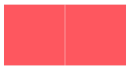
Totally Reconfigurable Engineering Solutions: Module Abstraction Layers

Chris Gammell
Head of Hardware and Developer Relations
Golioth

Module Abstraction Layers

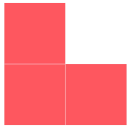
PROBLEM SPACE

Why I considered making a module as opposed to some other method of assembling and deploying electronics?



WHAT WE'VE BUILT

We'll take a look at the process of designing a module and finding a form factor that makes sense.



IMPLEMENTATION ANALYSIS

Diving into the Zephyr side of the implementation and how we plan to put it all together (and whether we should)

About Golioth

WHAT IS GOLIOTH?

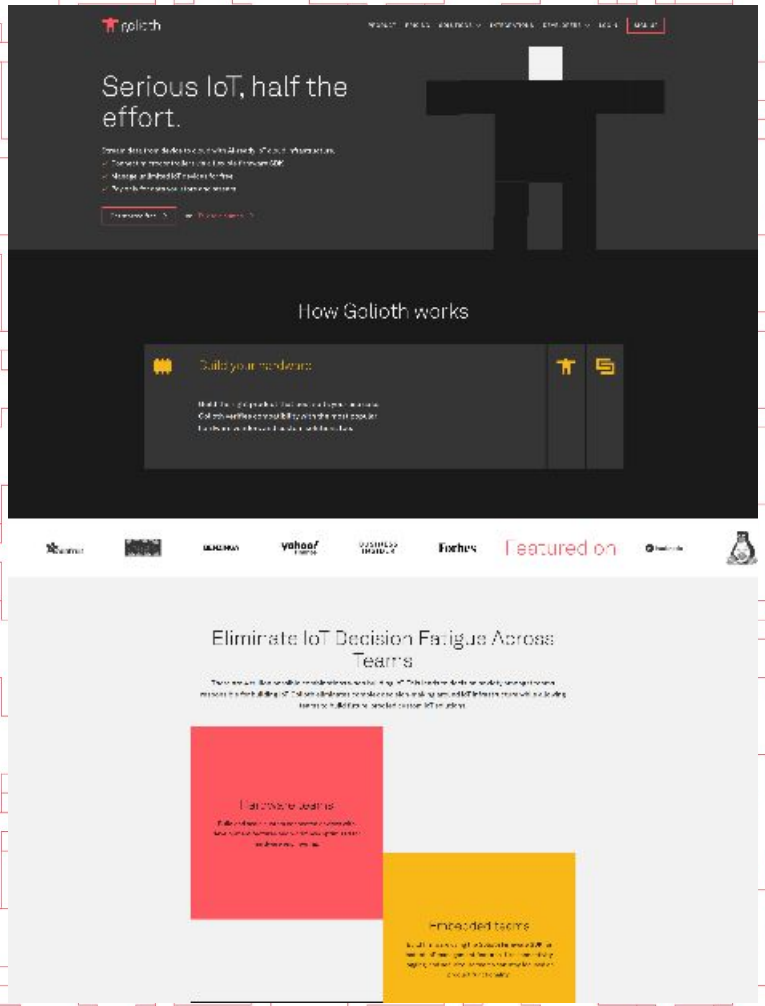
Golioth is a universal IoT platform where developers can prototype, connect, and manage IoT fleets.

OUR MISSION

It's our mission to improve everyday reality with IoT. We help organizations do more with connected sensors: from building new revenue streams to increasing resource efficiency.

OUR APPROACH

We believe developers should build IoT their way. They choose what to build and what to do with the data. Golioth is the universal connector that enables them to connect everything up securely and efficiently without the stress.



Problem We Want To Solve

As always, my story starts with hardware

What if I need to swap out the “essence” (core) of my board?

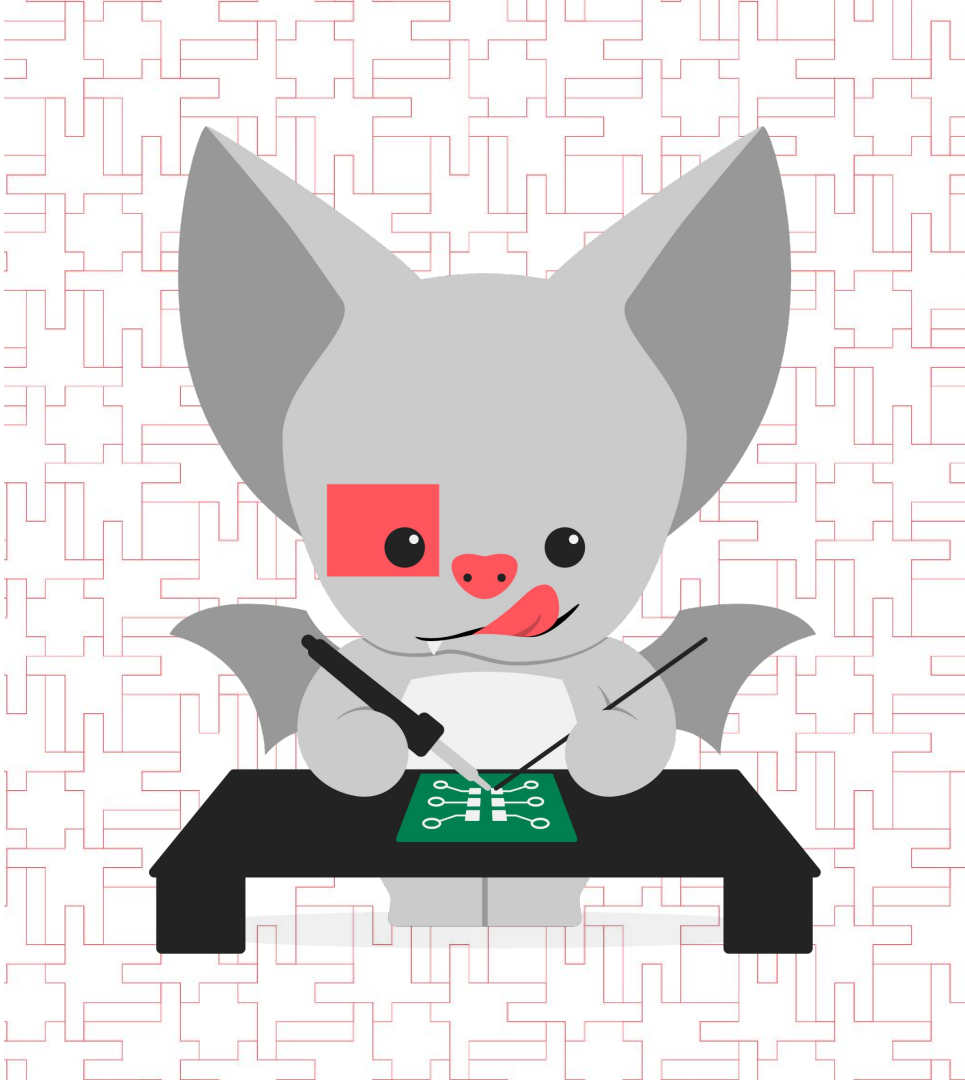
This has happened to me many times before

I WANT TO TARGET A NEW PROCESSOR

- Perhaps I want to take advantage of the wide range of processors available in Zephyr (growing every day, it seems!)
- Or I want to try out a new IDE to see how it works
- Maybe I got a good deal on a particular part and would like to try that out in the design to have a lower cost version of the overall product
- Perhaps it's 2022 and no parts can be found...

I WANT A NEW COMMUNICATION MECHANISM

- Most commonly, I want to target a different communication technology
- Retargeting Wi-Fi vs Cellular vs Ethernet vs Bluetooth has a bunch different silicon considerations

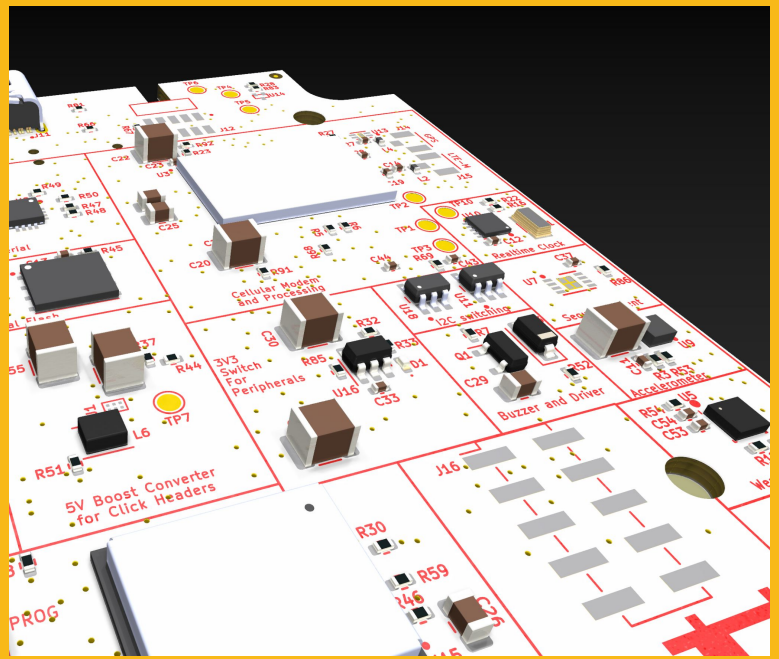


On the parts/comms Zephyr has us covered!

(if this is new info to you, we'll find you some people to talk to right after this presentation!)

The Downsides of Redesigning The Core

- **Lack of standardization** over time, because individual designs will find efficiencies and won't be constrained by the same upfront set of requirements.
- **Duplicated work** both on the hardware side and in the firmware that needs to support it (imagine all the overlay files!)
- **Speed of implementation** is lower than it could be, because each time we need to re-route the core section (potentially lower with reusable layout segments in eCAD tools)



What if we took the modularity of a connector or shield...

...and applied that same abstraction to a module edge?

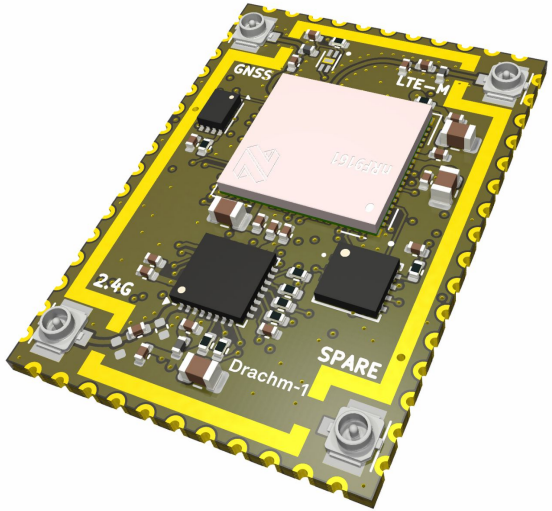
So...a module?

TAKING A GUESS AT FORM FACTOR

- We want it to be compact enough that it matches the expected size of other modules
- But it needs to be big enough to fit different (off-the-shelf) parts on there, including some that will be modules themselves
- The current module has 34 available pins (on a 2mm spacing) and the module size is 34x24

OPTIMIZATION FOR OUR USE CASE

- Flexible, while not introducing unnecessary complexity. We didn't have 100 edge pins with 60 of them set as "reserved".
- Keying off of the needs of the Elixir as a MVP of which pins we'd need in the future.



What We’ve Built (So Far)

I grabbed 4 configurations of parts / comms

- **nRF9151** (main processor, LTE-M/NB-IOT/GPS) with **ESP32-C3** (secondary, Wi-Fi/BT)
- **nRF52840** (main processor, BT) with **BG77** (secondary, LTE-M/NB-IOT/GPS)
- **ESP32-C3** (main processor, Wi-Fi/BT) with **BG77** (secondary, LTE-M/NB-IOT/GPS)
- **ESP32-S3** (main processor, Wi-Fi/BT) with **BG77** (secondary, LTE-M/NB-IOT/GPS)

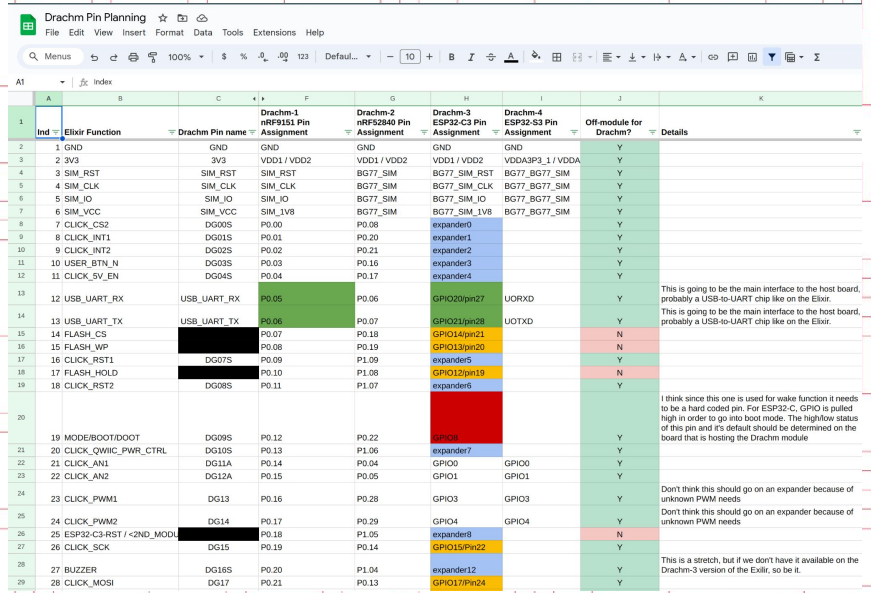
Pulling soft requirements from the Elixir board

KEEPING THE SAME FORM FACTOR

- I have a form factor that I like that includes breakouts for things like MikroBus Click boards
- There are peripherals on board that I want to keep, though those might change in the future (ie. sensors RTC, eSIM, etc)

BALANCING WHAT IS ON MODULE VS NOT

- Truly, what is the “essence” of a module?
- What is the bare minimum feature set of parts that will allow communications and processing?
- How do we balance the need for different pin functions on different chipsets?



Ind	Elixir Function	Drachm Pin name	Drachm-1 nRF9151 Pin Assignment	Drachm-2 nRF52840 Pin Assignment	Drachm-3 ESP32-C3 Pin Assignment	Drachm-4 ESP32-S3 Pin Assignment	Off-module for Drachm?	Details
1	GND	GND	GND	GND	GND	GND	Y	
2	3V3	3V3	VDD1 / VDD2	VDD1 / VDD2	VDD1 / VDD2	VDDA3P3_1 / VDDA	Y	
3	SIM_RST	SIM_RST	SIM_RST	BG77_SIM	BG77_SIM_RST	BG77_BG77_SIM	Y	
4	SIM_CLK	SIM_CLK	SIM_CLK	BG77_SIM	BG77_SIM_CLK	BG77_BG77_SIM	Y	
5	SIM_IO	SIM_IO	SIM_IO	BG77_SIM	BG77_SIM_IO	BG77_BG77_SIM	Y	
6	SIM_VCC	SIM_VCC	SIM_1V8	BG77_SIM	BG77_SIM_1V8	BG77_BG77_SIM	Y	
7	CLICK_CS2	DG00S	P0.00	P0.08	expander0		Y	
8	CLICK_INT1	DG01S	P0.01	P0.20	expander1		Y	
9	CLICK_INT2	DG02S	P0.02	P0.21	expander2		Y	
10	USER_BTN_N	DG03S	P0.03	P0.16	expander3		Y	
11	CLICK_5V_EN	DG04S	P0.04	P0.17	expander4		Y	
12	USB_UART_RX	USB_UART_RX	P0.05	P0.06	GPIO20/pin27	U0RXD	Y	This is going to be the main interface to the host board, probably a USB-to-UART chip like on the Elixir.
13	USB_UART_TX	USB_UART_TX	P0.06	P0.07	GPIO21/pin28	U0TXD	Y	This is going to be the main interface to the host board, probably a USB-to-UART chip like on the Elixir.
14	FLASH_CS	[redacted]	P0.07	P0.18	GPIO14/pin21		N	
15	FLASH_WP	[redacted]	P0.08	P0.19	GPIO13/pin20		N	
16	CLICK_RST1	DG07S	P0.09	P1.09	expander5		Y	
17	FLASH_HOLD	[redacted]	P0.10	P1.08	GPIO12/pin19		N	
18	CLICK_RST2	DG08S	P0.11	P1.07	expander6		Y	
19	MODE/BOOT/DOOT	DG09S	P0.12	P0.22	GPIO8		Y	I think since this one is used for wake function it needs to be a hard coded pin. For ESP32-C, GPIO is pulled high in order to go into boot mode. The high/low status of this pin and it's default should be determined on the board that is hosting the Drachm module
20	CLICK_QWIIC_PWR_CTRL	DG10S	P0.13	P1.06	expander7		Y	
21	CLICK_AN1	DG11A	P0.14	P0.04	GPIO0	GPIO0	Y	
22	CLICK_AN2	DG12A	P0.15	P0.05	GPIO1	GPIO1	Y	
23	CLICK_PWM1	DG13	P0.16	P0.28	GPIO3	GPIO3	Y	Don't think this should go on an expander because of unknown PWM needs
24	CLICK_PWM2	DG14	P0.17	P0.29	GPIO4	GPIO4	Y	Don't think this should go on an expander because of unknown PWM needs
25	ESP32-C3-RST / <2ND_MODU	[redacted]	P0.18	P1.05	expander8		N	
26	CLICK_SCK	DG15	P0.19	P0.14	GPIO15/Pin22		Y	
27	BUZZER	DG16S	P0.20	P1.04	expander12		Y	This is a stretch, but if we don't have it available on the Drachm-3 version of the Exilir, so be it.
28	CLICK_MOSI	DG17	P0.21	P0.13	GPIO17/Pin24		Y	

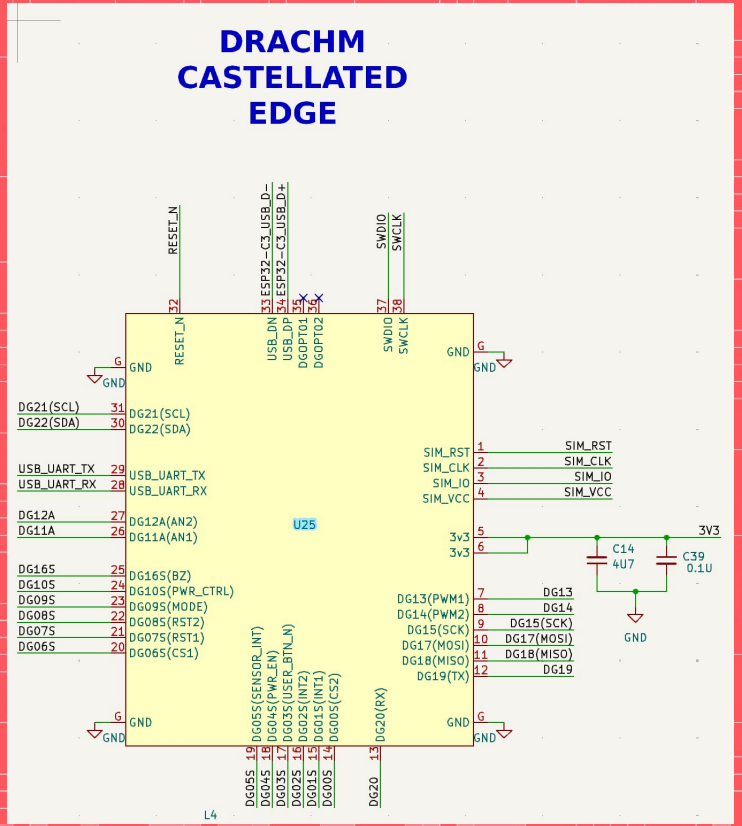
Trade-offs, Trade-offs, Everywhere

PIN CAPABILITIES ARE VARIED

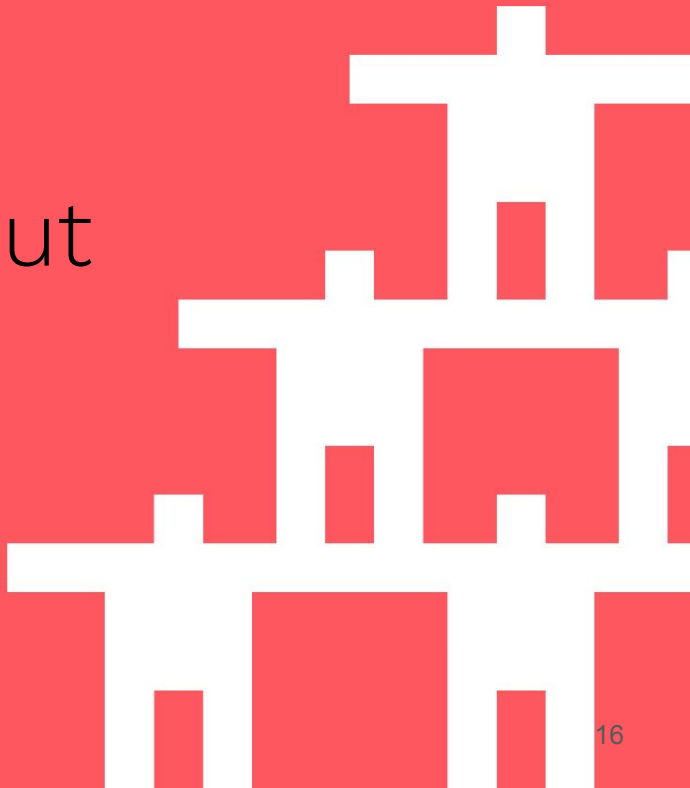
- Some pins are “GPIO only” in case we need to put certain parts on a GPIO expander
- ADC pins are limited (2)
- USB pins are available for parts that have direct connections

ANYTHING NOT ON THE CORE REQUIRES PINS

- eSIMs won’t always be necessary for a design, so it makes more sense to move those off module (might also want a SIM connector)



Analyzing Output



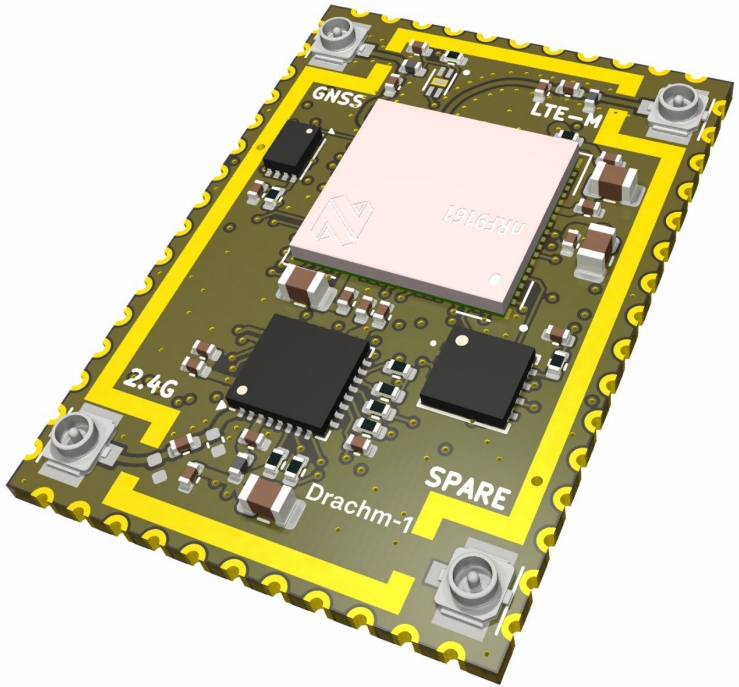
Meet Drachm-1

MAIN FEATURES

- nRF9151 as the main processor (LTE-M, NB-IOT, GNSS)
- External flash memory
- Security module on-board (ATECC608B)
- ESP32-C3 as the secondary processor (Wi-Fi, Bluetooth via the ESP-AT firmware)

EXTERNAL REQUIREMENTS

- Apply regulated 3V3
- Ground
- Programming via **`USB_UART_RX/TX`**



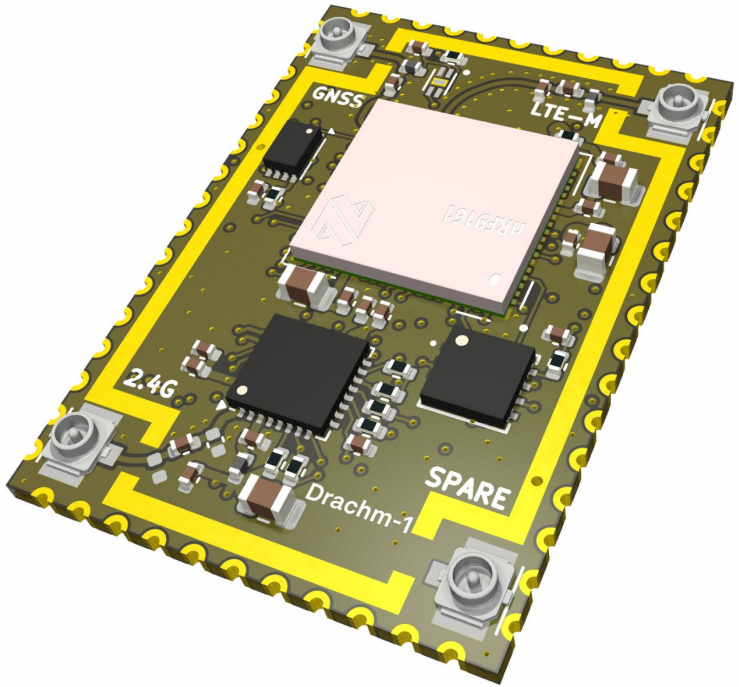
Meet Drachm-1

RF IMMUNITY AND FLEXIBILITY

- It’s immediately obvious this is not a commercial RF module because there’s no basis for pre-certification (aside from the modules onboard). We only utilized u.FL connectors to lean on unknown use cases
- Later in the process I figured out we should probably plan to have an RF shield on top, in case we (or someone else) want to take it towards certification.

SPACE CONSTRAINTS

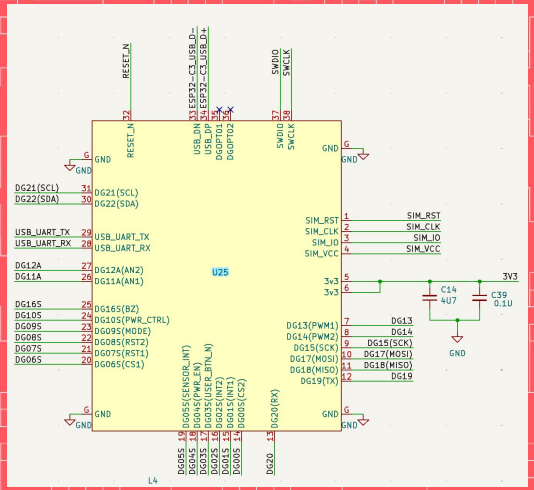
- This pass ended up being “easy” for space constraints, but will likely require re-analysis when a BG77 or equivalent cellular module is introduced.



Zephyr Considerations

MAPPED AT THE SOC LEVEL? BOARD LEVEL?

- Uh, good question! Hardware person here
- Many of the pins we have defined will be reused in common ways, listed on the schematic symbol
- I assigned generic pin names that map the GPIO type (analog, simple GPIO) and a name on the module edge (ie. `GP16S`)
- So is it a SOC? Or a board? Neither?
- We'll use abstract HW interfaces (see Chris Wilson's great article)



Step 1: Using Abstract Interfaces on the Base

NODE LABELS

- Name the peripherals to match the peripheral names on the module, ie. **`drachm_i2c`**
- We'll come back to this in the next slide

GPIO NEXUS NODES

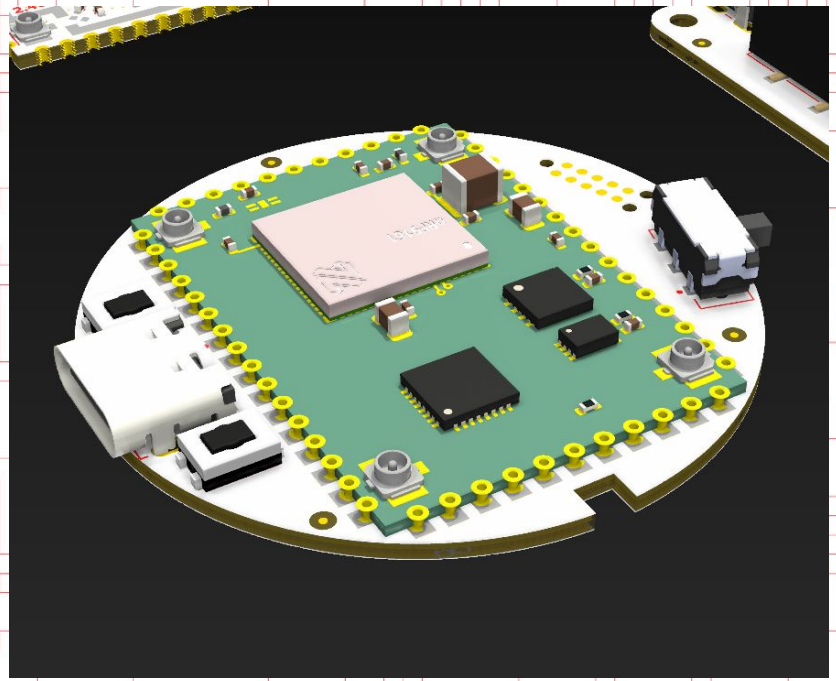
- This allows you to create aliases for GPIOs in Zephyr. We'll map pins based on their capabilities (simple, analog, general)
- Because we have a pin assignment to the edge connector. This will be something like **`<&drachm_gpio_s 16 0>`** and will map to the equivalent GP16S on the footprint, and will ultimately map to the gpio mapping on the underlying chip **`<&gpio 0 20>`** on the nRF9151 for the Drachm-1 board

```
mikrobus_header: mikrobus-connector {
    compatible = "mikro-bus";
    #gpio-cells = <2>;
    gpio-map-mask = <0xffffffff 0xffffffc0>;
    gpio-map-pass-thru = <0 0x3f>;
    gpio-map =    <0 0 &gpio1 9 0>,    /* AN  */
            /* Not a GPIO*/         /* RST */
            <2 0 &gpio0 20 0>,    /* CS   */
            <3 0 &gpio1 2 0>,    /* SCK  */
            <4 0 &gpio1 3 0>,    /* MISO */
            <5 0 &gpio0 26 0>,    /* MOSI */
                        /* +3.3V */
                        /* GND */
            <6 0 &gpio1 8 0>,    /* PWM  */
            <7 0 &gpio0 17 0>,    /* INT  */
            <8 0 &gpio1 24 0>,    /* RX   */
            <9 0 &gpio1 25 0>,    /* TX   */
            <10 0 &gpio1 30 0>,    /* SCL  */
            <11 0 &gpio1 21 0>;    /* SDA  */
                        /* +5V */
                        /* GND */
};
```

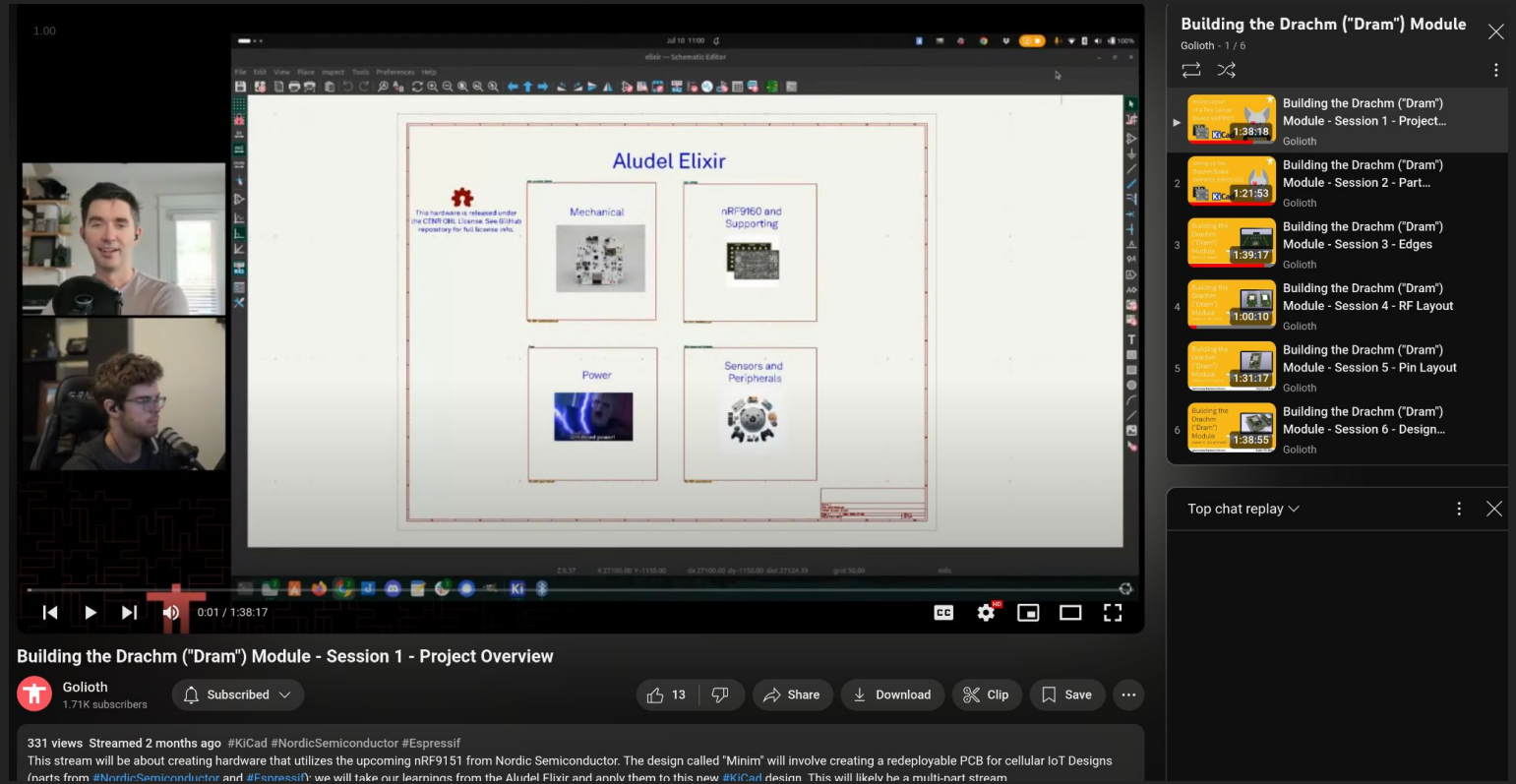
Step 2: The Device “Carrying” the Drachm is a “Shield” in Zephyr

CREATE AN OVERLAY FILE FOR THE SHIELD

- The overlay will call out the node labels we created in the previous step, ie **`drachm_i2c`**
- The label will act as an alias for when we are associating peripherals being used on the carrier board (white)
- Can write the overlay for the module interface without knowing which main/secondary processor are there
- The circular board (“sloshy mcwashy”, since you asked) has a footprint for the Drachm module and we can switch in whichever price point / comms / speed we need.



Want to see more? Watch it all unfold on YouTube



Go build something big

golioth