Avoid_Latch," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/tree/main/Lab2/2_Avoid_Latch/Sim

Capítulo III

Flip-Flop

I Introduction

I: i Background

1. What is the flip-flop:

   It is a basic digital memory circuit that can store one-bit data.

2. Key function:

   The key function of the flip-flops is to store data, holding a binary state (either 0 or 1) until a signal prompts a change.

3. Importance:

   The "memorizing“ ability is unique to the circuit.

4. Key differences between synchronized and non-synchronized:

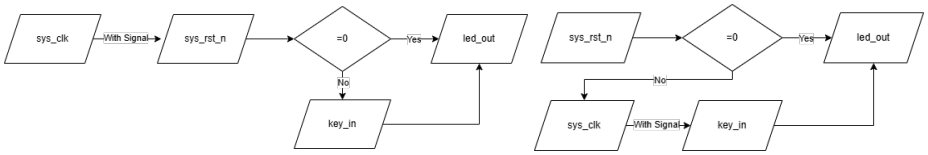
   Synchronized flip-flops will change state only when they respond to a specific clock signal. However, non-synchronized flip-flops will change immediately since the input signal is applied.

I: ii Purpose

In this experiment, we aimed to compare flip-flops with synchronized and non-synchronized reset.

II Materials and Methods

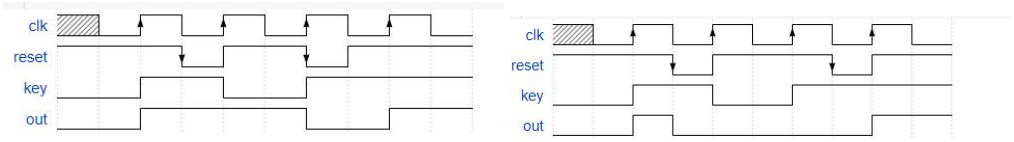
II: i Structure Block Diagram



(a) Sync flip-flop (b) Async flip-flop

Figura 4: Logic Design of Flip-Flop

II: ii Signal Waveforms



(a) Sync flip-flop (b) Async flip-flop

Figura 5: Designed signal waveform of both flip-flop

As (a) shows in the Async flip-flop, the reset key sets the output to 0 at its negative edge; while as shown in (b) in the Sync flip-flop, the value of output only changes when the positive edge of the clock signal.

II: iii Resources in Board

The whole circuit requires three inputs and one output, they are $sys\_clk/asys\_clk$, sys_rst_n, key_in, and led_out respectively. We used $FPGA\_CLK$, $RESET$, Key_1, and LED_0 on board.

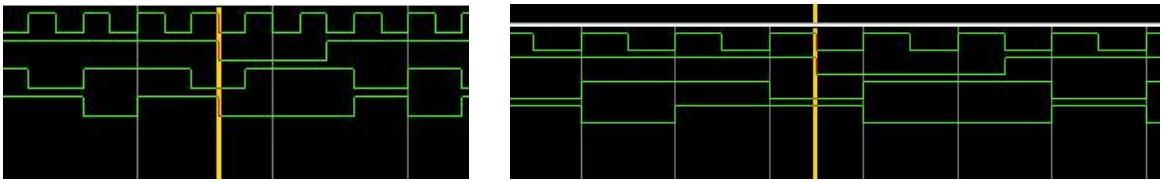
III Results

III: i Testbench

In this experiment, the testbench was built to accept three inputs: CLK, $RESET$, and key_in. The value of key_in is assigned between 0 and 1 randomly. The output was presented by the waveform. The expected results are listed below:

1. RESET is pressed, CLK will be reset and restarted, and LED state will be set to the initial state.
2. When RESET is not pressed, the system enters the judgment of the key_in state.
3. When key_in is pressed/not pressed, the state of LED will be set to the same state as key_in.

III: ii Simulated Result



(a) Async Flip-flop (b) Sync Flip-flop

Figura 6: Flip-flop

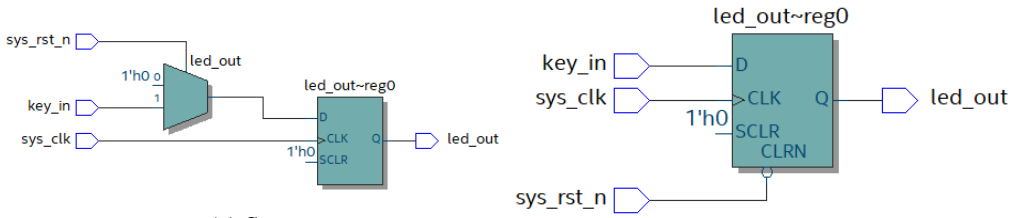
The waveform above in figure 6 have four signals, from top to bottom are the *Clock, Reset, Key, Output*respectively. The cursor shows an off-clock negative edge of the *Reset* signal. We can notice, in the asynced flip-flop, the output was set to zero immediately, while in the synced flip-flop, the output signal is unchanged as there isn't a clock signal.

III: iii Development Board

On the development board, we can't see the difference between sync and async flip-flop with the naked eye, but the general flip-flop functions as intended. The video can be find on Youtube.

IV Conclusion

IV: i Difference between Sync and non-Sync



(a) Sync (b) Async

Figura 7: Flip-flop

According to figure 7a, Sync flip flop achieves reset function by adding a multiplexer before its D input, when updating the signal, if the reset is 0, then the input will be 0 regardless of the Key_in; on contract, the Async flip-flop used the embedded reset function to reset the output as 0 regardless of the clock.

IV: ii Conclution

At first, the concept of synchronized and non-synchronized were not clear to me. In this experiment, we learned the difference between synchronized flip-flops and unsynchronized flip-flops, their reactions are very different in the way when the input is received.

V Appendices

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

Referências

[1] pikipity, Aug, 2024, "FPGA-Laboratory/../3_Flip_Flop," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/tree/main/Lab2/3_Flip-Flop/RTL

[2] pikipity, Aug, 2024, "FPGA-Laboratory/../3_Flip_Flop," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/tree/main/Lab2/3_Flip-Flop/Sim

Capítulo IV

Counter

I Introduction

I: i Purpose

In this experiment, we want to design a counter to control a flashing LED light circuit. We will learn more about the differences between synchronized and unsynchronized flip-flops.

I: ii Design and Key-results

The resulting circuit should be able to drive an LED flashing light at a frequency of 1Hz, and features a reset key to reset the output to the initial state.

Using Verilog, we have successfully designed the circuit, and tested it in both the simulator and the development board.

II Materials and Methods

II: i Structure Block Diagram

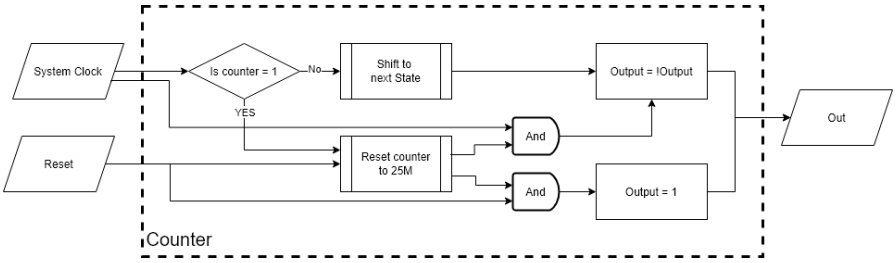


Figura 8: Logic Design of Counter

II: ii Signal Waveforms

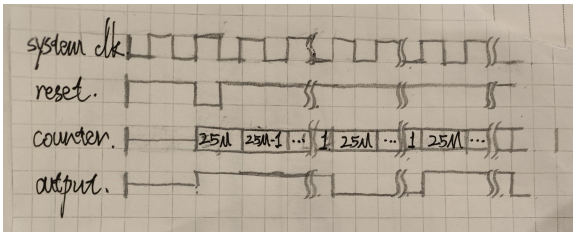


Figura 9: Design Wave of Counter

As the structure block diagram (fig8) and the design wave (fig9) shows, at the time RESET is pressed, the counter is set to $\frac{50Million}{2}$ and the system starts running. When the CLK impulse comes, the counter will count down. Once the counter counts to 0, the system will flip the output, and reset the counter to $\frac{50Million}{2}$.

II: iii Resources in Board

The whole circuit requires two inputs and one output, they are sys_clk, sys_rst_n, and led_out respectively. The output is displayed on the development board using LED light. And there is a register time_counter used to record time.

In this experiment, we used LED_0, $FPGA\_CLK$, and $RESET$ on the development board.

III Results

III: i Testbench

In this experiment, the testbench was built to accept two inputs: CLK and RESET. They are generated with values between 0 and 1 randomly. The output was presented by the waveform. The expected results are listed below:

1. When the negative edge of the Reset Signal (the moment reset key is pressed) is detected, the counter assigned the initial value of 25 million, and output
2. When time_counter counts down to zero after half a second, the LED light reverses its state.

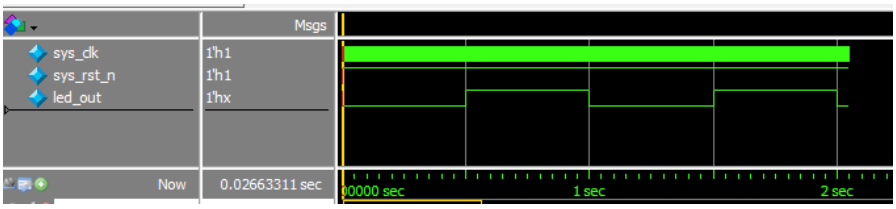


Figura 10: The simulation result of the Waveform

III: ii Simulated result

Above is the simulated waveform. As the label on the left indicates, the third signal is output signal, we can see it's reset to 0 when the *Reset* key is pressed at the crusor. And from the time axis at the bottom, we can see it filps its value every half a second, and generates a positive edge at a frequency of 1Hz.

This behavior aligns with our expectations, so we conclude the design pass the simulation test.

III: iii Development Board

On the development board, test results are the same as we predicted.

IV Discussions and Conclusion

IV: i Discussions

We noticed the simulation is running in a relatively slow speed, and it takes a while to simulate the situation in one second. This significantly slowed down our speed in debugging the circuit. We temporarily change its designed output frequency higher so we can observe its behavior faster. This trick turned out to be very helpful in the next experiment when the counter no longer our main focus.

In addition, we met the problem of assigning values with the wrong carry system. This caused huge issues while we were testing the code, the output waveform plot was not shown as we expected. It could not be found in log files because it isn't a typo, which cause to us wasting lots of time on debugging.

IV: ii Conclusion

In this experiment, we learned more about the difference between synchronized flip-flops and unsynchronized flip-flops, their reactions are very different in the way when the input is received.

V Appendices

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

Referências

[1] pikipity, Aug, 2024, "FPGA-Laboratory/../3_Flip_Flop," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/blob/main/Lab2/3_Flip-Flop/RTL/asyn_flip_flop.v

[2] pikipity, Aug, 2024, "FPGA-Laboratory/../3_Flip_Flop," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/blob/main/Lab2/3_Flip-Flop/Sim/tb_syn_flip_flop.v

Capítulo V

Flowing Light

I Introduction

I: i Purpose

In this experiment, we want to design a flowing LED circuit, using the concept of top-down design.

We will learn further about the differences between synchronized and unsynchronized flip-flops. Moreover, during the experiment, we will learn how to use shift operands to shift bits.

I: ii Design and Key-results

We want to implement a circuit that drives a 4-LED flow light using Verilog for the FPGA development board. Each LED will be light up for a second, and then it turn off, and next LED will be light up

We using the idea of top-down design, and the final circuit contains three modules, a slowdown module that generates a 1 Hz clock, a shift counter module that represents 4 states of the LED lights, and an LED driver module that lights up four LEDs based on the value of the shift counter.

II Materials and Methods

II: i Structure Block Diagram

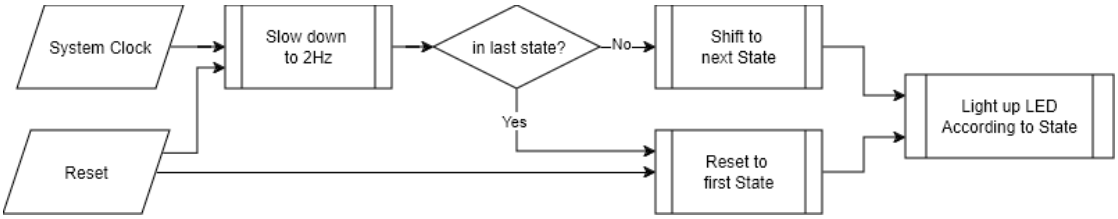


Figura 11: Structure Diagram

As figure 11 shows, the system first reduces the frequency to 1 Hz using the counter module from the previous experiment. Then the slow-downed clock signal is used to drive the shift counter, which shifts the cycle state of the LED lights.

II: ii Signal Waveforms

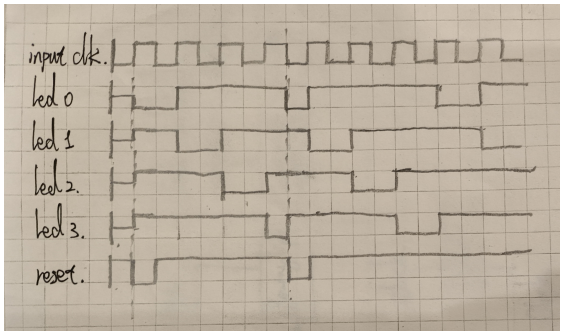


Figura 12: Designed Waveforms

As figure12 shows, once the signal of RESET goes downward, the LED_0 starts lighting up. Then after one input clk, bits shift to the next state. Until LED bits shift to the last LED light, then it starts over from LED_0.

II: iii Resources in Board

The whole circuit requires two inputs and four outputs, they are sys_clk, sys_rst_n, and led_0 to led_3 respectively. The output is displayed on the development board using LED light. And there is a register time_counter used to record time and a 1 Hz Clock.

In this experiment, we used $FPGA\_CLK$, $RESET$, and LED_0 to LED_3 on the development board.

III Results

III: i Testbench

In this experiment, the testbench was built to accept two inputs: CLK and RESET. They are generated with values between 0 and 1 randomly. The outputs were presented by the waveform. The expected results are listed below:

1. RESET is pressed, Time_counter and LED are set to the initial state.
2. RESET is unpressed, all signals remain.
3. Time_counter counts down to zero, the state of LED light shifts to the next state.

III: ii Simulated Result

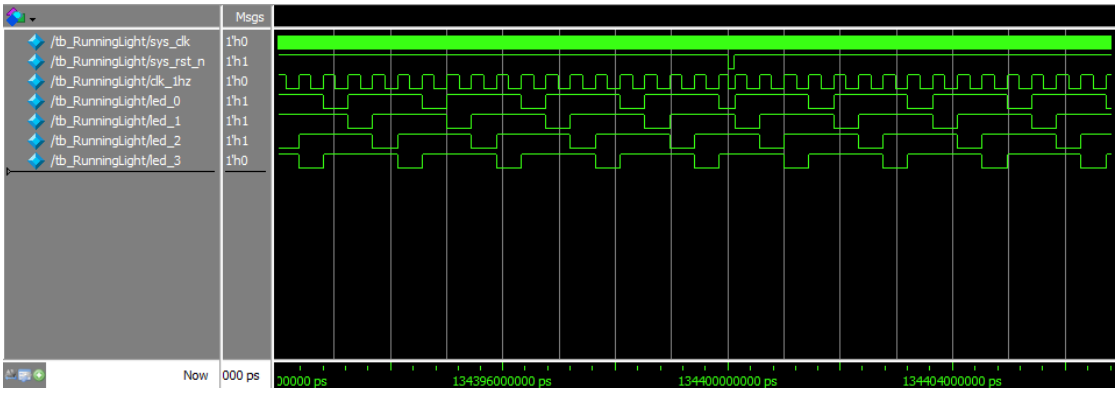


Figura 13: Waveform of Testbench

III: iii Development Board

In this experiment, the test failed to take on-board. The test situation will be discussed below.

IV Discussions and Conclusion

IV: i Discussions

In this experiment, we encountered lots of problems during coding and testing.

We met the problem of assigning value, the error code 10028 occurred when we tried to set an initial value for all LED lights, then in the if-else judgment to decide the signal of each LED, the issue of multiple constant drivers for the net. We solved the problem after removing the initiation part, and then we set the reset key as an initiation and start key.

IV: ii Conclusion

In this experiment, we successfully design the flowing light driver and tests it using the simulator.

We have learned and practiced the workflow of top-down design, split this drive into a time-counter module that slows down the frequency of the clock signal, and a main module that change the state of the LEDs once received a clock singal. Through out this experiment, we learned further about the difference between synchronized flip-flops and unsynchronized flip-flops. We also learned how to use the shift-bit operand. Last but not least, we learned how to do a top-down design, and how to split files from a main file, making it more modulized.

V Appendices

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

Referências

[1] pikipity, Aug, 2024, "FPGA-Laboratory/../3_Flip_Flop," distributed on Github, https://github.com/pikipity/FPGA-Laboratory/blob/main/Lab2/3_Flip-Flop/Sim/tb_syn_flip_flop.v