Task 1

Step 1

Commands used in the datasheet,

1) make clean
2) make build
3) sudo make flash

The blinking LED project conducted successfully (in the blink_led file)

The led_blue , led_red, led_green control the LED

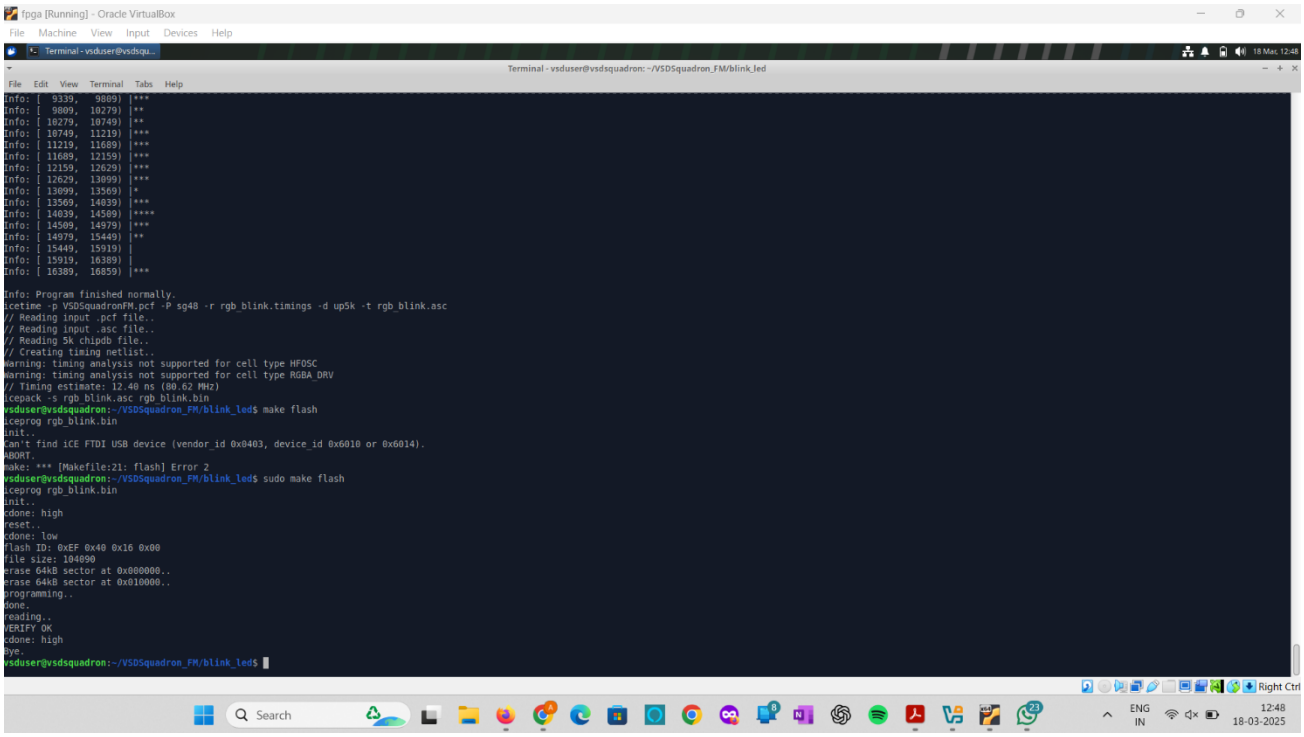
The testwire is the output

The hw_clk is the input for the signal given according to SB_HFOSC

The test wire takes input from hw_clk

frequency counter logic is driven by the internal oscillator.

And used in the hardware oscillator. Example: if it exceeds 12mHz it gets terminated or goes to initial stage.



Verilog code functionality can be summarized based on its purpose and the hardware design it models. Typically, Verilog code is written to describe the behaviour or structure of digital systems, such as FPGAs or ASICs. For combinational circuits, the code describes how outputs depend on the current inputs. For sequential circuits, the functionality is based on clock edges and how the system transitions between states. In this project we had various .v files, they were used for different

functions (red led, blue led, green led). The make file was the same for all these, there was difference only in the .v files.

The .Verilog file must be implemented in the board to get a blinking LED.

The RGB LED driver is a crucial component in controlling the illumination of an RGB LED, which consists of three separate LEDs (red, green, and blue) encapsulated in a single package. The driver enables each LED to be controlled individually, allowing for the creation of a wide spectrum of colours by adjusting the intensity of each colour channel.

Outputs from the internal logic and oscillator act as control signals for the RGB LED driver.

These outputs form the link between the internal logic of the circuit and the external RGB LED, driving its behaviour based on the input design.

Purpose of the Module:

The purpose of this Verilog module is to serve as a foundation for digital circuit design, showcasing key functionalities such as clock oscillation, internal logic, and driving an RGB LED. It simulates real-world hardware components while demonstrating how they interact within a system.

Description of Internal Logic and Oscillator:

1. Internal Logic:

- The module includes a D flip-flop, which is a fundamental sequential logic element. It captures the input data on the rising edge of the clock signal and stores it, producing the corresponding output.
- This internal logic models the behaviour of memory elements commonly used in digital circuits for data storage and state retention.

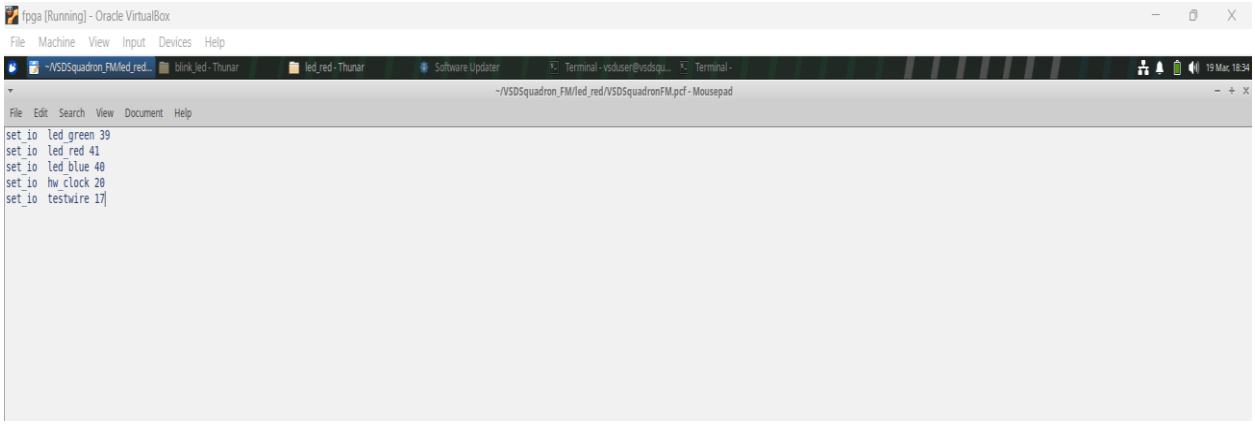
2. Oscillator:

- The clock oscillator is implemented using a counter, which divides the frequency of the input clock signal. This creates a slower clock signal to simulate the behaviour of an oscillator.
- Oscillators are essential in digital designs, providing timing signals for synchronization and driving sequential elements like flip-flops.

Functionality of the RGB LED Driver and Its Relationship to the Outputs:

- The RGB LED driver is responsible for controlling the individual red, green, and blue components of an RGB LED.
- In this module, the outputs from the internal logic (e.g., the flip-flop's output) can be connected to the RGB LED driver.
- `x` from the flip-flop could control one of the LED's colour channels.
- The oscillator's slower clock signal could toggle another channel, creating a blinking effect.
- This relationship demonstrates how internal logic and oscillators work together to manipulate and control external hardware, such as an RGB LED.

Step 2



The pcf file is created successfully

FNC	Pin Type	BANK	Differential Pair	Pin number
IOB_0a	PIO	2	-	48
IOB_2a	DPIO	2	TRUE_of_IOB_3b	47
IOB_3b_G6	DPIO/GBIN	2	COMP_of_IOB_2a	44
IOB_4a	DPIO	2	TRUE_of_IOB_5b	48
IOB_5b	DPIO	2	COMP_of_IOB_4a	45
IOB_6a	PIO	2	-	2
IOB_8a	DPIO	2	TRUE_of_IOB_9b	4
IOB_9b	DPIO	2	COMP_of_IOB_8a	3
IOB_10a	DPIO	1	TRUE_of_IOB_11b	-
IOB_11b_G5	DPIO/GBIN	1	COMP_of_IOB_10a	-
creset_b	CONFIG	1	-	8
IOB_12a_G4_CDONE	CONFIG/DPIO/GBIN	1	TRUE_of_IOB_13b	-
CDONE	CONFIG	1	-	7
IOB_13b	DPIO	1	COMP_of_IOB_12a	6
IOB_16a	PIO	1	-	9
IOB_18a	PIO	1	-	10
IOB_20a	PIO	1	-	11
IOB_22a	DPIO	1	TRUE_of_IOB_23b	12
IOB_23b	DPIO	1	COMP_of_IOB_22a	21
IOB_24a	DPIO	1	TRUE_of_IOB_25b	13
IOB_25b_G3	DPIO/GBIN	1	COMP_of_IOB_24a	20
IOB_29b	PIO	1	-	19
IOB_31b	PIO	1	-	18
IOB_32a_SPI_SO	DPIO/CONFIG_SPI	1	-	14
IOB_33b_SPI_SI	DPIO/CONFIG_SPI	1	-	17
IOB_34a_SPI_SCK	DPIO/CONFIG_SPI	1	-	15
IOB_35b_SPI_SS	DPIO/CONFIG_SPI	1	-	16
VCCPLL	VCCPLL	-	-	29
IOT_36b	DPIO/I3C	0	COMP_of_IOT_37a	25
IOT_37a	DPIO/I3C	0	TRUE_of_IOT_36b	23
IOT_38b	DPIO	0	COMP_of_IOT_39a	27
IOT_39a	DPIO	0	TRUE_of_IOT_38b	26
IOT_41a	PIO	0	-	28
IOT_42b	DPIO	0	COMP_of_IOT_43a	31
IOT_43a	DPIO	0	TRUE_of_IOT_42b	32
IOT_44b	DPIO	0	COMP_of_IOT_45a	34
IOT_45a_G1	DPIO/GBIN	0	TRUE_of_IOT_44b	37
IOT_46b_G0	DPIO/GBIN	0	-	35
IOT_47a	PIO	0	-	-
IOT_48b	DPIO	0	COMP_of_IOT_49a	36
IOT_49a	DPIO	0	TRUE_of_IOT_48b	43
IOT_50b	DPIO	0	COMP_of_IOT_51a	38
IOT_51a	DPIO	0	TRUE_of_IOT_50b	42
RGB2	LED	0	-	41
RGB1	LED	0	-	40
RGB0	LED	0	-	39
GND	GND	GND	-	Paddle
GND	GND	GND	-	Paddle
GND	GND	GND	-	Paddle
VCC	VCC	VCC	-	5
VCC	VCC	VCC	-	30
VCCIO_0	VCCIO	0	-	33
SPI_Vccio1	VCCIO	1	-	22
VCCIO_2	VCCIO	2	-	1
VPP_2V5	VPP	VPP	-	24

- Pin 41,40 and 39 are the LED according to the datasheet so they are basically the outputs…
- Pin 5,30 supply the VCC

The **paddle pin** in the context of **VCCPLL** often refers to the exposed metal pad or ground plane underneath the chip. It serves as a thermal and electrical connection point. In FPGA designs like those involving PLLs (Phase-Locked Loops), **VCCPLL** is the dedicated power supply for the PLL circuitry.

- Pin 29 is the VCCPLL also called CPU PLL voltage. The GND is therefore supported by the paddle pin. A **CONFIG pin** on an FPGA is typically used during the **configuration process**, which is how the FPGA loads its design data into internal logic blocks from external memory or sources. – pin 7 (CDONE)

Input Pins

- UART RX(Receiver)
- **Function in Verilog**: Declared as input, they allow your FPGA design to monitor and react to external stimuli.
- **Significance**: Input pins are essential for providing dynamic control or feeding real-time data into the FPGA logic.

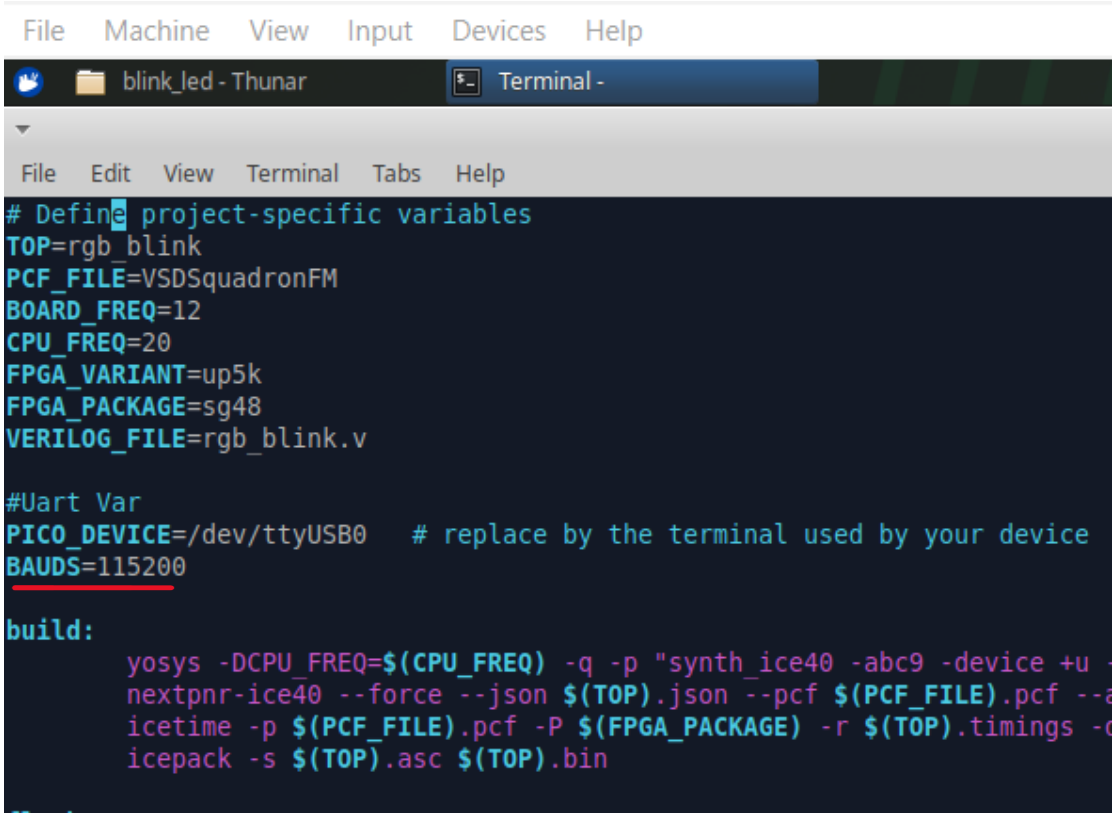
Output Pins

- UART TX (transmitter)
- **Function in Verilog**: Declared as output, they carry signals generated by your FPGA design to the connected hardware.
- **Significance**: Outputs provide feedback, drive external loads, or communicate with other systems.

UART (Universal Asynchronous Receiver-Transmitter) is a serial communication protocol used for data transmission between devices. Also, it controls BAUD_RATE.

Baud Rate: Number of symbols transmitted per second.

- **EXAMPLE: Baud Rate = 115200** means the UART will transmit **115200 symbols** per second.



Step 3

While executing this, I got a few issues,

1)

```
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make build
yosys -DCPU_FREQ=20 -q -p "synth_ice40 -abc9 -device u -dsp -top top -json top.json" top.v.
ERROR: Can't guess frontend for input file `top.v.' (missing -f option)!
make: *** [Makefile:15: build] Error 1
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$
```

Problem: there was a syntax error which I hadn't noticed,

Solution: the dot at the end was removed

2)

```
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make clean
rm -rf top.blif top.asc top.bin top.json top.timings
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make build
yosys -DCPU_FREQ=20 -q -p "synth_ice40 -abc9 -device u -dsp -top top -json top.json" top.v
top.v:11: ERROR: syntax error, unexpected TOK_INPUT, expecting ',' or '=' or ')'
make: *** [Makefile:15: build] Error 1
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$
```

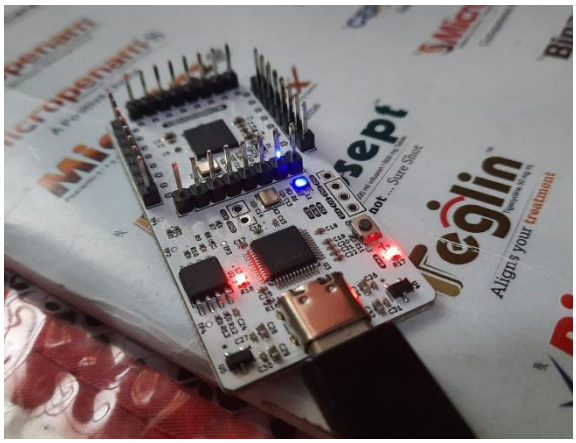
Solution: copy pasted the data from
https://github.com/thesourcerer8/VSDSquadron_FM/blob/main/led_blue/top.v

```
Terminal - vsduser@vsdsquadron: ~/VSDSquadron_FM/led_blue
File  Edit  View  Terminal  Tabs  Help
yosys -DCPU_FREQ=20 -q -p "synth_ice40 -abc9 -device u -dsp -top top -json top.json" top.v
nextpnr-ice40 --force --json top.json --pcf VSDSquadronFM.pcf --asc top.asc --freq 12 --up5k --package sg48 --opt-timing -q
Warning: unmatched constraint 'hw_clock' (on line 4)
ERROR: IO 'hw_clk' is unconstrained in PCF (override this error with --pcf-allow-unconstrained)
ERROR: Loading PCF failed.
1 warning, 2 errors
make: *** [Makefile:16: build] Error 255
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make clean
rm -rf top.blif top.asc top.bin top.json top.timings
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make clean
rm -rf top.blif top.asc top.bin top.json top.timings
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make build
yosys -DCPU_FREQ=20 -q -p "synth_ice40 -abc9 -device u -dsp -top top -json top.json" top.v
nextpnr-ice40 --force --json top.json --pcf VSDSquadronFM.pcf --asc top.asc --freq 12 --up5k --package sg48 --opt-timing -q
icetime -p VSDSquadronFM.pcf -P sg48 -r top.timings -d up5k -t top.asc
// Reading input .pcf file..
// Reading input .asc file..
// Reading 5k chipdb file..
// Creating timing netlist..
Warning: timing analysis not supported for cell type HFOSC
Warning: timing analysis not supported for cell type RGBA_DRV
// Timing estimate: 6.29 ns (159.10 MHz)
icepack -s top.asc top.bin
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make clean
rm -rf top.blif top.asc top.bin top.json top.timings
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ make build
yosys -DCPU_FREQ=20 -q -p "synth_ice40 -abc9 -device u -dsp -top top -json top.json" top.v
nextpnr-ice40 --force --json top.json --pcf VSDSquadronFM.pcf --asc top.asc --freq 12 --up5k --package sg48 --opt-timing -q
icetime -p VSDSquadronFM.pcf -P sg48 -r top.timings -d up5k -t top.asc
// Reading input .pcf file..
// Reading input .asc file..
// Reading 5k chipdb file..
// Creating timing netlist..
Warning: timing analysis not supported for cell type HFOSC
Warning: timing analysis not supported for cell type RGBA_DRV
// Timing estimate: 6.29 ns (159.10 MHz)
icepack -s top.asc top.bin
vsduser@vsdsquadron:~/VSDSquadron_FM/led_blue$ sudo make flash
[sudo] password for vsduser:
iceprog top.bin
init..
cdone: high
reset..
cdone: low
flash ID: 0xEF 0x40 0x16 0x00
file size: 104090
erase 64kB sector at 0x000000..
erase 64kB sector at 0x010000..
programming..
addr 0x00FE00  62%
```

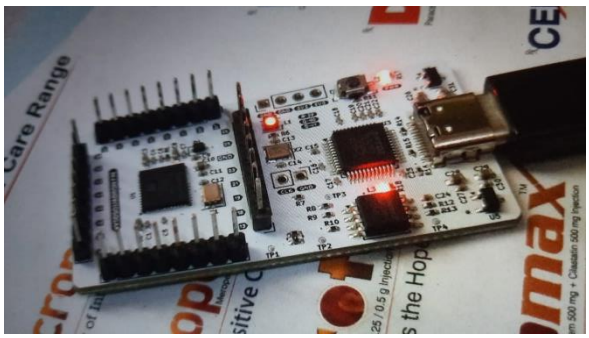
The final code..

Executed the same troubleshoot mechanism for led_red , led_green

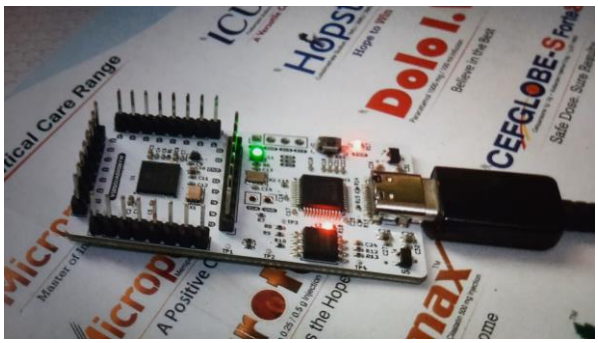
The results were



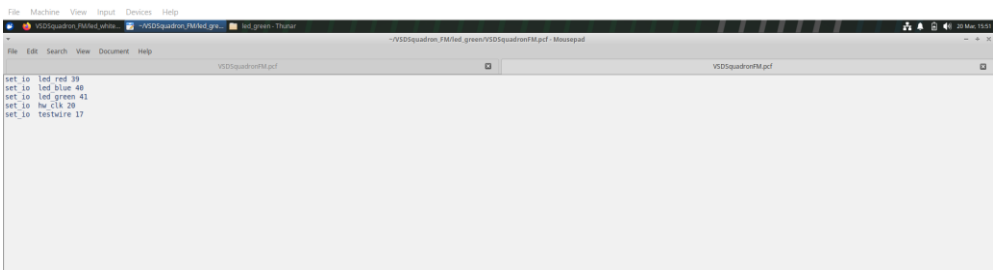
led_blue



led_red

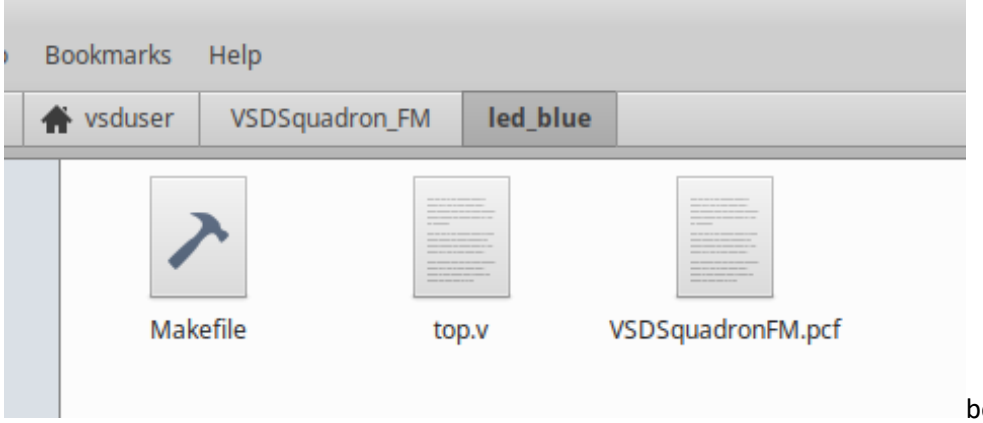


led_green

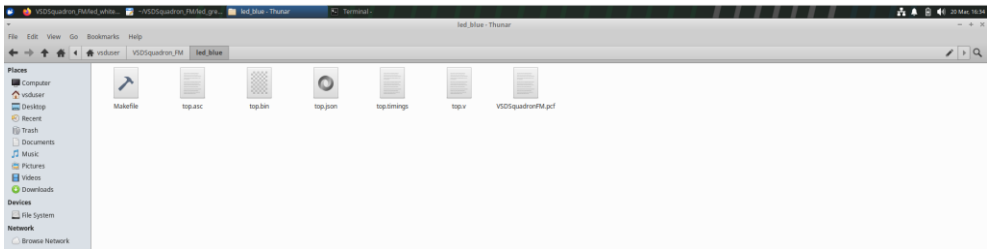


The .pcf file

An observation,



before flash



After flash

- Only the top.v , makefile , VSDSquadronFM.pcf were created by me.
- During make build the top.json, top.asc were created as mentioned in the make file.
- During flash, the top.bin was created.

What are pcf files?

PCF files (Physical Constraint Files) are text-based files used in FPGA design to map the logical signals defined in your Verilog (or VHDL) code to specific physical pins on the FPGA device. They play a crucial role in ensuring that the synthesized design correctly interfaces with the board's external hardware.

What is a makefile?

A Makefile is a text file that contains a set of directives used by the make build automation tool to compile and build projects. It defines how to derive target programs or files from source files, automating the process of building and managing dependencies in a project.

What is a .json file?

A .json file is a file that contains data in **JavaScript Object Notation (JSON)** format, which is a lightweight, human-readable format used to store and exchange structured data.

What is a .bin file?

A .bin file is a generic binary file that stores data in a non-human-readable, binary format instead of plain text

What is a .asc file?

Its meaning depends on the context of its use.

In FPGA design, .asc files are often used with Lattice iCE40 FPGAs. These files represent the physical configuration (bitstream) of the FPGA design in an ASCII format.

These .asc files are often converted to .bin (binary) format for programming the FPGA.

- **FPGA**:
  - An **FPGA (Field-Programmable Gate Array)** is a type of IC that can be programmed and reprogrammed after manufacturing to perform specific tasks. FPGAs are highly flexible and can be configured to suit a wide range of applications.
  - FPGAs can execute multiple operations simultaneously, which is great for high-speed and real-time applications.
  - used in telecommunications, aerospace, automotive, and industrial sectors for tasks like signal processing, encryption, and machine learning.

- **FPGA and Arduino boards:**
  - An FPGA gives you the power to design custom hardware, while an Arduino is more focused on providing a quick and easy platform for controlling devices and interacting with sensors.
  - FPGA boards and Arduino boards serve different purposes and are built on distinct technologies. Here's how they differ:

Aspect	FPGA Board	Arduino Board
Purpose	Used for custom digital logic design and parallel processing tasks.	Designed for simple microcontroller tasks and prototyping.
Flexibility	Highly customizable hardware; logic is defined by programming in HDLs like Verilog or VHDL.	Fixed hardware with pre-defined functionalities that can be programmed using C/C++.
Programming	Programmed using HDLs (Verilog/VHDL) and tools like Yosys.	Programmed using Arduino IDE with simple C/C++-based code.
Category of knowledge	Steeper, requires knowledge of digital logic design and HDLs.	Beginner-friendly; great for hobbyists and rapid prototyping.

----------Presented by Archana Bhat----------