

## General Tips:

Please skim through this entire page before attempting the lab. It's important to understand the big picture as well as the minutia, or as an engineer would say, the system and the microarchitecture.

## Objectives

- Learn the very basics about Field-programmable gate arrays (FPGAs) | [FPGA Wiki Page](#) | [FPGA High Level Tutorial](#) | [FPGA YouTube Video \(37 minutes\)](#) | [FPGA YouTube Video \(1 minute\)](#)
- Familiarize with the Hardware Description Language (HDL) Verilog | [Verilog vs. VHDL](#) | [Verilog Tutorial](#)
- Learn to implement a hardware module and a testbench
- Familiarize with the IcebreakerV1 (iCE40 Ultra 5K) FPGA Board and utilizing Lattice Open Source software

## Background

In this lab you will be utilizing (software) and the IcebreakerV1, or more specifically the Lattice iCe40 UltraPlus5K. This board is created by Lattice Semiconductors and is beginner friendly. Clicking [here](#) will allow you to download and read through the datasheet for the device

## Project Description:

The goal of this Project is to create a step-by-step tutorial to implement various Verilog onto a new board - The Lattice iCEbreaker V1. Once that is complete we will design a full lab, similar to the ones utilized in class, that can be ran through by new FPGA students with no issues.

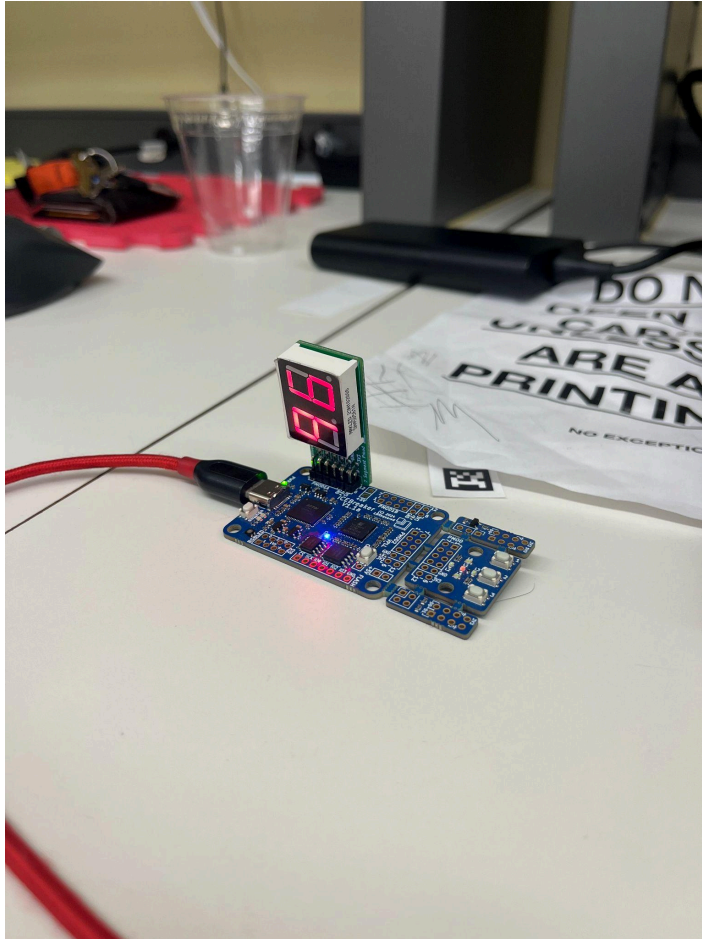
## Key Objectives:

- Download Software for Lattice FPGA Board
- Implement 7-segment display onto the FPGA Board
- Create a fully operational lab using the steps that we ourselves used.

## Expected Outcomes:

We hope to deliver a finished lab that will allow any student with the FPGA Board to run a 7-segment display or other files without the need of outside assistance.

## Hardware setup:



Solder the 7-segment display to the iCEBreaker board in the same fashion to this, connecting all 12 pins into the PMOD1A interface.

You could solder the 7-segment display into a different PMOD interface (other than 1A), but then you will need change the pinout in the physical constraints file (\*.pcf). The supplied \*.pcf file assumes it is in the PMOD1A port shown here.

## Software setup:

- 1st make sure you have a recent version of Python installed on your computer.
  - Windows: Make sure when you install python that select “Add python.exe to Path” so that the Python Package Manager pip is available
  - Linux: make sure your package manager is all up to date with “sudo apt update”, “sudo apt upgrade”, “sudo apt install python3-pip –fix-missing”
- We will use the [APIO open source ecosystem](#) which includes yosys for snthesis, next-pnr for placement and routing, and icepack for bitstream generation. We will note the most important steps but if you want further details see these [install instructions](#).
- Next, run “sudo pip install -U apio” on Linux or “pip install -U apio” on Windows

- Then type “apio” after it is successfully installed. Make sure the command works and you get an output like this:

```
floranicolas@DESKTOP-J6IQ92L:~$ apio
Usage: apio [OPTIONS] COMMAND [ARGS]...

Work with FPGAs with ease

Options:
  --version    Show the version and exit.
  -h, --help   Show this message and exit.

Project commands:
  build        Synthesize the bitstream.
  clean        Clean the previous generated files.
  graph        Generate a a visual graph of the verilog code.
  lint         Lint the verilog code.
  sim          Launch the verilog simulation.
  test         Launch the verilog testbench testing.
  time         Bitstream timing analysis.
  upload       Upload the bitstream to the FPGA.
  verify       Verify the verilog code.

Setup commands:
  drivers      Manage FPGA boards drivers.
  init         Manage apio projects.
  install      Install apio packages.
  uninstall    Uninstall packages.

Utility commands:
  boards       Manage FPGA boards.
  config       Apio configuration.
  drivers      Manage FPGA boards drivers.
  examples     Manage verilog examples.
  raw          Execute commands directly from the Apio packages
  system       System tools.
  upgrade      Check the latest Apio version.
```

- Next run “apio install oss-cad-suite”

```

floranicholas@DESKTOP-J6IQ92L:~$ apio install oss-cad-suite
File version.txt downloaded!
Version: 0.0.9
Installing oss-cad-suite package:
Download tools-oss-cad-suite-linux_x86_64-0.0.9.tar.gz
Downloading [ ] 100%
Unpacking.. [ ] 100%
Package 'oss-cad-suite' has been successfully installed!

```

- Next run “apio install gtkwave”
  - Linux: If that does not work, run “sudo apt install gtkwave”
- Next run “apio install examples”

```

floranicholas@DESKTOP-J6IQ92L:~$ apio install examples
File version.txt downloaded!
Version: 0.0.36
Installing examples package:
Download apio-examples-0.0.36.zip
Downloading [ ] 100%
Unpacking.. [ ] 100%
Package 'examples' has been successfully installed!

```

- Next run “apio examples -l”. This will give a large output of various different example project names and their descriptions for different boards supported by the APIO toolsuite. The board we are using for this tutorial is the iCEBreaker, so scroll until you see the four examples shown below.

```

iCEBreaker/blink
Blink the on board Red and Green LEDs as well as the 5 LEDs found on the default PMOD P2 module.

iCEBreaker/buttons
Example of controlling the LED using the tact buttons on the board.

iCEBreaker/Blinky
Blink the on board Green LED

iCEBreaker/led-green
Turning the on board Green LED on but all the other controllable LED off.

iCESugar 1.5/Blinky

```

- Now pick an example program to run. We will do blinky in this tutorial, which simply blinks a green LED on the board. This is as simple as it gets and a basic hello world of the embedded or FPGA industry.
- Next, change directories to wherever you want your example project to be copied to. Run “apio examples -d iCEBreaker/Blinky” and this will create an iCEbreaker folder with a Blinky subfolder. In the screenshot below the command is executed from the home folder ~, so that is where the iCEBreaker folder goes.

```

floranicholas@DESKTOP-J6IQ92L:~$ ls
CS3130
floranicholas@DESKTOP-J6IQ92L:~$ apio examples -d iCEBreaker/Blinky
Creating iCEBreaker/Blinky directory ...
Example 'iCEBreaker/Blinky' has been successfully created!
floranicholas@DESKTOP-J6IQ92L:~$ ls
CS3130  iCEBreaker
floranicholas@DESKTOP-J6IQ92L:~$ cd iCEBreaker
floranicholas@DESKTOP-J6IQ92L:~/iCEBreaker$

```

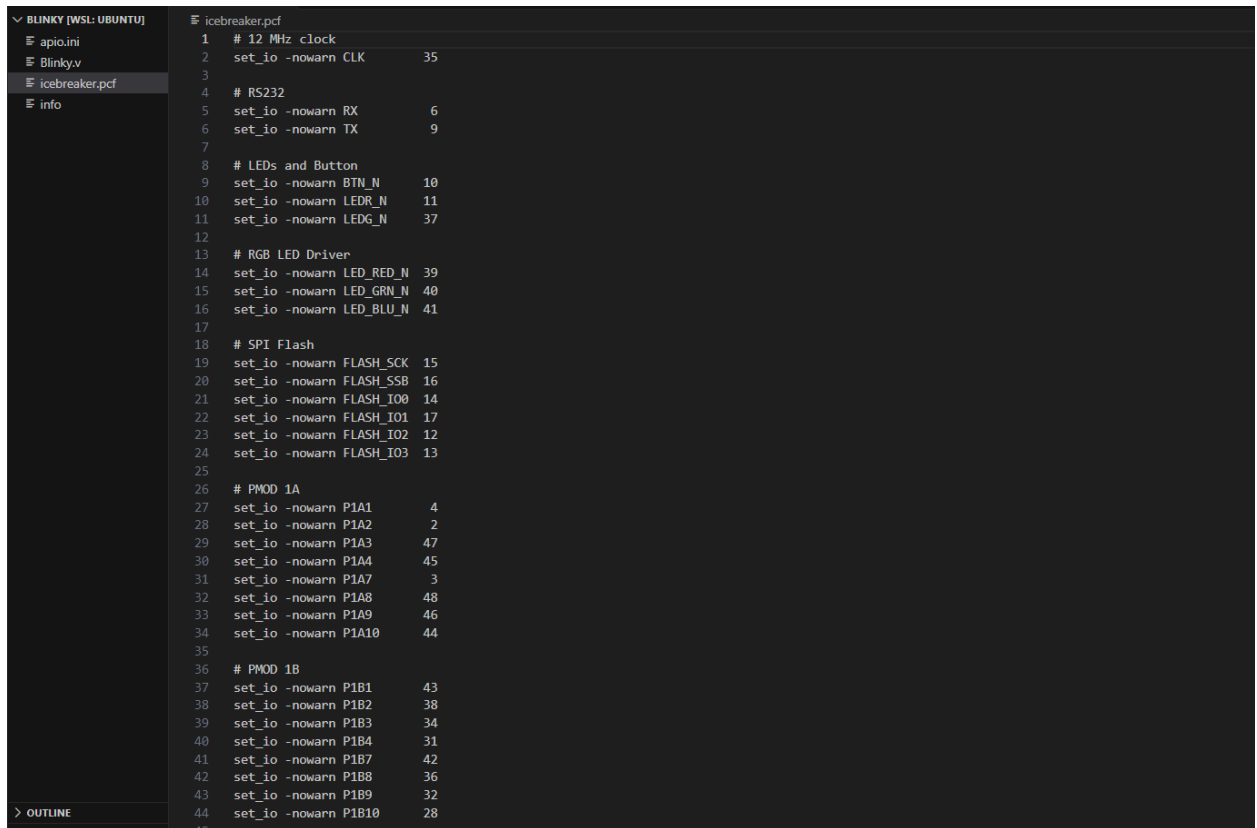
- Open the Blinky folder in your favorite text editor such as VSCode to have a look at the files. You will notice 4 files. There is an “info” file with a description of the example. A “Blinky.v” verilog RTL file.

```

1  //-----
2  //-- Blinking LED
3  //-- The Green LED is blinking. The other LEDs are turned off
4  //-----
5  module blink_test (
6      input CLK,          //-- 12 Mhz
7      output LEDG_N,      //-- Green led to blink
8      output LEDR_N,
9      output LED1,
10     output LED2,
11     output LED3,
12     output LED4,
13     output LED5
14 );
15
16 reg [23:0] counter = 0;
17
18 always @(posedge CLK)
19     counter <= counter + 1;
20
21 //-- Toggle the green LED
22 assign LEDG_N = counter[23];
23
24 //-- Turn off the other LEDs
25 assign LEDR_N = 3'b1;
26 assign {LED5, LED4, LED3, LED2, LED1} = 5'b0;
27
28 endmodule
29

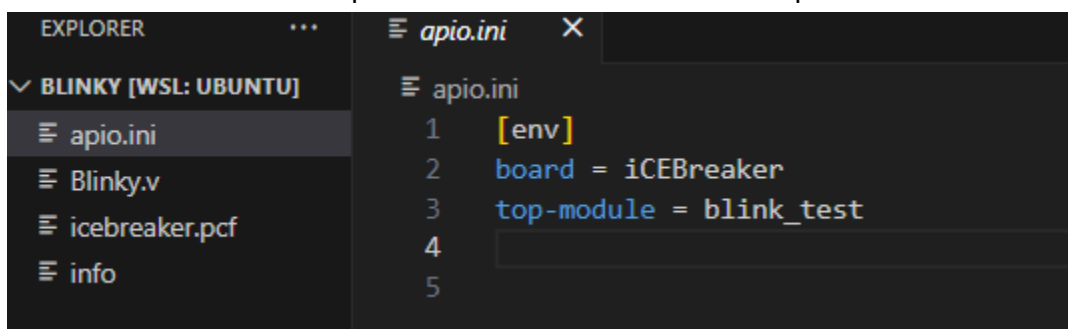
```

- There will be an “icebreaker.pcf” file. PCF stands for physical constraints file. This is how you map between input and output on your top most module in the design and physical pins in the FPGA. This is equivalent to the Quartus Pin Planner or the Xilinx Design Constraint (.xdc) file if you are familiar with those vendors.



```
1 # 12 Mhz clock
2 set_io -nowarn CLK 35
3
4 # RS232
5 set_io -nowarn RX 6
6 set_io -nowarn TX 9
7
8 # LEDs and Button
9 set_io -nowarn BTN_N 10
10 set_io -nowarn LEDR_N 11
11 set_io -nowarn LEDG_N 37
12
13 # RGB LED Driver
14 set_io -nowarn LED_RED_N 39
15 set_io -nowarn LED_GRN_N 40
16 set_io -nowarn LED_BLU_N 41
17
18 # SPI Flash
19 set_io -nowarn FLASH_SCK 15
20 set_io -nowarn FLASH_SSB 16
21 set_io -nowarn FLASH_IO0 14
22 set_io -nowarn FLASH_IO1 17
23 set_io -nowarn FLASH_IO2 12
24 set_io -nowarn FLASH_IO3 13
25
26 # PMOD 1A
27 set_io -nowarn P1A1 4
28 set_io -nowarn P1A2 2
29 set_io -nowarn P1A3 47
30 set_io -nowarn P1A4 45
31 set_io -nowarn P1A7 3
32 set_io -nowarn P1A8 48
33 set_io -nowarn P1A9 46
34 set_io -nowarn P1A10 44
35
36 # PMOD 1B
37 set_io -nowarn P1B1 43
38 set_io -nowarn P1B2 38
39 set_io -nowarn P1B3 34
40 set_io -nowarn P1B4 31
41 set_io -nowarn P1B7 42
42 set_io -nowarn P1B8 36
43 set_io -nowarn P1B9 32
44 set_io -nowarn P1B10 28
```

- Lastly, there will be an “apio.ini” file which contains configuration info for the APPIO toolsuite. It specifies the board we are using, the iCEBreaker, and specifies the top module in our design. If you are using multiple verilog files, you must specify which one is the “top-module” that connects to FPGA pins.



```
1 [env]
2 board = iCEBreaker
3 top-module = blink_test
4
5
```

- Next run “apio build”. This is going to run a series of 3 commands. First it will run the yosys tool which does synthesis, outputting a synthesized netlist to **hardware.json**. Next it runs nextpnr while does placement and routing, and outputs an implemented floor plan as **hardware.asc**. Lastly, it runs icepack which does bitstream generation creating **hardware.bin**.



```
PS C:\Users\gjd11> apio drivers --ftdi-enable
Launch drivers configuration tool
```

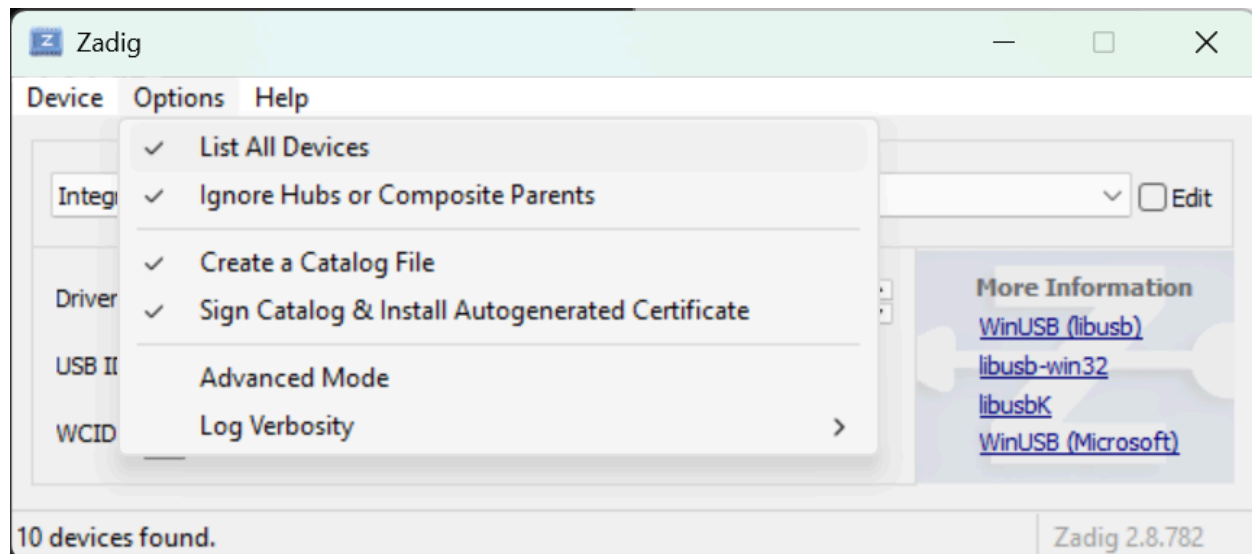
## FTDI driver installation: Usage instructions

1. Connect the FTDI FPGA board
2. Select (Interface 0)
3. Replace driver by "libusbK"
4. Reconnect the board
5. Check `apio system --lsftdi`

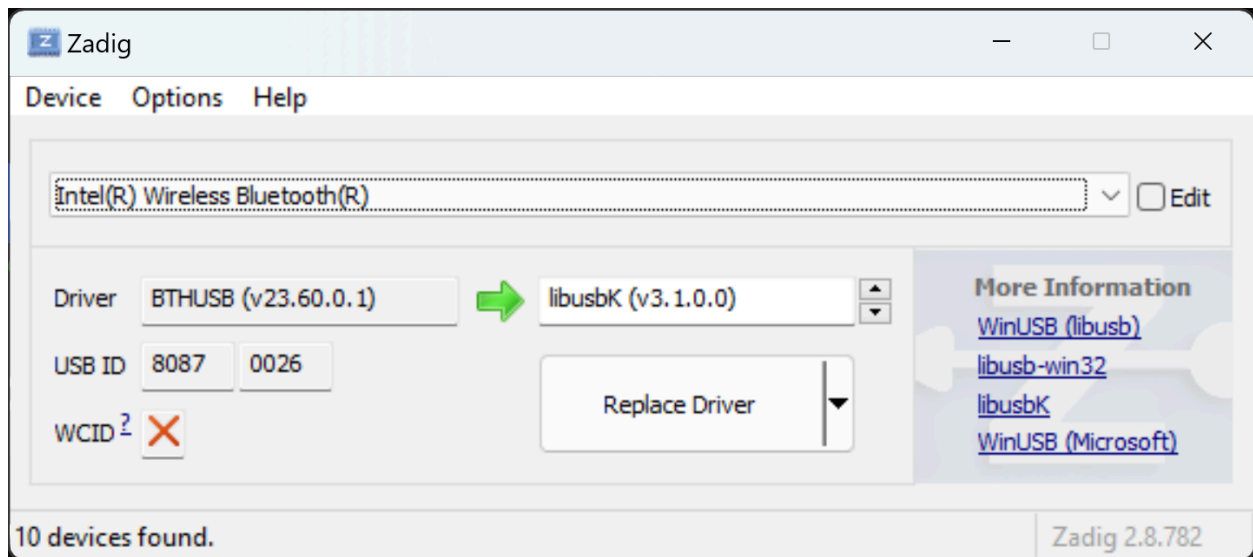
FTDI drivers configuration finished

The program Zadig will launch.

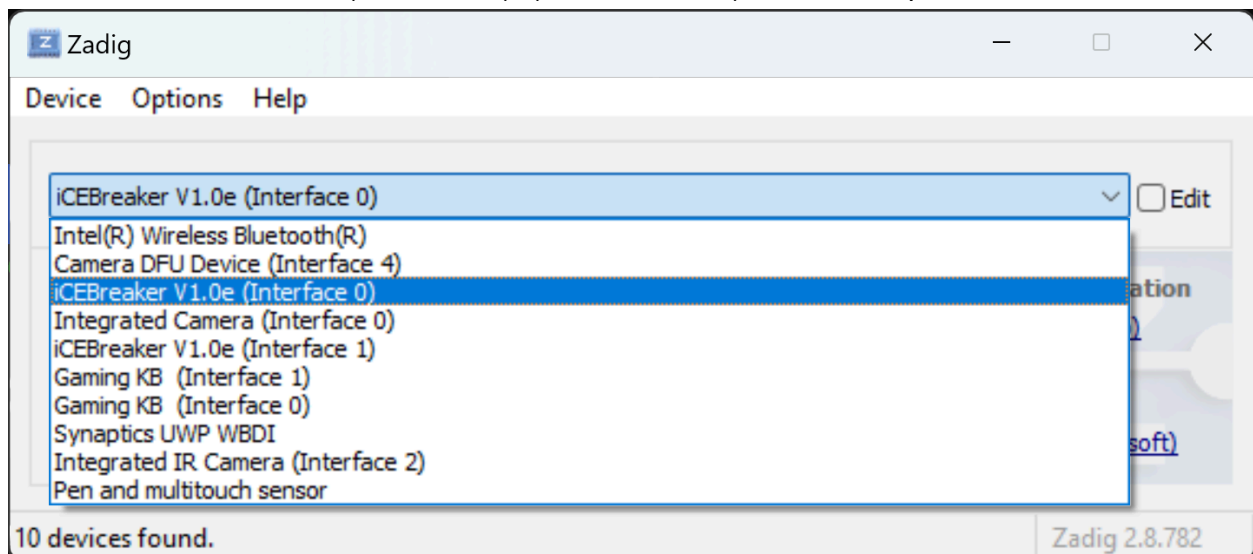
Click Options > List All Devices and make sure it is enabled.



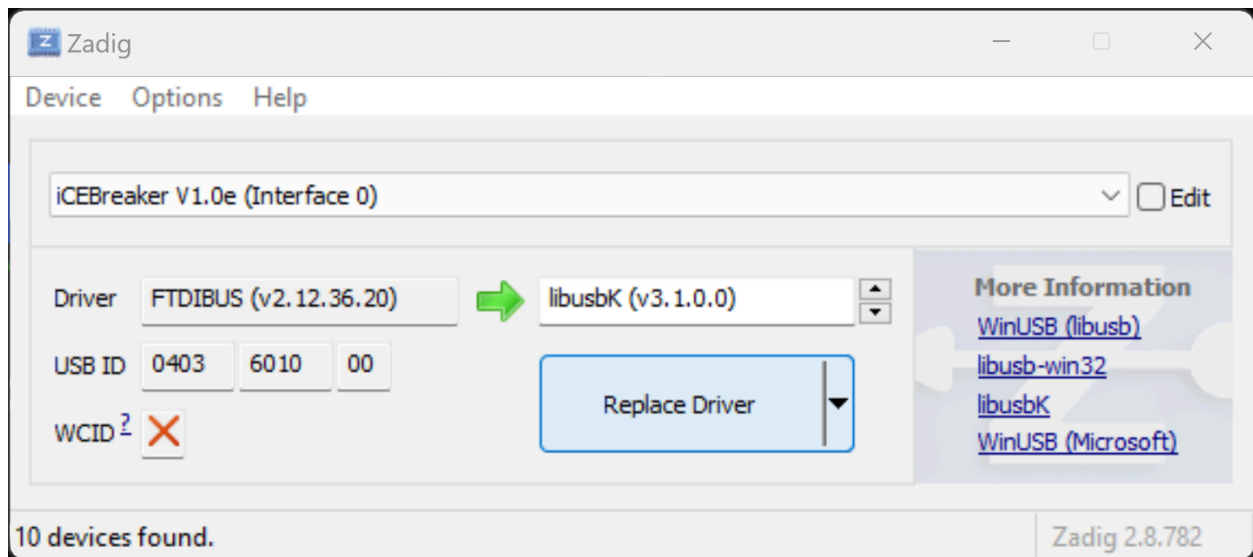




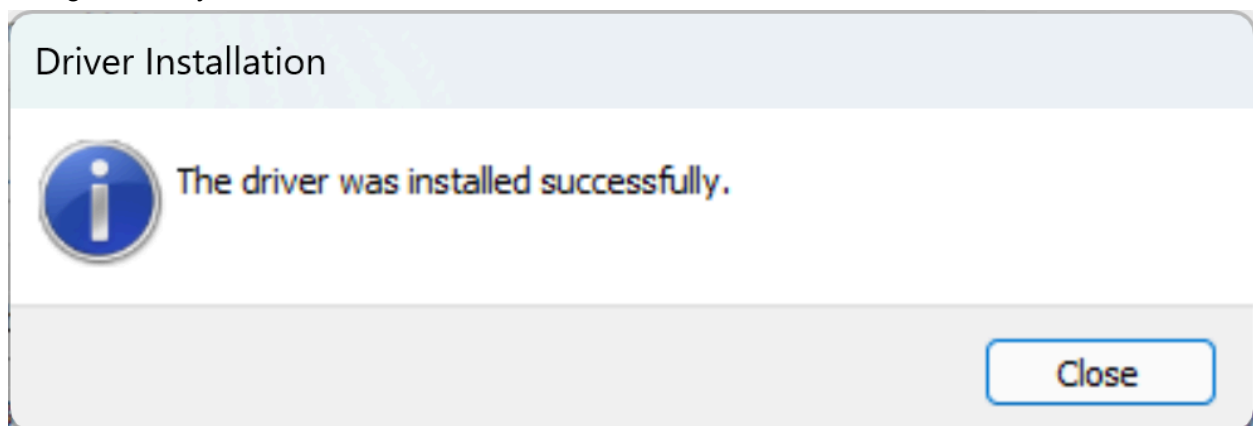
Select "iCEBreaker V1.0e (**Interface 0**)" (**not** Interface 1) from the drop down menu



Make sure the driver to the right of the green arrow says "libusbK". If it does not use the arrow to select this driver. Then click "Replace Driver":



If all goes well you should see this:



- Now open a terminal at the directory containing the **hardware.bin** bitstream file that was previously generated with apio build command. From this directory run the command “apio upload”

```
Windows PowerShell
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\gjd11\Documents\4Y\FPGA\lab\iCEBreaker\Blinky> ls

Directory: C:\Users\gjd11\Documents\4Y\FPGA\lab\iCEBreaker\Blinky

Mode                LastWriteTime         Length Name
----                -
-a-----         3/27/2025   1:29 PM           2474 .sconsign.dblite
-a-----         3/27/2025   1:29 PM              0 abc.history
-a-----         3/27/2025  11:22 AM           50 apio.ini
-a-----         3/27/2025  11:22 AM          678 Blinky.v
-a-----         3/27/2025   1:29 PM       725336 hardware.asc
-a-----         3/27/2025   1:29 PM       104090 hardware.bin
-a-----         3/27/2025   1:29 PM       401028 hardware.json
-a-----         3/27/2025  11:22 AM        1498 icebreaker.pcf
-a-----         3/27/2025  11:22 AM          29 info

PS C:\Users\gjd11\Documents\4Y\FPGA\lab\iCEBreaker\Blinky> apio upload
[Fri Apr  4 16:17:26 2025] Processing iCEBreaker

iceprog -d i:0x0403:0x6010:0 hardware.bin
init..
cdone: high
reset..
cdone: low
flash ID: 0xEF 0x40 0x18 0x00
file size: 104090
erase 64kB sector at 0x000000..
erase 64kB sector at 0x010000..
done.
VERIFY OK
cdone: high
Bye.
===== [SUCCESS] Took 15.79 seconds =====
PS C:\Users\gjd11\Documents\4Y\FPGA\lab\iCEBreaker\Blinky>
```

- Look at your iCEBreaker board. If everything went well, there should be a green LED labeled “G” blinking near the USB C port!

If you are having trouble with getting the open source programmer or the drivers to work properly, you can use the Diamond Programmer from Lattice.

However, if the “apio upload” command worked for you and the LED blinked, **feel free to skip this next section** and move onto the simulation section.

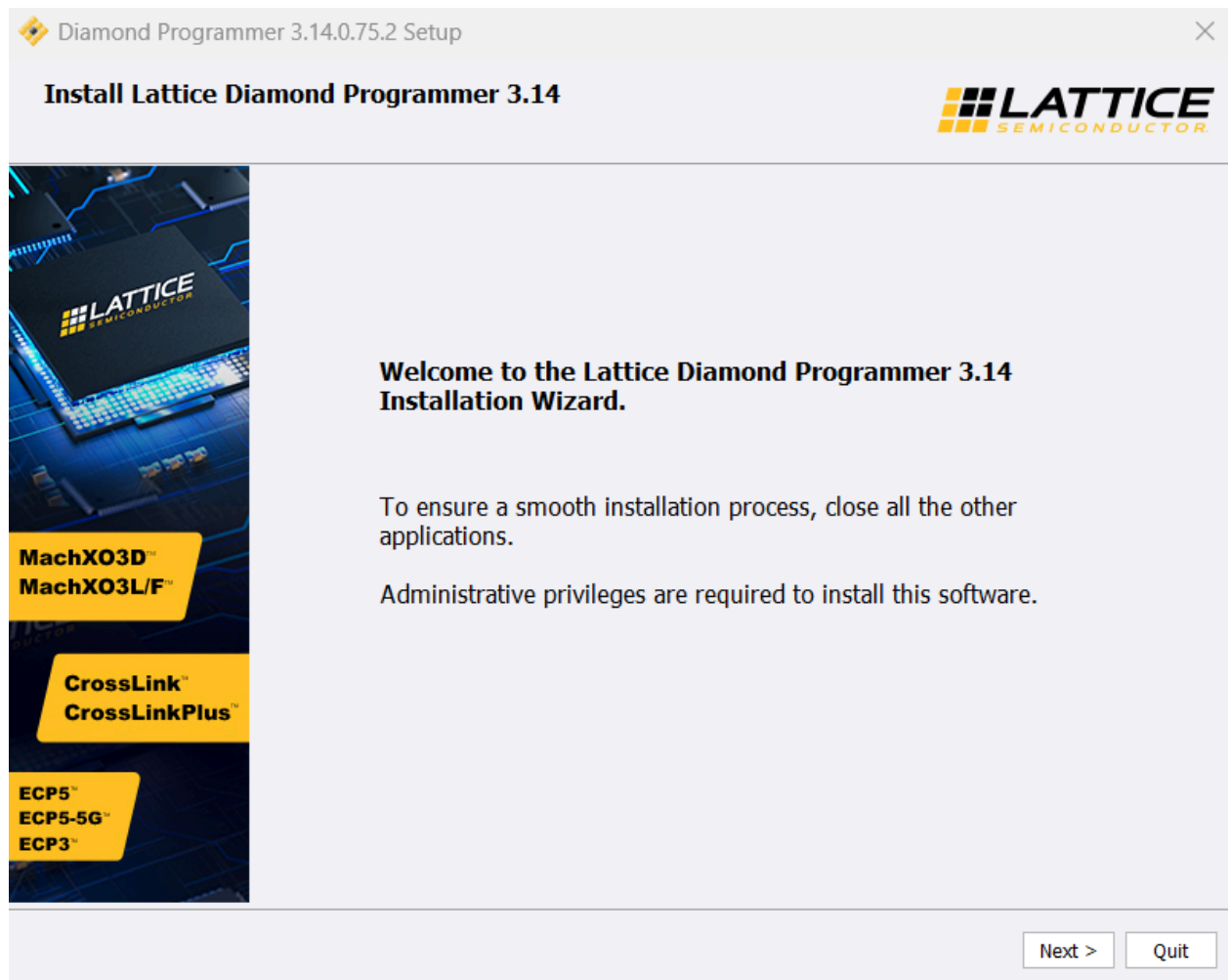
## Lattice Diamond Programmer:

- While this programmer is not open source, it is free. You will need an account from Lattice to download it, but there is no license or payment required.
- Go to this link to get the [Lattice Diamond programmer](#)
- Next, make an account for the website in the provided link. After making an account, you are able to download the following:

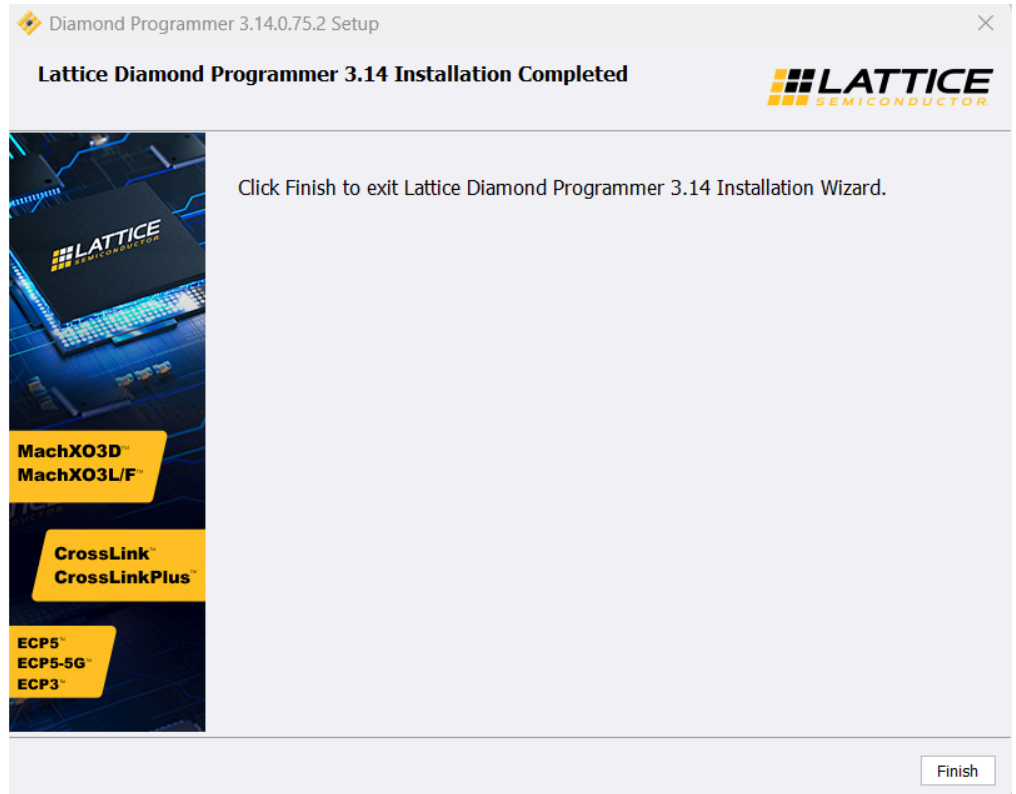
|                          |   |      |            |     |         |
|--------------------------|---|------|------------|-----|---------|
| <input type="checkbox"/> | <a href="#">Programmer Standalone 3.14 64-bit for Linux</a>   | 3.14 | 10/16/2024 | ZIP | 55.7 MB |
| <input type="checkbox"/> | <a href="#">Programmer Standalone 3.14 64-bit for Windows</a> | 3.14 | 10/16/2024 | ZIP | 53.3 MB |

For this tutorial, we downloaded the “Programmer Standalone 3.14 64-bit for Windows”

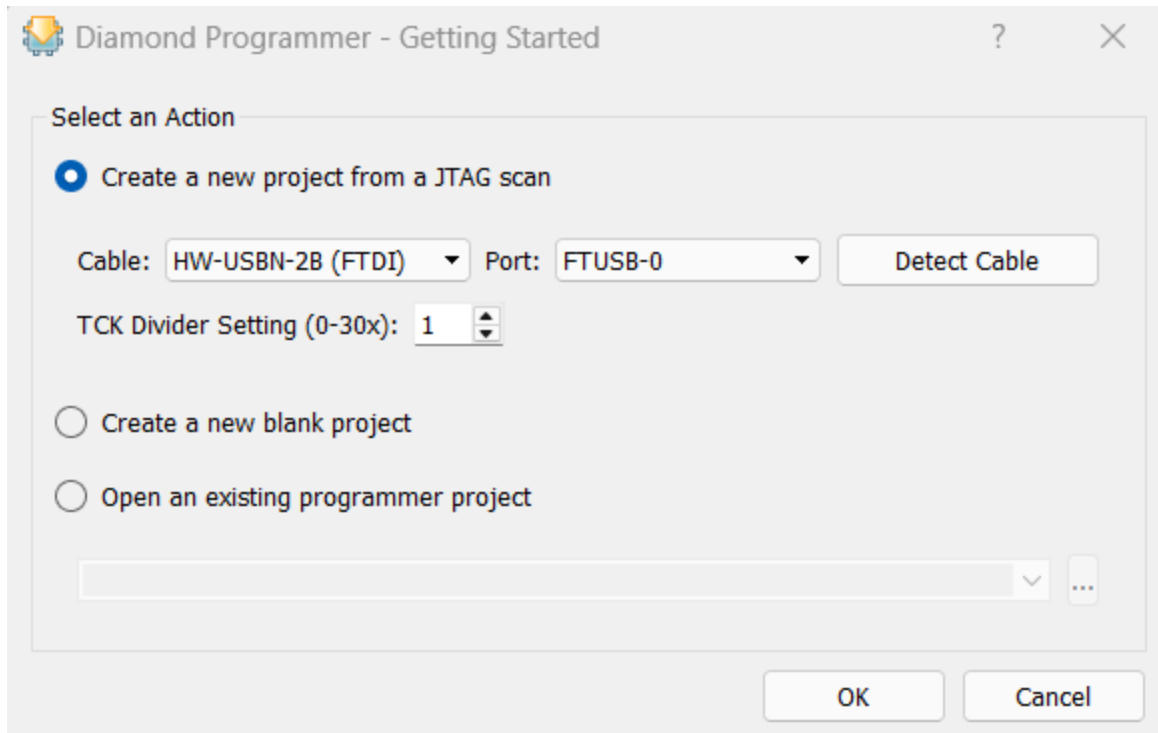
- After downloading the zip file, open the file and click on the application. Say yes to allowing it to run, and the following window comes up:



- Click next, chose the desired directory (in this tutorial we stuck to the default), then click next, click next again selecting "programmer", then select "I accept the license.", then hit next, next, then "install."



- After clicking finish, launch the “Diamond Programmer” from your start menu.
- Next, plug in the board and click “Detect Cable”. An example of the changed cable settings is shown below. Then click “OK”, if you get a windows network access prompt say “allow”.



- Set up your device family and device as shown below. This is the FPGA on the iCEBreaker board. Device Family: iCE40 UltraPlus. Device: iCE40UP5K

| Enable | Status                              | Device Family   | Device    | Operation   | File Name | File Date/Time | Checksum | USERCODE |
|--------|-------------------------------------|-----------------|-----------|-------------|-----------|----------------|----------|----------|
| 1      | <input checked="" type="checkbox"/> | iCE40 UltraPlus | iCE40UP5K | ... Program |           |                |          |          |

- Next click on “Operation” to the right of “Device”. Fill in the settings to match the screenshot. For the programming file, make sure to locate the “hardware.bin” file you generated earlier with “apio build”. Set up the SPI flash as indicated in the screenshot. Vendor: WinBond. Device: W25Q128JV. Package: 8-pin SOIC. This is the SPI flash memory chip on the iCEBreaker board. Then, click “OK.”

iCE40 UltraPlus - iCE40UP5K - Device Properties

General Device Information

Device Operation

Access mode: SPI Flash Programming

Operation: SPI Flash Erase,Program,Verify

Programming Options

Programming file: /itu/home/floranicholas/iCEBreaker/Blinky/hardware.bin ... 0xDD41

SPI Flash Options

Family: SPI Serial Flash

Vendor: WinBond

Device: W25Q128JV

Package: 8-pin SOIC

SPI Programming

Data file size (Bytes): 104090 Load from File

Start address (Hex): 0x00000000

End address (Hex): 0x00010000

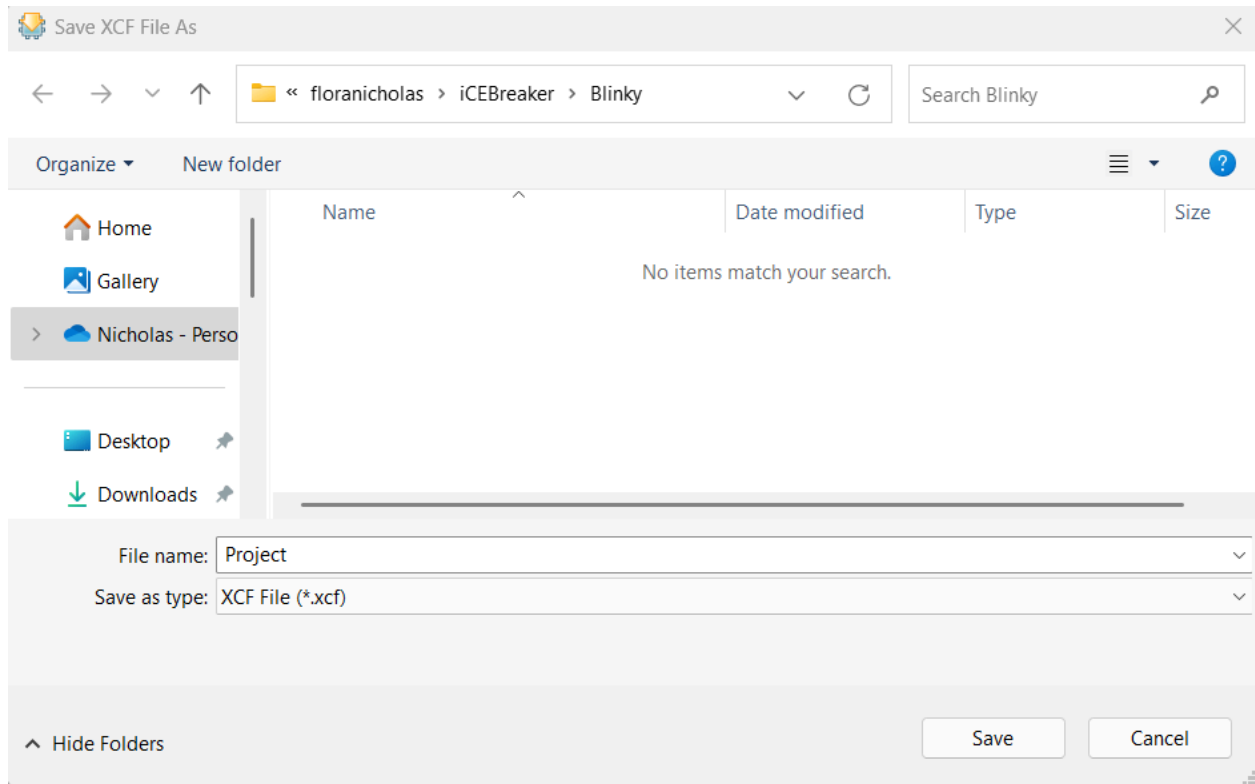
☐ Turn off addresses auto updating


☐ Erase SPI part on programming error

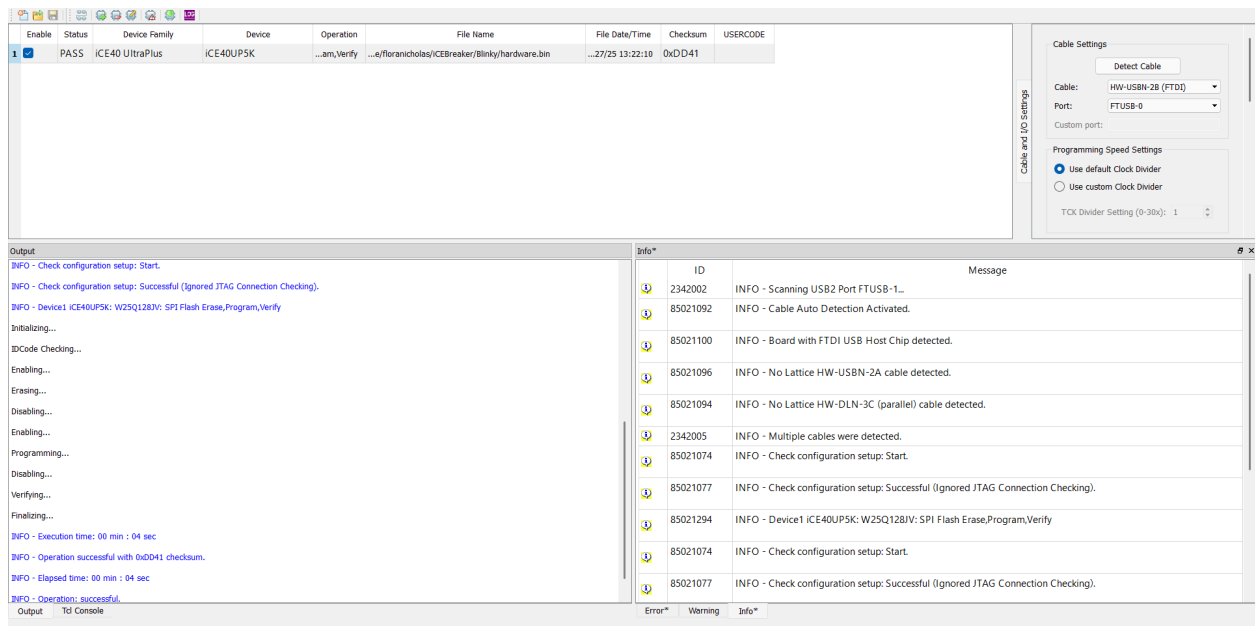
☐ Secure SPI flash golden pattern sectors

OK Cancel

- Go to file save and you can save all these configurations so don't have to re-input them every time you want to program the board.

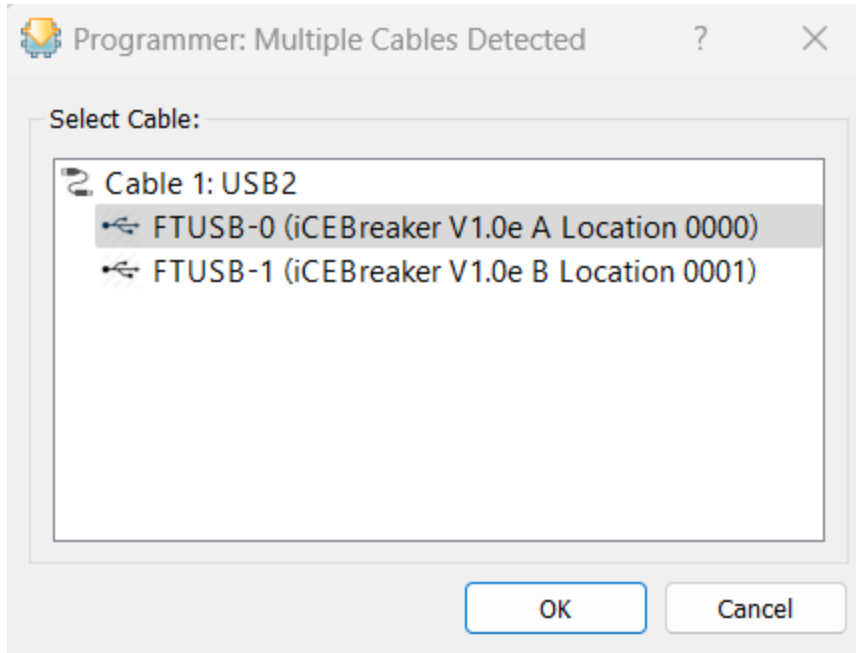


- Next, click program, which is the middle green simple symbol: , and the program should quickly and successfully pass, as shown below.



- If it doesn't, and shows an error message, switch click "Detect Cable" and switch the port as shown below. For this tutorial, we were on "FTUSB-0".





- Look at your iCEBreaker board. If everything went well, there should be a green LED labeled “G” blinking near the USB C port!

## Simulator:

Navigate to the 7\_segment folder and run the command “apio sim”

```
PS C:\Users\gjd11\Documents\4Y\FPGA\lab\2025-fpga-design-projects-lattice-fpga\7_segment> apio sim
iverilog -o SS_tb.out -D VCD_OUTPUT=SS_tb -D INTERACTIVE_SIM -D NO_ICE40_DEFAULT_ASSIGNMENTS "C:\Users\gjd11\apio\packages\tools-oss-cad-suite\share\yosys\ice40\cells_sim.v" SS_top.v SS_tb.v
vvp SS_tb.out
VCD info: dumpfile SS_tb.vcd opened for output.
7-Segment Display Testbench
DONE
SS_tb.v:44: $finish called at 2000000000000 (1ps)
gtkwave --rcvar "splash disable on" --rcvar "do initial zoom fit 1" SS_tb.vcd SS_tb.gtkw
```

This will run the verilog testbench “SS\_tb.v” with the Icarus Verilog (iverilog) simulator. It may take a few minutes, depending on how fast your computer is. It is going to log a few signals in the design to a SS\_tb.vcd value change dump (VCD) file.

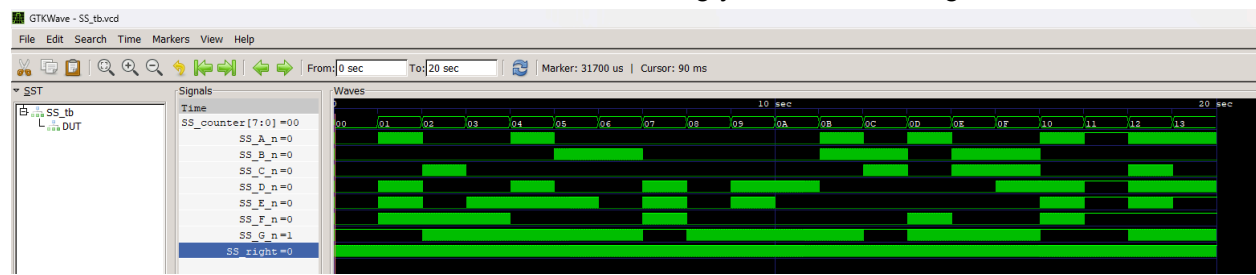
See this part of the code below for what signals are being logged. Alternatively, you can do “\$dumpvars(0, SS\_tb);” to log everything.

```

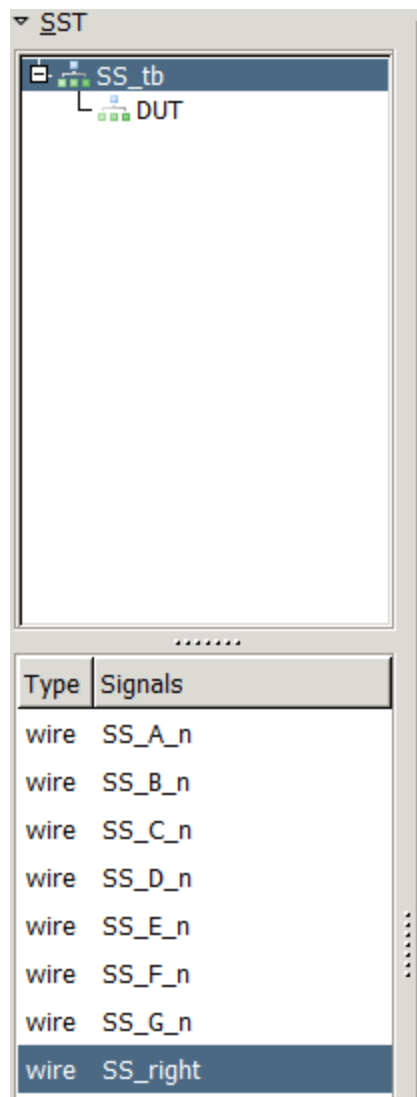
27     initial begin
28         $dumpfile("SS_tb.vcd");
29         // outputs
30         $dumpvars(0, SS_tb.SS_A_n);
31         $dumpvars(0, SS_tb.SS_B_n);
32         $dumpvars(0, SS_tb.SS_C_n);
33         $dumpvars(0, SS_tb.SS_D_n);
34         $dumpvars(0, SS_tb.SS_E_n);
35         $dumpvars(0, SS_tb.SS_F_n);
36         $dumpvars(0, SS_tb.SS_G_n);
37         $dumpvars(0, SS_tb.SS_right);
38         // DUT internals
39         $dumpvars(0, SS_tb.DUT.SS_counter);

```

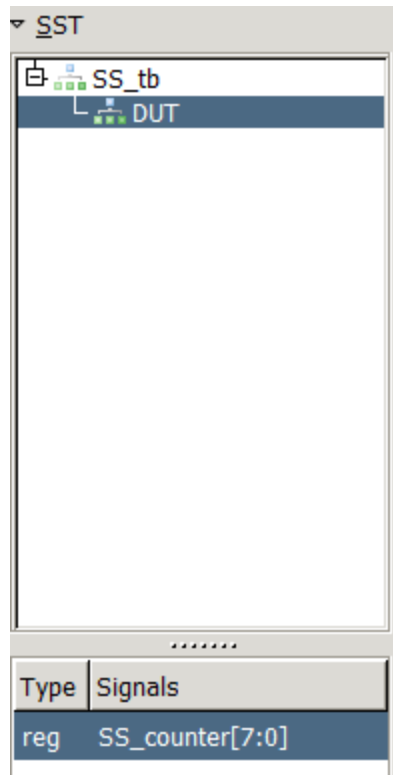
After that, GTKWave wave viewer will launch, allowing you to view the signals in the testbench.



Navigate on the left panel to add signals to the wave by double clicking on them:

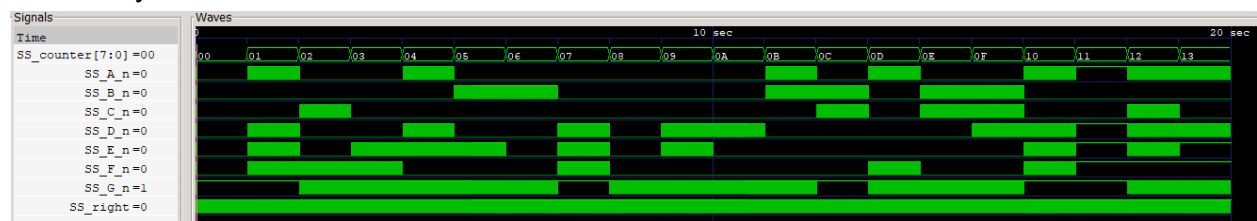


You can recursively descend the hierarchy and view the signals of the modules instantiated within each module.



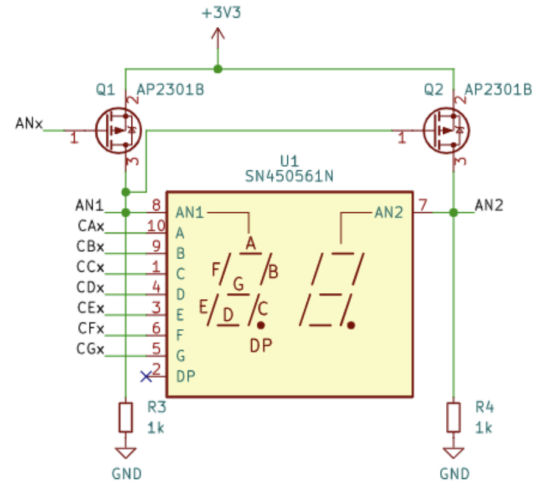
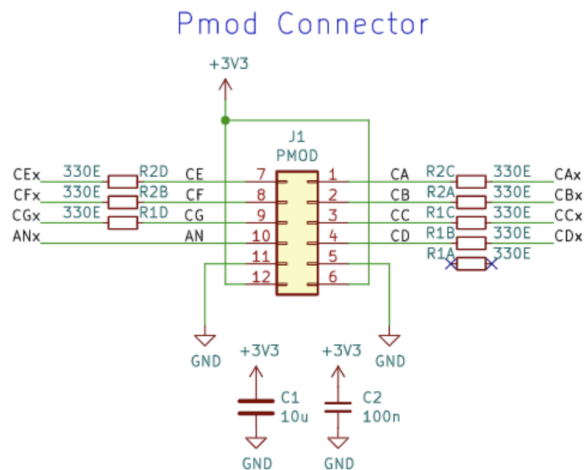
The signals that will be available will depend on what you chose to log in the SS\_tb.v file above.

The supplied testbench runs for 20 seconds, it shows that the SS\_counter register counts up once every second.



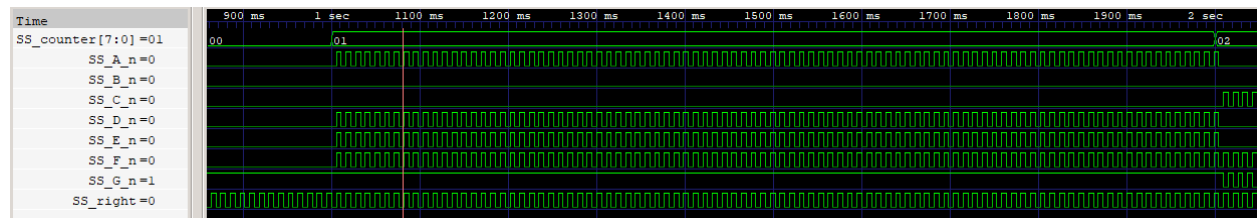
To understand how it works we must look at the 7 segment display.

Here is a link to the [7 segment display schematic](#) if you want the pdf. And here is a screenshot of it for convenience:

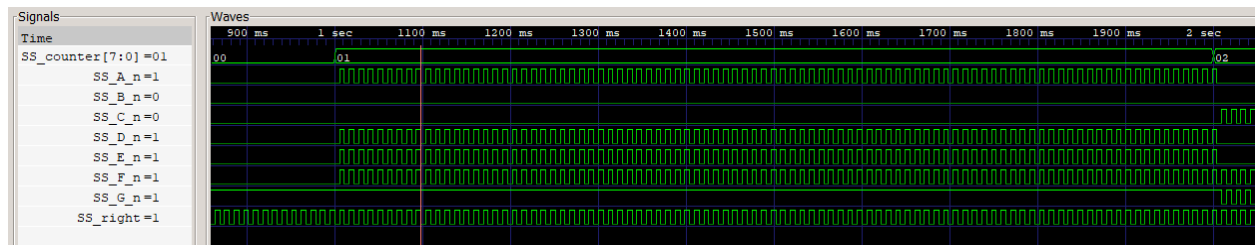


The seven output signals of our design “SS\_A\_n” - “SS\_G\_n” correspond to the 7 wires A-G in each digit of the seven segment display. The “\_n” indicates that the signal is active low, meaning 0 turns the segment on and 1 turns the segment off. The SS\_right output signal determines whether we are driving the right digit (if 1) or the left digit (if 0).

Let’s look at when the seven segment display outputs “01”, in other words it has been running for 1 second. Notice that SS\_right oscillates on and off fast enough that the human eye cannot detect the flicker of switching between the two digits. When SS\_right=0, the left digit is 0, so the only segment turned off is G for the middle to make the 0 shape. Remember the segment signals are active low.



Now look at when the SS\_right=1. When the right digit is displayed it outputs a 1 so the only 2 segments that are on are B and C. This corresponds with the vertical right side segments that make up the shape of 1.



Feel free to explore the other outputs across the first 20 seconds and understand how the 7-segment display works so that it appears as one 2-digit hex number to the human eye, while actually oscillating back and forth between the 2 digits.

## Questions:

- Why do we use the letters a-f for hex values? Why can't we count in decimals?
- What is the frequency of the oscillator in Blinky.v? What would happen if the oscillator was faster?
- What is the point of the virtual simulator? What can it give you that the board can't?
- When is it "ok" to define logic asynchronously, outside a process, and when is it not? (When a logic needs a clock, when is it ok to define it without a clock?) (Think also about combinational vs sequential logic.)