EEE205 – Digital Electronics II

Lab Report

Student Name: *Yukun.Zheng22*

Student ID: *2251625*

Submission date: *20/04/2025*

ABSTRACT:

This lab introduces digital system design using FPGA with Altera Quartus II and DE1 development board. Activities involve schematic-based design with 7448 BCD-to-7-segment decoder, AHDL-based design of a 7-segment decoder, and design of counters and frequency dividers with sequential logic. Simulations were conducted with the waveform editor to support functional verification before FPGA programming. Hardware testing was completed with successful display output verification and timing. This lab helps improve understanding of graphical design as well as code-based digital design methodologies.

Contents

1. INTRODUCTION

1.1 Background

As digital systems continue to diversify and increase in complexity, traditional methods like Application Specific Integrated Circuit (ASIC) falls short in long developing period and limited flexibility. However, rapidly evolving technology Field Programmable Gate Array (FPGA) has gained wide applications in modern embedded computer system due to its high flexibility, strong reprogrammability. Quartus II developed by Altera provides college students with efficient developing environment, ensuring students to be familiar with the whole process from software circuit design to hardware verification. Through this lab, students can understand the significance of FPGA-based development and acquire foundational knowledge in preparation for the Year 4 module EEE339.

1.2 Experiment

This experiment is designed to guide students step by step through a complete FPGA-based digital system design process, covering core topics relevant to future EDA (Electronic Design Automation) engineering careers. The lab consists of three main phases.

In the first phase, students use the Block Diagram Editor in Quartus II to develop a schematic involving the dec7448 integrated circuit (IC) and a seven-segment decoder on the DE1 board. Through this, they connect software design with hardware output by verifying the actual board behavior.

The second phase is AHDL implementation. Instead of directly searching for chips in Quartus, students need to write AHDL codes to compile and self-design the logic and then observe the reactions of hardware.

The last phase focuses on sequential logic design, students will use D FF (Flip-Flop) to construct mod-10 / mod-12 counter respectively, then cascade two mode-12 counters together to finish multi-digits display on the board.

1.3 Organization

This report is divided into 5 chapters.

- ✧ Chapter 1 **INTRODUCTION** outlines the structure of the report and provides a brief overview of the contents of each section.
- ✧ Chapter 2 **METHOLOGY** records the key techniques in each sub-lab, including circuit schematic drawing, AHDL coding, simulation setting, and pin assignments.
- ✧ Chapter 3 **RESULTS** records all the screenshots required by the lab manual with brief descriptions.
- ✧ Chapter 4 **DISCUSSIONS** first analyzes the challenges encountered during the lab, and then provides comparisons and improvement suggestions.

- ✧ Chapter 5 **CONCLUSIONS** effectively summarizes the acquisition and basis for future study.

2. METHODOLOGY

2.1 General Information

The lab mainly covers the contents of FPGA digital system design, including schematic drawing, AHDL coding, functional simulation, and hardware testing. Students will use Altera Quartus II as their software and DE1 developing board equipped with Cyclone II FPGA chip as their hardware to design digital systems.

2.2 Schematic Design & Simulation

In the beginning of section 2, students draw the circuit schematic based on 7448 BCD-to-7 segment decoder first, then assign the input and output pins to control the digital component on the DE1 board. Note that three control pins LTN, RBIN, BIN should be set 1 to function the decoder.
After that, students should create a ***.vmf*** file to simulate the waveform. Since dec-7448 is active-low in default, the simulation result appears inverted. By setting the properties of dec-7448's outputs "inversion", the re-simulation waveform becomes correct, and the schematic automatically updates with inversion bubbles to indicate NOT logic.

2.3 AHDL Combinational Design

In section 3, students learn to use AHDL language taught in class to implement the 7-segment LED, trying to observe the hexadecimal outputs from 0 to F. Students are suggested to use "TABLE" keyword in their coding.
After finishing encoding, students have to generate the symbol and insert into ***.bdf*** file, then assign the I/O pins to the switches in order to control the LED. Subsequently, they create ***.vmf*** file as they did in section 2 to simulate the waveforms, checking their understanding that that the output matches the expected hardware behavior.

2.4 AHDL Sequential Design

Section 4 focuses on the application of D Flip-Flop, counter, and frequency divider. Students first construct a mod-10 counter with enable and clear function by AHDL, and then extend this concept to design a mod-12 counter that uses the RCO (Ripple Carry Output) signal for cascading.
To perform "jumping per second" on DE1 board, students design a second-pulse generator module (sec_cnt) that divides the 27 MHz system clock. Finally, students finish building the system by cascading two mod-12 counter.

2.5 Hardware Testing

After completing the design of each stage, students have to load their computer design into FPGA board. Since students have assigned different pins on their design, they first turn on and off different physical switches on the DE1 board to simulate the binary inputs, then verify whether the outputs of the 7-segment LED coincide with the simulation results. During the lab, students test the combinational logic and sequential logic for the counter to ensure functional correctness and proper carry generation in the counters.

3. RESULTS

This section will present the experimental results of each stage, including the software simulation results and the measured effects of the hardware. The content is divided into two parts: 3.1 is the software simulation and design results, and 3.2 is the hardware operation effect on the DE1 board.

3.1 Software Results

3.1.1 Section 2: Schematic Design

This section presents the design process and simulation results of the 7448 digital display decoder circuit drawn based on the Quartus II schematic tool.

① Section 2.4：The design

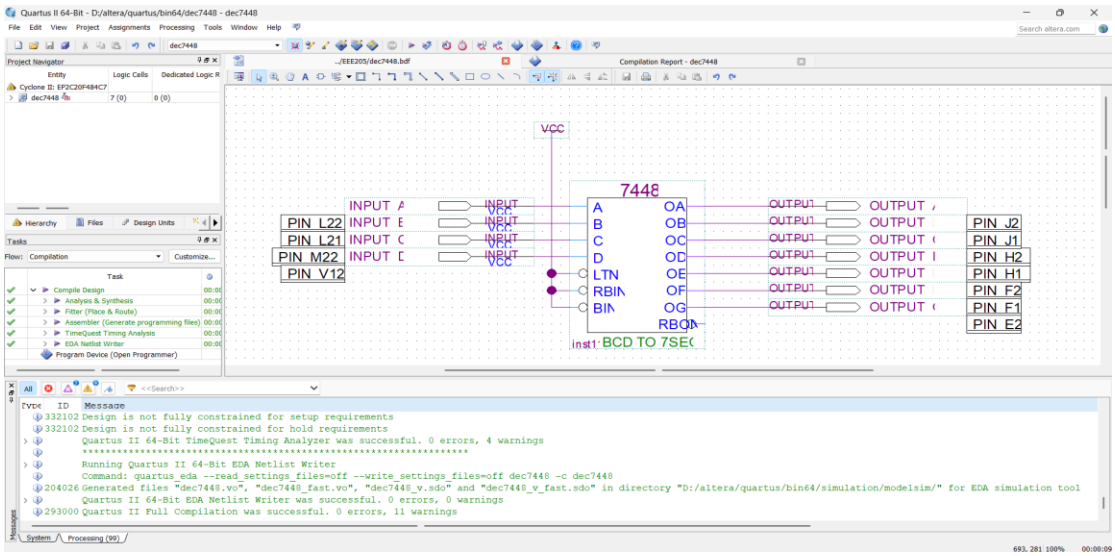
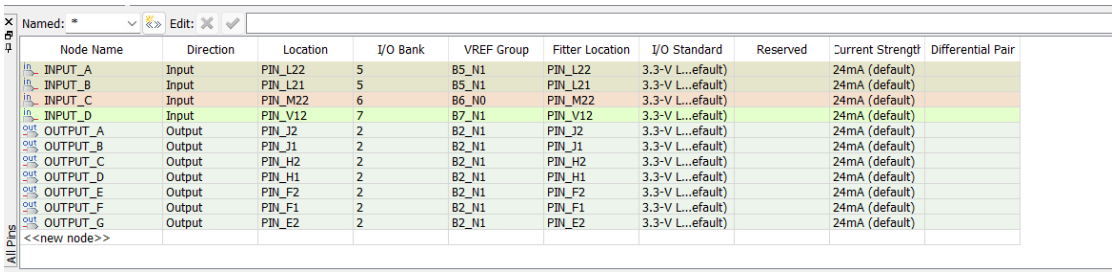


Figure 1: Initial schematic design based on 7448



Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
INPUT_A	Input	PIN_L22	5	B5_N1	PIN_L22	3.3-V L...efault)		24mA (default)	
INPUT_B	Input	PIN_L21	5	B5_N1	PIN_L21	3.3-V L...efault)		24mA (default)	
INPUT_C	Input	PIN_M22	6	B6_N0	PIN_M22	3.3-V L...efault)		24mA (default)	
INPUT_D	Input	PIN_V12	7	B7_N1	PIN_V12	3.3-V L...efault)		24mA (default)	
OUTPUT_A	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V L...efault)		24mA (default)	
OUTPUT_B	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V L...efault)		24mA (default)	
OUTPUT_C	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V L...efault)		24mA (default)	
OUTPUT_D	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V L...efault)		24mA (default)	
OUTPUT_E	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V L...efault)		24mA (default)	
OUTPUT_F	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V L...efault)		24mA (default)	
OUTPUT_G	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V L...efault)		24mA (default)	
<<new node>>									

All Pins

Figure 2: Decoder 7448 pin assignment

The above two figures show the original decoder structure drawn using the Quartus II graphics editor, without the logical inversion setting at the output end. Since the font of I/O blocks is too large to show the full name of the terminal, the second figure is introduced to demonstrate our design.

② *Section 2.6：The simulation waveform*

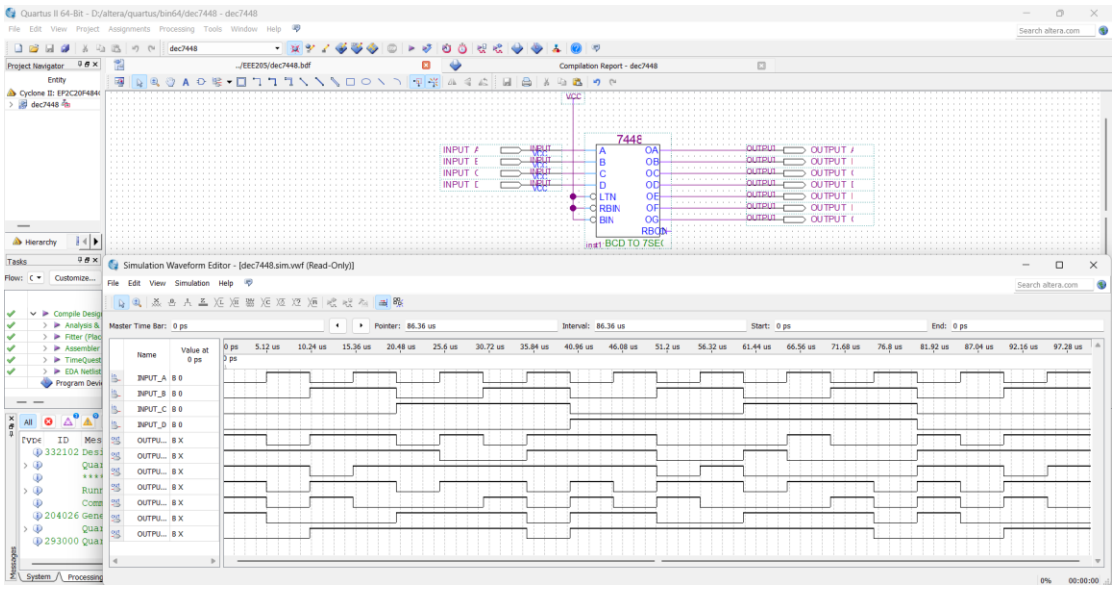


Figure 3: Original simulation waveform (not inverted, incorrect output)

Since the default configuration of the output of 7448 is active-low, the simulation results do not conform to expectations, and some digital segments are displayed abnormally.

③④ Section 2.7：The modified design & re-simulation

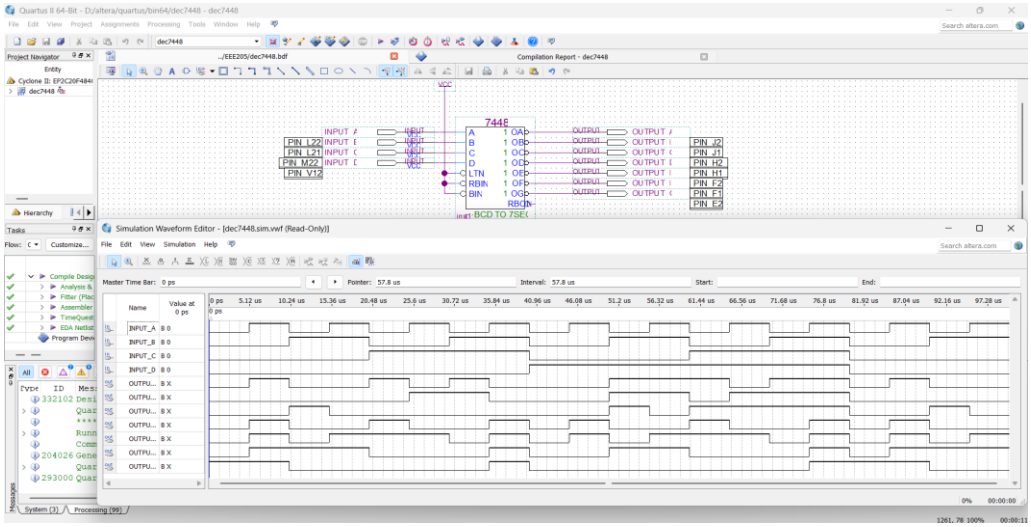


Figure 4: Modified schematic design (inverted outputs) & Simulation waveform

After inverting the output terminals, the simulation waveform of the seven-segment digital display was consistent with the expectation, and the digital display has been correctly lightened up.

3.1.2 Section 3.9: AHDL Combinational Logic Design

This section presents the design and simulation process of building a 7-segment decoder module from scratch using the AHDL language, covering the complete process from code writing, module encapsulation to functional simulation verification.

⑤ 7segment.tdf：AHDL codes

```
SUBDESIGN 7segment
(
    i[3..0]                 : INPUT;
    a, b, c, d, e, f, g : OUTPUT;
)
BEGIN
    TABLE
        i[3..0] => a, b, c, d, e, f, g;
        H"0"    => 1, 1, 1, 1, 1, 1, 0;
        H"1"    => 0, 1, 1, 0, 0, 0, 0;
        H"2"    => 1, 1, 0, 1, 1, 0, 1;
        H"3"    => 1, 1, 1, 1, 0, 0, 1;
        H"4"    => 0, 1, 1, 0, 0, 1, 1;
        H"5"    => 1, 0, 1, 1, 0, 1, 1;
        H"6"    => 1, 0, 1, 1, 1, 1, 1;
        H"7"    => 1, 1, 1, 0, 0, 0, 0;
        H"8"    => 1, 1, 1, 1, 1, 1, 1;
        H"9"    => 1, 1, 1, 1, 0, 1, 1;
        H"A"    => 1, 1, 1, 0, 1, 1, 1;
        H"B"    => 0, 0, 1, 1, 1, 1, 1;
        H"C"    => 1, 0, 0, 1, 1, 1, 0;
        H"D"    => 0, 1, 1, 1, 1, 0, 1;
        H"E"    => 1, 0, 0, 1, 1, 1, 1;
        H"F"    => 1, 0, 0, 0, 1, 1, 1;
    END TABLE;
END;
```

Figure 5: The AHDL source code structure of the 7-segment module

This figure shows the source code structure of defining the mapping relationship between the input i[3..0] and the output segment codes a~g using the TABLE statement taught in lecture.

⑥ hexdec.bdf: Top-level module encapsulation diagram

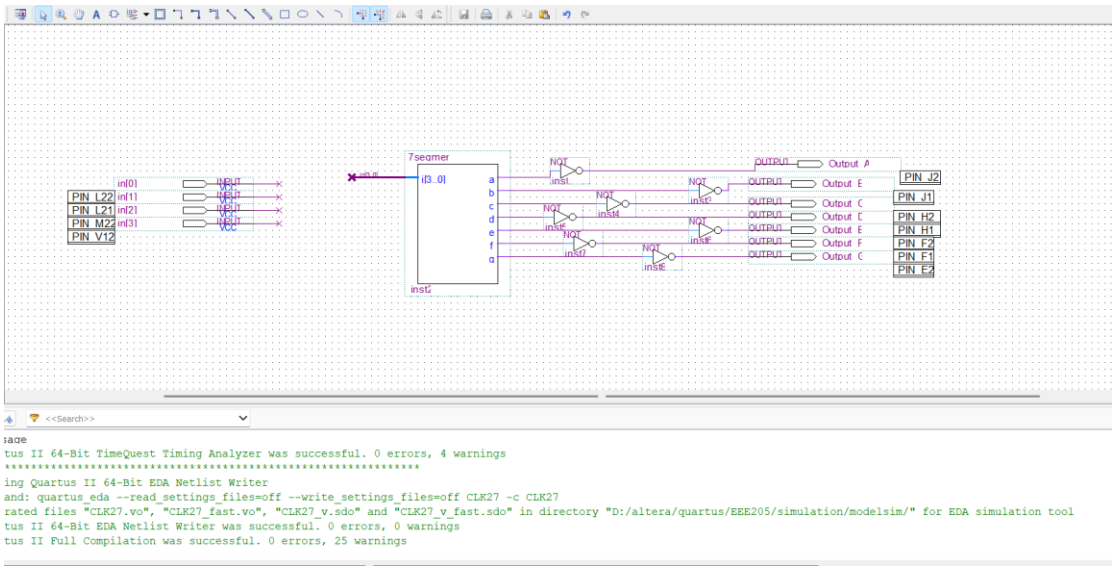


Figure 6: Top-level schematic structure of hexdec.bdf

After creating the 7-segment module as a symbol, the top-level schematic connection diagram constructed is used to drive the actual DE1 display device.

⑦ Simulation of hexdec

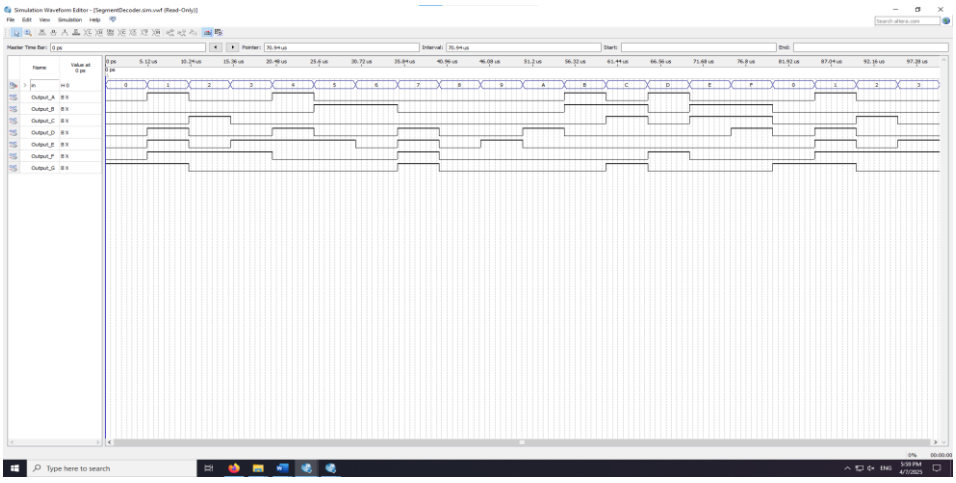


Figure 7: Simulation Waveform of hexdec module Function (Input i[3.0])

The figure above successfully shows the changes of the seven-segment output when the input i[3..0] changes from 0 to F to verify the decoding logic in simulation.

3.1.3 Section 4: AHDL Sequential Logic Design

Section 4 shows the implementation and simulation process of AHDL for the counter and frequency divider module, including the design of mod-10 and mod-12 counters, as well as the generation of one-second pulses and the cascaded display of dual modules.

⑧ dec_count.tdf: Decimal counter

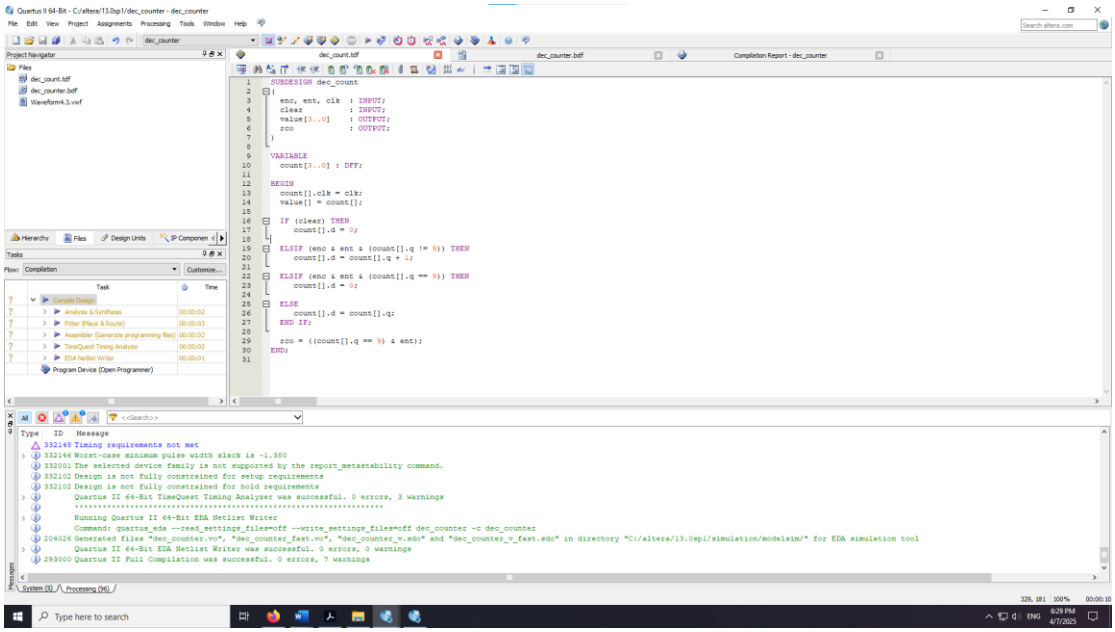


Figure 8: AHDL source code structure of the dec_count module

The code here correctly includes functions of the enable, clear control logic and rco output performed in the circuit.

⑨ bdf of Section 4.3：Top-level structure of dec_count

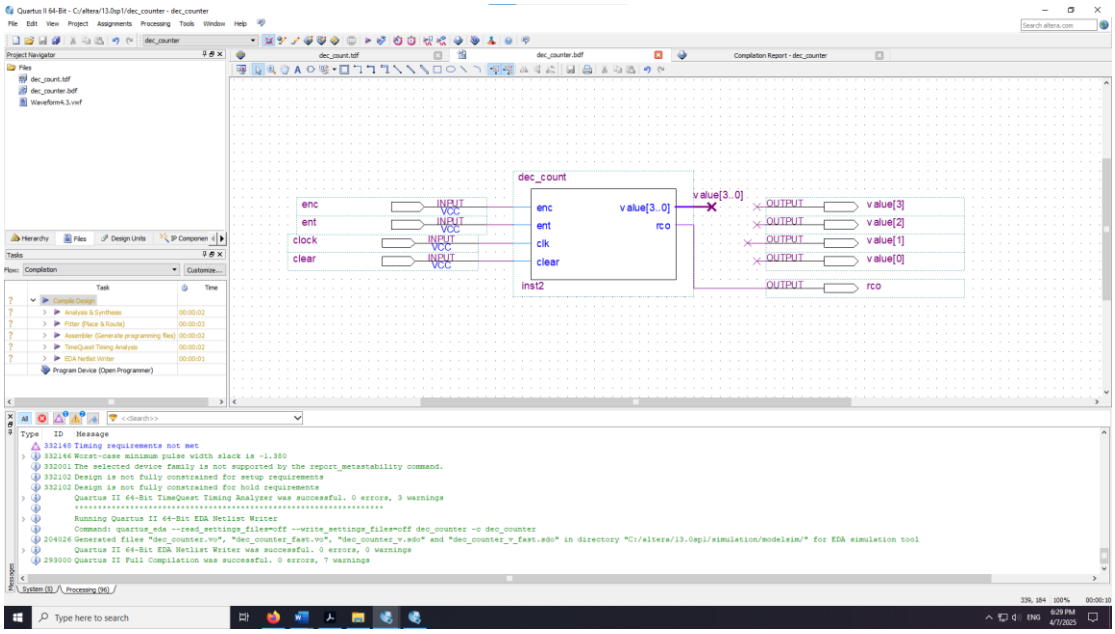


Figure 9: Top-level structure diagram of the counter module

The top-level circuit structure after encapsulating the dec_count module is presented, which is used to connect the switch input and the seven-segment digital display output.

⑩ Simulation of Section 4.3

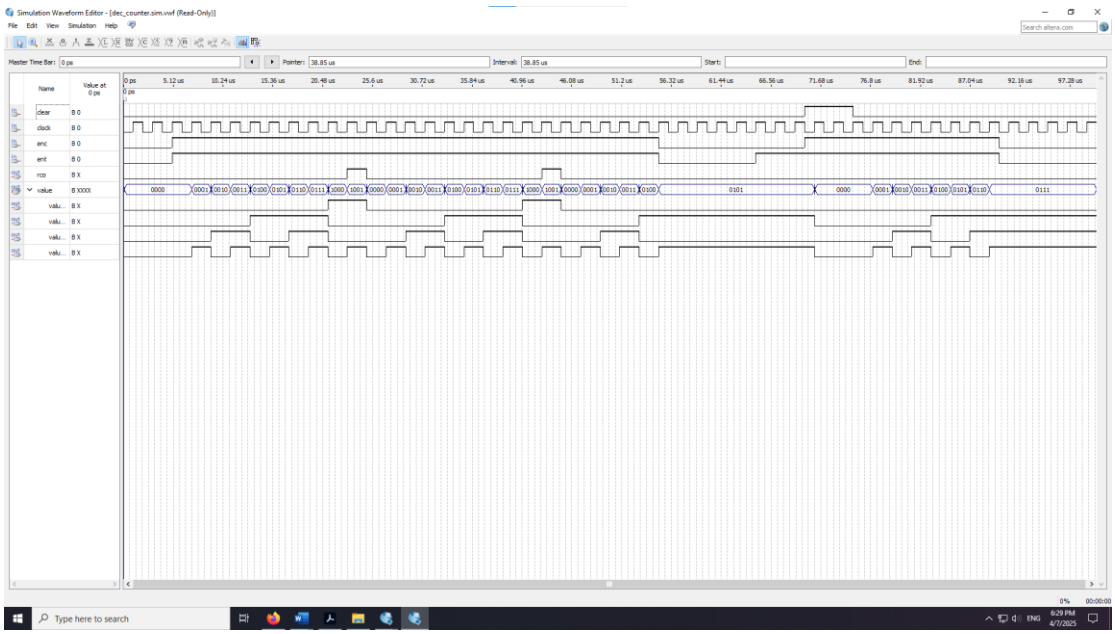


Figure 10: Simulation waveform of mod-10 counter function

The counting process from 0 to 9 and the rco signal flipping are displayed to verify the functional correctness of the counter.

⑪ sec_cnt.tdf：Frequency divider in Section 4.4

```
SUBDESIGN sec_cnt
(
  clk     : INPUT;
  second  : OUTPUT;
)

VARIABLE
  count[25..0] : DFF;

BEGIN
  count[].clk = clk;

  IF (count[].q == 27000000) THEN
     count[].d = 0;
     second = VCC;

  ELSE
     count[].d = count[].q + 1;
     second = GND;

  END IF;
END;
```

```
SUBDESIGN sec_cnt
(
  clk     : INPUT;
  second  : OUTPUT;
)

VARIABLE
  count[3..0] : DFF;

BEGIN
  count[].clk = clk;

  IF (count[].q == 5) THEN
     count[].d = 0;
     second = VCC;

  ELSE
     count[].d = count[].q + 1;
     second = GND;

  END IF;
END;
```

Figure 11: A comparison diagram of the AHDL source code of the frequency divider module performed in hardware and software

The left picture shows the version used for hardware download, while the right picture shows the version used for software simulation (*the upper limit of the count is adjusted to 5 to improve the simulation efficiency*).

Since the actual hardware needs to be accurate to 1Hz output, the upper limit of the count of the hardware frequency division module is set at 27,000,000. However, to accelerate the simulation process, the simulation version temporarily sets this value to 5 to shorten the simulation cycle for verifying the function.

⑫ bdf of Section 4.4：The structure diagram containing sec_cnt (without cascaded)

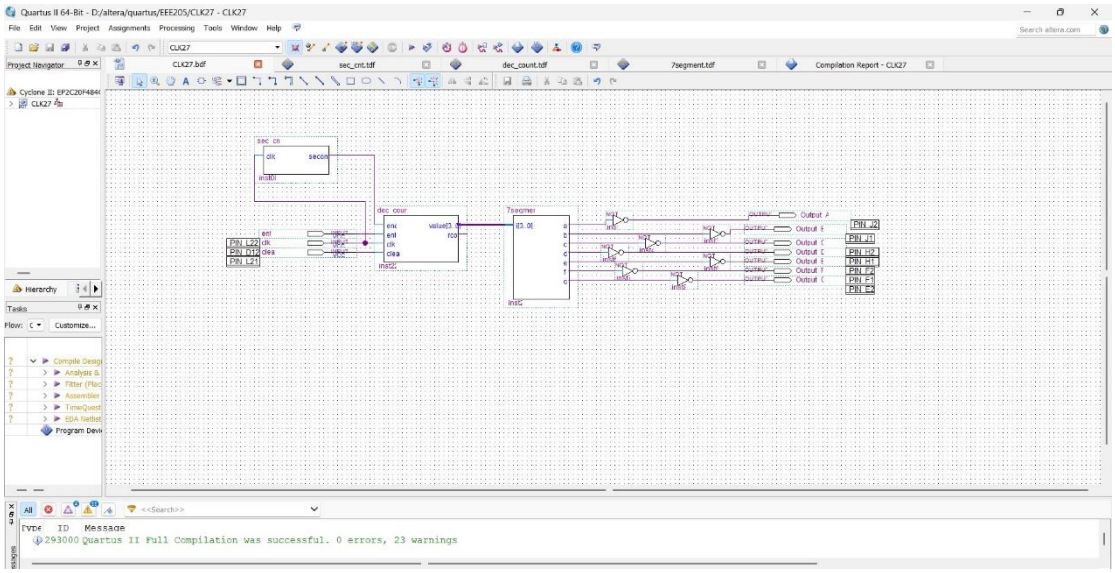


Figure 12: Counter system structure with frequency division control (without cascaded)

The top-level system framework composed of individual counters under the control of the frequency divider is shown in the above figure.

⑬ Simulation of Section 4.4

✧ *Case 1: None-cascaded counter*

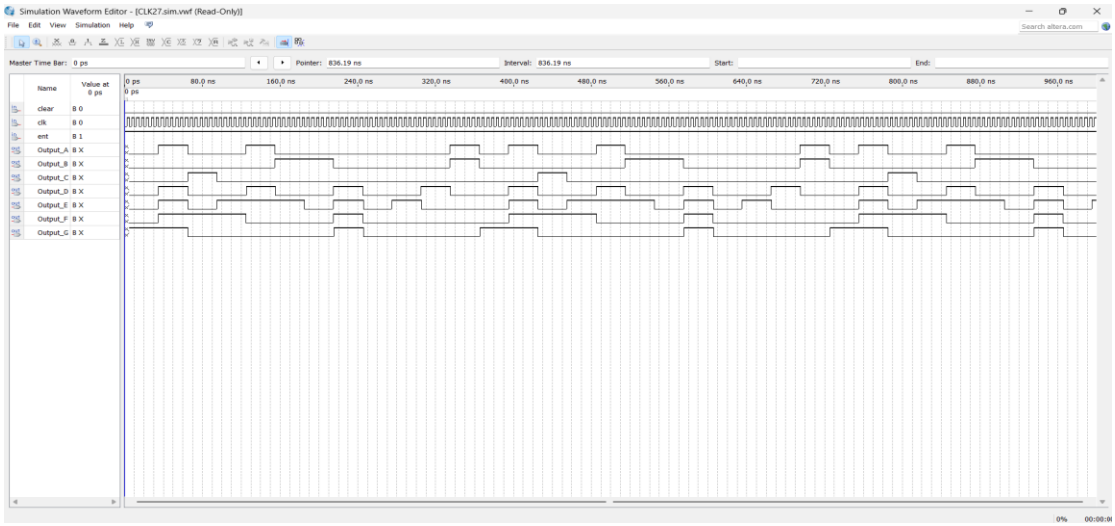


Figure 13: Simulation diagram of the one-second pulse-controlled counter jumping (without cascaded)

The simulation results of the counter jumping once per second under the second pulse control are presented.

✧ *Case 2: Cascaded counter*

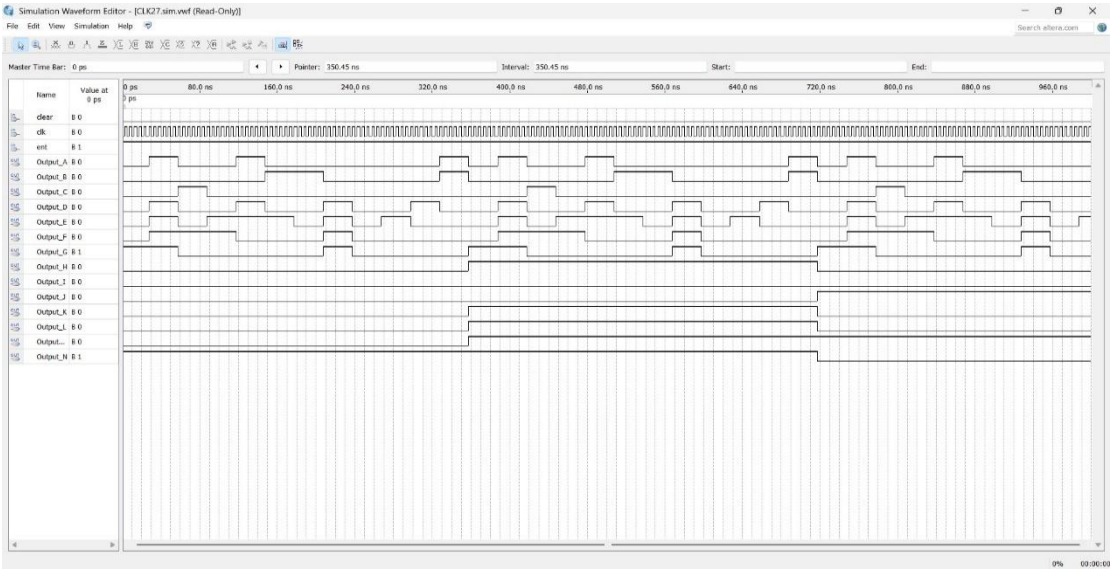


Figure 14: Simulation diagram of dual-mode 12 counter and dual digital displays (after cascaded)

This diagram shows that the two counter modules flip and output in sequence to achieve the double-digit display function.

⑭ bdf for divide-by-12 and dual 7-segment display

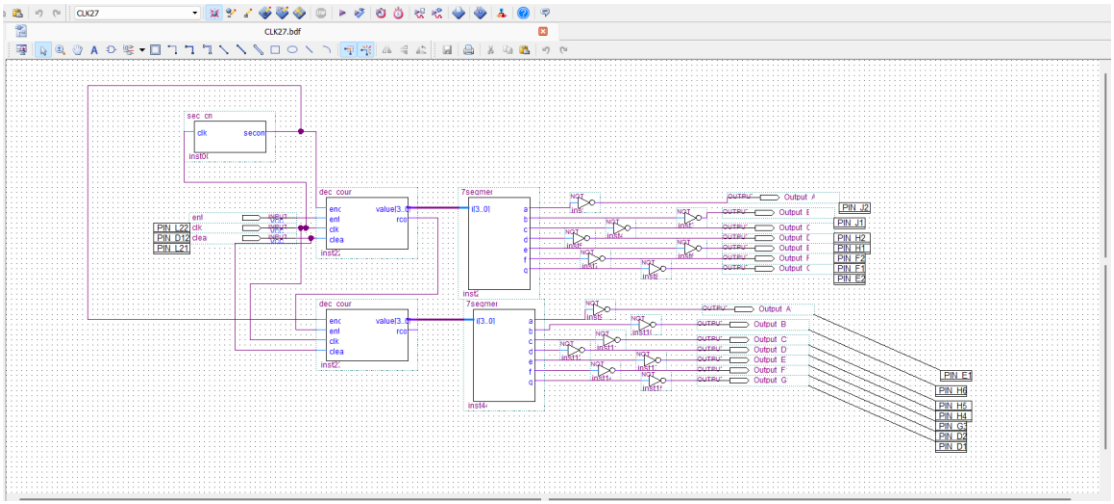


Figure 15: Top-level system structure diagram of dual-mode-12 counter and dual digital displays (after cascaded)

The final overall design structure of the cascade counter is presented, supporting the display function of multi-digits 7-segment digital display.

3.2 Hardware Results

This Section shows the hardware operations of the key parts in the experiment, including the display of the 7448 digital display after inverting the output in the graphical design (Section 2.7), and the dynamic display of the two-bit seven-segment digital display implemented by cascading the MO12 counter (Section 4.4).
Note that the decoder design in Section 3.9 did not shoot the corresponding hardware demonstration video, the pictures of its physical implementation are not included in this section.

3.2.1 Section 2.7

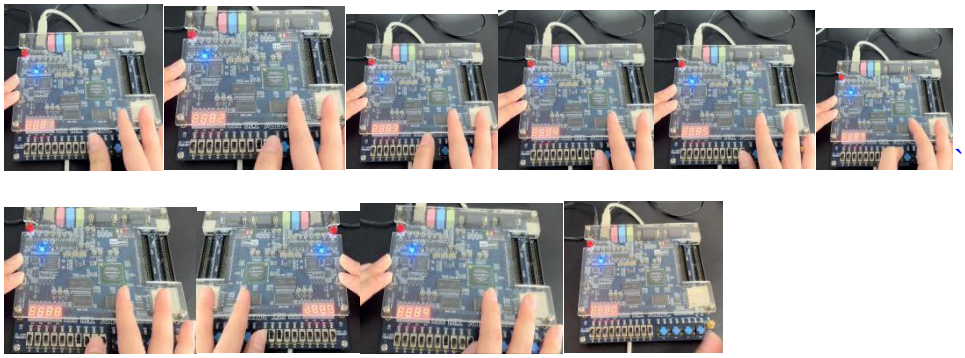


Figure 16: Hardware implementation of the corrected schematic design in Section 2.7

The LED lights of hardware show the modified schematic diagram successfully displaying hexadecimal numbers on the digital display after being downloaded to the DE1 development board and input through the actual operation switch. In the initial experiment, since the 7448 chip has an active-low output and the phase inversion period was not set, the lighting logic was completely reversed. By adding a NOT gate or setting the output pin property to "inverting", after recompiling, the segment lighting logic becomes correct.

3.2.2 Section 4.4

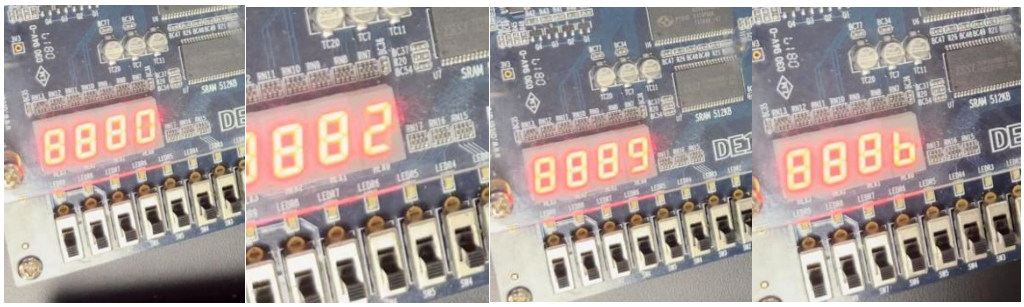


Figure 17: Hardware demonstration of the cascaded dual-mod-12 counters from Section 4.4.

This figure shows the two-bit digital display by cascading two mod-12 counters. The control of carrying in the MSB after LSB is fully counted is achieved by connecting the RCO signal. Combined with the pulse output by the sec_cnt module, the two-digit hexadecimal display with automatic increment per second is realized.

As shown in the figure, the two digital LED display digits HEX0 and HEX1 are updated in turn. This figure reflects the result of the collaborative work of the sequential logic module and also demonstrates the system integration capability throughout the entire experiment.

4. DISCUSSIONS

This chapter will analyze the problems that occur at each stage of the experiment process, and provide corresponding solutions based on practical operations, further reflecting on the key points in the design logic and engineering implementation.

4.1 Section 2：Schematic Design

✧ *Problem 1: Incorrect use of connection dots in schematic*

In the initial circuit setup, as shown in the figure, when multiple output ports were connected in front of the output pins, the " **purple dot**" node was mistakenly added.

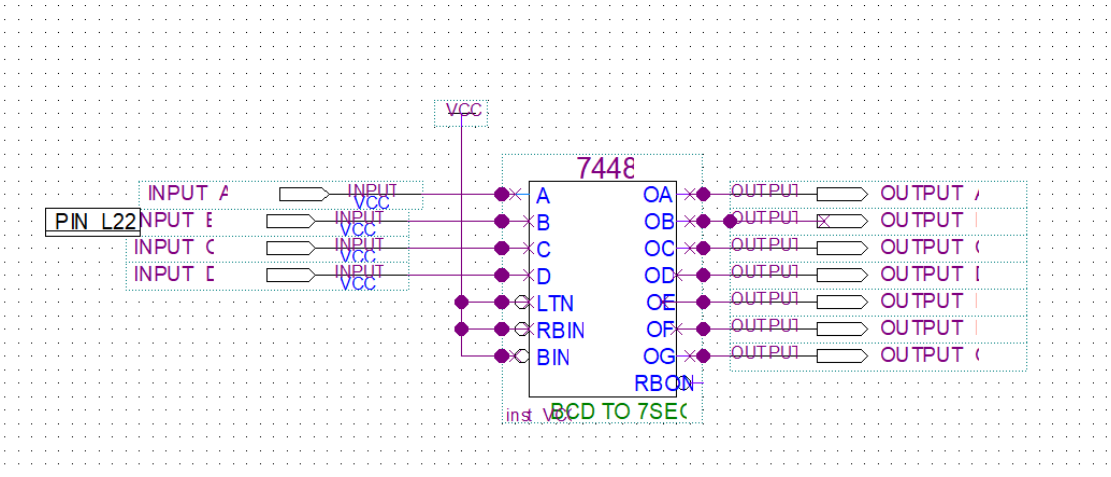


Figure 18: Connection errors related to purple dots

This symbol in Quartus II indicates "multiple lines intersect at this node". However, in the connection scenario of this experiment, the digital 7-segment display signal should be a one-way point-to-point connection and dots should not be used. Adding this node incorrectly will be recognized by the system as multiple output driver signals, thereby triggering a conflict prompt during compilation.

✧ *Solution 1: Removal of redundant dots and signal connection correction*

This problem was discovered only after multiple failed attempts at compilation by checking the schematic diagram and referring to the component connection specifications. Subsequently, by deleting the redundant nodes and ensuring that each signal line is connected to only one set of I/O, the problem was successfully solved.

✧ *Problem 2: Improper project folder structure for simulation*

In the early stage of the experiment, when creating the structure of the project folder, the recommended path structure in the instruction manual was not followed. After wrongly placing the project folder ***under the db folder*** automatically generated by Quartus, the system was unable to correctly identify and access the corresponding design file when conducting ***.vwf*** waveform simulation, resulting in continuous error reports from the simulator.

✧ *Solution 2: Rebuilding project in correct directory structure*

After a long period of fruitless attempts, the problem was solved after rechecking the file structure and recreating the project in a **parallel directory** of the ***db folder***. This experience also reminds us that in the development of digital systems, the file structure management of the toolchain cannot be ignored either. A visual comparison between the correct and incorrect project folder structures is shown in Figure 18.

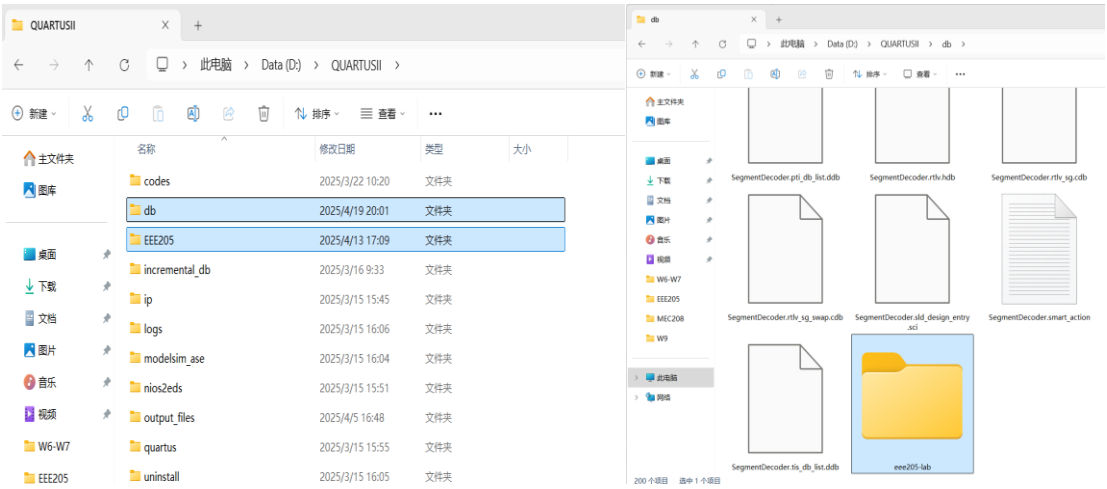


Figure 19: Comparison of correct and incorrect folder placement in Quartus project

The left screenshot shows the correct setup where the project folder is created **parallel** to the ***db folder***. The right screenshot shows an incorrect setup where the project folder is mistakenly placed ***inside the db directory***, which causes simulation errors.

4.2 Section 3：AHDL Combinational Logic Design

✧ *Problem 1: Failure to call symbol in top-level diagram*

After completing the writing of the AHDL file and attempting to generate module symbols, it was found that no matter how the search was conducted, the corresponding 7-segment module could not be added in the graphic editor. At the beginning, it was suspected that the operation step of "Create Symbol Files for Current File" was not executed correctly, resulting in the symbol not being generated successfully.

✧ *Solution 1: Standardizing naming to match Quartus symbol requirements*

After several failed attempts and consulting the teaching assistant, I noticed the first reminder in the "Important Tips" on the second page of the experimental manual: The project name and the folder name must be consistent, and neither of them can be the same as the AHDL file name. Due to not reading the prompt carefully at the beginning, in the actual operation, there was a partial overlap between the project name and the AHDL file name, which caused Quartus to be unable to index the module correctly and fail to search for the corresponding symbol. After recreating the folder and standardizing the naming, the module was successfully generated and inserted into the top-level graphic diagram, and the problem was solved.

4.3 Section 4：AHDL Sequential Logic Design

✧ *Problem 1: Unmodified counter value caused simulation failure*

When initially simulating the timing system, it was not realized that the upper limit of the count in the frequency divider module still remained at 27,000,000 required by the hardware. Under this setting, the simulator needs to perform tens of millions of clock cycles to output a high-level pulse, which imposes a huge burden on the processing capacity of the simulation system, resulting in a long period of unresponsiveness during the simulation process and the inability to observe any effective waveforms. Additionally, the wrong waveform by applying 27 MHZ is presented as below for better illustration.

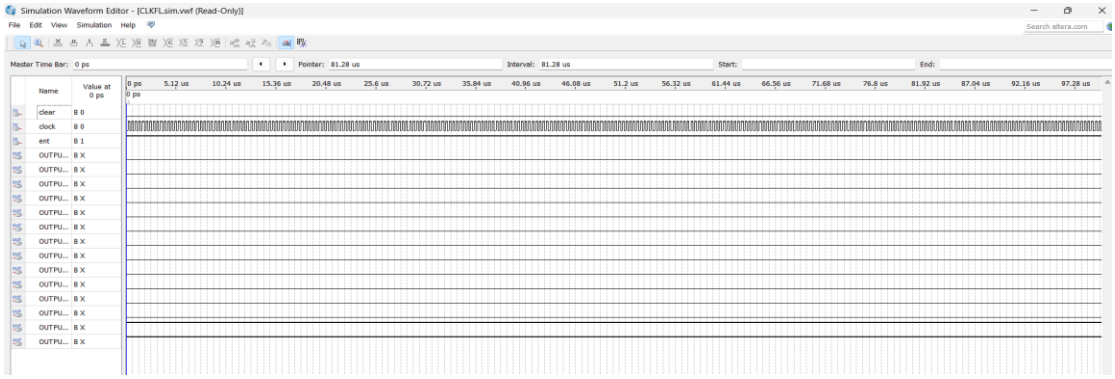


Figure 20: Wrong simulation waveform applying 27MHZ in software

✧ *Solution 1: Reducing count limit for simulation optimization*

After repeated simulations with no results for this problem, by referring to the relevant instructions in the experimental manual and consulting the teaching assistant, it was learned that in the simulation environment, the upper limit value of the counter should be temporarily modified to a smaller value 5 to shorten the simulation cycle, accelerate waveform generation, and facilitate the observation of system behavior and verification of logical functions.

The reason why the upper limit of the count is set at 27,000,000 in the hardware design is that the DE1 board uses a 27MHz system clock. To generate a 1Hz pulse signal, a complete count must be made once. However, if this parameter is still used in the software simulation, the simulator will need to go through a large number of clock cycles, resulting in no output of the waveform for a long time.

5. CONCLUSION

This experiment was carried out around the basic process of FPGA digital system design. Based on the Quartus II software platform and the DE1 development board, the graphical logic design, AHDL combinational logic modeling and the implementation of the sequential logic system were completed successively.

Through experiments, I have mastered how to flexibly switch design methods among different design levels. In section 2, graphical design is used to quickly construct the circuit structure. In section 3, the logical functions are finely described through AHDL. Finally, in section 4, the multi-module cascading system is implemented through sequential logic. Meanwhile, the practical problems encountered in the experiment, such as incorrect signal connection, improper engineering structure settings, and low simulation debugging efficiency, have also stimulate me to enhance my ability to locate and solve problems in practice.

Although I plan to become an analog IC engineer in the future and focus on the research of semiconductor devices, this laboratory changed my initial impression: analog systems and digital systems were completely separated. However, I discovered some practical cases like GaN-on-Si and GaN-on-SOI technologies, which GaN power devices are integrated on the same chip with CMOS or FinFET digital controllers. In this type of SoC, a deep N-well and a protection ring must be used to isolate the high-voltage analog module, and the power network must be separated to reduce interference. Moreover, before tape-out of chips, many enterprises will first use FPGA to quickly verify the function. At this time, the block diagram design in Quartus is the pilot version of the chip logic. These design strategies highlight the close interaction between the analog module and the digital module, not only at the functional level but also at the layout level.

This experiment is not merely a training of digital logic; it also provides a valuable entry point for system-level thinking. It makes me realize that mastering FPGA design can not only enhance digital skills, but also lays a crucial foundation for the future development of analog devices.