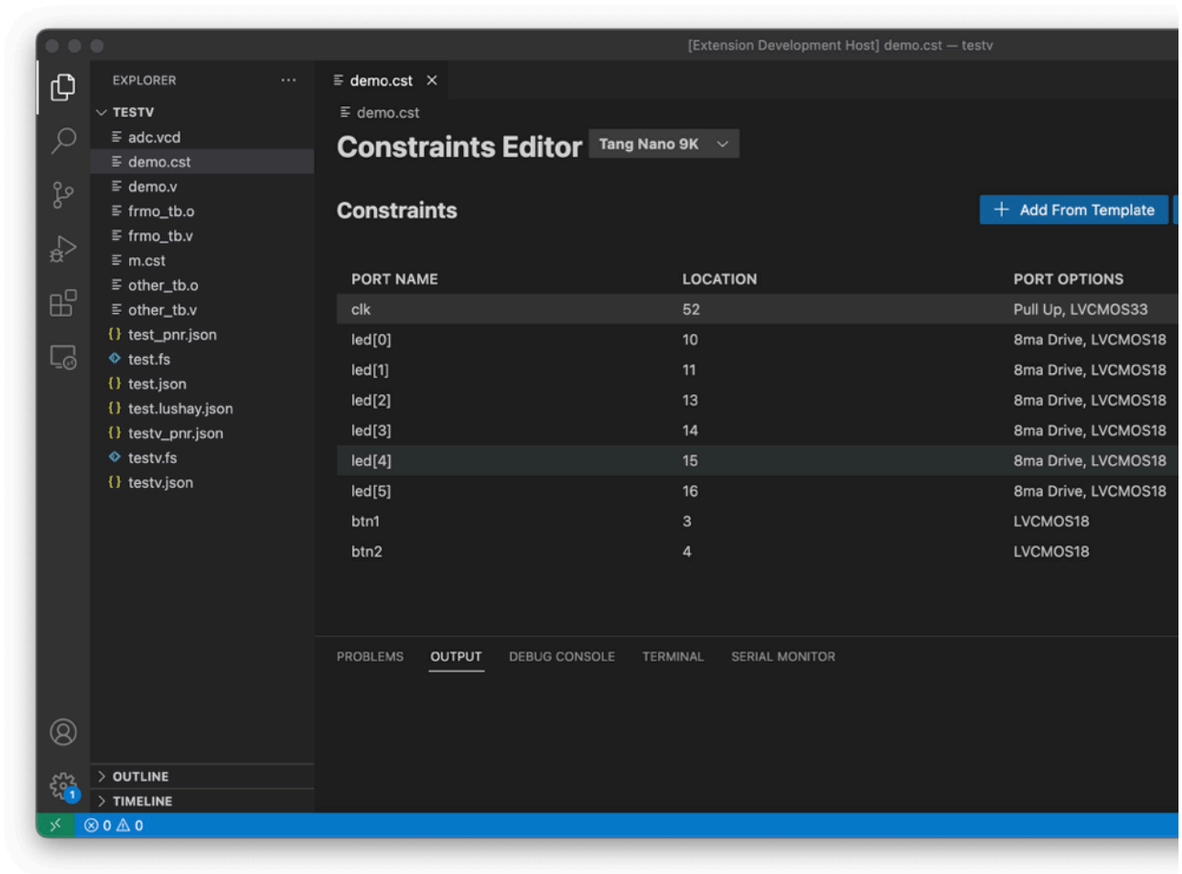
Tang Nano 9K: Getting Setup

Lushay Labs • Aug 13, 2022 • 17 min read



In this article we will be going over how to get started developing for the Tang Nano FPGA using a fully open source toolchain. We will start with an overview of FPGAs and the toolchain required to develop them, we will then go through setting up your development environment and finish off with a simple example so we can test and verify that everything is working.

What's an FPGA ?

An FPGA is a type of IC that you "program" with digital hardware circuits as apposed to microcontrollers or CPUs that you program with software. A CPU has a fixed internal architecture and only understands how to process its own machine codes / assembly language. Developers then

compile programming languages into these machine codes and the CPU will run them as it goes through the code.

With FPGAs the internal architecture is a blank canvas, you can choose to create a CPU type architecture and design your own assembly language if you like - there are even open-source architectures like RISC-V - but you can also decide to custom build your own application specific circuit (ASIC) developing your application in hardware instead of software.

A general purpose CPU is convenient in the sense that one design can be used to perform a lot of different tasks - its general purpose - but the downside is its not very efficient for any specific task. Take for example a single-core 2 ghz CPU, even though the clock pulses 2 billion times per second it doesn't mean you can actually do 2 billion things in your application. Each line of code will get converted to many assembly instructions, each of which will be handled serially - one at a time - and each can take multiple clock cycles. Modern CPUs try to pipeline or parallelize these multiple clock cycles but for the most part you have a lot of overhead. Add on top of that you usually have other things that are running like an operating system which also are scheduled and take clock cycles.

On the other hand with FPGAs you are programming your application in hardware, you can react to every clock pulse directly. There is no central processing unit, each part of the application can be handled in parallel, so not only do you actually get the full clock speed, but you get the effect of having as many parallel cores as needed.

This also means that with a microcontroller for example, each extra operation that you would like to add to your application will negatively effect all the current operations since they are all fighting for the same resources. In an FPGA - unless you specifically want it to block - adding new cores / applications to your FPGA has no effect on the existing cores as they are completely separate hardware.

So how does it work ? How can an FPGA dynamically reprogram its internal hardware ?

At its core an FPGA is made up of lookup tables, flip-flops and multiplexers. A lookup table can be thought of as a reconfigurable logic gate, its a prebuilt table where for each set of inputs there is a predefined output. For example a lookup table for an AND gate would look something like this:

Input 1	Input 2	Output
0	0	0
1	0	0
0	1	0
1	1	1

Usually the lookup tables (LUTs) in an FPGA have more inputs and outputs so they are a bit bigger, but by being able to reprogram these tables you can change the AND gate into an OR gate as follows:

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1

So as you can see in this example by just changing two of the outputs you "changed" the internal hardware from an AND gate to an OR gate.

Combine these reprogrammable gates with flip-flops which allow for storing data and multiplexers which allow for routing connections and you have a method of dynamically "programming" the internal electrical circuit.

This is a bit of a simplified view, usually FPGAs also come with built in memory / ram, DSPs or even dedicated math functions but at a high level I think the parts that LUTs, flip-flops and multiplexers are the core.

Tang Nano 9K

In this series we will be focusing on the Tang Nano 9K FPGA development board (available in our store) which includes an FPGA from Gowin's LittleBee family the GW1NR-9. This board IMHO currently provides the best bang for your buck both in terms of the FPGA itself number of LUTs / Flip-flops / ram and also the development board exposes a good number of IOs and has on-board LEDs, buttons and a built-in UART debugger and other peripherals. The open-source toolchain support is also very good supporting all the features we will be using.

FPGA Family	Gowin LittleBee	MachXO2	Lattice ECP5	Lattice Nexus	Lattice iCE40	Intel Cyclone V
Project	Apicula	Facade/Trellis	Trellis	prjoxide	Icestorm	Mistral
Logic Slice	✓	✓	✓	✓	✓	🚧 (no extended-LUT7)
├ LUTRAM	🚧	❌	✓	✓	❌ (N/A)	🚧 (no MLAB init)
└ ALU	✓	❌	✓	✓	✓	🚧 (no share chain)
IO buffer	✓	✓	✓	✓	✓	🚧
├ IO standards	✓	⚠	✓	🚧	✓	❌
└ DDR/SERDES	🚧	❌	✓	🚧	✓	❌
Block RAM	❌	❌	✓	✓	✓	🚧 (no TDP, no 8x20 mode)
DSP	❌	❌	✓	✓	✓ (up5k only)	❌
PLL	❌	❌	✓	⚠	✓	❌
Routing	✓	✓	✓	✓	✓	⚠
├ Local	✓	✓	✓	✓	✓	⚠
├ Global	✓	🚧	✓	✓	✓	⚠
└ Long wires	✓	❌ (N/A)	❌ (N/A)	❌ (N/A)	❌ (N/A)	❌ (N/A)
Timing	✓	❌	✓	✓	✓	🚧 (nominal delays only)
├ Logic	✓	❌	✓	✓	✓	🚧 (nominal delays only)
├ Routing	✓	❌	✓	✓	✓	🚧 (nominal delays only)
└ IO buffer	✓	❌	🚧	🚧	❌	❌

Open source toolchain Supported Features by FPGA family

For all these reasons, and because of its relatively low price the tang nano 9K has been chosen for this series.

The Programming Process

The process of programming your own core onto an FPGA goes through 3 stages:

1. Synthesis
2. Place and Route
3. Bitstream Generation

Synthesis

The first step is to take a design written in an HDL (Hardware description language) like Verilog and convert it into primitives which make up the FPGA.

In verilog you write your program in terms of modules with registers / connections and logic gates you also have abstractions like conditional statements or math operations. The job of the synthesis tool is to take all this and convert it to primitives like LUTs and flip-flops.

Place and Route

Once you have a list of connections and FPGA primitives from the synthesis stage the next step - like the name suggests - is to map each of the primitives to their physical counterpart. Like to map each of the generated look up tables to one of the physical 8640 internal luts.

This step must take into account all the routing requirements when selecting where to place each component, the output of this stage is similar to the output of the synthesis stage, except each of the components are physically mapped.

Bitstream Generation

The third and final step is generating the bits required so that the FPGA itself understands the layout the the place and route stage generated. Each FPGA manufacturer has their own internal format which open-source toolchains need to reverse engineer in-order to understand the exact format required to program the FPGA.

After running these three stages you will have a file which can be programmed onto the FPGA which will reconfigure the internal hardware to match your design

The Open-Source Toolchain

The open source toolchain is a collection of open-source tools for each of the steps mentioned in the previous section. We have Yosys for synthesis, NextPnR for placing and routing and Apicula which reverse-engineered the Gowin architecture and provides the bitstream generation tools. We will also be using openFPGALoader for programming the final bitstream onto the FPGA.

We recommend setting up the toolchain with OSS-CAD-Suite and our VS code extension. OSS-CAD-Suite is a project maintained by the yosys team which provides pre-built binaries for MacOSX, Windows and Linux. Our extensions is a wrapper around these pre-built binaries allowing you to visually configure and run the OS toolchain.

If you would prefer to install and use the toolchain manually we have the instructions for that here.

The first step to get this setup is to download and install Visual Studio Code:

VSCode

This is a popular lightweight cross-platform editor with many plugins some of which specifically for FPGA development. The editor itself can be downloaded from here.

Once installed, you can open it up and go to the Extensions tab to install plugins. To install our plugin you can search for "Lushay Code" or by going to the following link.

Besides for that we also recommend these two plugins, although not mandatory we find these useful.

1. Verilog-HDL/SystemVerilog - provides syntax highlighting support for Verilog
2. WaveTrace - provides built in waveform viewing which can be used for debugging.

With VsCode and the 3 extensions installed the next thing we need is OSS-Cad-Suite, you can find the latest version download from here.

> ⚠️ There are some synthesis issues in the latest version of OSS-CAD-Suite while they are going through a redesign of the OSS toolchain. It is recommended to download the 2023-02-10 version of OSS-CAD-Suite for best results: https://github.com/YosysHQ/oss-cad-suite-build/releases/tag/2023-02-10

On the release page there are several different versions depending on your OS and CPU architecture. For OS, linux and windows are just called windows and linux, and for MacOSX you need the darwin version. Next you have the architecture, usually either arm or x64, arm is for apple silicon or an arm-based linux, x64 is for a standard intel / amd cpus.

Download the file for your OS and then extract it somewhere you should now have a folder called `oss-cad-suite` on your computer. Place this folder wherever you want to keep the toolchain. It is worth noting on windows you don't download a zip but rather a self-extracting executable, but it is the same, you just double click it to extract it as a folder called `oss-cad-suite`.

> ⚠️ There is an issue when the path or folder to oss-cad-suite contains spaces, to be safe you can for example put it at the root of your drive and name it "oss-cad-suite" without spaces

With the folder ready open up VSCode and click on the button on the bottom right hand side called "FPGA Toolchain":

You should receive a popup that says the OSS-CAD-Suite path is not setup, click on the "Setup Now" button and a file browser will appear, you need to select the OSS-CAD-Suite folder we just extracted you should select the actual folder called `oss-cad-suite` (the one with the `bin` folder inside).

Once selected you have successfully setup the toolchain.

Ubuntu USB Permission Fix

On Ubuntu you will need to give permissions to you user so that we can run openFPGALoader without being root. To do this you need to run the following from a terminal:

```
curl -sSL https://raw.githubusercontent.com/lushaylabs/openfpgaloa
```

And then once completed you will need to log-out and log back in so that the user permissions take effect and you will need to unplug and replug in the tang nano board.

Windows USB Driver Fix

On windows the default driver does not work with `openFPGALoader` so to get it to work we need to replace the default driver. To do this you need a program called Zadig which can be downloaded here.

Download the tool and then open it up, once open you need to select "Options" > "List All Devices" from the top menu to show all connected usb devices. Next in the dropdown you should see two devices:

1. JTAG Debugger (Interface 0)

2. JTAG Debugger (Interface 1)

Interface 0 is the JTAG interface `openFPGALoader` needs to program the Tang Nano while Interface 1 is the serial interface used for the UART communication.

It is important to leave interface 1 as-is and only replace interface 0 here.

So select "JTAG Debugger (Interface 0)" and underneath you should see it show that it will change the driver to "WinUSB" if WinUSB is not selected you can use the up/down arrows to select it.

Once selected just the replace driver button and wait a minute or two, once done it should work in openFPGALoader.

If you ever want to uninstall this driver and go back to the default driver (for example to use the official gowin IDE) all you have to do is go to "Device Manager" select the device called "JTAG Debugger" under "Universal Serial Bus devices" right-click on it and press "uninstall device" from the popup also select the checkbox to attempt to remove the driver. Once removed right click on any item in the device manager window and press "Scan for hardware changes" this should reconnect the device back with the original driver.

Other Recommended Software

Besides for the toolchain itself there is some other software which can aid in the development of FPGA cores, and will be used throughout this series.

Node.js

During the course of development you may find yourself needing to prepare / convert data or even communicate with the FPGA from your computer. Node.js is just my personal preference and is what I will be using in this series, but any programming language will probably have a way to do the same things if you have a different preference.

To install node.js you can either grab the official installer from the node.js site or install using a node version manager like so:

Mac / Ubuntu / Windows WSL

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/inst
nvm install v17.0.1
nvm use v17.0.1
nvm alias default v17.0.1
```

Tabby

Tabby is a cross-platform terminal which has support for serial terminal which will allow us to communicate over UART with the tangnano9k. Here as-well it is a matter of personal preference, I chose to mention tabby since it supports all operating systems and has some extra features like choosing how the data will be read / shown (ascii, hex) and each side can be configured independently.

Tabby can be downloaded here

If you only need basic serial support, the Lushay Code extension comes with a basic built-in serial console. But I still recommend Tabby since the extra features can be nice.

Creating a New Project

We should now have everything we need to effectively develop an FPGA core. To get started open up your editor (VSCode), from the explorer tab you will have a button to open a folder, click it to choose a new directory that we can use for this project.

Once inside you should see your folder in the explorer tab and we can now create some files.

Every FPGA project needs at least two files:

1. A file that describes your circuit (in our case a verilog `.v` file)
2. A file that describes the connection between your circuit and your physical board. (in our case a physical constraints `.cst` file)

Verilog is an HDL language which is what get's synthesized. Inside the code you will usually want to use specific pins from the actual FPGA, to do this in verilog you create a name for the pin (it can be whatever you want), and then in the `.cst` file you define the mapping of pin name to pin along with other configuration for the pin.

So let's start with a simple example which display a binary counter using the on-board LEDs.

Now onboard we have 6 leds and if each will represent one bit in our counter then it means we can count up from 0 to 63 (binary `111111`). Now our onboard clock frequency is 27 Mhz so counting up every clock pulse would mean the counter would cycle through all the options around 422 thousand times a second, t0o fast to be able to see the counting. So let's say we want it to count once every half a second. Then we would need to wait 13,500,000 clock cycles between incrementing the counter.

We can also say that our counting module will require the clock signal on-board as input since we will need it to react to clock cycles and we know we want our output to be the status of the 6 leds.

So now that we have a general plan, let's create a verilog file called `counter.v` and we can start off with what we know.

```
module top
(
    input clk,
    output [5:0] led
);

localparam WAIT_TIME = 13500000;
reg [5:0] ledCounter = 0;

endmodule
```

This code defines a new module called top with an input called `clk` for the clock signal and 6 outputs called `led`. Again the names here for the inputs and outputs will be mapped to actual pins in the `.cst` file later, so the names can be anything you like.

The format for defining an array's size or accessing a group of bits in general is `[MSB:LSB]` which is why 6 bits can be represented as `[5:0]` meaning bit 5 is the most significant bit and bit 0 is the least significant bit, totaling 6 bits.

The next line defines a local constant which will be used at compile time, we have already calculated that we will need 13.5 million clock cycles per counter increment in-order to reach the desired half second delay.

The final line is our actual counter again using the same notation with the `[5:0]` to say it will be a 6 bit wide register.

We will also need a counter for the clock cycles. Looking at the binary representation of 13,500,000 we see it needs 24 bits to contain so we can create a counter with the signature `[23:0]` for 24 bits.

```
reg [23:0] clockCounter = 0;
```

The next thing we need to do is increment our counters on every clock pulse. This can be done with an `always` block:

```
always @(posedge clk) begin
    clockCounter <= clockCounter + 1;
    if (clockCounter == WAIT_TIME) begin
        clockCounter <= 0;
        ledCounter <= ledCounter + 1;
    end
end
```

The always block starts with it's trigger list or sensitivity list. Essentially what needs to happen for the circuit defined inside to occur. In our case we are saying whenever there is a positive edge on the clock pin (a.k.a clock rise) we want the following to happen.

First we will increment the clock counter and then check if we have reached the wait time defined above. If we have reached the wait time then we would like to reset the clock counter to zero and increment the led counter.

We don't need to deal with ever resetting the led counter as it has exactly the number of bits as the desired counter so it will automatically roll over back to 0 once it increments 63.

It is also worth noting that the `<=` operator is not like a standard assignment operator from most programming languages. This operator sets the value for the input of the flip-flop which will only propagate to the output on the next clock signal. This means that even though we increment it on the first line of the block, the value of `clockCounter` will only equal 1 on the next clock pulse, and for the remainder of the current block the value will still be 0. Same thing when we increment the ledCounter, The change will only be seen on the next clock signal.

There is a way to immediately assign a value using the blocking `=` operator instead, but I like to stick only with the non-blocking assignment operator `<=` when working with registers, both for simplicity, consistency and I think it is a better practice.

The last and final thing to finish our verilog module is to connect the value of our register to the leds.

```
assign led = ledCounter;
```

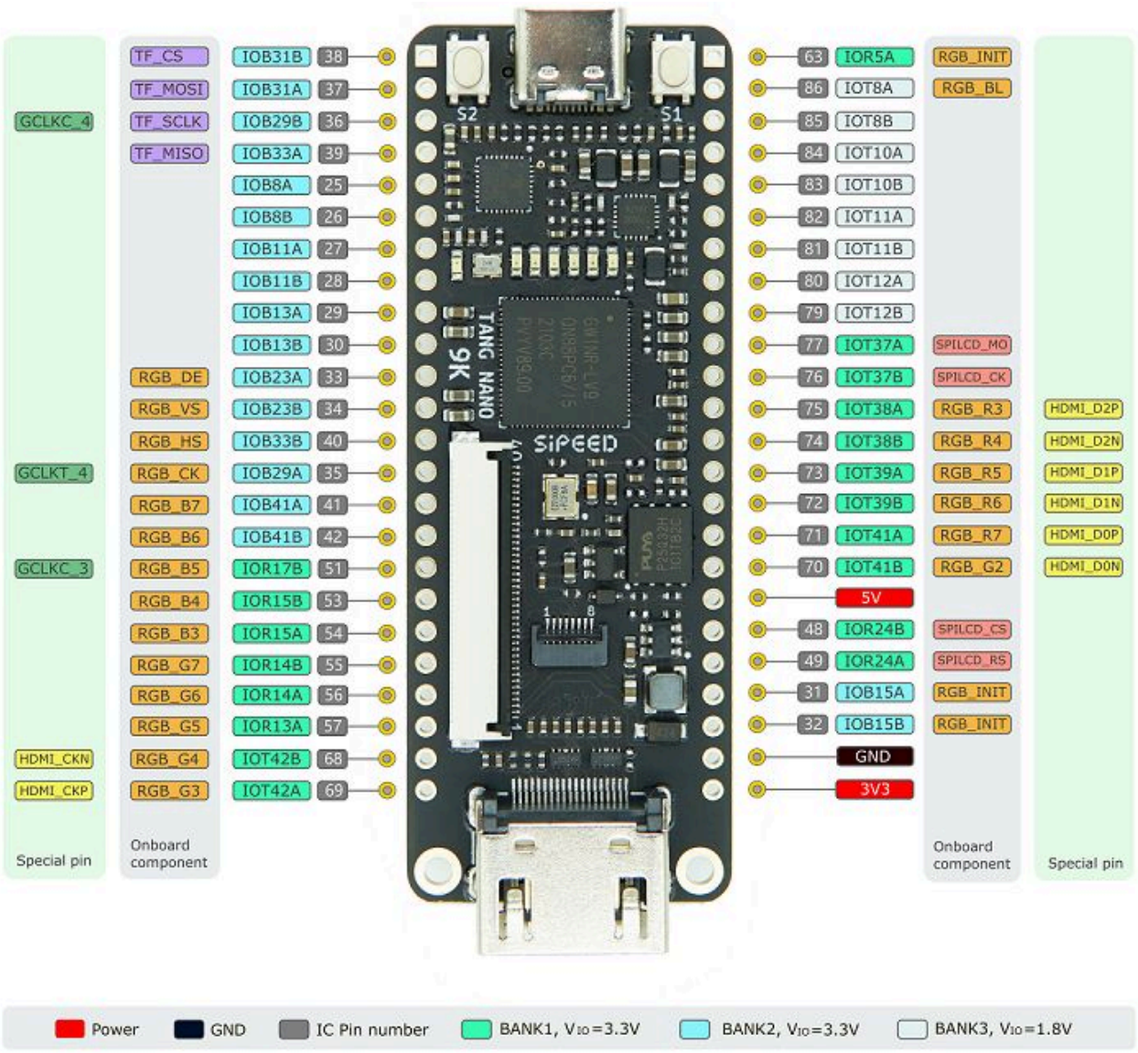
Outside the always block we use the `assign` and `=` to define the value of wires. Wires (which is the default input/output type) unlike registers don't store values so we need to simply define what they are connected to and they will always equal that value (since they will be physically connected to them).

The final code should look like the following:

```
module top
(
    input clk,
    output [5:0] led
);

localparam WAIT_TIME = 13500000;
reg [5:0] ledCounter = 0;
reg [23:0] clockCounter = 0;

always @(posedge clk) begin
    clockCounter <= clockCounter + 1;
    if (clockCounter == WAIT_TIME) begin
        clockCounter <= 0;
        ledCounter <= ledCounter + 1;
    end
end

assign led = ledCounter;
endmodule
```

The next file we need to create is the `.cst` file where we will need to define the I/Os we used `clk` and `led`. First we will take a look at doing it manually so you can get a sense for how it works under the hood, and then we will take a look at doing it through our VScode extension.

To do this manually we need to know some information about the pins. First of all we need to know the pin number. For the GPIO of the board its pretty easy, all the pins are listed in the following image:



So for example the top left pin's number is 38 (IOB31B) and the top right pin (IOR5A) is pin 63, and so on. It's worth noting also that most GPIO are 3.3V but to pay attention that all the BANK3 pins (pin 79-86) are 1.8V pins and should only be used with devices or sources working with 1.8v.

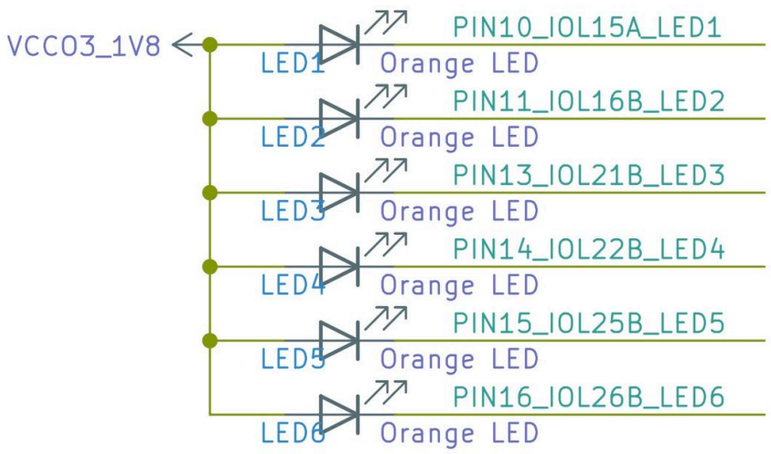
But with all that said, in our example we are not using any of the GPIO pins we are using pins of the FPGA which are routed to on-board components (the clock and LEDs).

Tang Nano 9K 3672 schematic

Tang_Nano_9K_3672_schematic.pdf • 461 KB

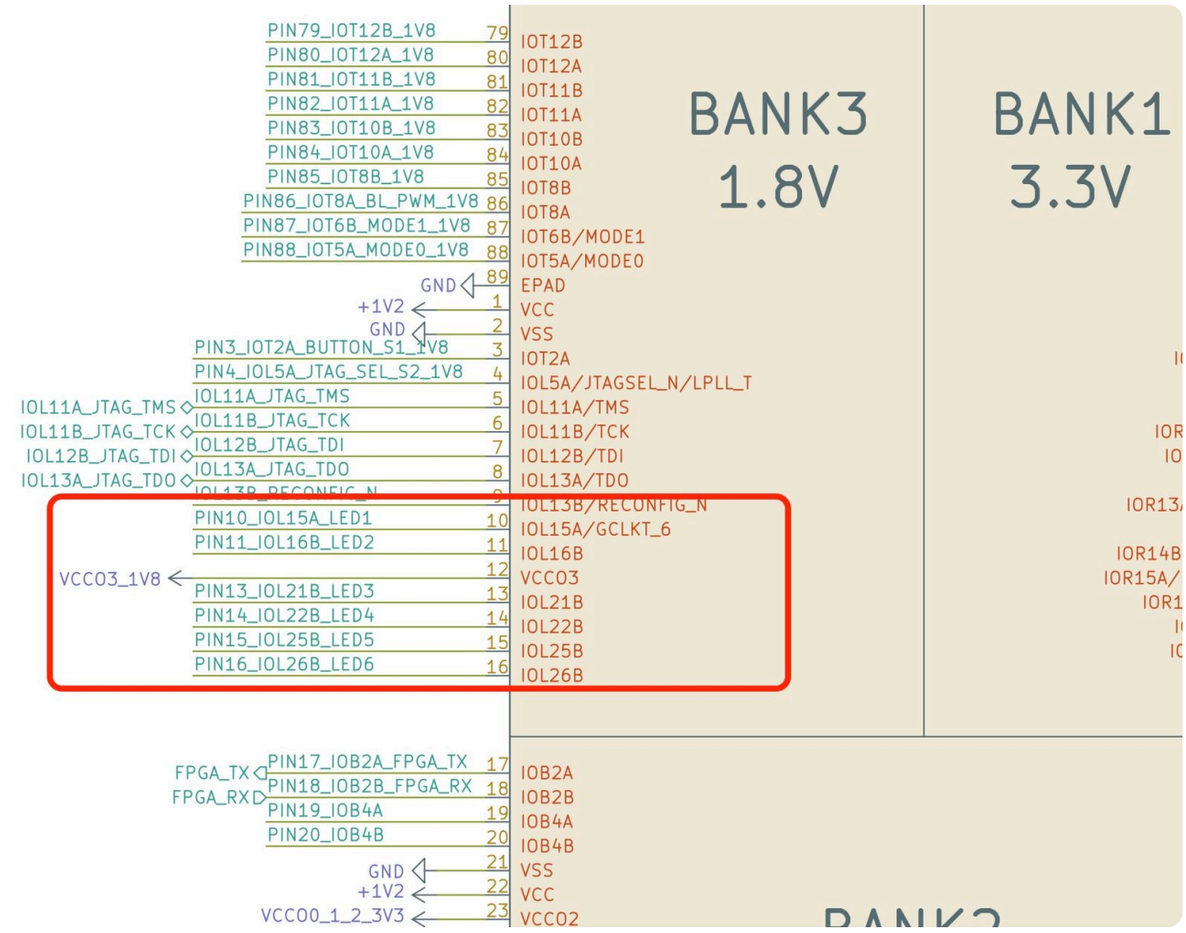
Opening up the file we can see a section called "LED x 7" defining how the 6 LEDs are connected (not sure why x7 probably a mixup with the 7th LED which is connected to the UART module.

LED x 7



A side note - something thing that can be seen here is that the LEDs are common anode all being supplied power directly from the 1.8 voltage source which means we need to pull our pin down to low (0) in order for the led to light up.

But back to what we are looking for. We see the names defined to the connections and then we can see where these references end up on the actual FPGA chip. Above this section we have the FPGA chip itself and we can see they end up as part of bank 3's pins:



More specifically we can see they are pins 10,11,13,14,15,16. Using the same process we can find the clock pin in the xtal (crystal oscillator) section.

Showing that the pin is pin 52.

With that mini investigation complete we would need to create a cst file where we would define the pins like so:

```
IO_LOC "clk" 52;
IO_PORT "clk" PULL_MODE=UP;
IO_LOC "led[0]" 10;
IO_LOC "led[1]" 11;
```

```
IO_LOC "led[2]" 13;
IO_LOC "led[3]" 14;
IO_LOC "led[4]" 15;
IO_LOC "led[5]" 16;
```

IO_LOC defines a location constraint, meaning we are defining that when placing and routing our design the io called clk must be placed on a specific pin defining its specific location.

IO_PORT defines port level constraints like pull ups / pull downs, logic types drive strength etc.

Going through our file we define `clk` to be pin 52 with a pull up resistor, and we are defining the 6 bits of our `led` output using the pin numbers we found.

It's worth noting we are adding the bit index to them only for convenience so that we can receive them all together instead of 6 separate outputs in verilog. But if we wanted we could have written `IO_LOC "led1" 10;` and then in verilog have each LED as a separate output.

You can view a little bit more information about this format from the official physical constraints document from Gowin Appendix A.

SUG935E appendixA

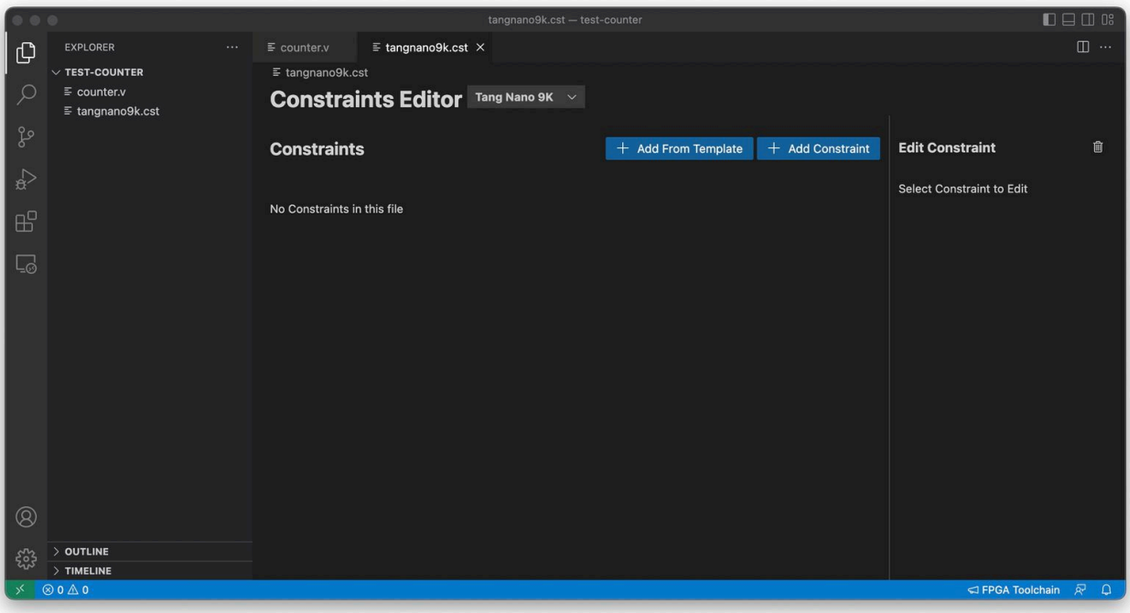
SUG935E-appendixA.pdf • 283 KB

Or more in-depth by searching for the system IO and GPIO specification documents if you would like a more in-depth read. To some extent you can also view some of the options in the apicula longvals documentation here.

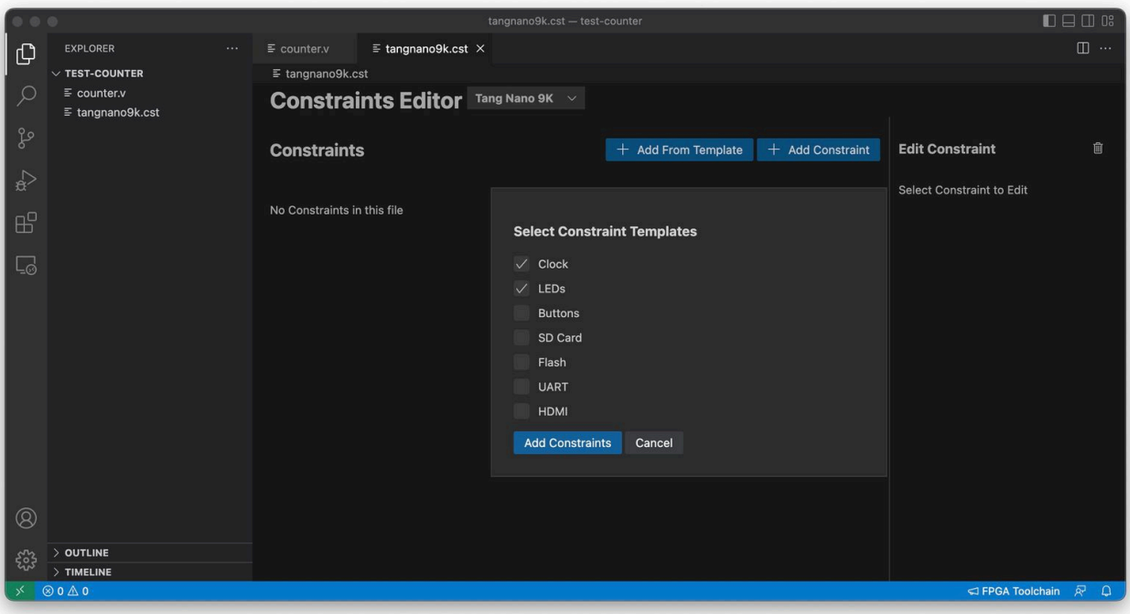
Constraints with LushayCode

With the extension simply create a new file in vscode for example `tangnano9k.cst` and clicking on it you should see something like the

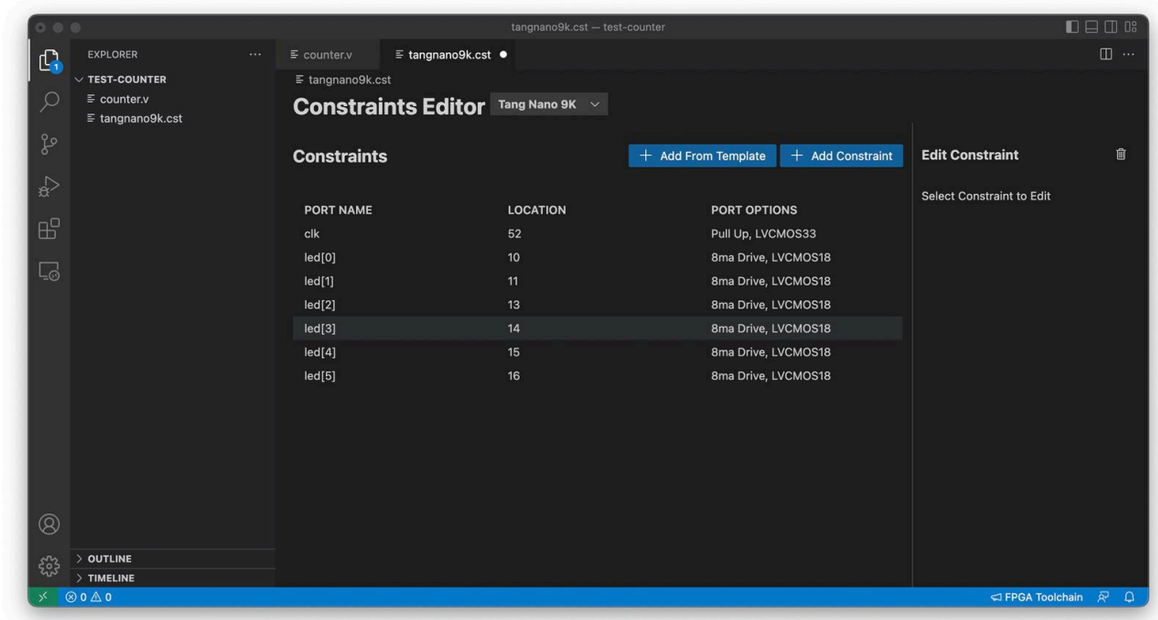
following:



Since both the LEDs and the clock are part of the development board they have an included template, so we can simply press "Add From Template" and select both the clock and leds.



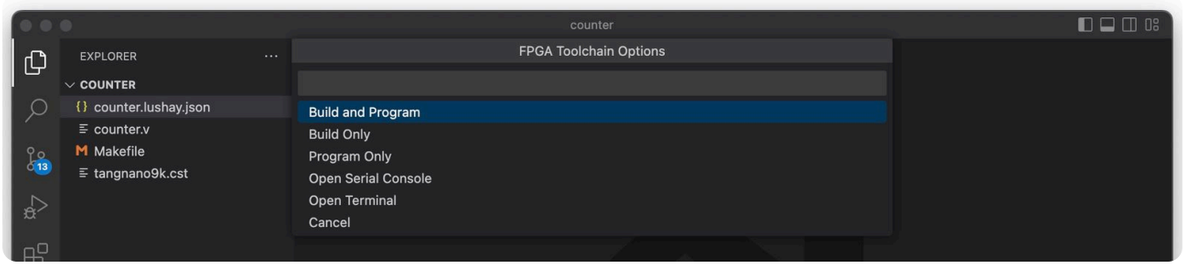
Clicking on "Add Constraints" will add them to the current file and you will see them in the table:



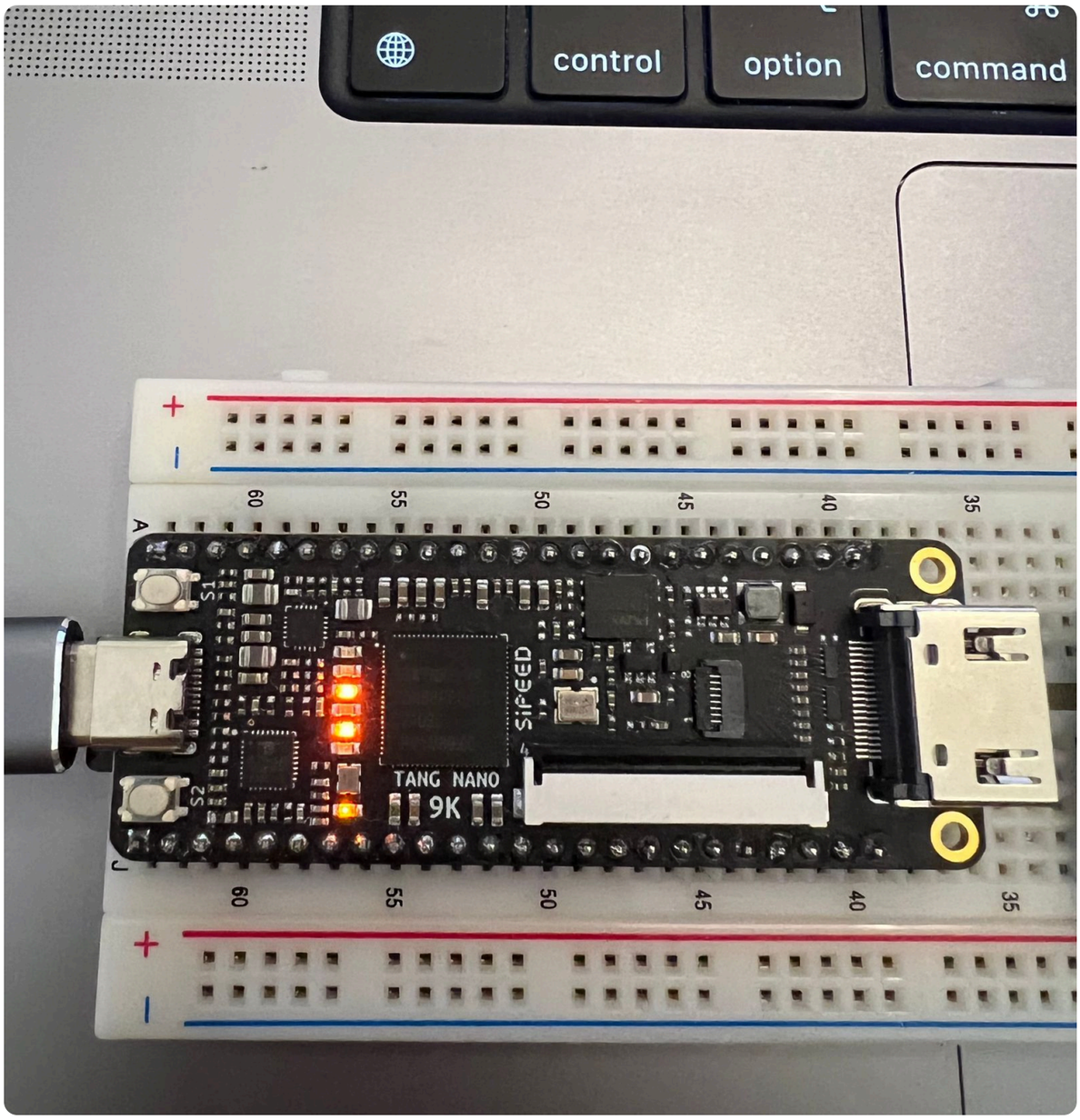
Just save the file and we are done with the constraints. You can see that the pin numbers are the same as we saw when doing the manual investigation.

Running the Example

Our project is now ready so how do we compile it with the toolchain. To do this you can click again on the "FPGA Toolchain" button from our VSCode extension and you should get a dropdown with the different options:



Select "Build and Program" and if all went well you should now see the tangnano counting up over it's LEDs.



The only thing is that the counter is inverted. This is due to the fact that the LEDs use a common anode and a value of 0 means to light up not a value of 1. To fix this we can make a small change to our code.

To update the verilog code in our example invert the counter so that the leds match the bit status using the `~` operator. The final line of our module should be:

```
assign led = ~ledCounter;
```

Re-click on "Build and Program" from the "FPGA Toolchain" button to rebuild and program the FPGA with our fix.

Conclusion

In this article we went through a lot of different topics getting the development environment setup and running a basic example. In the next part of this series we will go more in depth on debugging FPGA projects in-order to visually see what's going on internally.

I would like to thank you for reading and if you have any questions / comments feel free to leave a comment below or reach out on twitter @LushayLabs.

All the code from the example is also available on in our github repo here, and if you would like to order a tang nano 9K and support this site you can do so from our store.

Core Setup

Share this article:

Newer article

Tang Nano 9K: Debugging & UART

Aug 20, 2022 • 23 min read

J

Jonie 1

3 years ago

Hello.
I m new to FPGA 's programing.
I followed this tutorial and everything works, but is there a way to adapt this toolchain to work with Tang Nano 1K?
When i try to flash it with .fs file generated by toolchain im met with id missmatch, but if i take fs file from examples it works.

1 0 Reply

Lushay Labs Mod → Jonie 1

3 years ago

Hey, apologies for the delay replying, yes it is possible, you need to create a file called `<project>.lushay.json` for example here you can call it `counter.lushay.json` and inside you need to add the "board" key to select a specific board. For example:

```
{
"board": "tangnano1k"
}
```

This will change the board when building your project, you also need to select the board from the dropdown in the constraints editor (the .cst file) if you want to use the cst templates or to choose a pin from the board visually.

There are also other settings you can add to the `.lushay.json` file you can see it being used in all our examples on github for example here: https://github.com/lushayla... and you can see the full list of options here https://marketplace.visuals...

0 0 Reply

D

Drew Marold

3 years ago

Great tutorial. I just got my Nano 9k as part of the latest Hackerbox and they recommended this tutorial for getting your toolchain setup. I'm on Windows 10 and everything worked just fine first time through.

1 0 Reply

perry

6 months ago

Hello,
I am having an issue where when I try to build and program I get an error message
Error: unable to open frdi device: -4 (usb_open() failed)
Error: JTAG init failed with: unable to open frdi device

I have used zadig to change the driver for interface 0 so that shouldn't be the problem.

0 0 Reply

philtor

10 months ago

Wow, you've done an excellent job with your VSCode extension. I had the counter example running in about 10 minutes (mostly spent downloading) on my Tang Nano 9k board. Good work!

Will the Tang Nano 20k be supported?

0 0 Reply

K

kh101207

a year ago

I'm new to learning fpga, how do I solve this problem?
Error: Error: Failed to claim FPGA device: mismatch between target's idcode and bitstream idcode
Error: bitstream has 0x1100481B hardware requires 0x0000081b

0 0 Reply

P

Phil Rogers

a year ago

When I run Zadig to replace the driver, VS Code works fine, but it stops my Gowin toolchain from working (can't open the port for programming).
If I reinstall Gowin it resets the driver to the original and works fine, but then VSCode stops working (can't open the port for programming).

Does anyone know how to set it up so that both will work?
Using Windows 11.

0 0 Reply

J

Juan Pepe Guajolotero

a year ago

Hi, I'd like to use the GOWIN IDE as well. I love your VSCode plugin, but I'm comfortable continuing with VHDL. I can upload my code to the GOWIN IDE without issues, but it doesn't seem to work in your environment. Is there any connection between your plugin and the GOWIN IDE? I can do the same in your environment, but not in GOWIN's IDE.

0 0 Reply

J

Juan Pepe Guajolotero Juan Pepe Guajolotero

a year ago

Never mind, I was able to solve the problem. I had to download the newest IDE, https://wiki.sipeed.com/hardware/en/tang/Tang-Nano-Doc/questions.html#error-found . i have to ask, will your plugin admit VHDL code ever? THHXSS :)

0 0 Reply

F

florinn2520

2 years ago

Hello, I have two questions:
- Can we use systemverilog on this toolchain?
- how to program ram instead flash

0 0 Reply

LUSHAY LABS

Lushay Labs Mod → florinn2520

2 years ago

Hey, currently systemverilog is not supported, for programming to ram instead of flash you need to add to your project file (*.lushay.json file) the setting: `"programMode": "ram"` you can take a look here to get a general idea of a project file if you don't currently have one https://github.com/lushayla... and also more info on the file format here https://learn.lushaylabs.co... under Configuration

1 0 Reply

F

Frug

2 years ago

I'm on Ubuntu 22.04.03
ran the permission fix
did everything and ran build program

it seems to do stuff but stops on the last bit here (no flashing LEDs)

Finished Bitstream Generation
Starting FPGA Programming with OpenFPGALoader
write to flash
Jtag frequency : requested 6.00MHz found 1 devices
index 0:
idcode 0x100481b
manufacturer Gowin
family GW1N
model GW1N(R)-9C
irlength 8
File type : fs
Parse file Parse /home/frug/projects/tang9ktest/tang9ktest.fs:
checksum 0x863a

see more

0 0 Reply

LUSHAY LABS

Lushay Labs Mod → Frug

2 years ago

hmm, The permission fix seems to have worked (without it it can't use the USB device at all, and here it is able to read the idcode and communicate) but it looks like there is some other issue here with the ID Verify failed it looks like it is in some kind of loop and I don't see that it was flashed. What version of the open source toolchain are you using, I will try and reproduce it with your OS and version and see if I have the same issue, also another thing you can try is to try overwriting your tang9ktest.fs file with the file from here https://github.com/lushayla... name it `tang9ktest.fs` and overwrite your file, this file has been tested on other devices so we can rule out and bitstream generation issues.

0 0 Reply

P

Paul → Lushay Labs

2 years ago

Hey! Not OP but I have the same OS, did everything you suggested, but I'm still getting the same errors. The extension version I have is v0.0.18, which seems to be the latest. Any luck fixing it?

0 0 Reply

Lushay Labs Mod → Paul

2 years ago

What version of OSS-cad-suite are you using, after testing different versions I seem to have a problem using openFPGALoader in the newer releases as-well (not the same error message as above but it fails to program), I would suggest trying this version: https://github.com/YosysHQ/... just replace your existing folder with this one and let me know if it solves the programming issue, it seems to have some kind of issue regarding verifying the ID although from what I see in the logs the IDs look fine both for the file and chip

0 0 Reply

Virantha Ekanayake

→ Lushay Labs

2 years ago edited

Was definitely having the same ID Verify Failed issue using the manual tool-chain and your counter.fs. It appears to be a problem with the latest v0.11.0 of openFPGALoader. If I checkout v0.10.0 instead, then your provided counter.fs downloads and works fine.

Still having an issue if I compile my own bitstream using my own compiled yosys and nextpnr-gowin, it programs fine but doesn't blink the LEDs. Will update if I figure out what was the breaking change.

0 0 Reply

Lushay Labs Mod

→ Virantha Ekanayake

2 years ago

Thank you for getting to the bottom of this and updating, I will look into this

0 0 Reply

Virantha Ekanayake

→ Lushay Labs

2 years ago

So, I finally have the most up-to-date versions of the flow that actually generate a working bitstream for the counter.v example. I started

with the oss-cad-suite tools from Feb-10, 2023 as you mention, and this set of yosys/nextpnr/apycula/openfpgaloader makes my leds blink correctly.

I had an inkling there was something wrong with the clock placer/routing in the current versions because I could in the RTL set the LEDs statically (initial conditions) but anything based off of the clock would not toggle. So I knew at least most of the design was getting generated correctly.

After a lot of triage, I have the following versions that work together fine. The key was finding that apycula newer than 0.8.2 (as

see more

0 0 Reply

HA **Hussein Ahmed**
2 years ago

Hello.
i have some troubles with zadig, i can't find the jtag options like in the tutorial. Is it because i haven't installed openfpgaloader?

0 0 Reply

Lushay Labs Mod → Hussein Ahmed
2 years ago

Hey, sorry for the delay, Zadig is a separate piece of software that can be downloaded from here, https://zadig.akeo.ie/ it is only required on Windows to update the driver to be compatible with the open-source toolchain.
So if you are on Windows you can download it and run it to change the driver for Interface 0 only to "WinUSB" then it should be picked up by openfpgaloader.
With the driver setup, you can download oss-cad-suite and put it in the root of your drive (the name cannot have spaces in it) and then through the Lushay Code plugin in vscode you should be able to link to this folder and start building / programming the tang nano

0 0 Reply

C **chin keong Lam**
2 years ago

how to solve this problem : apycula/gowin_pack.py", line 984, in main
AttributeError: module 'importlib.resources' has no attribute 'files' ?

0 0 Reply

C **chin keong Lam** → chin keong Lam
2 years ago

problem solved by using latest tgz

1 0 Reply

Fedor Goncharov
2 years ago

Hi,

I followed the tutorial for Tang Nano 4K and met some problems when launching VSCode extension - Lushay Labs. So the OSS-build was failing on PnR-stage, but this was essentially the problem that by default the toolchain was running for Tang Nano 9K - don't know why. As explicitly put the config file <projectname>.lushay.json with {"board": "tangnano4k" ..} everything worked out smoothly. To say - I find

this tutorial great and authors did really a great job to keep things simple but essential. Bravo. I will continue your set of tutorials adapting them where I can.

P.S. Can you check in your VSCode extension if returning error messages from openFPGALoader is correct?
I checked the related openfpgaloader-program.js and it has a weird line in method onCommandPrintErrorLine(...) { ... if (!this.detectedError) { #print some error msg }. If detectedError = false, then you should not print anything, no? I noticed this because I recieved log-error messaged on openFPGALoader stage with CRC-checksum, though all was running smoothly and program was saying that CRC is correct.

0 0 Reply

Lushay Labs Mod → Fedor Goncharov
2 years ago

Thank you for reporting this, regarding the first issue do you have multiple project files by chance in the same workspace in vscode, and another question was the project file selected from the bottom or does this happen when it is on "<auto detect="" project="">"

Regarding the second issue, it's a bit confusing there, but if detectedError = false inside the print error function means we got an error message from openFPGALoader, but we are not sure what it is, so we print it out as is, as opposed to "device not found" for example which once we detect what the issue is we won't print out the raw error and rather in onCommandEnd we will print out a formatted version of the error message.

Not sure why the CRC checksum would end up there if it is ok, I will look into this

Thanks again

1 0 Reply

Shadowtrance
2 years ago

Hi
I'm setting up the toolchain etc with your vscode extension while I wait for my tang nano 20k to arrive.
Just messing with the counter example, unchanged (as you have it on github for the 9k) and I'm running into an issue when building.
I haven't figured out how to change it all for the 20k yet but I'll get there, just wanted to make sure it was all working first.
Anyway, i hit the snag at this point in building...

> Finished PnR
> Starting Bitstream Generation with Apicula
> testing. You are advised not to use it for production.
>
> Traceback (most recent call last):
> File "c:\tang-nano\oss-cad-suite\bin\gowin_pack-script.py", line 33, in <module>

> sys.exit(load_entry_point('Apycula==0.8.2a1', 'console_scripts', 'gowin_pack')())

see more

0 0 Reply

caljer12 Shadowtrance

2 years ago

I had the same issue on MacOS (Monterey 12.6.6) and reverting to oss-cad-suite version 2023-04-06 did the trick. In addition, and prior to that issue showing up, there was an issue with Qt framework files (even after running ./activate in terminal) and I had to manually and one by one by one allow execution via System Preferences... > Security & Privacy tab.

0 0 Reply

Lushay Labs Mod Shadowtrance

2 years ago edited

Hey, I think this may be an issue with the specific oss-cad-suite version, since it has automated nightly builds of oss-cad-suite, sometimes the versions of the different tools inside can be mismatched causing issues. Could you tell me which version you downloaded so that I can test it out. Also if you want you can try using this version:
https://github.com/YosysHQ/oss-cad-suite-build/releases/tag/2023-02-10
or this version:
https://github.com/YosysHQ/oss-cad-suite-build/releases/tag/2023-04-06

these have been reported to be working by others windows.

Also regarding changing board, you need to create a project file (which is a file ending in `.lushay.json`) which can be created manually or via the <auto detect project> button in the toolbar. With the file created you can change the board like so:

```
{
"board": "tangnano20k"
}
```

You can see the full list of options that can be added to the project file here: https://learn.lushaylabs.co...

0 0 Reply

Shadowtrance Lushay Labs

2 years ago edited

I downloaded the latest version at the time which was 2023-06-25 oss-cad-suite-windows-x64-20230625.exe and now there's a newer version (2023-06-26).
I'll try an earlier version and see how it goes.
Ok i can confirm 2023-04-06 works with the default project for the 9k. And after some digging it seems 20k isn't supported yet in nextpnr.

0 0 Reply

tuxinoo

3 years ago

Hello,

Thanks for you tutorials that gave me the need to buy a Tang Nano 9k dev board. I receive my own recently, and unfortunately, the code doesn't work as is depending on version of your GOWIN chip.
Currently, the source code seems to have been tested and validated for GW1RN-**9C** chip. But, with some boards, you buy en AliExpress for instance, like mine, you can have an issue when trying to burn your `.fs` file to your board with an ID product error.
So, I did the test with `counter` and you have to change the `FAMILY` property in the `Makefile` if you want to succeed. I don't know the difference between those chips and I didn't try all tutorials source code on GitHub repo yet. So, when you search on Google, we will find a link to a Gowin PDF document with a mention to "*GW1RN-9C revised to GW1RN-9*" on 06/12/2020 in history revision...

0 0 Reply

Lushay Labs Mod → tuxinoo

3 years ago

Hey, thank you for reporting this and bringing it to my attention, I am glad you found a workaround, but I would like to get to the bottom of this, I spoke with Sipeed, and they said they did not change the chip to something in the GW1RN-9 family, so I am wondering if it is a software issue, where for example the family name changed in the open source toolchain, or a hardware issue, where maybe the wrong chip was included. To help debug it I was wondering, did you have to change the device as well or did you keep it as `GW1NR-LV9QN88PC6/I5` ? also which version of the toolchain are you using ? so that I can try and reproduce the issue on my end.

0 0 Reply

Tuxinoo

3 years ago

Hello,
Thanks for you tutorials that gave me the need to buy a Tang Nano 9k dev board. I receive my own recently, and unfortunately, the code doesn't work *as is* depending on version of your GOWIN chip. Currently, the source code seems to have been tested and validated for GW1RN-**9C** chip. But, with some board you buy en AliExpress for instance, like mine, you can have an issue when trying to burn your `.fs` file to your board with an ID product error. So, I did the test with `counter` example and you have to change the `FAMILY` property in the `Makefile` (ie https://github.com/lushayla... to GW1RN-**9** if you want to succeed.
I don't know the difference between those chips and I didn't try all source code modules in GitHub repo yet. So, when you search on Google, we will find a link to a Gowin PDF document with a mention to "*GW1RN-9C revised to GW1RN-9*" on 06/12/2020 in history revision...

0 0 Reply

Frank

3 years ago

I received a message of termination with error. Thinking that moving oss-cad up the directory structure to C:\ might solve the problem, I moved it there. Now when I click "FPGA Toolchain" I get no response. How can I re-link the new location to Lushay Code extension?

0 0 Reply

Lushay Labs Mod → Frank

3 years ago

Hey Frank, If you go to the extension settings you can clear (just delete the text) in the OSS Cad Suite Path text box, then the next time you click on "FPGA Toolchain" it will ask you to reconnect it. I am adding a link to a PDF explaining how to get to the extension settings page https://drive.google.com/file/d/1aTQX3x-oP4zTg2nTuz7RPXK8c63yiSIK/view?usp=sharing let me know if this helps

0 0 Reply

Vince Bednarz

3 years ago

I am having a problem understanding the "module name" requirements. In the readme.md there is a statement that says there must be a module named "top". Which appears to be the case since if I name a module "counter" as I've found in your github repo, yosys fails, indicating module top was not found. But in the UART example there is no module named "top", which is why it refuses to build. So, what am I missing. Is "top" like "main" in c/cpp? If it's not going to be used is it simply defined as " module top(); endmodule "?
Should I be asking this question elsewhere, like below the UART example?

0 0 Reply

Lushay Labs Mod → Vince Bednarz

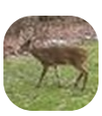
3 years ago edited

Hey, thank you for reaching out. In a verilog project you can have many modules, but you always need one "main" module like you mentioned in a similar way to c/cpp. The difference between a module you instantiate vs the main module, is that the main module receives all its ports (inputs and outputs) from predefined locations in the FPGA via constraints (for example you connect the clock / IO, etc) whereas modules you instantiate you need to connect them to either registers / wires you created or to one of the ports you received in your main module. The main module also allows the toolchain to better optimize and understand if you have included a module in your project for example that was never used it will be able to remove it as part of the optimisations. This process requires a reference point which is this main module.

The name of this module is classically called `top` but you are able to configure this if you would like a different name. So for example with the counter demo, since it is the main module (it

see more

0 0 Reply



Vince Bednarz → Lushay Labs

3 years ago

Thanks for the response, at least I was in the right area as to what the "module top" function is in terms of the Verilog source. As I was writing this reply, more information suddenly appeared in your response. So, now I believe you have answered my questions.

I decided to look through your other posts and

I decided to look through your other posts and tutorials. I just stumbled upon the Lushay Code page and I see a discussion about the project file. I will continue studying that document.

I came upon your site and tools, via the latest "hacker box" project, so I may have missed a reference to that page in my rush to get something working on the tangnano. If it's not already there, I will let folks know on the Instructables hacker box project page.

Coming from the world of embedded microcontrollers, it appears that there is much to learn to get beyond the "hello world" FPGA "program"!

0 0 Reply

Lesley Vaden

3 years ago

I think its okay now. I re-installed the driver one more time and it worked on my windows 10 machine. I think windows 11 and mac os didn't things as much but windows 10 was better

0 0 Reply

Lushay Labs Mod → Lesley Vaden

3 years ago

That's great to hear that you got it working, and thank you for letting us know about windows 11 and mac os, we have tested those in the past but maybe something has changed, we will retest those OSs to see if anything needs updating

0 0 Reply

Lesley Vaden

3 years ago

I've tried re-installing the driver, I've got all 3 extensions in vs code. I have everything here...I've been using vs code and node already so several of the programs are familiar to me. Do i need to check my settings? since I've been working with different things?

What is this .fs file extension? and why does it need an IDCODE? what syntax should I use?

0 0 Reply

This comment was marked as spam.

Lushay Labs Mod → Lesley Vaden

3 years ago

Sounds good, did it generate the fs file now? regarding the issue your are getting now have you replaced the default driver with the WinUSB driver provided by Zadig https://zadig.akeo.ie/ . You need to download Zadig and once open you need to select "Options" > "List All Devices" and then replace the driver for JTAG Debugger (Interface 0) with the WinUSB driver. this is required for it to work with openFPGALoader on windows

0 0 Reply

Lushay Labs Mod → Lesley Vaden

3 years ago

The IDCODE is used to identify the FPGA device itself so that it can know before programming that the bitstream (.fs) file was generated for this specific device or not so not to accidently program a bitstream for a different FPGA. See my comment below, you don't need to manually create the fs file, it will be generated if the toolchain finishes successfully. Another thing you can try is to run the commands manually and see if you get more info. You can press the "FPGA Toolchain" button and then on "Open Terminal" you can then run the commands manually like:

```
yosys -p "read_verilog counter.v; synth_gowin -top counter -json counter.json"
nextpnr-gowin --json counter.json --write counter_pnr.json --freq 27 --device GW1NR-LV9QN88PC6/I5 --family GW1N-9C --cst tangnano9k.cst
gowin_pack -d GW1N-9C -o counter.fs counter_pnr.json
```

if running the counter example, it may provide more info on why it is not compiling

0 0 Reply

Lesley Vaden

3 years ago

so since it was asking for a FPGA.fs file, I made one and the build still fails but now the error is changed to:

```
Finished PnR
Starting Bitstream Generation with Apicula
failed to create process.
Finished Bitstream Generation
Starting FPGA Programming with OpenFPGALoader
write to flash
Jtag frequency : requested 6.00MHz found 1 devices
index 0:
idcode 0x100481b
manufacturer Gowin
family GW1N
model GW1N(R)-9C
irlength 8
File type : fs
Parse file Parse c:\Users\phosp\FPGA\FPGA.fs:
Warning: IDCODE not found

Warning: Unknown IDCODE
Error: Error: Failed to claim FPGA device: stoul
Finished FPGA Programming
Task finished with errors exiting
Toolchain Completed
```

0 0 Reply

Lesley Vaden

3 years ago

my build is failing with:

```
Finished PnR
Starting Bitstream Generation with Apicula
failed to create process.
Finished Bitstream Generation
Starting FPGA Programming with OpenFPGALoader
write to flash
```