Xelex®

INTERNSHIP

REPORT

Advisors: Jack Yeh
Dinh Quang Vu

Students: Dinh Xuan Quang

HO CHI MINH CITY, AUGUST 2024

Contents

1 Summary task

Week	Task description
1	Practice using Gowin EDA tool and revise HDL code
2	Work and practice with board Tang-nano 4k
3	FPGA GAO project and static time analysis
4	Research and work with Kicad tool
5	Research schematic and draw PCB-layout for FPGA board part1
6	Research schematic and draw PCB-layout for FPGA board part2
7	Research and work with board GW5A-ACG525 part1
8	Research and work with board GW5A-ACG525 part2
9	Research and work with board GW5A-ACG525 part3

Table 1: *Task summary*

2 Creating project with Gowin EDA tool

- Double-click the Gowin icon to lauch the Gowin software.

Figure 1: *Gowin EDA tool*

- The startup screen in Gowin is divided into 4 sections, as follows:
  - (1) Recent projects: Processes that have been opened recently.
  - (2) Quick Start: Quick Start, including New Project, Open Project, Open Example Project
  - (3) Tools: Tools, including FloorPlanner, Timing Constrains Editor, Programmer (data flow configuration or download)
  - (4) User Manuals: User manual, click to enter the product introduction of the official website of Gowin.

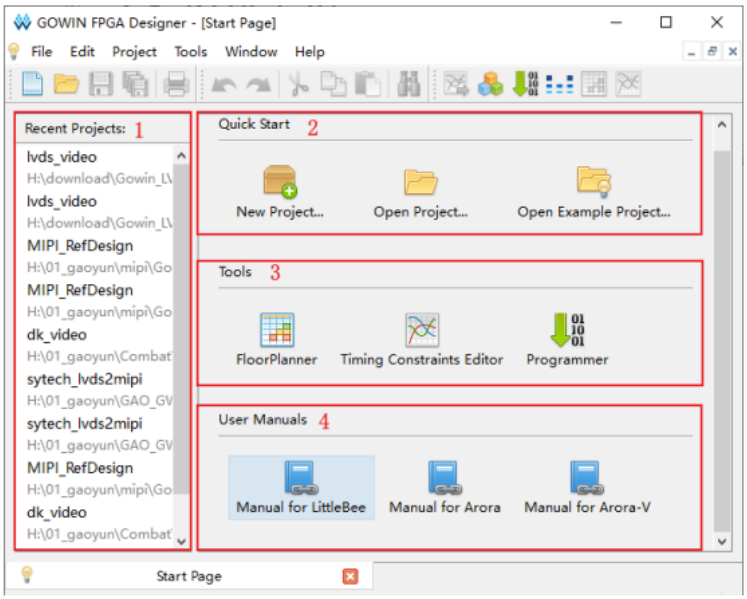


Figure 2: *Gowin EDA tool*

- Click the New Project,the New dialog box will appear, as shown in Figure 3 below, click the OK button.

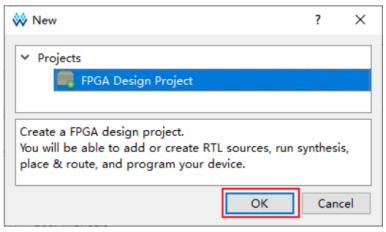


Figure 3: *Creating project interface*

- The Gowin wizard will appear,fill the project name and project path,then click to next.

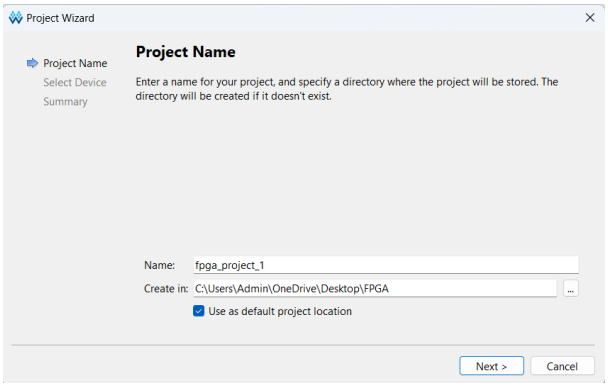


Figure 4: *Filling name and path project*

- The Select Device box will apppear,you can find your specific FPGA chip model used in design base on series,package,device,speed,device version.Then choose your chip and click OK.

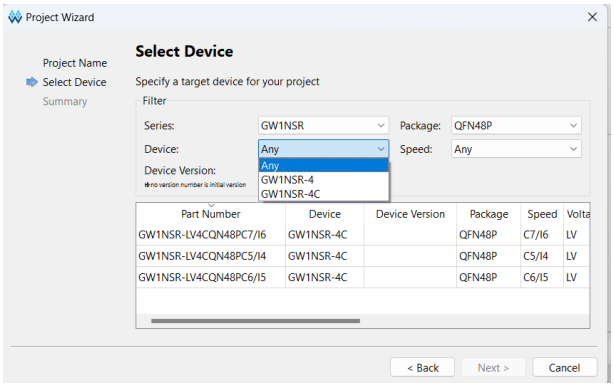


Figure 5: *Select your device*

- The summary of your chip information will appear for you to check it,then click back if you want to change your chip or finish.

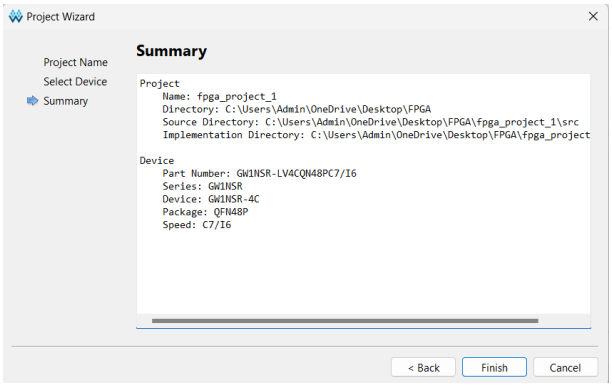


Figure 6: *Summary information*

- Next, you will see the Gowin design interface.

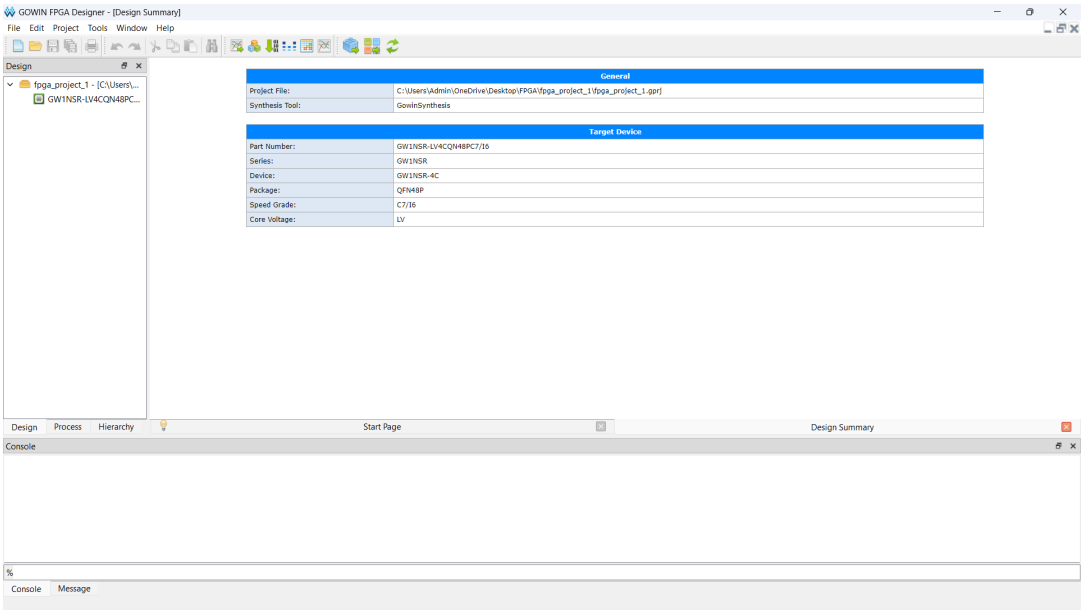


Figure 7: *Gowin design interface*

2.1 Creating verilog file

- Then you will create verilog file by clicking on the toolbar or click File->New,choose Verilog File,then click OK.

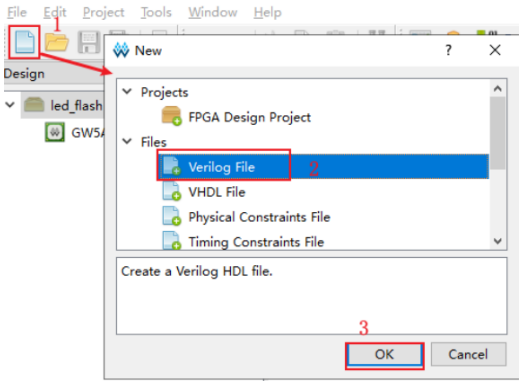


Figure 8: *Add the Verilog source file*

- Fill the name and path of the verilog file then click OK.

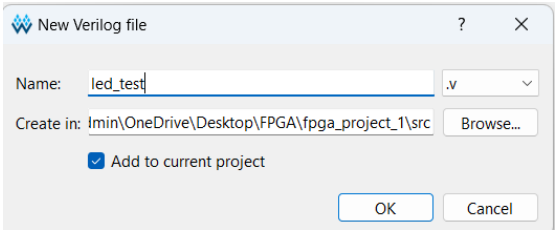


Figure 9: *Fill name and path of verilog file*

- After that, we can see the new folder Verilog Files include file verilog we just create.

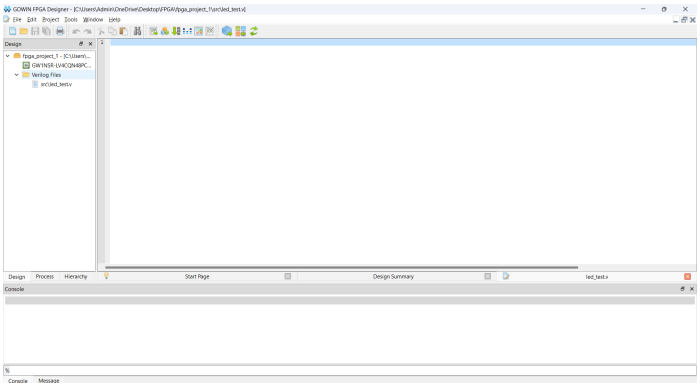


Figure 10: *Verilog design interfaces*

- Coding example:

Listing 1: *LED blinking after 1s deplay example*

```
module led_flash(
input clk,
input rst_n,
output reg led
);
reg [25:0] cnt;
always @ (posedge clk or negedge rst_n)
begin
if (!rst_n)
cnt <= 26'd0;
else if (cnt < 26'd49_999_999)
cnt <= cnt + 1'b1;
else
cnt <= 26'd0;
end
always @ (posedge clk or negedge rst_n)
begin
if (!rst_n)
led <= 1'b0;
else if (cnt == 26'd49_999_999)
led <= ~led;
else
led <= led;
end
endmodule
```

2.1.1 Synthesize

- Using ctrl+S or click file on taskbar and choose save your verilog file.Then click to process like the figure below.Next double-click to Synthesize or click icon run synthesis on the taskbar .If your code has errors there will be icon red on the synthesis and system will show your message error below,else there will be green icon in the syntheis.

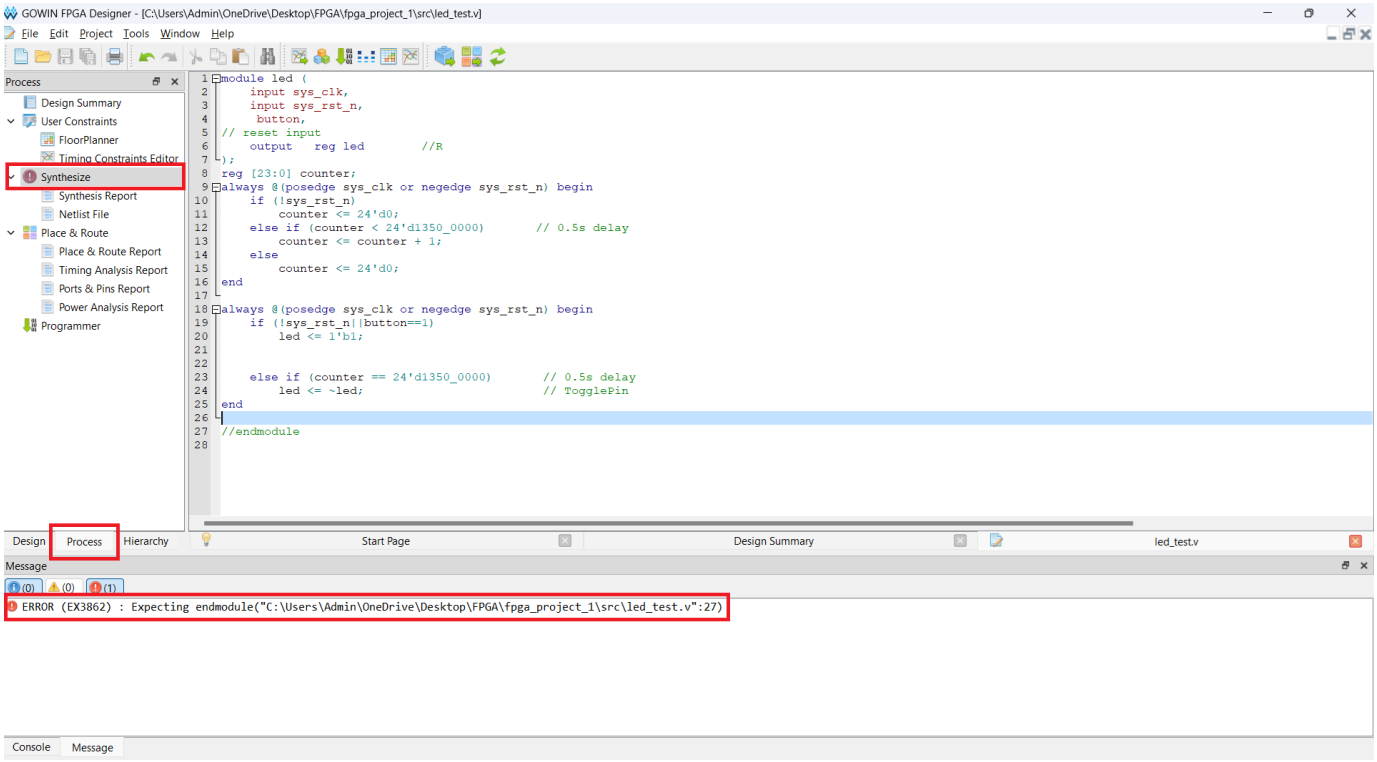


Figure 11: *Synthesis process*

- Consulting:the tool just check syntax errors, if your design have logical errors,system can not show message errors, for example: incorrect connections between modules.However,you are provided provided with the function of viewing RTL design analysis, click Tools->Schematic Viewer->RTL Design Viewer in turn, and then you can see the RTL description and analysis stage.

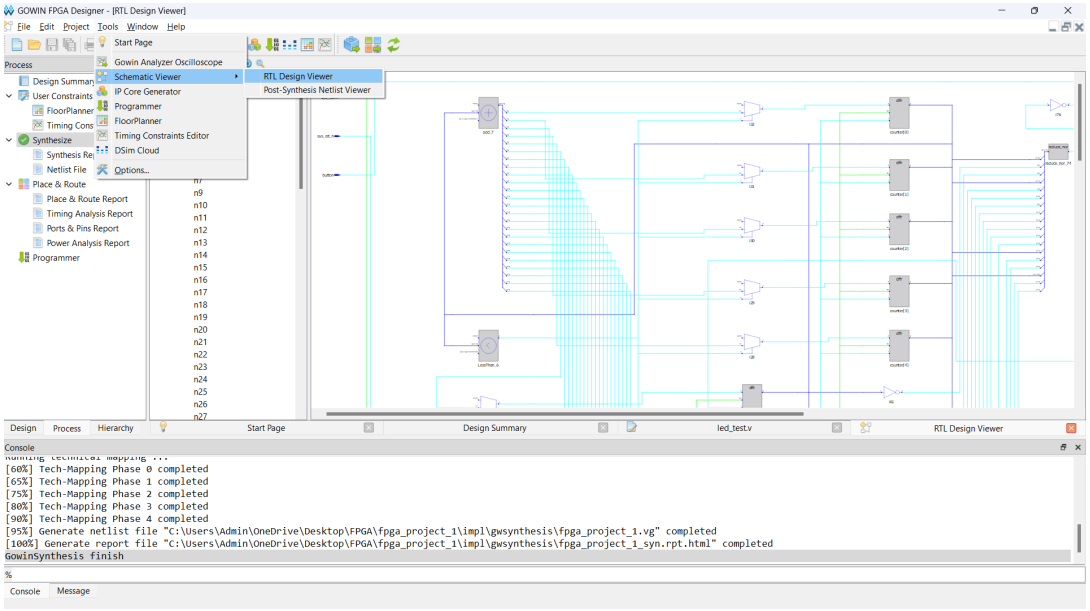


Figure 12: *RTL Design Viewer*

2.1.2 Hierarchy

- You can choose Hierarchy near process to see information about your module include:Regiesters,LUT,ALU,

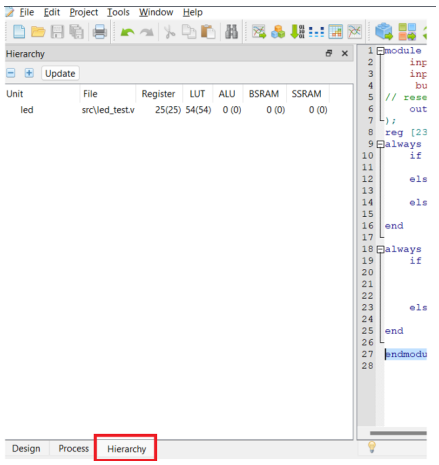


Figure 13: *Hierarchy information*

- In the hierarchy window, we place the mouse over the verilog file, right-click on the file, and set the file as the top-level file, as shown in below Figure.Once this is set, the file name will be bolded, and the synthesis will be re-analyzed

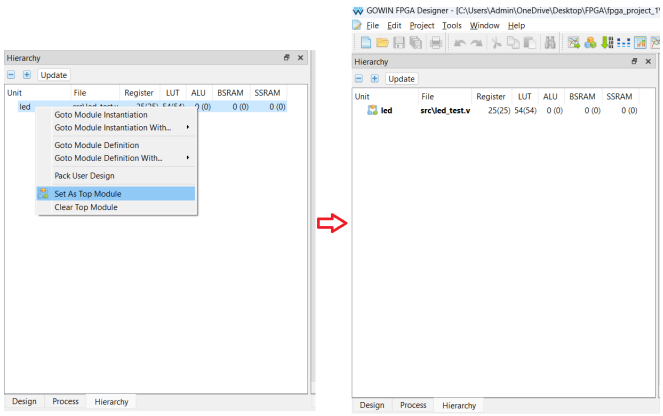


Figure 14: *Hierarchy set*

2.2 Physic constraints

2.2.1 FloorPanner

- Clicking on Process->FloorPanner for physical constraints or clicking directly on toolbar,as shown in figure below

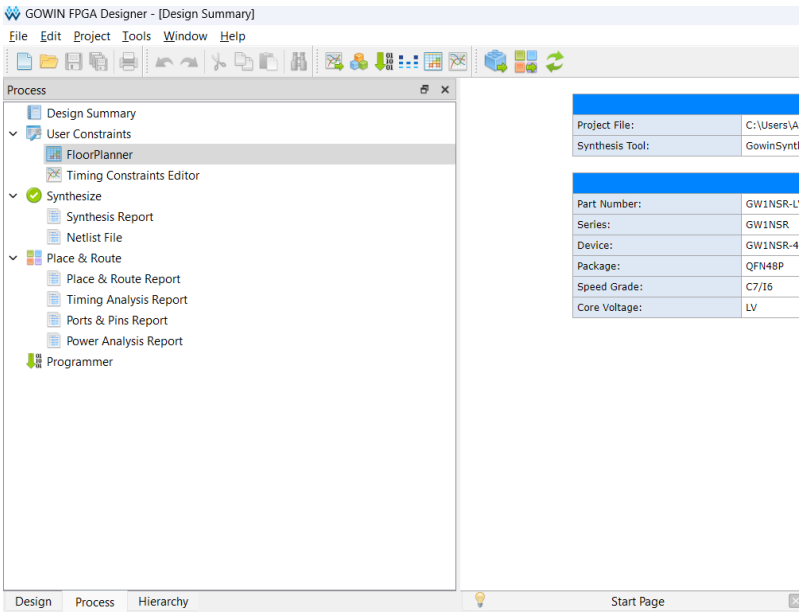


Figure 15: *Choosing floorPanner*

- Then the dialog box will appear, t since we don't create a new .cst file to store the physics constraints, a dialog box will pop up asking if we need to create a .cst file, here we can directly click OK.

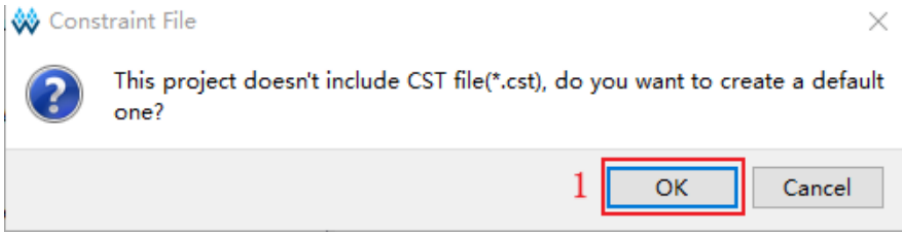


Figure 16: *The Create Physics Constraint File dialog box is displayed*

- Next,go to the FloorPlanner page, click I/O Constraints, perform I/O constraints, and then assign pins and level standards according to your board

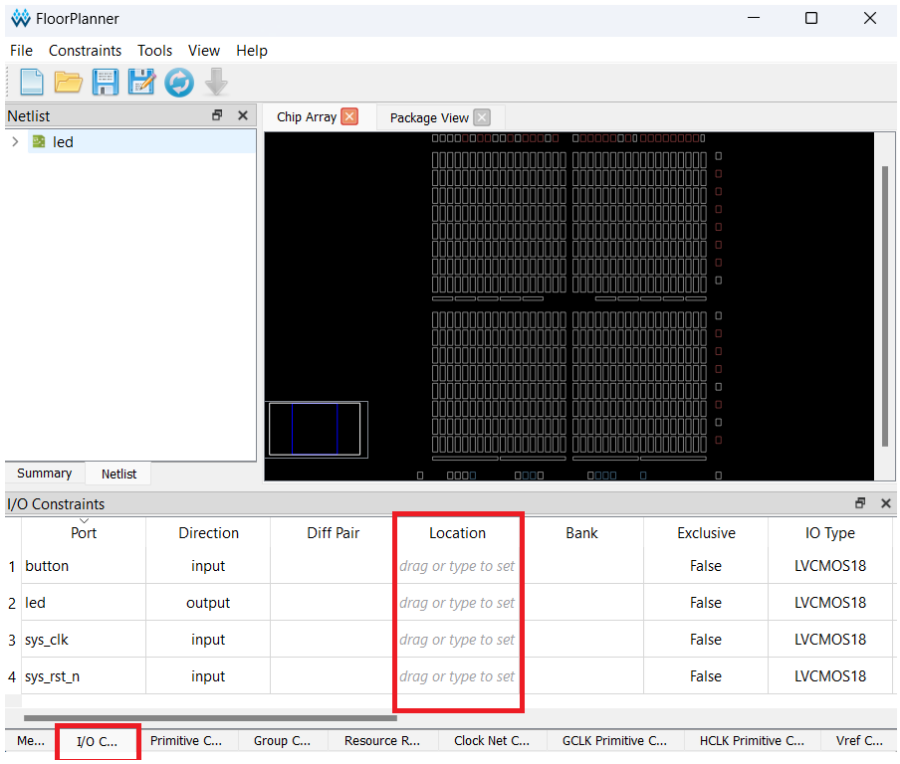


Figure 17: *I/O constraint interface*

- After you fill the location,click Save icon on the taskbar,then a file .cst will be created in Physical Constraint Files folder in design part.

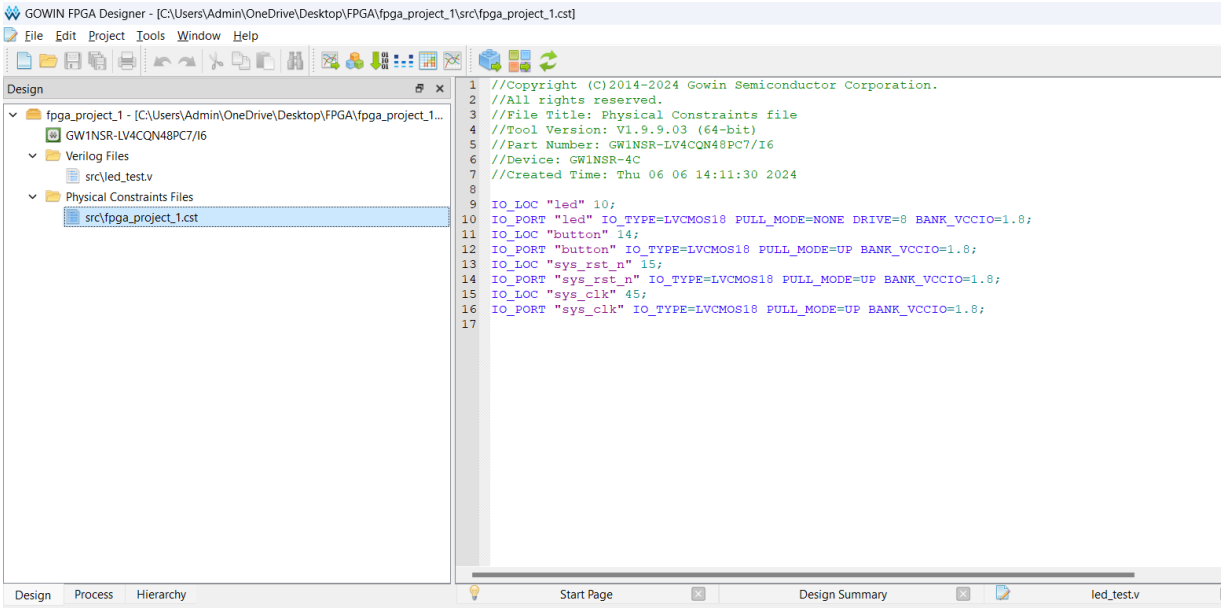


Figure 18: *View the contents of the physical constraint file*

2.3 Place and Route

- We click Process->Place and Route to place and route, after the place-route is successful, we can see "Bitstream generation completed", if it fails, please modify it according to the prompt information given in the Console

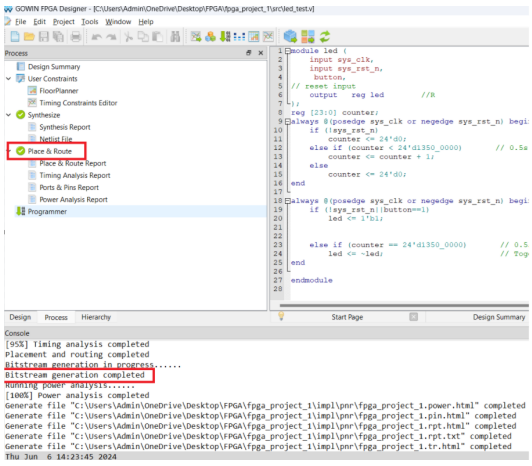


Figure 19: *Place-and-route successful*

2.4 Board-level validation

- Connect your device to your computer, then click on the Programmer in process.

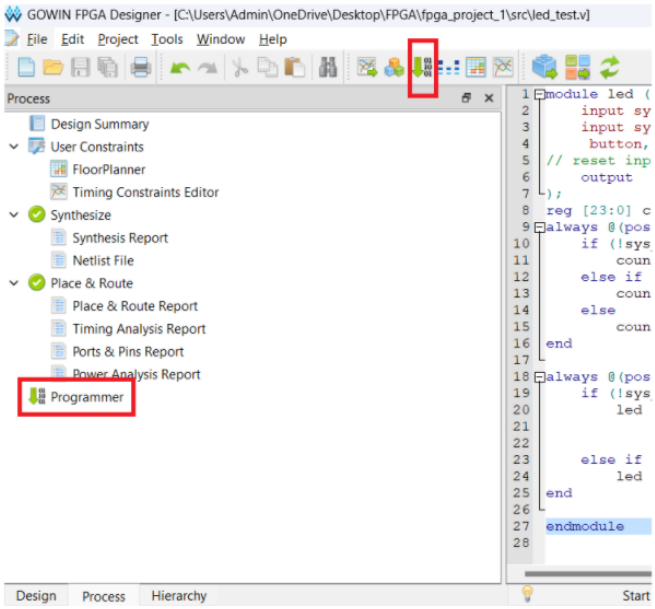


Figure 20: *Programmer image*

- After that, the following figure interface will pop up indicating that the device has been detected, and then click save. If you see No USB Cable Connection, check your hardware connection.

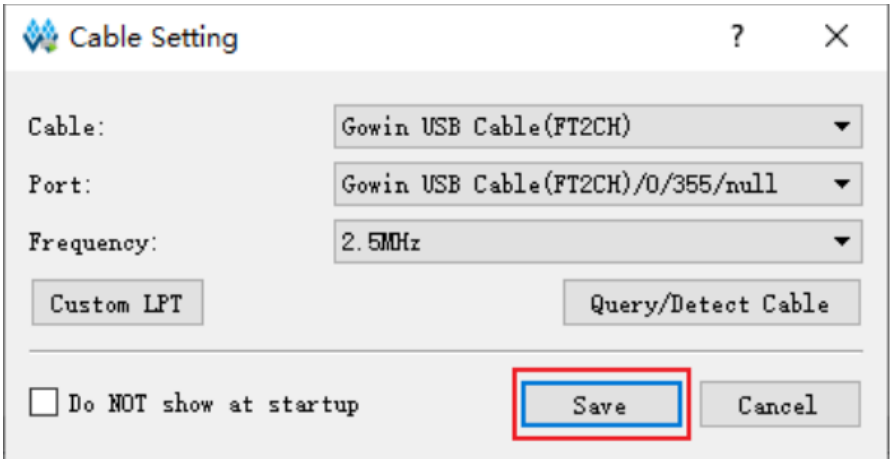


Figure 21: *A configuration prompt has been detected*

- Clicking on Programmer/configure to download the fs file to the FPGA chip.After the download is successfull, the message will be printed.

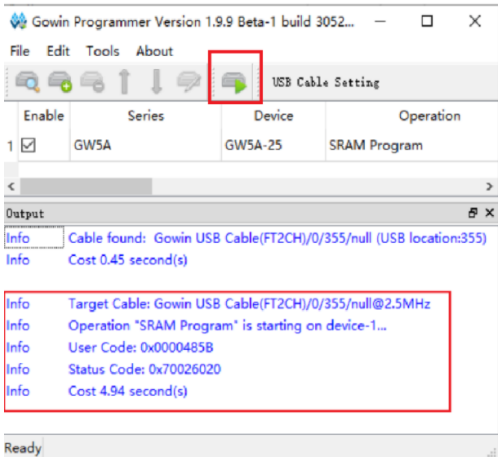


Figure 22: *Download the fs file and a message indicating that the file is successfully downloaded*

- Observation phenomenon: At this time, we can see that the corresponding LED0 light on the development board is flashing, which means that our experiment is successful. At this time, we can see that after the program is downloaded, the Led is blinking after 0.5s.

3 Gowin and Modelsim simulation experiments

3.1 Modelsim compiles the Gowin device library

- Gowin's IP CORE or primitives are only used in the corresponding development platform, and the third-party tool Modelsim cannot obtain the internal running logic results of Gowin's IP CORE, so it cannot be directly simulated, and the Gowin library needs to be compiled. This chapter begins with a tutorial on how to compile a Gowin library, which needs to be done the first time you use the Gowin FPGA, and the compiled library can be used at any time without having to compile it before each simulation.

3.1.1 Create a new Modelsim library

- Open the ModelSim software and click the [File]-> [Change Directory] option in the menu bar to enter the path switching page

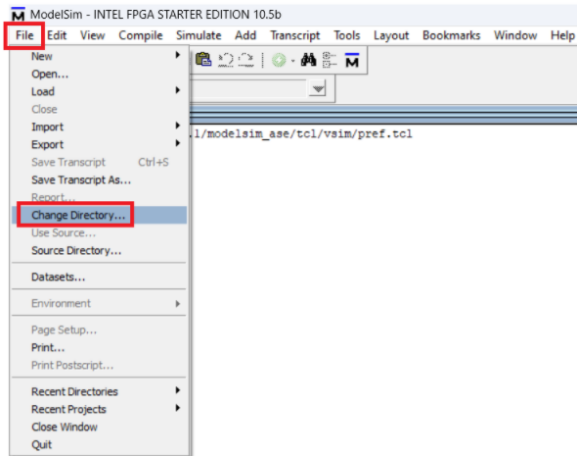


Figure 23: *Go to the Path Switching page*

- In the interface that opens, select the first item "a new library", "library Physical Name" and enter the library name, for example, we are compiling a device of the Gowin 5A series, name it "gw5a", and then click OK, as shown below

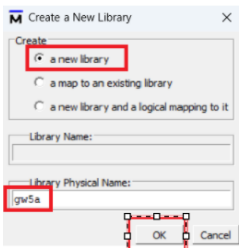


Figure 24: *Create a new library*

- If a gw5a(empty) library appears in the library list, and the path is the location set earlier (in the root directory of the project), the new library is complete.

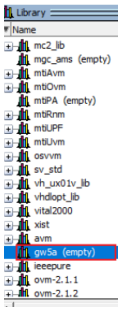


Figure 25: *The new library is completed*

3.1.2 Compile Gowin's device library files

- Click Compile -> Compile in the menu bar to open the compilation library file interface.

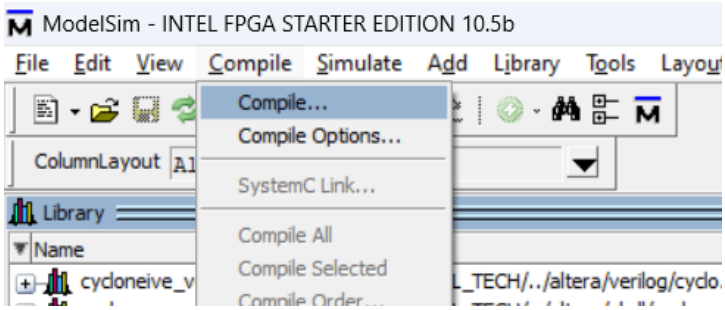


Figure 26: *Open the Compile Library file interface*

- You must select library as the `gw5a` we just created, and then modify the library path of the `gw5a` series device with the file path to Gowin software, specifically: `"...\GowinV1.9.9.03x64\IDE\simlib"`,

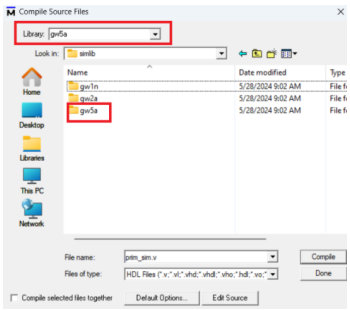


Figure 27: *Locate the Gowin series library file*

- Under this path,find the prim_sim.v file and click the Compile button to start compiling, as shown in Figure below. At this time, the transcript window of Modelsim will quickly refresh the compilation process information, and after the compilation is completed, click the Done button to complete the compilation.

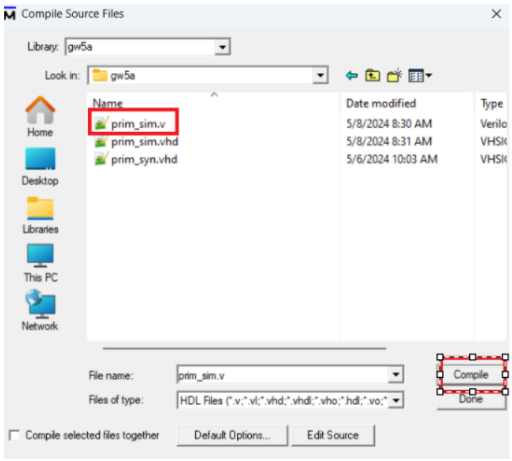


Figure 28: *Locate the library file to compile*

- Once compiled, the Transcript interface will display 0 errors, 0 warnings, and a variety of components under the gw5a library, as shown in Figure below.

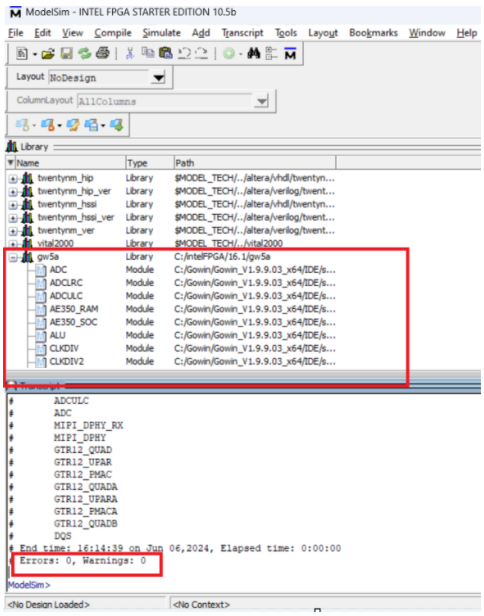


Figure 29: *The series device library has been compiled*

3.1.3 Add a library to the Modelsim default library list

- Find the installation location of Modelsim, this is a read-only file, right-click to select the Modelsim.ini file, click Properties, and check the read-only attribute in the property settings interface.

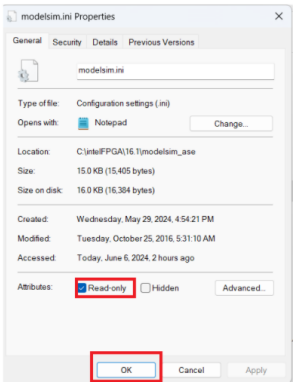


Figure 30: *Removed the Moderssim. Ini's read-only attribute*

- Then open the file with Notepad or Notepad++, and then about 17 lines of the file, add the specified statement of the `gw5a` library:

  ```
  gw5a = $MODEL_TECH/../gw5a
  ```

  and save as shown below.

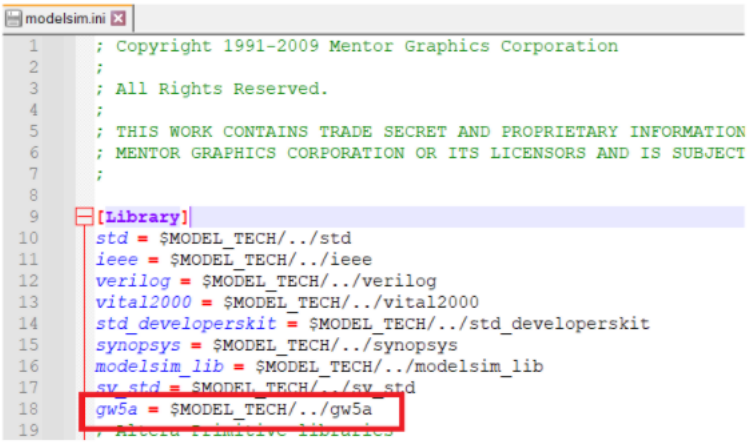


Figure 31: *Add the GW5a library file path*

- Restore the read-only properties of modelsim.ini. Now that we've covered how to compile the library for Gowin 5A series FPGAs using Modelsim, and the 1N and 2A series devices can also be compiled in the same way, and then we can use Modelsim to create simulation projects and simulate design projects.

3.2 Modelsim simulation validation process

3.2.1 Write Verilog simulation design code

- Create a verilog file name led_flash_tb in gowin,

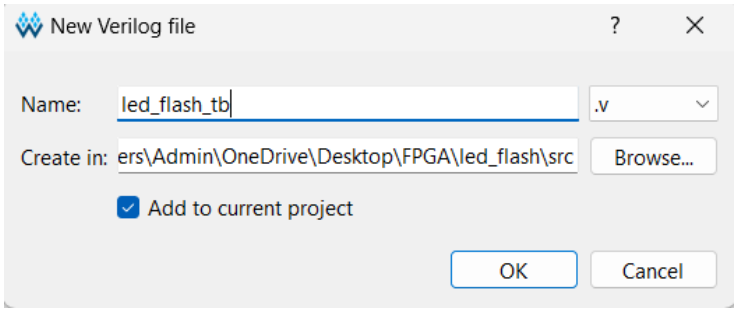


Figure 32: *Create simulation file name*

- Coding example:

Listing 2: *Testbench for led_flash*

```
`timescale 1ns/1ns
module led_flash_tb();
    reg Clk;
    reg Rst_n;
    wire Led;
led_flash led_flash(
.clk(Clk),
.rst_n(Rst_n),
.led(Led)
);
initial Clk = 1;
    always#10 Clk = ~Clk;
    initial begin
    Rst_n = 0;
    #201;
    Rst_n = 1;
    #200;
    #100000000;
end
endmodule
```

- Then remember to run synthesis to check error.

3.2.2 Create a Modelsim project and add simulation files

- Open Modelsim and create a new project in Modelsim. Select File->New->p roject, as shown in Figure below.

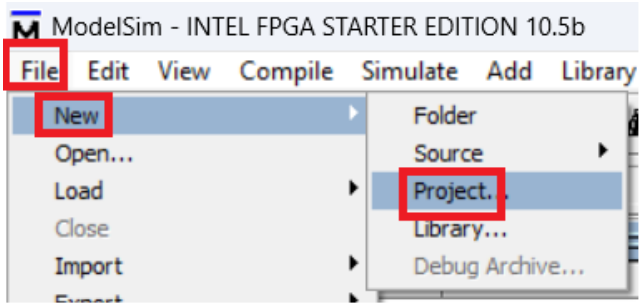


Figure 33: *Establish a Modelsim project*

- We name the project as the corresponding project name "led_flash", "Project Location" is the project path, click "Browse" to set, according to the needs of us save the simulation project to our design project, create a new sim folder and select, as shown in Figure below

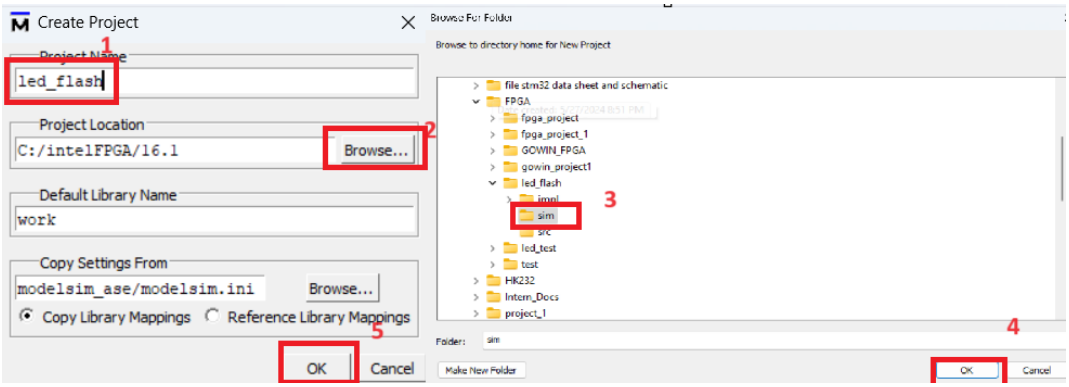


Figure 34: *Modelsim project settings*

- After that choose "Add Existing File"

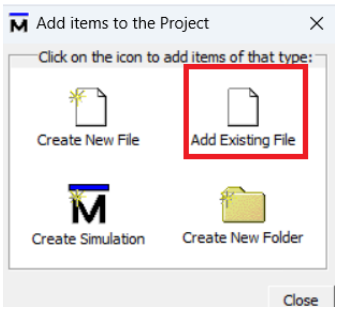


Figure 35: *Add File to Project dialog box*

- Selecting the Browse to select the project design file "led_flash.v" and "led_flash_tb.v", to add .

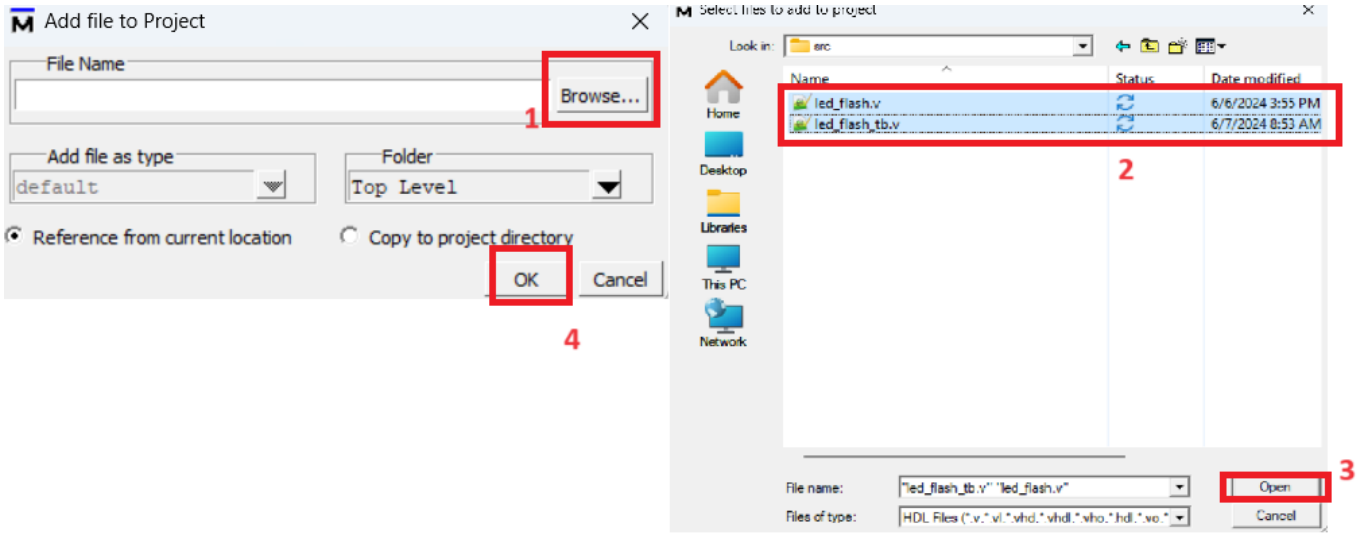


Figure 36: *Select the file to add*

- After the addition is complete, you can see the project files we added in the Modelsim.

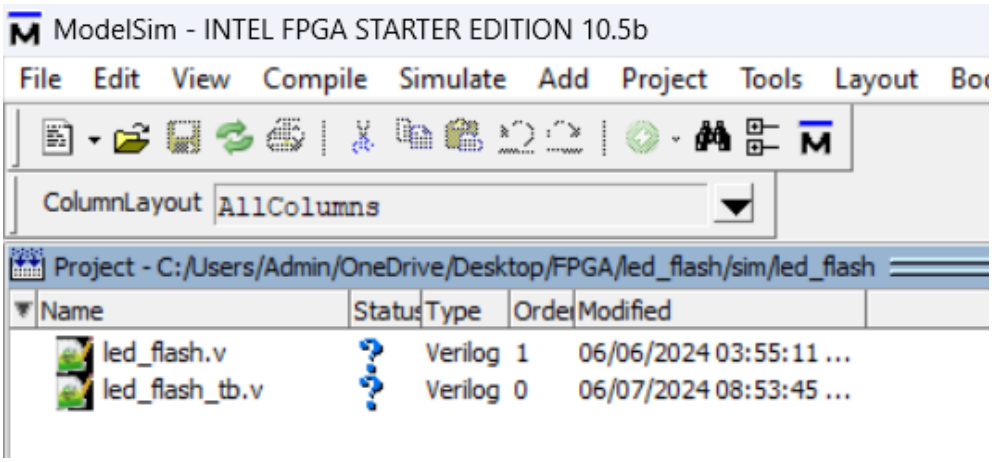


Figure 37: *After the file is added, the diagram is displayed*

3.2.3 Compile the simulation file

- There are two ways of compilation: the first is Compile Selected, the function of compiling selected files requires first selecting one or several files, and the translation of the selected can be completed after executing the command; The second is Compile All, which compiles all the files in the project in the order in which they are compiled. Here, we choose to compile all files and select Compile->Compile All, as shown in Figure below:

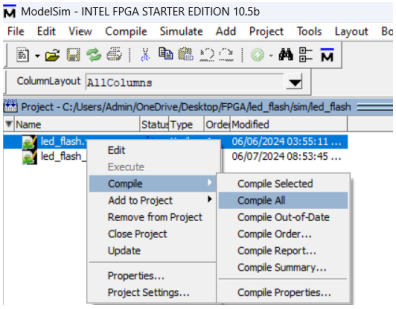


Figure 38: *Compile all project files*

- After compilation, the status at the end of the file changes indicates that the compilation has passed. The "Transcript" below also indicates that the file has been compiled successfully.

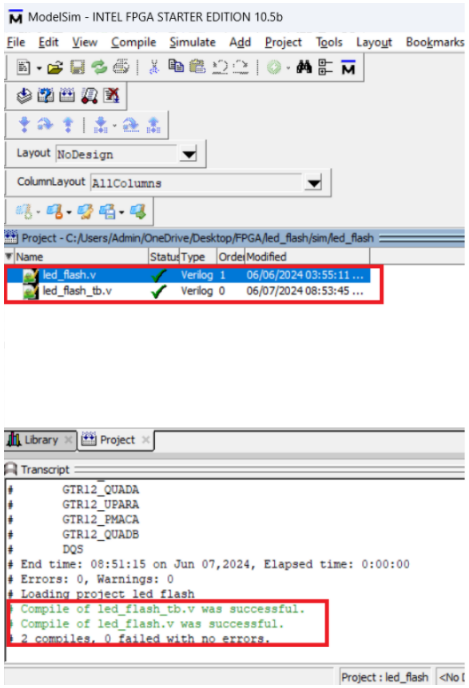


Figure 39: *Compile all project files*

3.2.4 Configure the simulation environment

- We right-click in the project status bar, then select "Add to Project"-> "Simulation Configuration..." and click on it

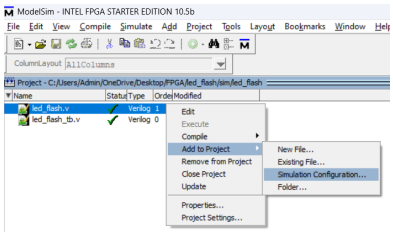


Figure 40: *Enter the operation diagram of configuring the simulation environment*

- In Add Simulation Configuration page, select the "led_flash_tb" module in the work library as the top level of the design in the Design tab page, and click Copy Module Name as the name of the simulation configuration "Simulation Configuration Name" to ensure that the naming is consistent, as shown in Figure below. In complex engineering design, we can design multiple different simulation configurations at the top level to simulate and test the project

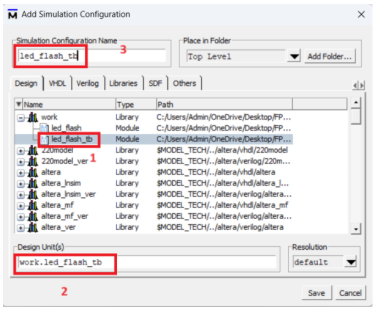


Figure 41: *Simulation environment interface configuration 1*

- Click on "Optimization Options...", and in "Optimization Options.." Select "Apply full visibility to all modules(full debug module)" in the settings bar and click "OK".

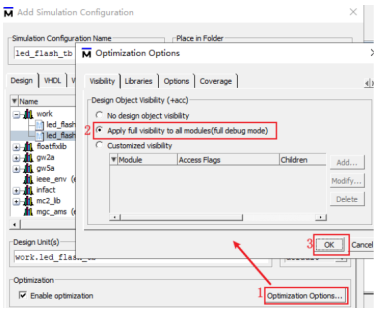


Figure 42: *Simulation environment interface configuration 2*

- Click on the "libraries" tab and click on "Add..." in the "Search libraries(-L)" section. Add the new Gaoyun library file "gw5a", select the library file "gw5a" in "Search Libraries First(-Lf)" as well, and finally click "Save" to save the settings, as shown in Figure 3-24 below.

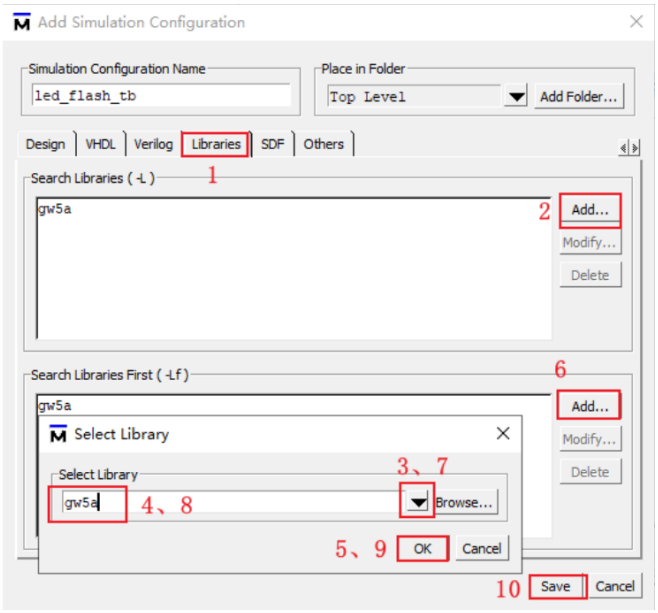


Figure 43: *Simulation environment interface configuration 3*

- After the above steps have been completed, and after saving the configuration, we can see that the simulation configuration file "led_flash_tb" is generated in the "project" column

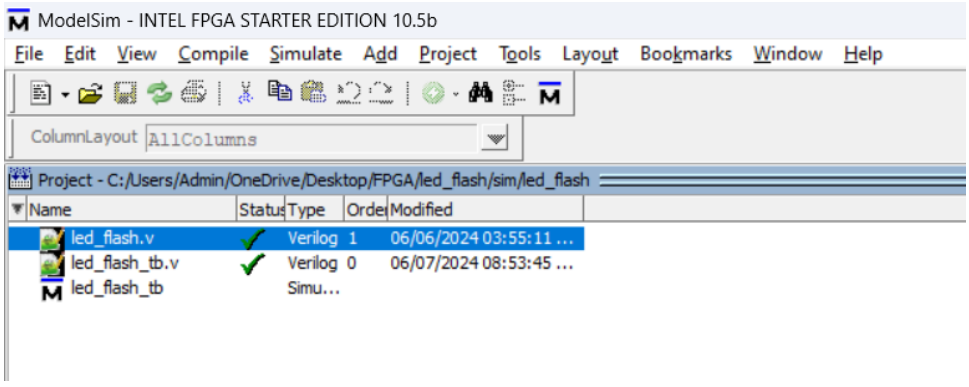


Figure 44: *Schematic diagram of successful generation of simulation file*

3.2.5 Simulation interface waveform observation

- We first need to double-click on the newly generated simulation file "led_flash_tb" to enter the "sim" field

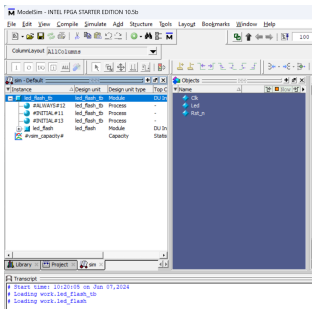


Figure 45: *Enter the SIM interface*

- In the "sim" interface, we can add the waveform of the module we want to observe, select the module, right-click and select "Add Wave"

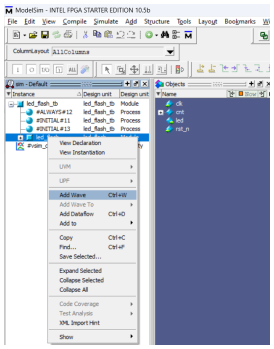


Figure 46: *Add the module waveform that needs to be observed*

- After adding the waveform, go back to the "Library" column, right-click on "Walker" and click Updat to update the led_flash_tb" file in the "Work" bar, as shown below.

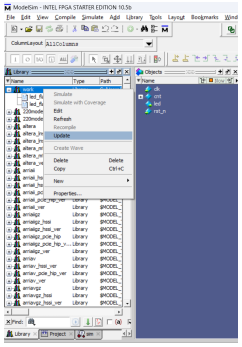


Figure 47: *Update the work bar*

- Click Run all to start running the simulation

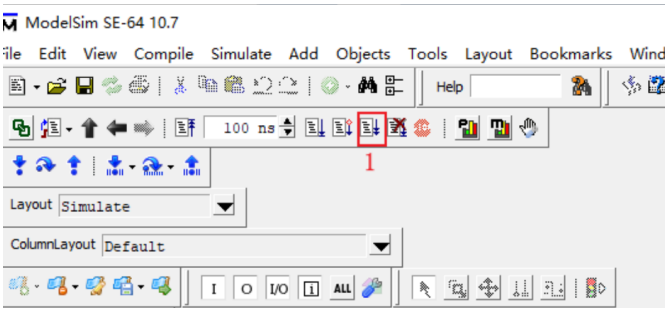


Figure 48: *Start the simulation*

- We will then jump to the Wave window, wait for the program to run for a while, and then we can see the waveform change.

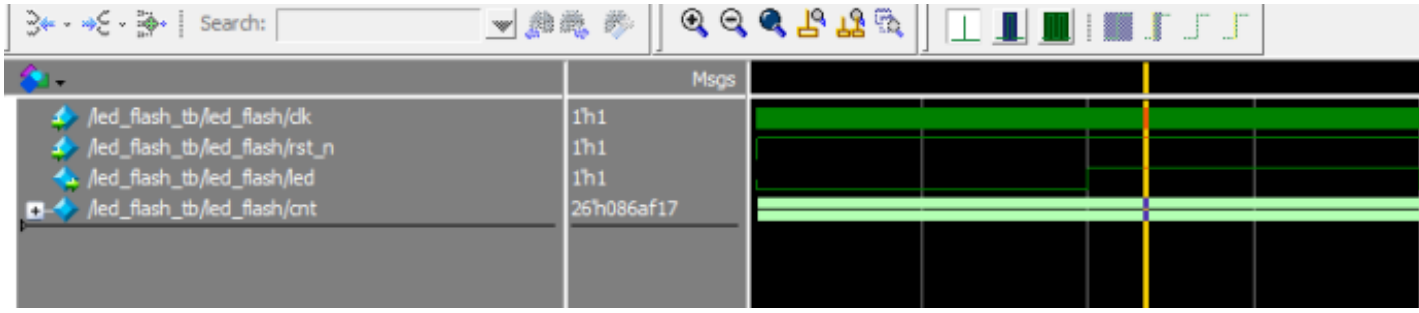


Figure 49: *Observe the waveform*

- If you want to stop running, click on the toolbar at the top of the Wave window, and start again, at this time, click "Zoom Full" to quickly observe the waveform, if you want to understand the function of each button in the toolbar, move the mouse over the button, you can see the explanation, such as continue to run the simulation button.

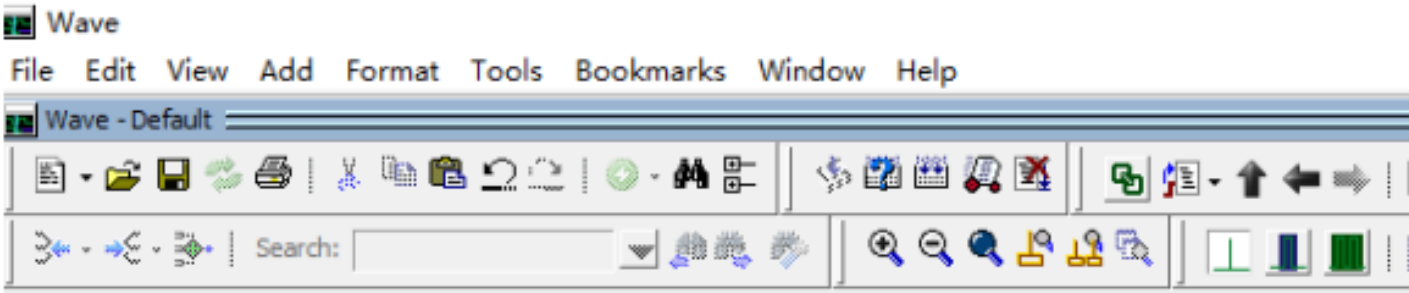


Figure 50: *Wave interface tool button function view*

- During development, after making changes to the design file, click Save and check that the syntax is correct before re-running the program. In Modelsim, we enter the "Library"

interface, re-"Recompile" the "led_flash_tb" and "led_flash" in the "Work" directory, and reload the simulated waveform

3.2.6 Save the waveform file

- First of all, the first step is to save the dataset sim, open the sim window, and click file->save dataset sim or Click Save .wlf file on the SIM page and name the file as follows.

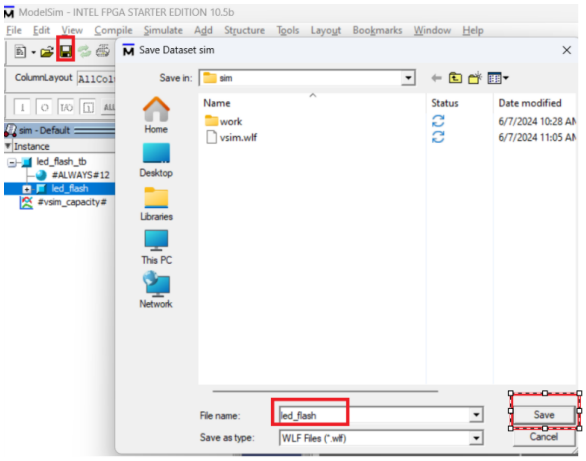


Figure 51: *Saving the .wlf file*

- In the second step, the Wave interface saves the waveform. After clicking the save icon, the waveform file "wave.do" will be saved in the project's simulation file sim by default.

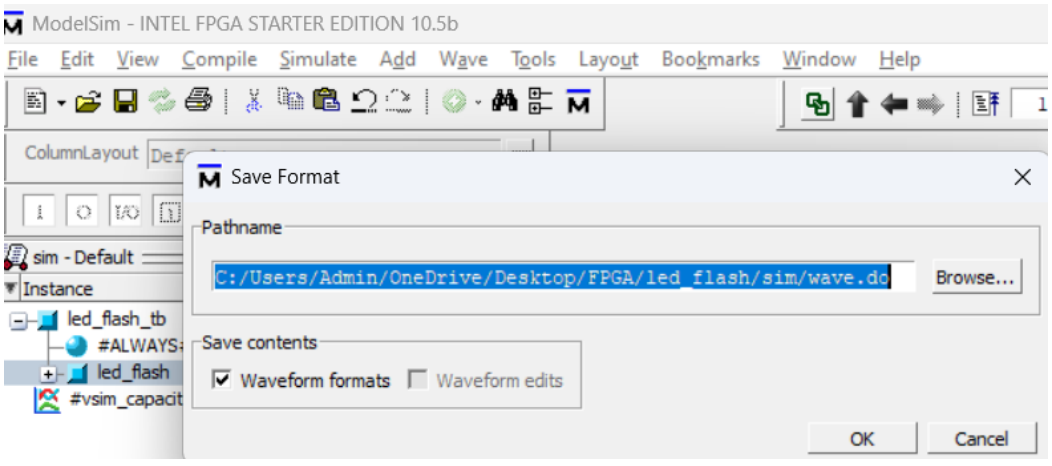


Figure 52: *Save the waveform file*

3.2.7 Reopen the simulation project

- After closing the simulation, the user needs to reopen the simulation by opening the .mpf file

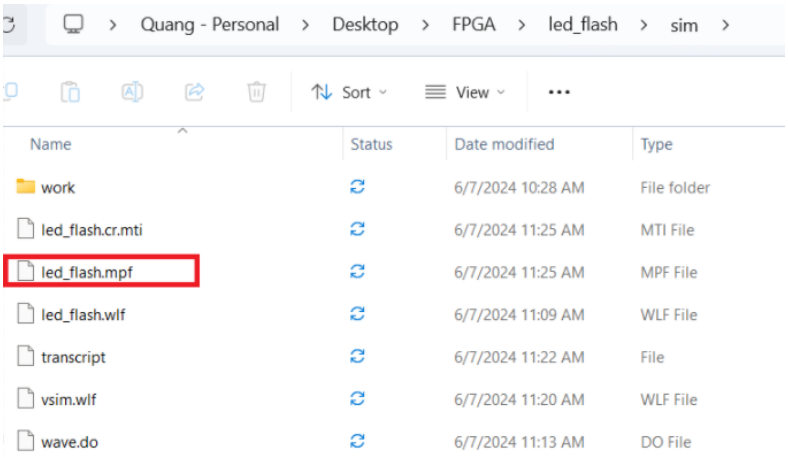


Figure 53: *Reopen the simulation project*

- When opening ".mpf" for the first time, the system will not choose to open it with Modelsim by default, readers need to manually set the way to open the ".mpf" file, and readers who install multiple versions of Modelsim also need to pay attention to setting the corresponding version of Modelsim software. If you have saved the waveform file before, after opening the Modelsim software, click file->all file to open the saved .wlf file, as shown in Figure 5 below

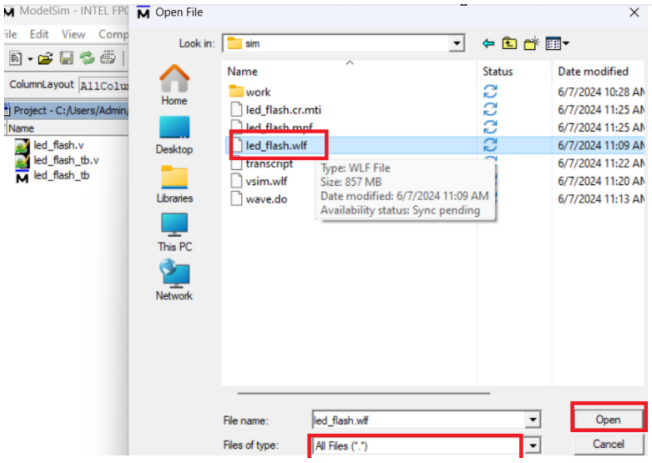


Figure 54: *Open the saved .wlf file*

- Then click file->load->Macro file to open the saved .do file, as shown in Figure below, and then you can see the waveform file we saved earlier.

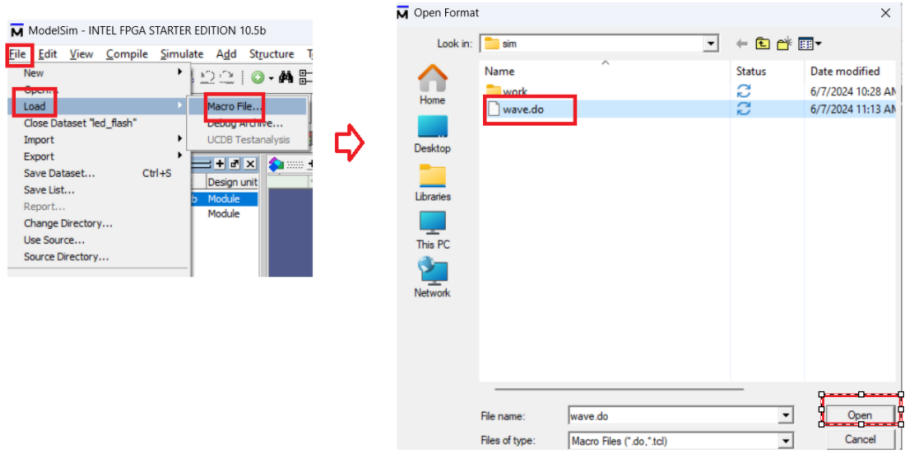


Figure 55: *Open the saved waveform file*

4 Gowin analyzer Oscilloscope

4.1 GAO file configuration

4.1.1 Create GAO file

- Open the Gowin project, click , and in the pop-up New dialog box, select Create a new GAO Config File

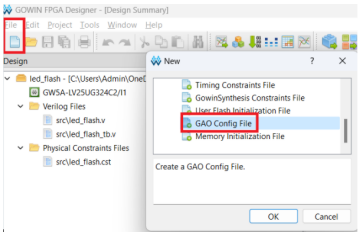


Figure 56: *Create a new Gao profile*

- The dialog box appears,there are two options for RTL Design and for Post-Synthesis Netlist.The "For RTL Design" type is used to capture the pre-RTL signal of the synthesis optimization and generate the configuration file extension .rao.The Forpos-Synthesis Netlister type is used to capture the Netlister signal after comprehensive optimization, generating a profile extension.Here we choose For RTL Design and mode standard

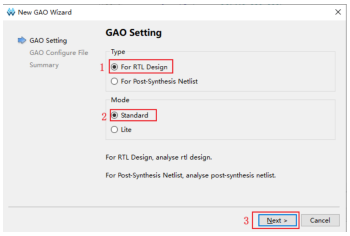


Figure 57: *GAO file configuration interface 1*

- Then enter the configuration file name in the "Name" edit box in the pop-up window, the default is the same as the project name.

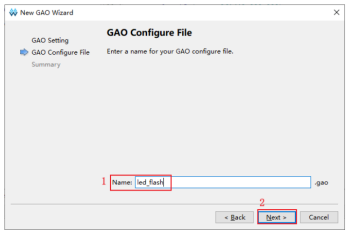


Figure 58: *GAO file configuration interface 2*

- Finally,the summary box appear, click finish.

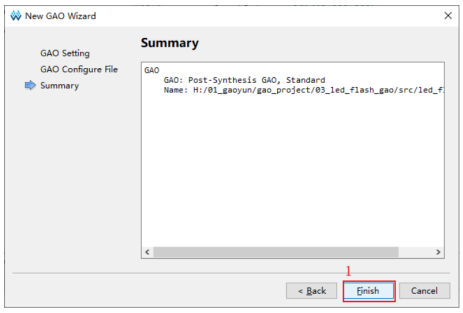


Figure 59: *GAO file configuration interface 3*

4.1.2 Launch the GAO configuration window

- After the GAO file is created, GAO Config Files will appear in the Design window

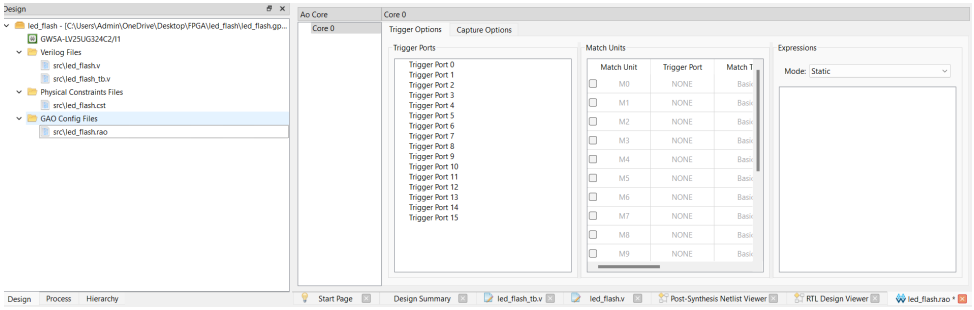


Figure 60: *Go to the GAO configuration window*

- Note that if the warning appears when double-clicking on the .gao file, we need to analyze the synthesize first, and then double-click the .gao file to enter the GAO configuration window.

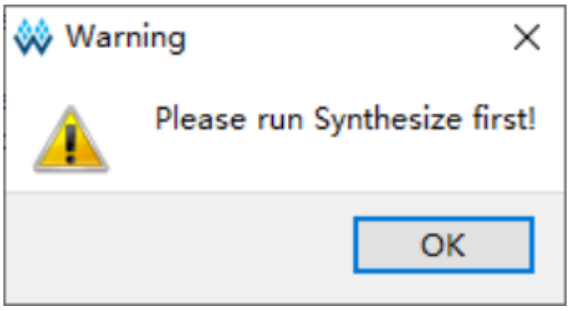


Figure 61: *GAO warning*

4.1.3 Configure Standard Mode GAO

- Configure the number of functional cores :The Ao Core view is used to display and manage the number of functional cores used in the current project, the default Core 0 can support up to 16 cores, you can click Add anywhere in the Ao Core view, or Remove to remove, as shown in Figure below, here we use the default Core 0.

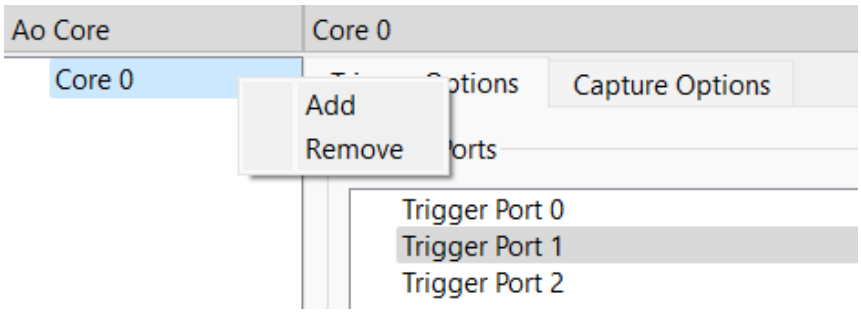


Figure 62: *A diagram of how to add or remove cores*

- In the Trigger Ports view of Configuring Trigger Port Core 0, you can see that there are a total of 16 trigger ports, and the width of each trigger port ranges from 1 64. Port 0 is used here, and after double-clicking, you can enter the port configuration interface Trigger Port, as shown in Figure below, where MSB and LSB represent the high and low positions of the trigger port, respectively.

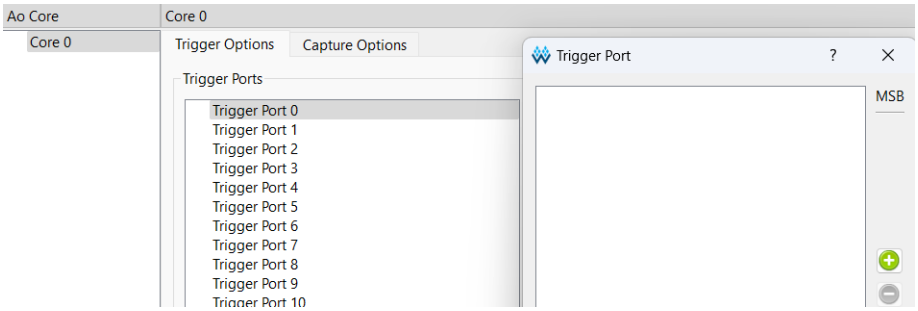


Figure 63: *The operation diagram of the trigger port configuration page is displayed*

- Click + for searching net ,enter the trigger method you want to add in the Name column, for example, the trigger mode you want to observe in this experiment is the status signal LED of the LED light, enter the led, and then click to search,as shown in figure below:

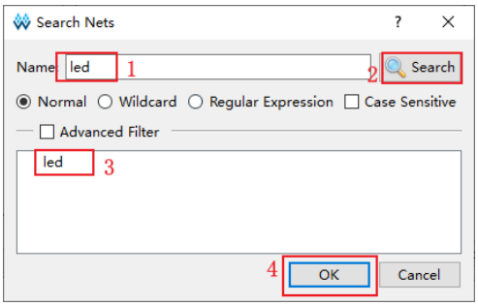


Figure 64: *Select the trigger signal*

- Then go back to the Trigger Port and click OK, as shown in Figure below.

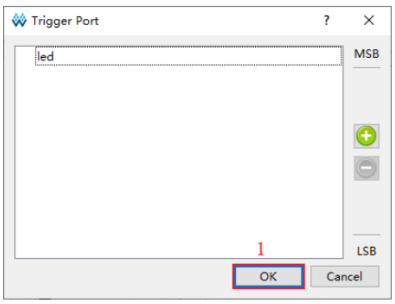


Figure 65: *Trigger signal added*

- Configure the matching unit,click on the match unit M0 the table of unit config appear,and you can configure match type,function,value,...

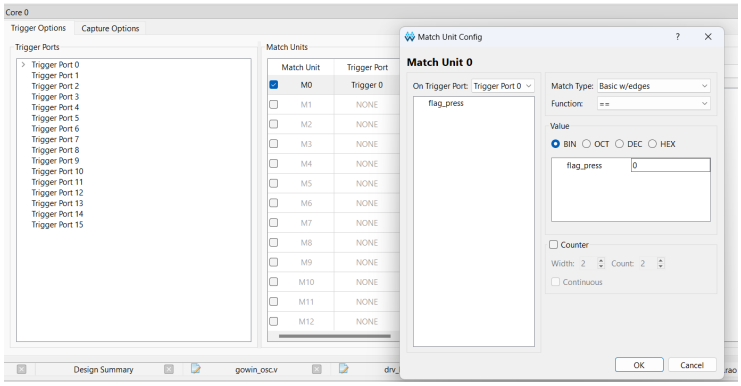


Figure 66: *The Trigger Condition Configuration page is displayed*

- The Expressions view is used to set up trigger expressions, with a maximum of 16 trigger expressions per feature core. Right-click anywhere in the Expressions view and select Add to bring up the Expression dialog box, as shown in below.

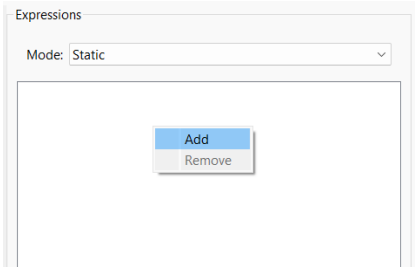


Figure 67: *Enter the Expressions dialog box*

- Here we only have one trigger matching unit M0, click M0, and then click OK, as follows

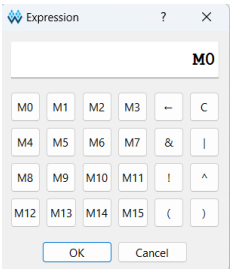


Figure 68: *Select the trigger expression*

- Once the addition is complete, as shown in Figure below, users can double-click to modify it

Figure 69: *The expression is added successfully*

- Click "Captur Options"this is used to configure the sampling clock, storage depth, sampling data signal sampling information, and display the number of Bsram resources used by the current Okor Captur Signels

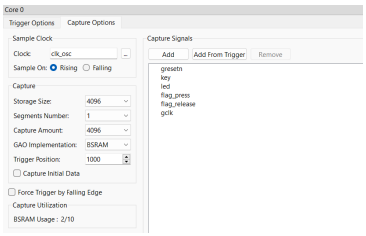


Figure 70: *Capture option*

4.2 Running analyzer oscilloscope

4.2.1 Place and route

- After each modification of the GAO file, you need to click "Place and Route" to rearrange the route.

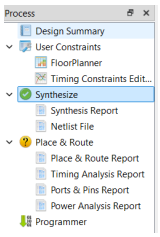


Figure 71: *place and route*

4.2.2 Program Device

- Click under FS file, and the operation is as shown below.

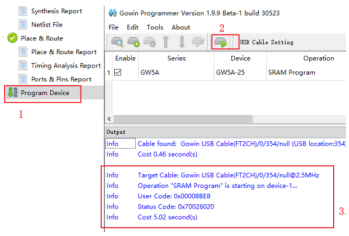


Figure 72: *Programing device*

4.2.3 Waveform grabbing

- Click on the symbol of the gowin analyzer oscilloscope, then set the usb cable,and run the program step as in the figure below.

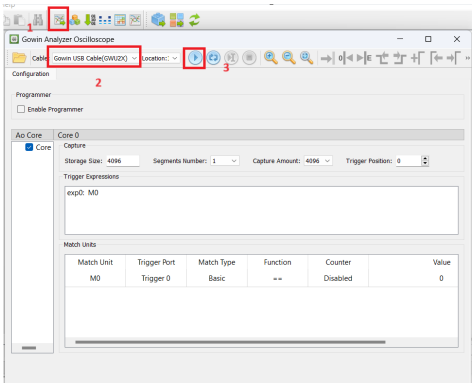


Figure 73: *Oscilloscope running*

- Then a window of waveform will appear,you can zoom in,out or change the format of signal.

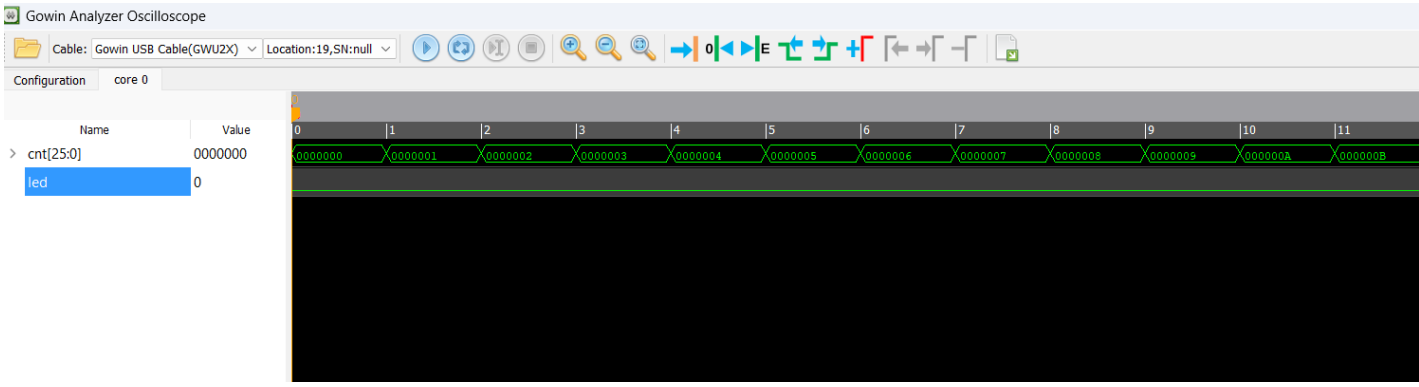


Figure 74: *Oscilloscope running*

5 GAO Project with Tangnano4k

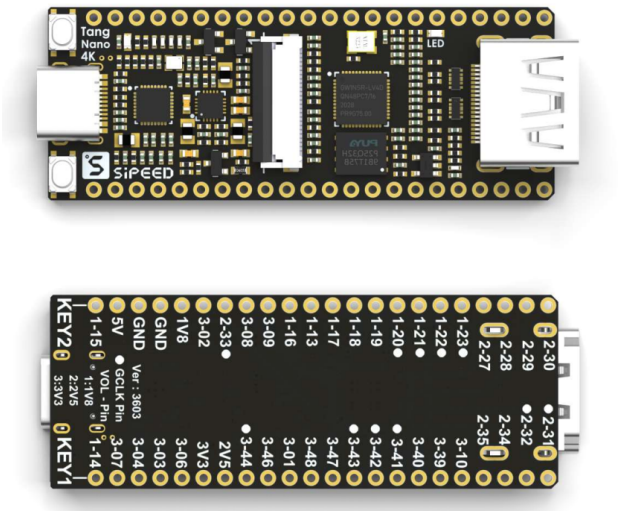


Figure 75: *Tangnano4k image*

5.1 Programming

5.1.1 Verilog Code

- This project code is used for debouncing the button(drv_key.v).

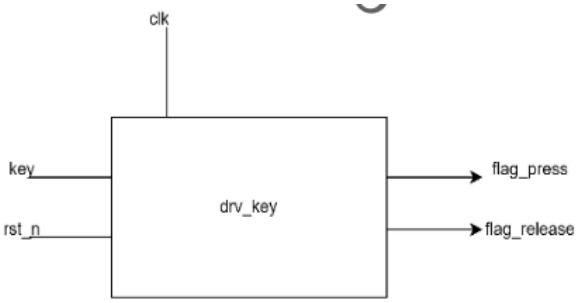


Figure 76: *drv_key module*

- Design block diagram

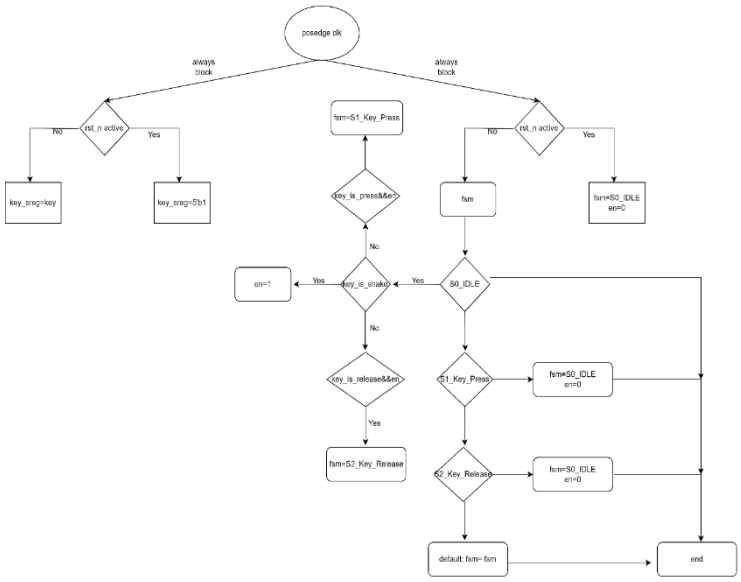


Figure 77: *Block diagram*

```
module drv_key(
    //Inputs
    input clk,
    input rst_n,
    input key,

    //Outputs
    output flag_press,
    output flag_release
);

//parameter
parameter KEY_PRESS = 1'b0;      //
parameter FILTER_TIME = 27_000_0; //

//localparam
localparam KEY_RELEASE = !KEY_PRESS;
localparam S0_IDLE = 0;
localparam S1_KEY_PRESS = 1;
localparam S2_KEY_RELEASE = 2;

//reg
reg [31:0] cnt_press;
```

```
reg [31:0] cnt_release;
reg [4:0] key_sreg;    //shift reg
reg [3:0] fsm;
reg en;

wire key_is_press = (key_sreg[4:1] == {4{KEY_PRESS}});
wire key_is_release = (key_sreg[4:1] == {4{KEY_RELEASE}});
wire key_is_shake = !(key_is_press || key_is_release);

//assign
assign flag_press = (fsm == S1_KEY_PRESS);
assign flag_release = (fsm == S2_KEY_RELEASE);

//always
//
always @ (posedge clk) begin
    if(!rst_n) begin
        key_sreg <= {5{KEY_RELEASE}};
    end
    else begin
        key_sreg <= key_sreg << 1 | key;
    end
end

always @ (posedge clk) begin
    if(!rst_n) begin
        fsm <= S0_IDLE;
        cnt_press  <= 0;
        cnt_release <= 0;
        en <= 0;
    end
    else begin
        case (fsm)
            S0_IDLE: begin
                if(key_is_shake) begin
                    cnt_press <= 'd0;
                    cnt_release <= 'd0;
                    en <= 1;
                end
                else if(key_is_press && en) begin
                    if(cnt_press < FILTER_TIME)
                        cnt_press <= cnt_press + 1;
                    else begin
```

```
                    cnt_press <= 0; //max = FILTER_TIME
                    fsm <= S1_KEY_PRESS;
                end
            end
            else if(key_is_release && en) begin
                if(cnt_release < FILTER_TIME)
                    cnt_release <= cnt_release + 1;
                else begin
                    cnt_release <= 0; //max = FILTER_TIME
                    fsm <= S2_KEY_RELEASE;
                end
            end
            else begin
                fsm <= fsm;
                cnt_press  <= 0;
                cnt_release <= 0;
                en <= 0;
            end
        end

        S1_KEY_PRESS: begin
            fsm <= S0_IDLE;
            en <= 0;
        end

        S2_KEY_RELEASE: begin
            fsm <= S0_IDLE;
            en <= 0;
        end

        default: begin
            fsm <= fsm;
        end
    endcase
  end
end

endmodule  //get_key end
```

- Then go to IP configuration choose Hard Module → Clock → OSC, configure the clock of OSC then click OK a file osc will appear like below.

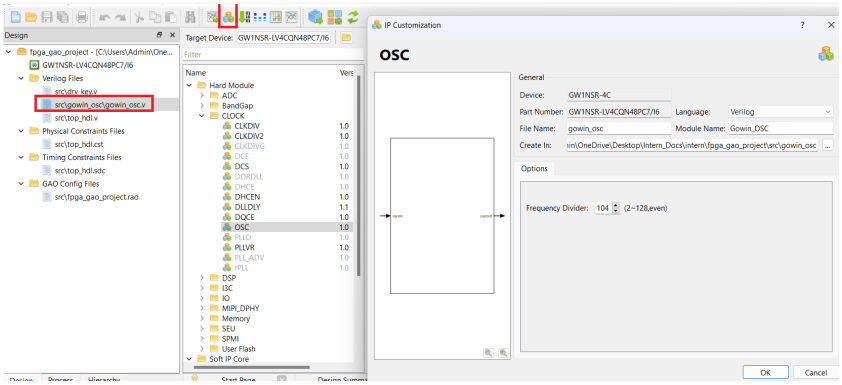


Figure 78: *Configure OSC clock*

- The next step is create a file top_hdl.v for running synthesis

```
module top_hdl(
    //Inputs
    input gclk,     // 27MHz
    input gresetn,
    input key,
    //Outputs
    output reg led
);
//localparam
localparam KEY_PRESS = 1'b0;
localparam KEY_RELEASE = !KEY_PRESS;
localparam LED_ON    = 1'b1;
localparam LED_OFF   = !LED_ON;
wire flag_press;
wire flag_release;
drv_key #(
    .FILTER_TIME(27_000_0*5), //50ms
    .KEY_PRESS(KEY_PRESS)
)drv_key_ut0(
    //Inputs
    .clk  (gclk),
    .rst_n(gresetn),
    .key  (key),
    //Outputs
    .flag_press (flag_press),
    .flag_release(flag_release)
```

```
);
always @ (posedge gclk) begin
    if(!gresetn) begin
        led <= LED_ON;
    end
    else if(flag_press) begin
        led <= !led;
    end
end
wire clk_osc;     //250/2=125MHz, gao use clock
Gowin_OSC Gowin_OSC_ut0(
    .oscout(clk_osc),
    .oscen(1'b1)
);
endmodule
```

- Setting Physic constraint

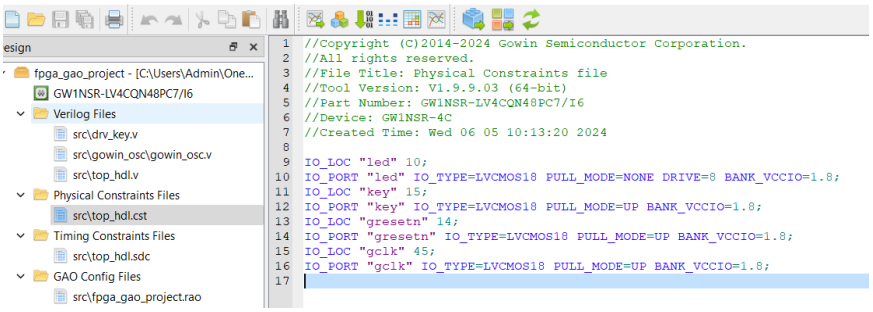


Figure 79: *Setting Physic constraint*

- Setting Time constraint

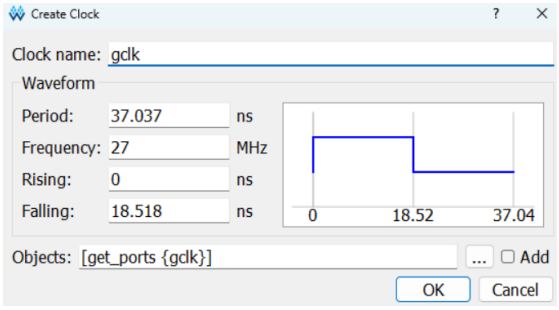


Figure 80: *Setting Time constraint*

- Running Gowin analyzer oscilliscope
- initial led:0(off),flag_press=1,press_release

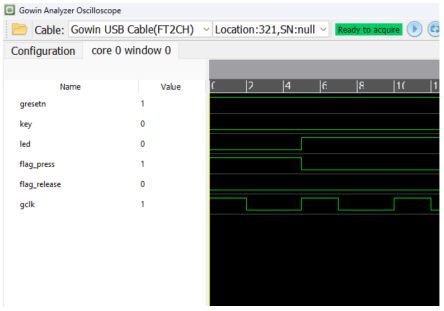


Figure 81: *GAO when led=0*

- initial led:1(on),flass_press=1:press_release

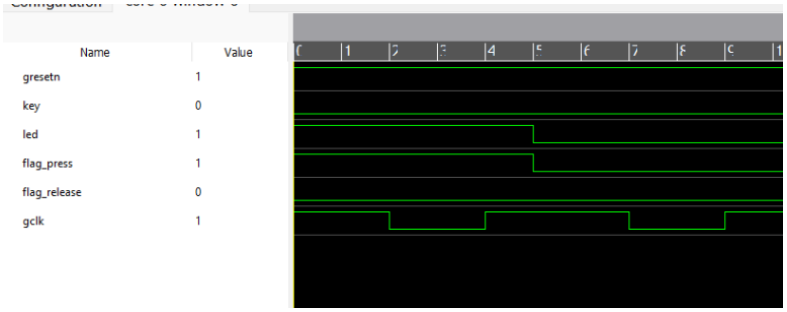


Figure 82: *GAO when led=1*

5.2 Static Timing analysis

5.2.1 Definition

- Setup Time:
  - Definition: The minimum amount of time before the clock edge that the input signal (data) must be stable (unchanged) to be correctly latched by the flip-flop or register.
  - Purpose: Ensures that the data is available and stable long enough for the flip-flop to capture it reliably.
- Hold time:
  - Definition: The minimum amount of time after the clock edge that the input signal (data) must remain stable to be correctly latched by the flip-flop or register.

– Purpose: Ensures that the data remains stable long enough after the clock edge for the flip-flop to latch it correctly.

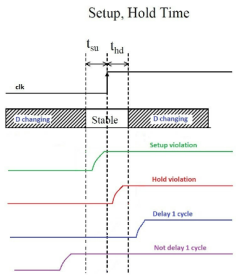


Figure 83: *Setup and hold time*

- Propagation delay:
  - Time it takes for a signal to travel from source to a destination
  - the longer the propagation delay,the slower clock is able to run.

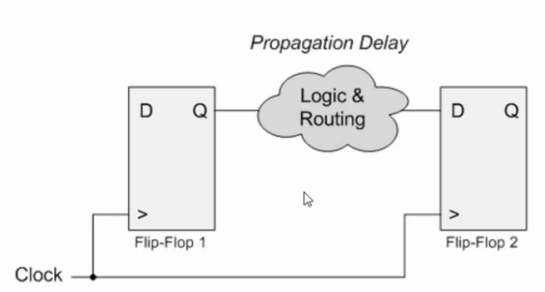


Figure 84: *Propagation delay*

- Setup time slack: a measure of how much extra time is available before the data must be stable at the setup time of the flip-flop.

Formula:

$$\text{Setup Slack} = T_{clk\_period} - (T_{clk\_q} + T_{comb\_logic} + T_{setup} + T_{skew})$$

Where:

- T_{clk_period}: Clock period
- T_{clk_q}: Clock-to-Q delay (time taken for the data to appear at the output after the clock edge)
- T_{comb_logic}: Propagation delay through the combinational logic
- T_{setup}: Setup time requirement of the flip-flop
- T_{skew}: Clock skew

- Hold Time Slack:a measure of how much extra time is available after the clock edge before the data must remain stable for the hold time of the flip-flop.

  Formula:

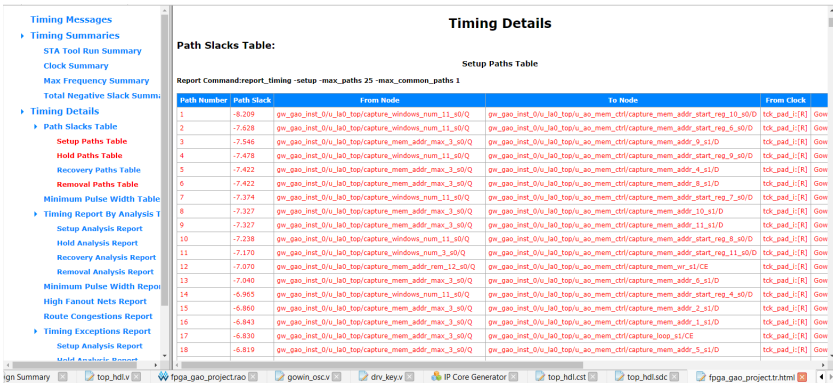
  $$\text{Hold Slack} = (T_{clk\_q\_min} + T_{comb\_logic\_min}) - T_{hold} - T_{skew}$$

  Where:

  - $T_{clk\_q\_min}$: Minimum clock-to-Q delay
  - $T_{comb\_logic\_min}$: Minimum propagation delay through the combinational logic
  - T_{hold}: Hold time requirement of the flip-flop
  - T_{skew}: Clock skew

5.2.2 Time error and fix

- When the slack is negative,time is error.



- There are two basic fixed:
  - Slow down your clock frequency
  - Break your logic up into stages (pipeline)

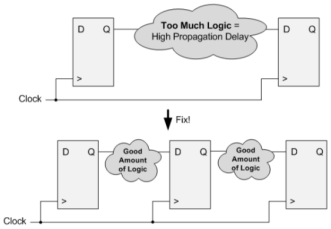


Figure 85: *Break logic*

- In our project we will set FRE_DIV to 104,set time constraint file: create_clock -name tck -period 37.037 -waveform 0 18.518 [get_ports tck_pad_i]

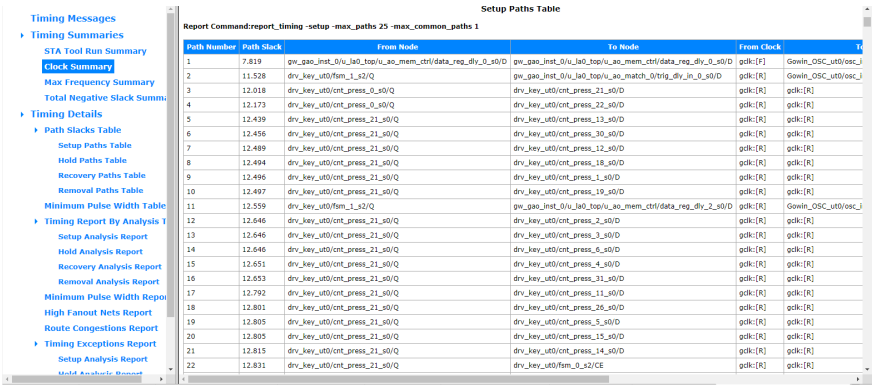


Figure 86: *Fixing time*

6 Kicad

- KiCad is an open-source software suite for creating electronic circuit schematics, printed circuit boards (PCBs), and associated part descriptions. KiCad supports an integrated design workflow in which a schematic and corresponding PCB are designed together, as well as standalone workflows for special uses. KiCad also includes several utilities to help with circuit and PCB design, including a PCB calculator for determining electrical properties of circuit structures, a Gerber viewer for inspecting manufacturing files, a 3D viewer for visualizing the finished PCB, and an integrated SPICE simulator for inspecting circuit behavior.
- KiCad runs on all major operating systems and a wide range of computer hardware. It supports PCBs with up to 32 copper layers and is suitable for creating designs of all complexity. KiCad is developed by a volunteer team of software and electrical engineers around the world with a mission of creating free and open-source electronics design software suitable for professional designers.

Figure 87: *Kicad software*

- Kicad support many tools you can see below

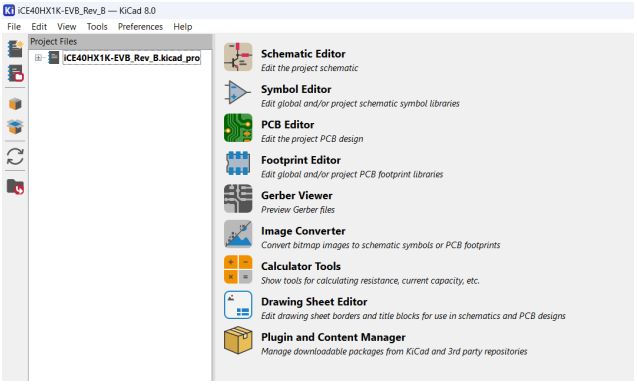


Figure 88: *Kicad application*

- The work flow to design PCB

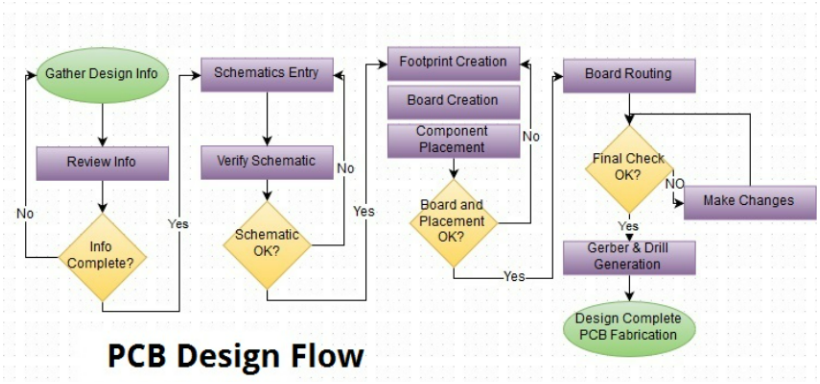


Figure 89: *Design workflow*

6.1 My practice project

6.1.1 Schematic

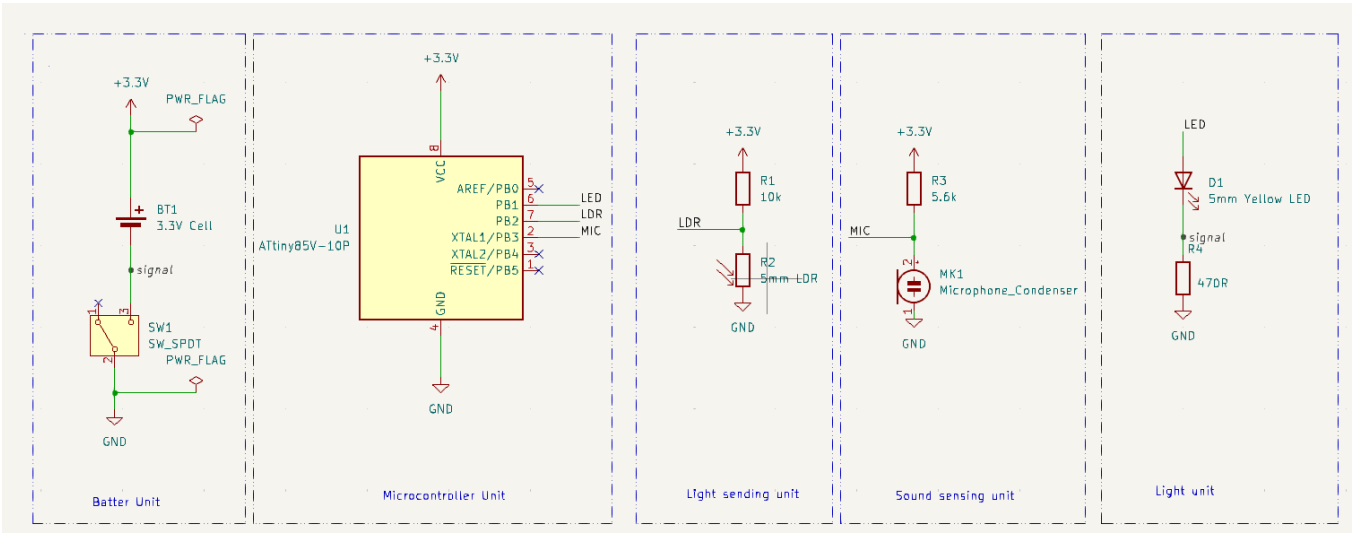


Figure 90: *Practice project schematic*

6.1.2 PCB

6.1.2.1 PCB_layout design view

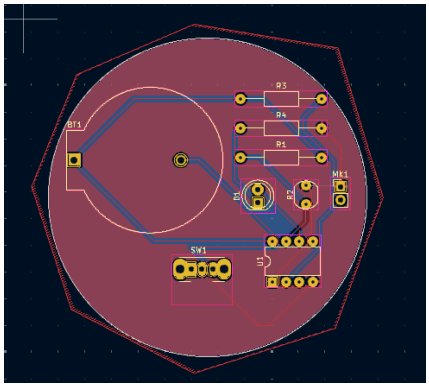


Figure 91: *Practice project PCB design*

6.1.2.2 PCB_layout 3D view

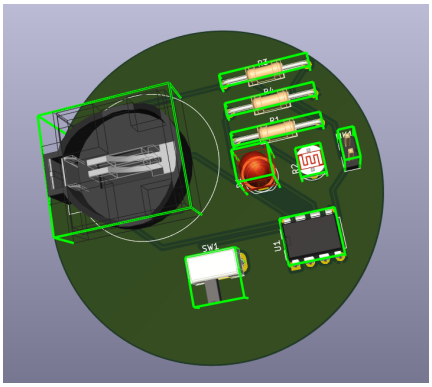


Figure 92: *Practice project PCB 3D*

6.2 FPGA project

6.2.1 Schematic

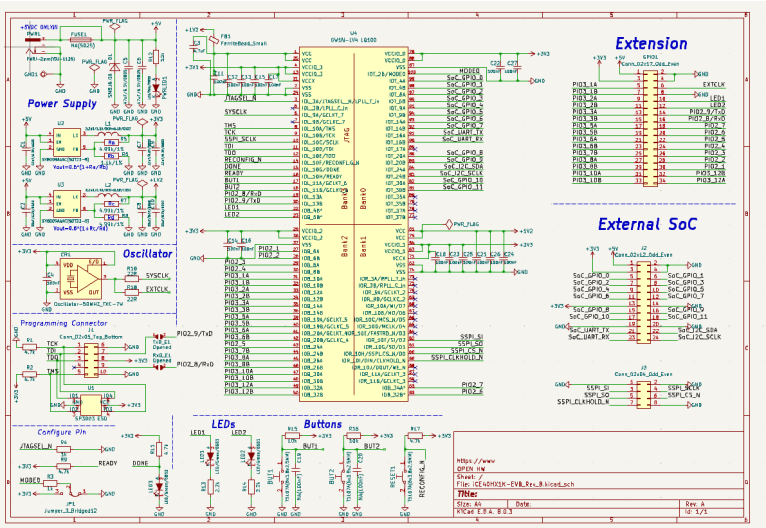


Figure 93: *FPGA schematic*

6.2.2 PCB

6.2.2.1 PCB_layout design view

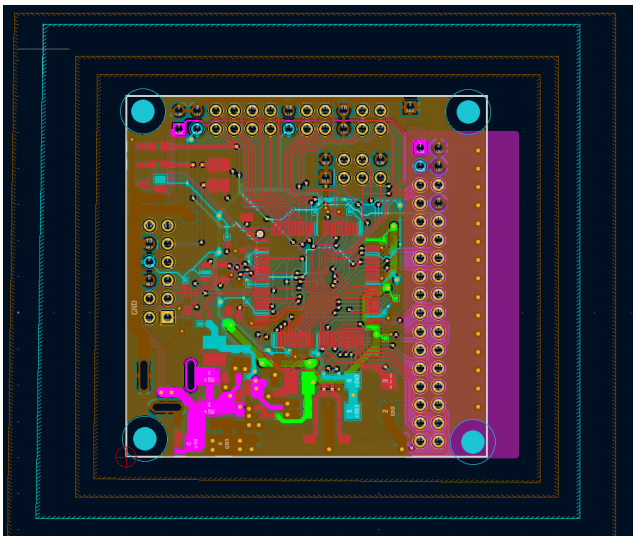


Figure 94: *FPGA PCB design*

6.2.2.2 PCB_layout 3D view

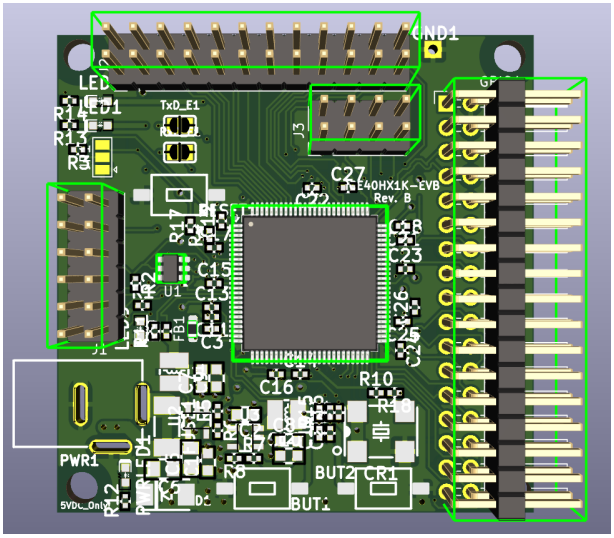


Figure 95: *FPGA PCB 3D front view*

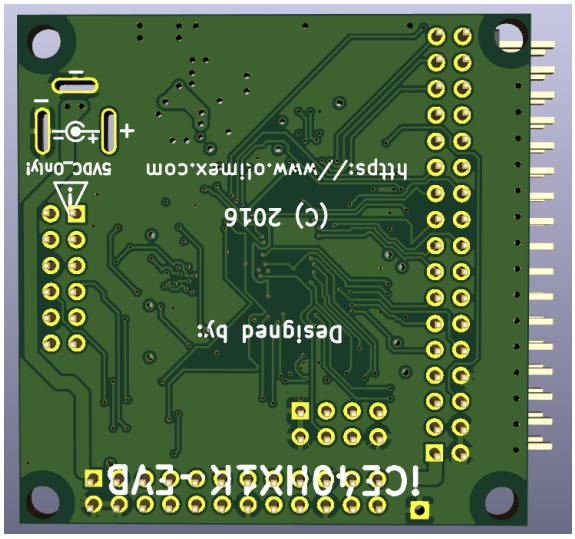


Figure 96: *FPGA PCB 3D back view*

- Not use 90 degree routing.
- Need to improve printing layer,remember to reduce the word name.
- Checking more clearly about component such as JTAG port.

7 UART_TX Design and verification of serial port transmitter module

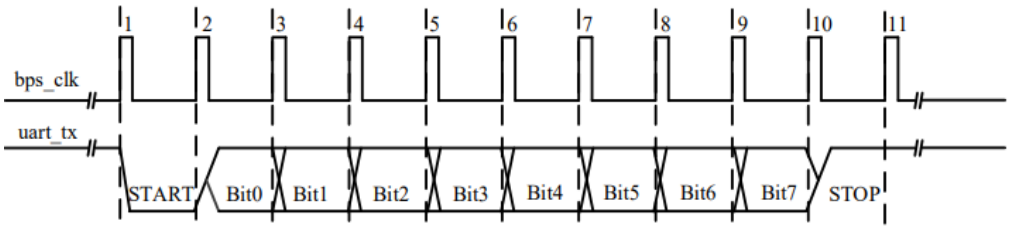


Figure 97: *process of uart transfer*

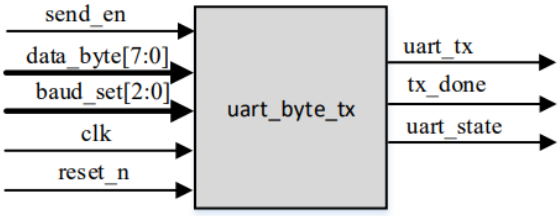


Figure 98: *Module of uart transfer*

- send_en:enable sending
- baud_set[2:0]:set value baudrate

7.1 Baurate clock module design

- Our system have clock frequency 50MHZ,formula:= $\frac{50x10^6}{baudrate} - 1$
- baudrate:9600->5207
- 19200->2603
- 38400->1301
- 57600->867
- 115200->433

```
always@(posedge clk or posedge reset)
   if(reset)
      bps_DR <= 16'd5207;
   else begin
```

```
    case(baud_set)
       0:bps_DR <= 16'd5207;// do baurate=9600;500000000/9600~~5207
       1:bps_DR <= 16'd2603;//baudrate=19200
       2:bps_DR <= 16'd1301;//baudrate=38400
       3:bps_DR <= 16'd867;//baudrate=57600
       4:bps_DR <= 16'd433;//baudrate=115200
       default:bps_DR <= 16'd5207;
    endcase
end
```

- Checking the uart_state

```
     always@(posedge clk or posedge reset)
    if(reset)
       uart_state <= 1'b0;
    else if(send_en)
       uart_state <= 1'b1;
    else if(bps_cnt == 4'd11)
       uart_state <= 1'b0;
    else
       uart_state <= uart_state;
```

- Checking reset signal:

```
always@(posedge clk or posedge reset)
    if(reset)
       data_byte_reg <= 8'd0;
    else if(send_en)
       data_byte_reg <= data_byte;
    else
       data_byte_reg <= data_byte_reg;
```

- Here we use a counter to generate a baud rate clock.

```
    //counter
    always@(posedge clk or posedge reset)
    if(reset)
       div_cnt <= 16'd0;
    else if(uart_state)begin
       if(div_cnt == bps_DR)
          div_cnt <= 16'd0;
       else
          div_cnt <= div_cnt + 1'b1;
```

```
end
else
   div_cnt <= 16'd0;

// bps_clk gen
always@(posedge clk or posedge reset)
if(reset)
   bps_clk <= 1'b0;
else if(div_cnt == 16'd1)
   bps_clk <= 1'b1;
else
   bps_clk <= 1'b0;
```

7.2 Data output module design

- determine the state of the data sending cycle by counting the baud rate clock

```
always@(posedge clk or posedge reset)
   if(reset)
      bps_cnt <= 4'd0;
   else if(bps_cnt == 4'd11)
      bps_cnt <= 4'd0;
   else if(bps_clk)
      bps_cnt <= bps_cnt + 1'b1;
   else
      bps_cnt <= bps_cnt;
```

- Also in order for the module to be able to control or call other modules, a byte is generated to signal the end of the transfer. After a data bit is transmitted, the tx_done signal outputs a high level of the clock.

```
always@(posedge clk or posedge reset)
   if(reset)
      tx_done <= 1'b0;
   else if(bps_cnt == 4'd11)
      tx_done <= 1'b1;
   else
      tx_done <= 1'b0;
```

- Data transmission status control module

```
 always@(posedge clk or posedge reset)
if(reset)
   uart_tx <= 1'b1;
else begin
   case(bps_cnt)
      0:uart_tx <= 1'b1;
      1:uart_tx <= START_BIT;
      2:uart_tx <= data_byte_reg[0];
      3:uart_tx <= data_byte_reg[1];
      4:uart_tx <= data_byte_reg[2];
      5:uart_tx <= data_byte_reg[3];
      6:uart_tx <= data_byte_reg[4];
      7:uart_tx <= data_byte_reg[5];
      8:uart_tx <= data_byte_reg[6];
      9:uart_tx <= data_byte_reg[7];
      10:uart_tx <= STOP_BIT;
      default:uart_tx <= 1'b1;
   endcase
end
```

7.3 TESTBENCH code for modelsim

```
`timescale 1ns/1ns
`define CLK_PERIOD 20

module uart_byte_tx_tb;

   reg clk;
   reg reset_n;
   reg [7:0]data_byte;
   reg send_en;
   reg [2:0]baud_set;

   wire uart_tx;
   wire tx_done;
   wire uart_state;

   uart_byte_tx uart_byte_tx(
      .clk(clk),
      .reset_n(reset_n),

      .data_byte(data_byte),
```

```
    .send_en(send_en),
    .baud_set(baud_set),

    .uart_tx(uart_tx),
    .tx_done(tx_done),
    .uart_state(uart_state)
);

initial clk = 1;
always#(`CLK_PERIOD/2)clk = ~clk;

initial begin
    reset_n = 1'b0;
    data_byte = 8'd0;
    send_en = 1'd0;
    baud_set = 3'b100;
    #(`CLK_PERIOD*500 + 1 )
    reset_n = 1'b1;
    #(`CLK_PERIOD*50);

 //send first byte
    data_byte = 8'haa;
    send_en = 1'd1;
    #`CLK_PERIOD;
    send_en = 1'd0;

    @(posedge tx_done)
    #(`CLK_PERIOD*5000);

 //send second byte
    data_byte = 8'h55;
    send_en = 1'd1;
    #`CLK_PERIOD;
    send_en = 1'd0;

    @(posedge tx_done)
    #(`CLK_PERIOD*5000);
    $stop;
end

endmodule
```

7.4 Coding test for real device

```
module uart_tx_test(
    input clk,
    input reset_n,

    output uart_tx,
    output led
);
    parameter MCNT = 49_999_999; //   1S

    reg [7:0] data_byte;
    reg send_en;

    //
    reg [25:0]cnt; //
    wire tx_done;


    always@(posedge clk or negedge reset_n)
    if(!reset_n)
        cnt <= 25'd0;
    else if(cnt == MCNT)
        cnt <= 25'd0;
    else
        cnt <= cnt + 1'b1;

    always@(posedge clk or negedge reset_n)
    if(!reset_n)
        send_en <= 1'b0;
    else if(cnt == MCNT)
        send_en <= 1'b1;
    else
        send_en <= 1'b0;

    always@(posedge clk or negedge reset_n)
    if(!reset_n)
        data_byte <= 8'b0;
    else if(tx_done)
        data_byte <= data_byte + 1'b1;
    else
        data_byte <= data_byte;
```

```
uart_byte_tx uart_byte_tx(
    .clk(clk),
    .reset_n(reset_n),

    .data_byte(data_byte),
    .send_en(send_en),
    .baud_set(3'd0),

    .uart_tx(uart_tx),
    .tx_done(tx_done),
    .uart_state(led)
);

endmodule
```

7.5 Simulation

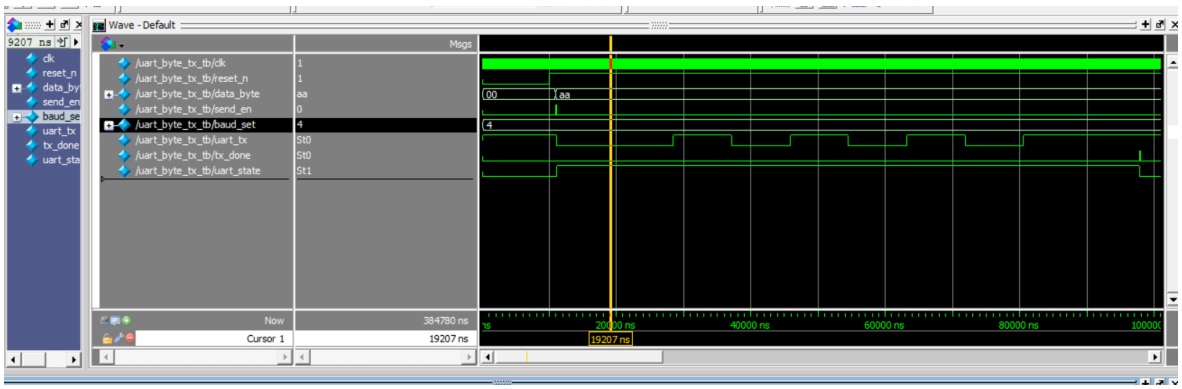


Figure 99: *Simulation with modelsim*

7.6 Testing with board

- We using hercules software:

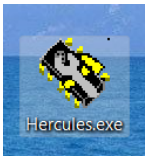


Figure 100: *hercules software*

- Setting up 8bit,choose the COM port connect to UART,configure baud,parity

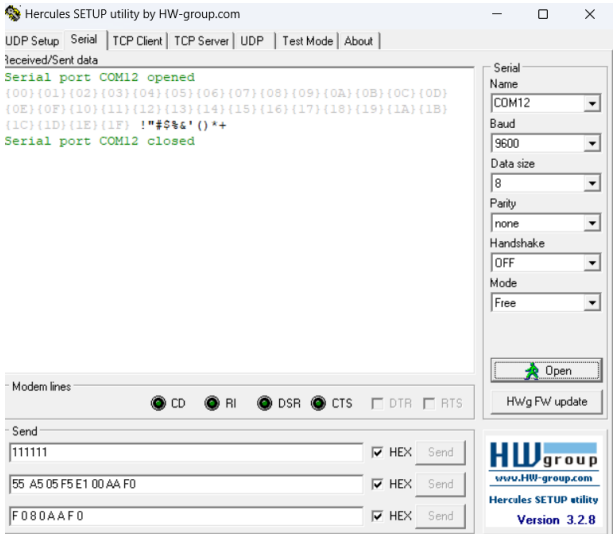


Figure 101: *process of uart transfer*

8 UART_RX / Design and verification of serial port receiver module

8.1 Analysis of the principle of serial port reception

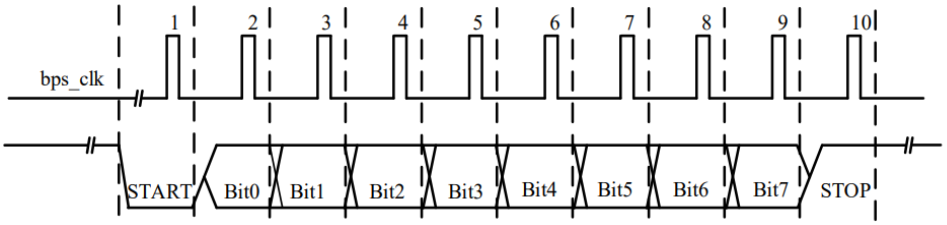


Figure 102: *process of uart transfer*

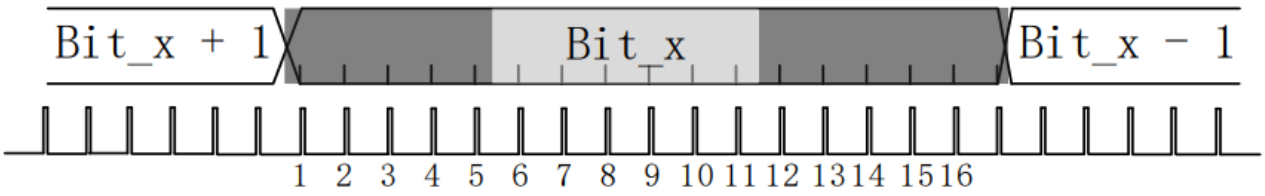


Figure 103: *Taking sample of 1 bit receive*

- Because data can be unstable when it change,specially in the dark gray part.The middle of period is relatively stable and represent the correct result.Taking 6 sample and find the probability of high and low.For example 1/1/1/0/1/1 result 1 and 0/1/0/0/0/0 is 0,when 0/0/0/1/1/1 enviroment is bad,data is no reliable , not processed.

8.2 Single-bit asynchronous signal synchronization design

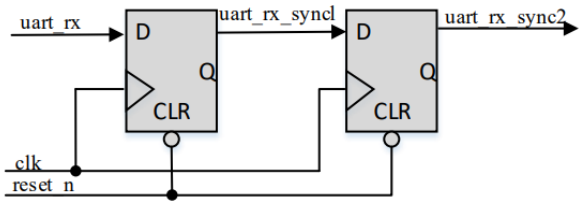


Figure 104: *Schematic logic design*

```
always@(posedge clk or posedge reset)
if(reset)begin
    uart_rx_sync1 <= 1'b0;
    uart_rx_sync2 <= 1'b0;
end
else begin
    uart_rx_sync1 <= uart_rx;
    uart_rx_sync2 <= uart_rx_sync1;
end
```

- Because asynchronous signal,signal is not stable(o or 1).
- First Synchronization Register (uart_rx_sync1): This register captures the asynchronous UART input signal uart_rx on the rising edge of the clock. If uart_rx changes state close to the clock edge, the output of this register might be metastable.
- Second Synchronization Register (uart_rx_sync2): This register captures the output of the first synchronization register on the next clock edge. By this time, the signal from uart_rx_sync1 is likely to have settled to a stable state, thus reducing the chance of metastability propagating further.
- The double synchronization ensures that any metastable state in uart_rx_sync1 has time to resolve before it affects the rest of the logic.

8.3 Edge detection

- Using for detect for falling edge to start uart state.

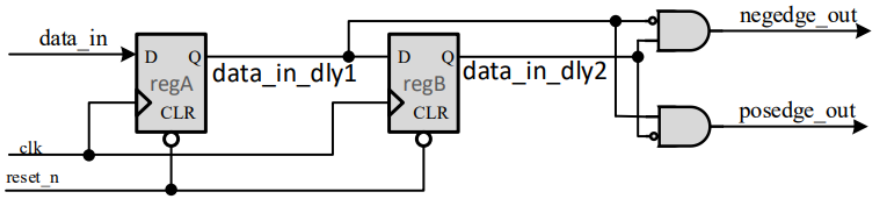


Figure 105: *principle of edge detection*

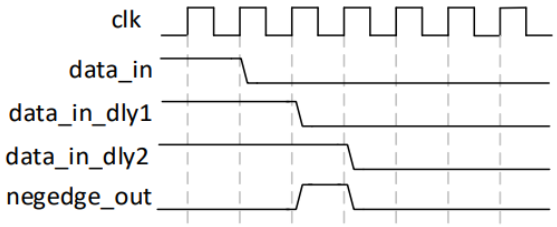


Figure 106: *Falling edge detection waveform*

```
always@(posedge clk or posedge reset)
if(reset)begin
    uart_rx_reg1 <= 1'b0;
    uart_rx_reg2 <= 1'b0;
end
else begin
    uart_rx_reg1 <= uart_rx_sync2;
    uart_rx_reg2 <= uart_rx_reg1;
end
assign uart_rx_nedge = !uart_rx_reg1 & uart_rx_reg2;
```

- For find rising edge , uart_edge=uart_rx_reg1 and !uart_rx_reg2

8.4 Design of the sample clock generation module

- If clock frequency =50MHZ,number of baurate clock bit is 16.(Formula:=(50000000/(16xbaudrate)))-1
  - baudrate:9600->324
  - 19200->162
  - 38400->80
  - 57600->53
  - 115200->26

```
always@(posedge clk or posedge reset)
if(reset)
    bps_DR <= 16'd324;
else begin
    case(baud_set)
        0:bps_DR <= 16'd324;
        1:bps_DR <= 16'd162;
        2:bps_DR <= 16'd80;
        3:bps_DR <= 16'd53;
        4:bps_DR <= 16'd26;
        default:bps_DR <= 16'd324;
endcase
```

- Counter for Baud Rate Clock Generation

```
always@(posedge clk or posedge reset)
if(reset)
   div_cnt <= 16'd0;
else if(uart_state)begin
   if(div_cnt == bps_DR)
      div_cnt <= 16'd0;
   else
      div_cnt <= div_cnt + 1'b1;
end
else
   div_cnt <= 16'd0;
```

- take enough clock for read one sample of bit

- Baud Rate Clock Generation (bps_clk)

```
always@(posedge clk or posedge reset)
if(reset)
   bps_clk <= 1'b0;
else if(div_cnt == 16'd1)
   bps_clk <= 1'b1;
else
   bps_clk <= 1'b0;
```

identify when start read a new sample bit.

- Baud Rate Counter (bps_cnt)

```
//bps counter
always@(posedge clk or posedge reset)
if(reset)
    bps_cnt <= 8'd0;
else if(bps_cnt == 8'd159 | (bps_cnt == 8'd12 && (START_BIT > 2)))
    bps_cnt <= 8'd0;
else if(bps_clk)
    bps_cnt <= bps_cnt + 1'b1;
else
    bps_cnt <= bps_cnt;
always@(posedge clk or posedge reset)
if(reset)
    rx_done <= 1'b0;
else if(bps_cnt == 8'd159)//(16x10-1)
    rx_done <= 1'b1;
else
    rx_done <= 1'b0;
```

- identify get enough sample of bits.
- WHY CHOOSING CASE =8'd12 and START_BIT>2:
  - because at 6->11 the sample of bit add to "START_BIT", the START_BIT We want to be low to start,position 12 we want to check that START_BIT,if >2 min there are more 3 sample with high mean START_BIT can be high not low.Therefore we stop becasue START_BIT is invalid.

8.5 Design of sampling data receiving module

```
always@(posedge clk or posedge reset)
   if(reset)begin
      START_BIT <= 3'd0;
      data_byte_pre[0] <= 3'd0;
      data_byte_pre[1] <= 3'd0;
      data_byte_pre[2] <= 3'd0;
      data_byte_pre[3] <= 3'd0;
      data_byte_pre[4] <= 3'd0;
      data_byte_pre[5] <= 3'd0;
      data_byte_pre[6] <= 3'd0;
      data_byte_pre[7] <= 3'd0;
      STOP_BIT <= 3'd0;
   end
   else if(bps_clk)begin
      case(bps_cnt)
         0:begin
        START_BIT <= 3'd0;
        data_byte_pre[0] <= 3'd0;
        data_byte_pre[1] <= 3'd0;
        data_byte_pre[2] <= 3'd0;
        data_byte_pre[3] <= 3'd0;
        data_byte_pre[4] <= 3'd0;
        data_byte_pre[5] <= 3'd0;
        data_byte_pre[6] <= 3'd0;
        data_byte_pre[7] <= 3'd0;
        STOP_BIT <= 3'd0;
      end
         6 ,7 ,8 ,9 ,10,11:START_BIT <= START_BIT + uart_rx_sync2;
         22,23,24,25,26,27:data_byte_pre[0] <= data_byte_pre[0] + uart_rx_sync2;
         38,39,40,41,42,43:data_byte_pre[1] <= data_byte_pre[1] + uart_rx_sync2;
         54,55,56,57,58,59:data_byte_pre[2] <= data_byte_pre[2] + uart_rx_sync2;
         70,71,72,73,74,75:data_byte_pre[3] <= data_byte_pre[3] + uart_rx_sync2;
         86,87,88,89,90,91:data_byte_pre[4] <= data_byte_pre[4] + uart_rx_sync2;
         102,103,104,105,106,107:data_byte_pre[5] <= data_byte_pre[5] + uart_rx_sync2;
         118,119,120,121,122,123:data_byte_pre[6] <= data_byte_pre[6] + uart_rx_sync2;
         134,135,136,137,138,139:data_byte_pre[7] <= data_byte_pre[7] + uart_rx_sync2;
         150,151,152,153,154,155:STOP_BIT <= STOP_BIT + uart_rx_sync2;
         default:
      begin
        START_BIT <= START_BIT;
        data_byte_pre[0] <= data_byte_pre[0];
```

```
        data_byte_pre[1] <= data_byte_pre[1];
        data_byte_pre[2] <= data_byte_pre[2];
        data_byte_pre[3] <= data_byte_pre[3];
        data_byte_pre[4] <= data_byte_pre[4];
        data_byte_pre[5] <= data_byte_pre[5];
        data_byte_pre[6] <= data_byte_pre[6];
        data_byte_pre[7] <= data_byte_pre[7];
        STOP_BIT <= STOP_BIT;
      end
      endcase
  end
```

- data_byte_pre is array with 8 register with 3bits ,store value .

8.6 Data status determination module

```
  always@(posedge clk or posedge reset)
  if(reset)
     data_byte <= 8'd0;
  else if(bps_cnt == 8'd159)begin
     data_byte[0] <= data_byte_pre[0][2];
     data_byte[1] <= data_byte_pre[1][2];
     data_byte[2] <= data_byte_pre[2][2];
     data_byte[3] <= data_byte_pre[3][2];
     data_byte[4] <= data_byte_pre[4][2];
     data_byte[5] <= data_byte_pre[5][2];
     data_byte[6] <= data_byte_pre[6][2];
     data_byte[7] <= data_byte_pre[7][2];
  end
```

- Because the data take 6 samples, so it should occur more than 3 so we take the value at position 3(if =1 mean 4,5,6 samples value of 1.)

8.7 TESTBENCHfor modelsim

```
`timescale 1ns/1ns
`define CLK_PERIOD 20

module uart_byte_rx_tb;
  reg clk;
  reg reset_n;
```

```
reg [2:0] baud_set;
reg [7:0] data_byte_tx;
reg send_en;
wire tx_done;
wire uart_state;
wire [7:0]data_byte_rx;
wire rx_done;
wire uart_tx_rx;

uart_byte_tx uart_byte_tx(
   .clk(clk),
   .reset_n(reset_n),

   .data_byte(data_byte_tx),
   .send_en(send_en),
   .baud_set(baud_set),

   .uart_tx(uart_tx_rx),
   .tx_done(tx_done),
   .uart_state(uart_state )
);

uart_byte_rx uart_byte_rx(
   .clk(clk),
   .reset_n(reset_n),

   .baud_set(baud_set),
   .uart_rx(uart_tx_rx),

   .data_byte(data_byte_rx),
   .rx_done(rx_done)
);

initial clk = 1;
always#(`CLK_PERIOD/2)clk = ~clk;

initial begin
   reset_n = 1'b0;
   data_byte_tx = 8'd0;
   send_en = 1'd0;
   baud_set = 3'd0;
   #(`CLK_PERIOD*20 + 1 );
   reset_n = 1'b1;
```

```
      #(`CLK_PERIOD*50);

    //send first byte
      data_byte_tx = 8'haa;
      send_en = 1'd1;
      #`CLK_PERIOD;
      send_en = 1'd0;

      @(posedge tx_done)
      #(`CLK_PERIOD*5000);

    //send second byte
      data_byte_tx = 8'h55;
      send_en = 1'd1;
      #`CLK_PERIOD;
      send_en = 1'd0;

      @(posedge tx_done)
      #(`CLK_PERIOD*5000);
      $stop;
   end

endmodule
```

8.8 Code for testing real device

```
`timescale 1ns/1ns
`define CLK_PERIOD 20

module uart_byte_rx_tb;
   reg clk;
   reg reset_n;
   reg [2:0] baud_set;
   reg [7:0] data_byte_tx;
   reg send_en;
   wire tx_done;
   wire uart_state;
   wire [7:0]data_byte_rx;
   wire rx_done;
   wire uart_tx_rx;

   uart_byte_tx uart_byte_tx(
```

```
    .clk(clk),
    .reset_n(reset_n),

    .data_byte(data_byte_tx),
    .send_en(send_en),
    .baud_set(baud_set),

    .uart_tx(uart_tx_rx),
    .tx_done(tx_done),
    .uart_state(uart_state )
);

uart_byte_rx uart_byte_rx(
    .clk(clk),
    .reset_n(reset_n),

    .baud_set(baud_set),
    .uart_rx(uart_tx_rx),

    .data_byte(data_byte_rx),
    .rx_done(rx_done)
);

initial clk = 1;
always#(`CLK_PERIOD/2)clk = ~clk;

initial begin
    reset_n = 1'b0;
    data_byte_tx = 8'd0;
    send_en = 1'd0;
    baud_set = 3'd0;
    #(`CLK_PERIOD*20 + 1 );
    reset_n = 1'b1;
    #(`CLK_PERIOD*50);

 //send first byte
    data_byte_tx = 8'haa;
    send_en = 1'd1;
    #`CLK_PERIOD;
    send_en = 1'd0;

    @(posedge tx_done)
    #(`CLK_PERIOD*5000);
```

```
    //send second byte
      data_byte_tx = 8'h55;
      send_en = 1'd1;
      #`CLK_PERIOD;
      send_en = 1'd0;

      @(posedge tx_done)
      #(`CLK_PERIOD*5000);
      $stop;
   end

endmodule
```

8.9 Simulation

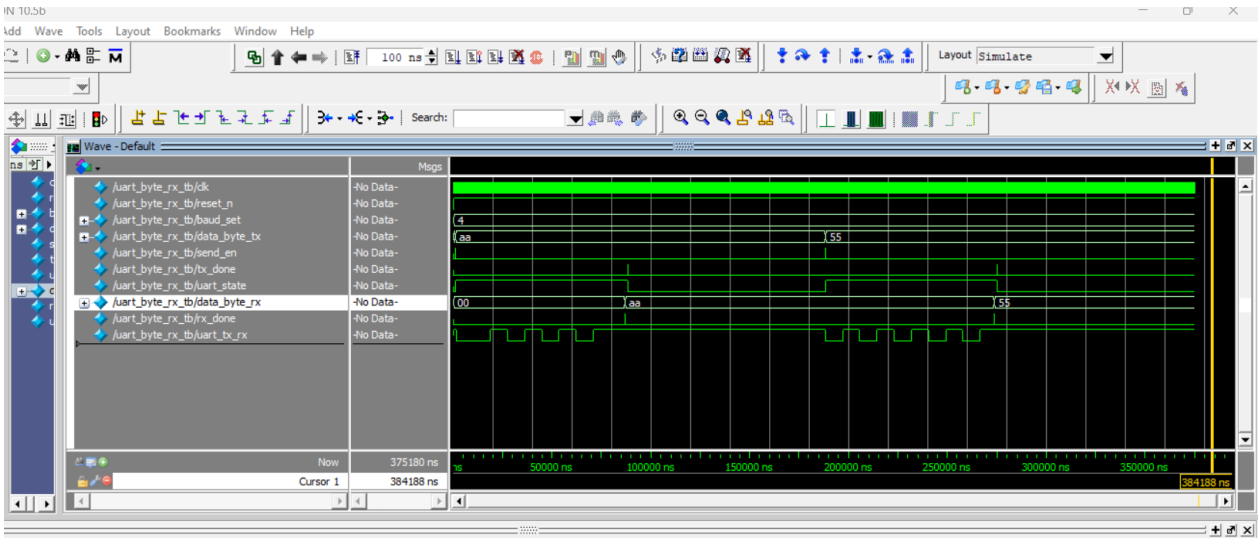


Figure 107: *UART receiver simulation with modelsim*

8.10 Testing with board

- Using the hercules software.

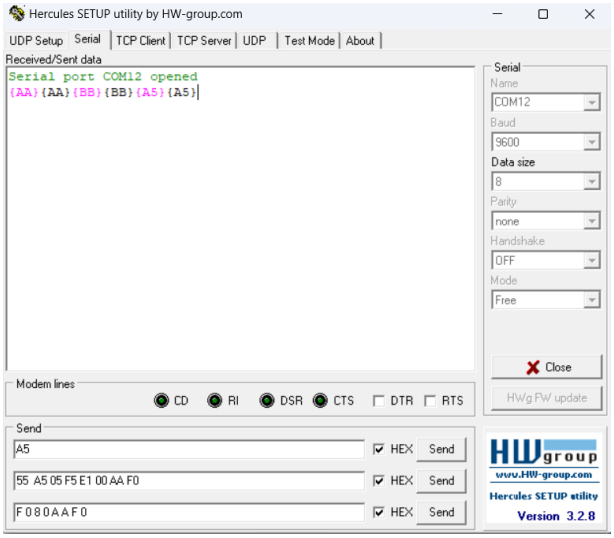


Figure 108: *UART receiver testing with board*

9 UART_RX Control LED

9.1 Defines the serial port transmission protocol frame

55	A5	time_set [31:24]	time_set [23:16]	time_set [15:8]	time_set [7:0]	ctrl [7:0]	F0

Figure 109: *Custom serial port transmission protocol frames*

- Header:
  - 0x55:First byte of header
  - 0xA5:Second byte of header
  - The header helps the FPGA identify the start of a valid data frame.
- Data:
  - time_set[31:24]:The most significant byte of the time setting
  - time_set[23:16]:The second significant byte of the time setting
  - time_set[15:8]:The third byte of the time setting
  - time_set[7:0]: The last byte of the time settting
  - ctrl[7:0]:the control word byte.
  - These 5 bytes contain the actual data to be used for controlling the LEDs. The time_set bytes might specify a timing parameter, while the ctrl byte specifies control commands.
- tail:
  - 0xF0: This byte marks the end of the frame.
  - The tail ensures that the entire frame is received before processing the data.

9.2 The principle and idea of serial port control LED

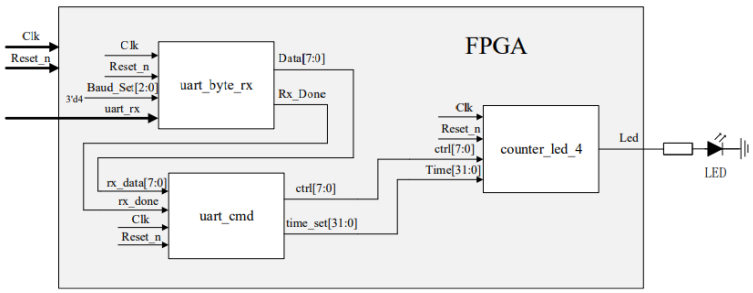


Figure 110: *Design a block diagram*

- 3 component and modules:
  - Serial data reception(uart_byte_rx)
    - Receiving serial datas .
    - Operating on a single byte at a time.
    - Output: Data[7:0] and signal indicate completion(Rx_Done)
  - Command conversion(uart_cmd)
    - Collecting multiple single-byte data output form uart_byte_rx
    - Caching 8 bytes of data and checks if the data forms a valid frame based on the predefined protocol.
    - If valid, it extracts the state control word (ctrl[7:0]) and the periodic control word (time_set[31:0]).
  - LED Control(counter_led_4)
    - Controlling words to manage the LED's behavior.
    - Using a counter driven by the periodic control word to determine the timing of LED state changes.
    - The state control word defines the specific on/off patterns of the LED.

9.3 Design of Serial Port Command Conversion Module (uart_cmd)

- The serial port receiving module will transmit one byte of data at a time, and according to our custom serial port transmission protocol frame, a minimum of 8 bytes is required to determine whether the data is used to control the LED. To do this, we need to cache the data, as follows:

```
input Clk;
input [7:0]rx_data;
reg [7:0] data_str [7:0];
always@(posedge Clk)
    if(rx_done)begin
        data_str[7] <= #1 rx_data;
        data_str[6] <= #1 data_str[7];
        data_str[5] <= #1 data_str[6];
        data_str[4] <= #1 data_str[5];
        data_str[3] <= #1 data_str[4];
        data_str[2] <= #1 data_str[3];
        data_str[1] <= #1 data_str[2];
        data_str[0] <= #1 data_str[1];
end
```

- It shift the data like below:

Figure 111: *example of shift data*

- The biggest advantage of this method is that the data cached is continuously changing and consistent with the frame length of the serial transmission protocol. In this way, we only need to perform a frame structure check on the 8 bytes of data in the data_str once when we receive new bytes of data, rather than checking all the data that has already been received. Once the data in the data_str satisfies the serial transmission protocol frame we defined, the corresponding control word output is taken from it. Here's the code:
- After receiving 8 bytes, the module checks if the received sequence matches the expected protocol frame:

```
input Reset_n;
output reg[7:0]ctrl;
output reg[31:0]time_set;
reg r_rx_done;
always@(posedge Clk)
    r_rx_done <= rx_done;
always@(posedge Clk or negedge Reset_n)
    if(!Reset_n) begin
        ctrl <= #1 0;
        time_set <= #1 0;
    end
    else if(r_rx_done)begin
        if((data_str[0]==8'h55)&&(data_str[1]==8'hA5)&&(data_str[7]==8'hF0))begin
            time_set[31:24] <= #1 data_str[2];
            time_set[23:16] <= #1 data_str[3];
            time_set[15:8] <= #1 data_str[4];
            time_set[7:0] <= #1 data_str[5];
            ctrl <= #1 data_str[6];
        end
    end
```

9.4 Counting drive module design(counter_led_4)

- The counting driver module needs to control the word count and generate the flag signal according to the period, and the specified IO according to the status control word control to generate high/low levels, and drive the LED to turn on and off. The generation of the flag signal is very simple, you only need to define a counter, and when the value of the counter reaches the period control word, let the flag signal add itself. Here's the code:

```
input Clk;
input Reset_n;
input [31:0]Time;
reg [31:0]counter;
always@(posedge Clk or negedge Reset_n)
if(!Reset_n)
    counter <= #1 0;
else if(counter >= Time - 1)
    counter <= #1 0;
else
    counter <= #1 counter + 1'b1;
    reg [2:0]counter2;
always@(posedge Clk or negedge Reset_n)
if(!Reset_n)
    counter2 <= #1 0;
else if(counter >= Time - 1)
    counter2 <= #1 counter2 + 1'b1;
```

- The counter2 bit width of the flag signal here is 3, which can be used to represent 0 7, and when it exceeds 7, it will automatically overflow to zero. The control word used to control the LED on and off state also has exactly 8 bits, so we can use the case statement to drive the LED on and off according to the value of counter2, the code is as follows:

```
always@(posedge Clk or negedge Reset_n)
    if(!Reset_n)
        Led <= #1 0;
    else case(counter2)
        0:Led <= #1 Ctrl[0];
        1:Led <= #1 Ctrl[1];
        2:Led <= #1 Ctrl[2];
        3:Led <= #1 Ctrl[3];
        4:Led <= #1 Ctrl[4];
        5:Led <= #1 Ctrl[5];
        6:Led <= #1 Ctrl[6];
        7:Led <= #1 Ctrl[7];
```

```
            default:Led <= #1 Led;
        endcase
```

- Whenever the counter reaches the counter value, the value of counter2 is self-incremented, causing the LED to change its drive. When the value of counter2 exceeds 7, it overflows to zero and becomes 0 again, so that the LED can cycle through the 8 states of the input at a certain frequency.

9.5 Create a top-level package

```
module uart_rx_ctrl_led(
    Clk,
    Reset_n,
    Led,
    uart_rx
);

    input Clk;
    input Reset_n;
    output Led;
    input uart_rx;

    wire [7:0]ctrl;
    wire [31:0]time_set;
    wire [7:0]rx_data;
    wire rx_done;

    parameter Baud_Set = 3'd4;

    counter_led_4 counter_led(
        .Clk(Clk),
        .Reset_n(Reset_n),
        .Ctrl(ctrl),
        .Time(time_set),
        .Led(Led)
    );

    uart_cmd uart_cmd(
        .Clk(Clk),
        .Reset_n(Reset_n),
        .rx_data(rx_data),
        .rx_done(rx_done),
```

```
        .ctrl(ctrl),
        .time_set(time_set)
    );

    uart_byte_rx uart_byte_rx(
        .Clk(Clk),
        .Reset_n(Reset_n),
        .Baud_Set(Baud_Set),
        .uart_rx(uart_rx),
        .Data(rx_data),
        .Rx_Done(rx_done)
    );

endmodule
```

9.6 TESTBENCH for simulation

```
`timescale 1ns / 1ps
module uart_rx_ctrl_led_tb();

    reg Clk;
    reg Reset_n;
    wire Led;
    reg uart_rx;
    wire [31:0] delay_time;

    uart_rx_ctrl_led uart_rx_ctrl_led(
        .Clk(Clk),
        .Reset_n(Reset_n),
        .Led(Led),
        .uart_rx(uart_rx)
    );
    parameter Baud_Set = 3'd4;

    assign delay_time = (Baud_Set == 3'd0) ? 20'd104166:
                        (Baud_Set == 3'd1) ? 20'd52083:
                        (Baud_Set == 3'd2) ? 20'd26041:
                        (Baud_Set == 3'd3) ? 20'd17361:
                                             20'd8680;


    initial Clk = 1;
```

```
always#10 Clk = ~Clk;

initial begin
   Reset_n = 0;
   uart_rx = 1;
   #201;
   Reset_n = 1;
   #200;

   uart_tx_byte(8'h55);
   #(delay_time*10);
   uart_tx_byte(8'ha5);
   #(delay_time*10);
   uart_tx_byte(8'h55);
   #(delay_time*10);
   uart_tx_byte(8'ha5);
   #(delay_time*10);
   uart_tx_byte(8'h00);
   #(delay_time*10);
   uart_tx_byte(8'h00);
   #(delay_time*10);
   uart_tx_byte(8'hc3);
   #(delay_time*10);
   uart_tx_byte(8'h50);
   #(delay_time*10);
   uart_tx_byte(8'haa);
   #(delay_time*10);
   uart_tx_byte(8'hf0);
   #(delay_time*10);


   uart_tx_byte(8'h55);
   #(delay_time*10);
   uart_tx_byte(8'ha5);
   #(delay_time*10);
   uart_tx_byte(8'h9a);
   #(delay_time*10);
   uart_tx_byte(8'h78);
   #(delay_time*10);
   uart_tx_byte(8'h56);
   #(delay_time*10);
   uart_tx_byte(8'h34);
   #(delay_time*10);
```

```
        uart_tx_byte(8'h12);
        #(delay_time*10);
        uart_tx_byte(8'hf1);
        #(delay_time*10);
        #15000000;
        $stop;
    end

    task uart_tx_byte;
        input [7:0]tx_data;
        begin
            uart_rx = 1;
            #20;
            uart_rx = 0;
            #delay_time;
            uart_rx = tx_data[0];
            #delay_time;
            uart_rx = tx_data[1];
            #delay_time;
            uart_rx = tx_data[2];
            #delay_time;
            uart_rx = tx_data[3];
            #delay_time;
            uart_rx = tx_data[4];
            #delay_time;
            uart_rx = tx_data[5];
            #delay_time;
            uart_rx = tx_data[6];
            #delay_time;
            uart_rx = tx_data[7];
            #delay_time;
            uart_rx = 1;
            #delay_time;
        end
    endtask


endmodule
```

9.7 Simulation

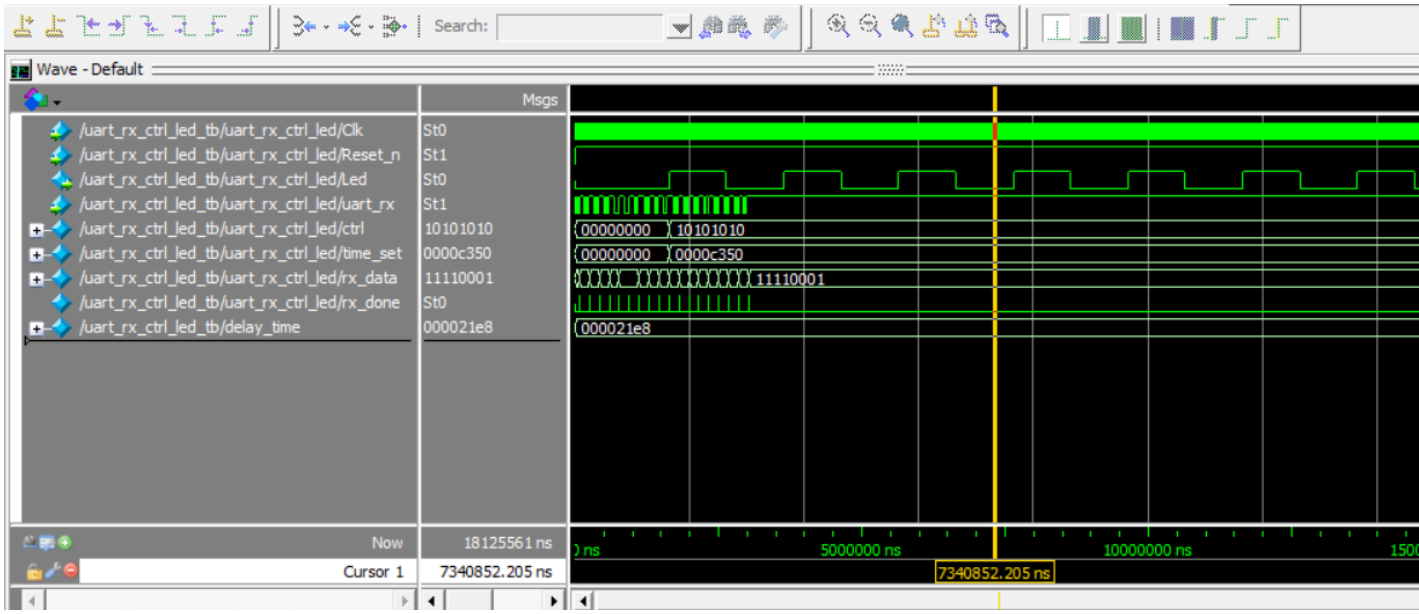


Figure 112: *Simulation with modelsim*

10 Addition and subtraction counter based on modular design

10.1 Module function division

- In order to realize the addition and subtraction of two button control LED lights according to the binary counting mode, the top module key_led_top can be divided into two button debounce modules and an LED control module, and the connection between the modules is shown in Figure below.

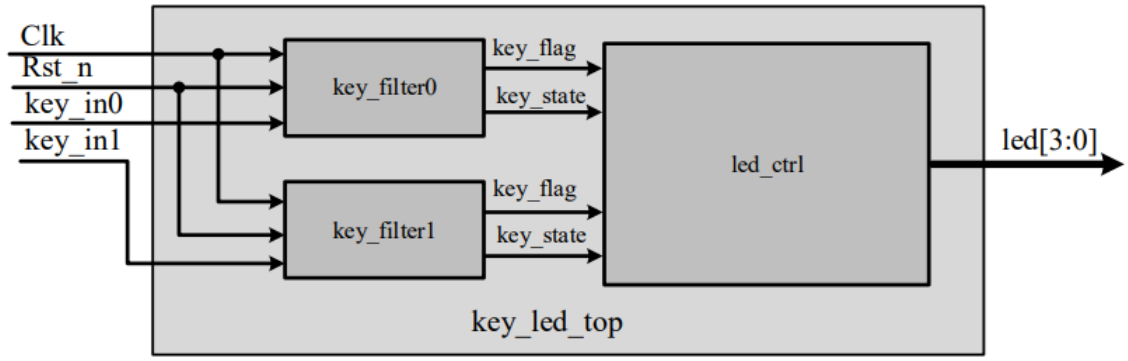


Figure 113: *key_led design module*

10.2 Function design of module

- The key_filter here has already been written in the previous sections, and can be called in the key_led_top. Now write a led_ctrl, and from the following figure you can get a list of its ports as follows.

```
input Clk;
input Rst_n;
input key_flag0,key_flag1;
input key_state0,key_state1;
output [3:0]led;
```

- This module needs to add and subtract the counter according to the state of the two buttons, and from the previous section, it can be seen that when key_flag and key_state is 1, it is the key pressed.

```
reg [3:0]led_r;
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
led_r <= 4'b0000;
else if(key_flag0 && !key_state0)
```

```
led_r <= led_r + 1'b1;
else if(key_flag1 && !key_state1)
led_r <= led_r - 1'b1;
else
led_r <= led_r;
```

- The initial value of the led_r of the counter is $4'B0000$, here when the key 0 is pressed, the counter is increased by one, and the counter becomes $4'B0001$, as can be seen from the LED light circuit diagram on the development board, the LED light is lit at a high level, and the value of led_r is directly output to the LED, and you can see that the LED light is on and off The effect is dim and dim.

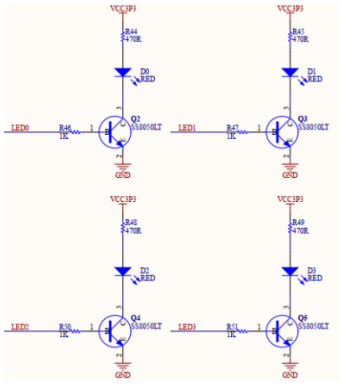


Figure 114: *led circuit diagram*

- Now that the individual modules are written, let's start the design of the top-level file, instantiate the three modules and set the internal signal type to wire in the new key_led_top.

```
module key_led_top(Clk,Rst_n,key_in0,key_in1,led);
    input Clk;
   input Rst_n;
   input key_in0;
   input key_in1;

   output [3:0]led;

   wire key_flag0,key_flag1;
   wire key_state0,key_state1;

   key_filter key_filter0(
      .Clk(Clk),
      .Rst_n(Rst_n),
      .key_in(key_in0),
```

```
    .key_flag(key_flag0),
    .key_state(key_state0)
);

key_filter key_filter1(
    .Clk(Clk),
    .Rst_n(Rst_n),
    .key_in(key_in1),
    .key_flag(key_flag1),
    .key_state(key_state1)
);

led_ctrl led_ctrl0(
    .Clk(Clk),
    .Rst_n(Rst_n),
    .key_flag0(key_flag0),
    .key_flag1(key_flag1),
    .key_state0(key_state0),
    .key_state1(key_state1),
    .led(led)
);

endmodule
```

11 LED_BCD

11.1 Digital tube drive principle

- This board is used common cathode. The common end is connected to the ground, the corresponding segment needs to be low.

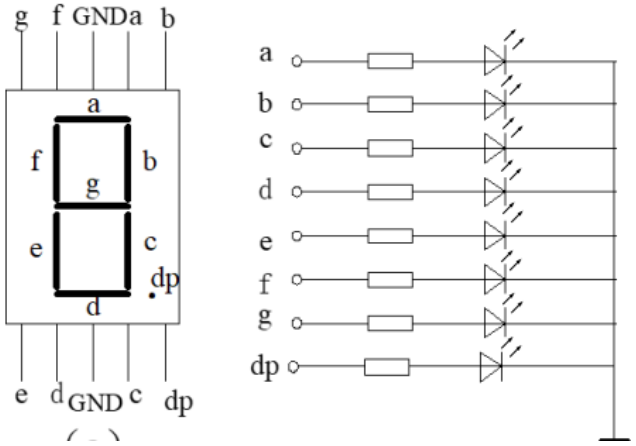


Figure 115: *LED type*

- Digital tube coding and decoding table

Data_disp	a	b	c	d	e	f	g	h	
0	0	0	0	0	0	0	1	1	8`hc0
1	1	0	0	1	1	1	1	1	8`hf9
2	0	0	1	0	0	1	0	1	8`ha4
3	0	0	0	0	1	1	0	1	8`hb0
4	1	0	0	1	1	0	0	1	8`h99
5	0	1	0	0	1	0	0	1	8`h92
6	0	1	0	0	0	0	0	1	8`h82
7	0	0	0	1	1	1	1	1	8`hf8
8	0	0	0	0	0	0	0	1	8`h80
9	0	0	0	0	1	0	0	1	8`h90
a	0	0	0	1	0	0	0	1	8`h88
b	1	1	0	0	0	0	0	1	8`h83
c	0	1	1	0	0	0	1	1	8`hc6
d	1	0	0	0	0	1	0	1	8`ha1
e	0	1	1	0	0	0	0	1	8`h86
f	0	1	1	1	0	0	0	1	8`h8e

Figure 116: *LED encoding and decoding table*

- There are two working modes of segment digital tube: static display mode and dynamic display mode. The static display is characterized by an 8-digit cable that must be connected to an 8-digit data line to maintain the displayed glyph code. When a glyph is fed once, the

display glyph can be maintained until a new glyph is fed. This method is because each digital tube needs independent data line, so the hardware circuit is more complex, the cost is higher, and it is rarely used.

- In order to save IO and cost, the circuit structure shown in Figure below is generally adopted, so that the 3 digital tubes connected together have 7*2 less I/O than the static one.

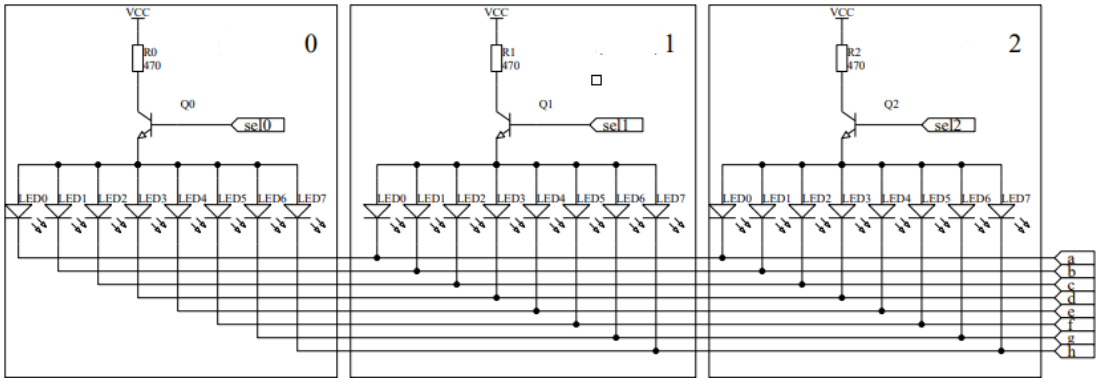


Figure 117: *Three-digit digital tube equivalent circuit diagram*

- This enables another display mode, dynamic display. The dynamic display is characterized by the parallel connection of the segment selection line of all digit code tubes, and the bit selection line controls which digit digital tube is valid. Select the bright digital tube collection display with dynamic scanning. The so-called dynamic scanning display is to take turns to send out the glyph code and the corresponding bit selection to each digital tube, using the afterglow of the luminous tube and the visual retention effect of the human eye, so that people feel as if all the digital tubes are displayed at the same time.

- Now for example, suppose the scanning time is set to 1S, the three digital tubes are divided into 3s, and the sel data line is 'b100 at the first second, then the digital tube 0 is selected, then a=0, and the LED0 of the digital tube 0 can be lit; In the second 2, the sel data line is 'b010, then the digital tube 1 is selected, then b=0, and the LED1 of the digital tube 1 can be lit; In the 3rd second, the sel data line is 'b001, then the digital tube 2 is selected, then c=0, and the LED2 of the digital tube 2 can be lit. At this time, the effect will be that the LED0 of the digital tube 0 will be on for one second, then the LED1 of the digital tube 1 will be on for one second, and finally the LED2 of the digital tube 2 will be on for one second, and so on again.

- In this way, if the 1ms refresh time is used, due to the afterglow effect of the digital tube and the persistence of human vision, the LED0 of the digital tube 0, the LED1 of the digital tube 1 and the LED2 of the digital tube 2 will be lit "at the same time" and there will be no flicker.

- This is schematic of led7 segments in board

Figure 118: *Cascade schematic of two 7-segment 4-digit digital tubes*

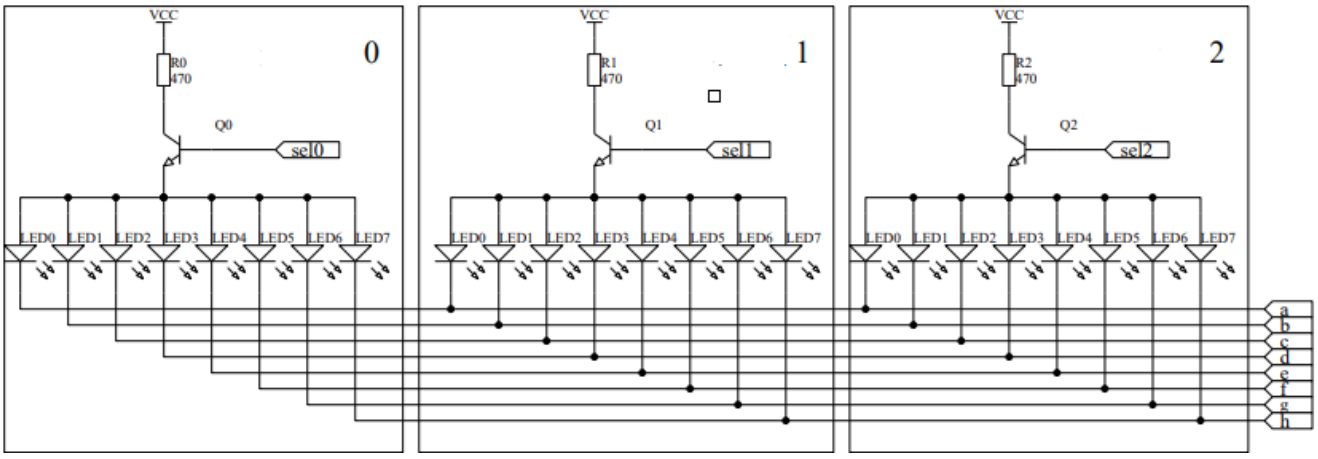


Figure 119: *The 7-segment, 8-digit digital tube uses the 74HC595 drive schematic*

11.2 Digital tube dynamic scan drive design

11.2.1 Module interface design and internal function division

- From the above analysis, we can get the input-output block diagram of Figure below, and the list of interfaces is shown in the following table

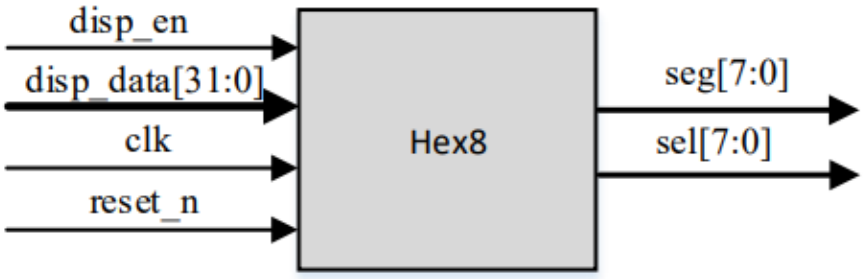


Figure 120: *Module hex8*

Signal	I/O	Description of the function
clk	I	Clock 50MHZ
reset_n	I	reset signal
disp_en	I	digital tube enable signal
disp_data[31:0]	I	8 digital tube data to be displayed , every four digits form a BCD code
sel[7:0]	O	digital tube position selection
seg[7:0]	O	digital tube segment selection, the current content to be displayed

Table 2: *List of module interfaces*

- According to the above analysis, it can be seen that there must be a driving clock with a period of 1ms first, so a frequency division circuit is required; When the position selection of the digital tube is carried out, a cyclic shift is required; After selecting the bits, a selector is required to strobe the data input bits; To realize the function of the digital tube codec table, a decoder is required. The logic circuit diagram of the digital tube driver module can be simplified to the following Figure , and the role of each part is shown in Table below.

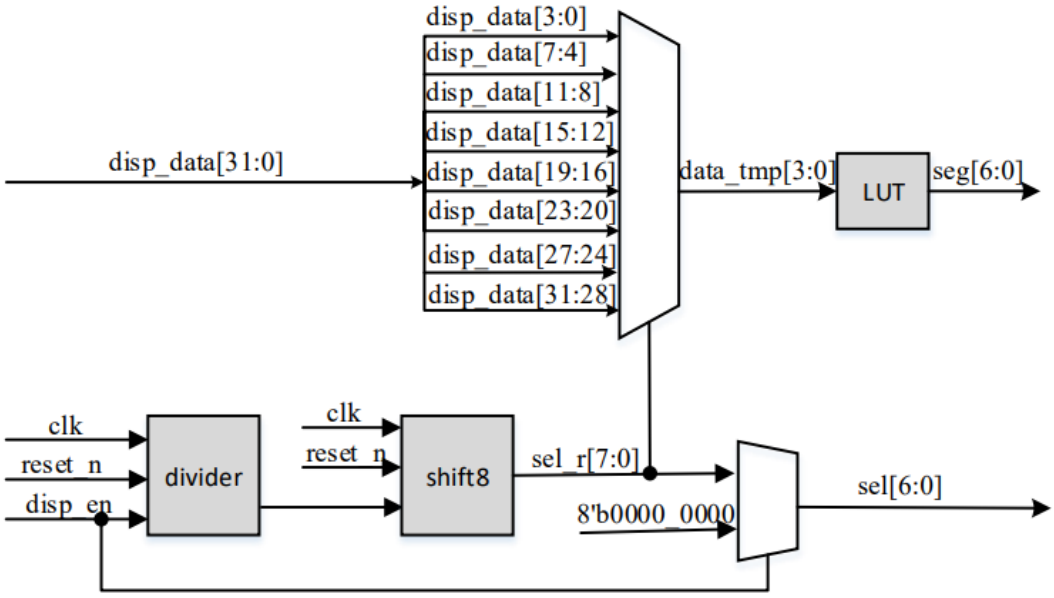


Figure 121: *Logic circuit diagram of the digital tube driver module*

Name	Function description
divder	The crossover generates a scan clock of 1KHz
shift8	8bit cyclic shift register
MUX8	Data input :select
MUX2	Enable
LUX	Data Decoder

Table 3: *Description of the function of the submodule*

11.3 74HC595 Drive module design

- The 74HC595 chip is used in the digital tube design on the development board, which functions as a shift register to save the pins of the FPGA by shifting it. The FPGA only needs to output 3 pins to achieve the purpose of sending digital tube data, which greatly saves IO design resources compared to the traditional segment selection and bit selection methods. The specific usage methods and technical parameters of the 74HC595 are as follows: As can be seen in the data sheet, the operating frequency values of the 74HC595 chip are different at different operating temperatures and operating voltages, as shown in Tables . Since the chip in the learning board uses 3.3V power supply, when designing its working frequency, it directly uses the clock after the 50M crystal four-way frequency as its working clock, so that it can work at 12.5M frequency at 3.3V.

SYMBOL	PARAMETER	Vcc(V)	MIN	TYP	MAX	UNIT
f_{max}	maximum clock frequency SH_CP and ST_CP	2.0	4.8	-	-	MHz
		4.5	24	-	-	MHz
		6.0	28	-	-	MHz
T_{amb} = -40to + 125℃						

Figure 122: *The 74HC595 drives the I/O ports*

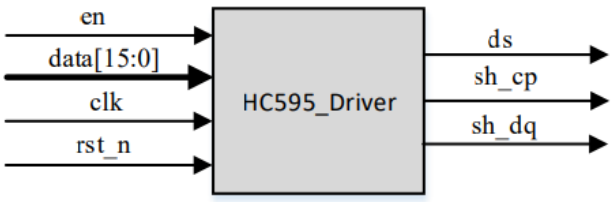


Figure 123: *74HC595 Driver I/O Port Function Descriptions*

Signal	I/O	Function description
clk	I	50M Clock
rst_n	I	Reset signal
en	I	Digital tube enable signal (1 enabled,0 off)
data[15:0]	I	8 Digital tube SEG and SEL data for digital tubes
ds	O	Serial data output
sh_cp	O	Clock output for shift register
st_cp	O	Clock output for memory registers

Table 4

11.4 Testing with Board

- Display "1234ABCD"

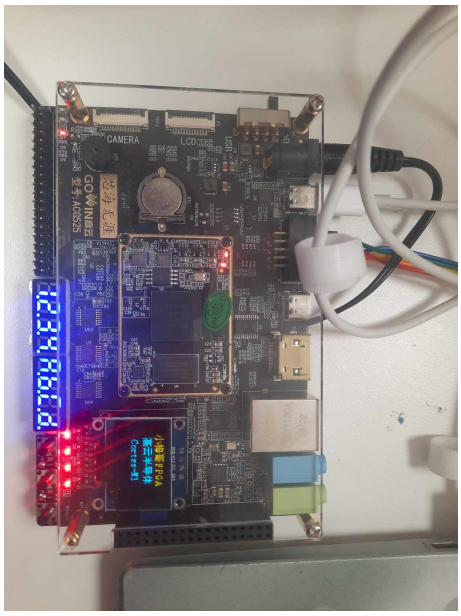


Figure 124: *Testing with board*

12 Result and knowledge achieved

- Having experience working in a company environment.
- Gained experience with FPGA programming and development tools.
- Enhanced understanding of hardware description languages such as VHDL or Verilog.
- Improved debugging and problem-solving skills related to FPGA design and implementation.
- Working and communicating with foreign engineer.
- Gain knowledge about reading schematic,drawing and routing PCB_layout design
- Having chance take part in workshop about technology.