

TEARDOWN 2025

# HAL no

The story of ch32v003fun  
And how you can make your own SDK

cnlohr on  
[YouTube](#)  
[Twitter](#) / [BSky](#) / [Chaos.Social](#)  
[Github](#)  
(Ask about the [Discord](#))

<https://github.com/cnlohr/halnoslides>

# The story of ch32fun

ch32fun Public

master 7 Branches 3 Tags

Pin Unwatch 39 Fork 190 Star 1.1k

[Go to file](#) [Add file](#) [Code](#)

**About**

Open source minimal stack for the ch32 line of WCH processors, including the ch32v003, a 10C 48 MHz RISC-V Microcontroller - as well as many other chips within the ch32v/x line.

[Readme](#) [MIT license](#)

[Activity](#)

1.1k stars 39 watching 190 forks

**Releases** 2

v1.0rc2 Latest on Jan 31

+ 1 release

**Packages**

No packages published [Publish your first package](#)

**Contributors** 85

+ 71 contributors

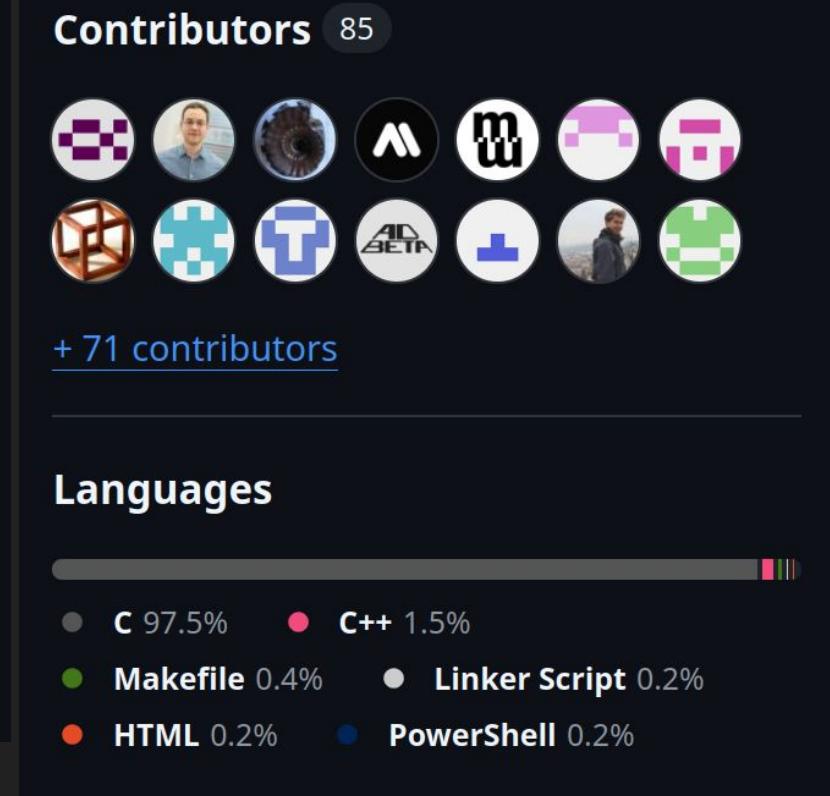
**Languages**

C 97.5% C++ 1.5%

Makefile 0.4% Linker Script 0.2%

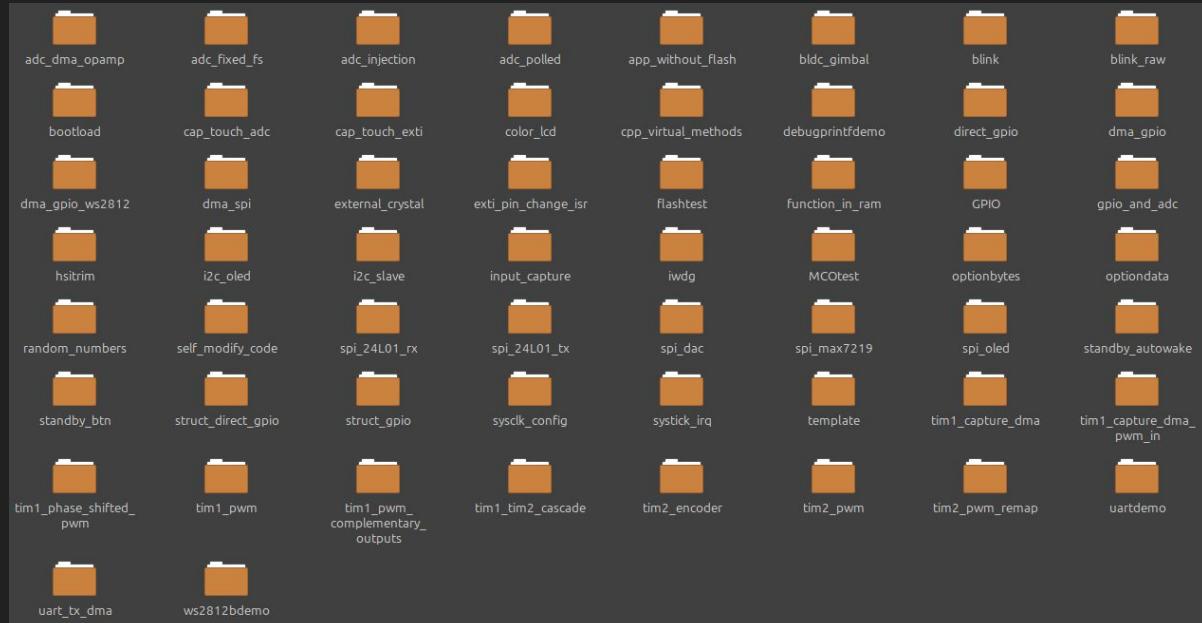
HTML 0.2% PowerShell 0.2%

[README](#) MIT license



# What is it even?

- Like an SDK
- Not a full HAL
- Lots of examples
- Minimal “features”
  - Startup code
  - Header
  - GPIO Config, On, Off
  - Delay
- But these examples...



# Typical Example

- Teaches how underlying hardware works
- Gives a working start
- Easy to grow/change

```
/*
 * initialize adc for polling
 */
void adc_init( void )
{
    // ADCCLK = 24 MHz => RCC_ADCPRE = 0: divide by 2
    RCC->CFGRO &= ~(0x1F<<11);

    // Enable GPIOD and ADC
    RCC->APB2PCENR |= RCC_APB2Periph_GPIOD | RCC_APB2Periph_ADC1;

    // PD4 is analog input ch1 ?
    GPIOD->CFGRL &= ~(0xf<<(4*4)); // CNF = 00: Analog, MODE = 00: Input

    // Reset the ADC to init all regs
    RCC->APB2PRSTR |= RCC_APB2Periph_ADC1;
    RCC->APB2PRSTR &= ~RCC_APB2Periph_ADC1;

    // Set up single conversion on ch1 ?
    ADC1->RSQR1 = 0;
    ADC1->RSQR2 = 0;
    ADC1->RSQR3 = 7; // 0-9 for 8 ext inputs and two internals

    // set sampling time for ch1 ?
    ADC1->SAMPTR2 &= ~(ADC_SMP0<<(3*7));
    ADC1->SAMPTR2 |= 7<<(3*7); // 0:7 => 3/9/15/30/43/57/73/241 cycles

    // turn on ADC and set rule group to sw trig
    ADC1->CTLR2 |= ADC_ADON | ADC_EXTSEL;

    // Reset calibration
    ADC1->CTLR2 |= ADC_RSTCAL;
    while(ADC1->CTLR2 & ADC_RSTCAL);

    // Calibrate
    ADC1->CTLR2 |= ADC_CAL;
    while(ADC1->CTLR2 & ADC_CAL);

    // should be ready for SW conversion now
}
```

# But I can do all this with a HAL? No?

- Maybe?
- “Bootloader” is a full USB stack in 1920 bytes of flash.
- Allows self-program
- Just nice to have USB on whatever pins you want

The screenshot shows a GitHub repository page for 'rv003usb' (Public). The repository is described as 'CH32V003 RISC-V Pure Software USB Controller'. The commit history lists 407 commits from cnlohr, starting with an update to new folder locations and ending with an update to README.md. The repository has 551 stars, 19 watchers, and 56 forks. It includes sections for Releases, Packages, and Contributors, and a Languages chart showing C (90.7%), C++ (3.4%), Assembly (3.4%), Makefile (0.9%), HTML (0.8%), Linker Script (0.7%), and Batchfile (0.1%).

rv003usb Public

master 4 Branches 1 Tag

Go to file Add file Code

**about**

CH32V003 RISC-V Pure Software USB Controller

Readme MIT license Activity 551 stars 19 watching 56 forks

**Releases**

Create a new release

**Packages**

No packages published Publish your first package

**Contributors** 13

**Languages**

- C 90.7%
- C++ 3.4%
- Assembly 3.4%
- Makefile 0.9%
- HTML 0.8%
- Linker Script 0.7%
- Batchfile 0.1%

# But really, why?

- Fast iteration time.
- Stop making business decisions on fictional technical reasons.
  - If complicated apps only take a few kB, no need for expensiver chips.
  - Paradoxically avoids vendor lock-in. *static\_i2c.h*
- Reason about available resources
- Stops rationale for mediocrity
- Find new solutions to problems

**How many clock cycles does digitalRead/Write take?**

Projects Programming

Jun 2017 post #1

**sherzaad Edison**

Hi,

Anyone know how many clock cycles does a digitalRead/Write take on an arduino board?

Sherzaad

...

**LarryD**

Jun 2017 post #3

write = ~50  
2-3uS UNO

Oh HAL,

We're not a match.  
I tried to be your pal  
but you make my head scratch.

Your code is a puzzle  
and the interface is jagged.  
While CPU cycles you guzzle  
my interrupt is run ragged!

But the hardware is elegant,  
you time wasting contraption.  
I've discovered registers most relevant  
now I'm done with this abstraction

<https://hackaday.io/project/159821-taptdoa/log/150490-ode-to-hal>

- Electromage

“An idiot admires complexity. A genius admires simplicity.  
[...] for] an idiot, the more complicated it is, the more he will  
admire it. If you make something so [confusing] he can’t  
understand it, he’s going to think you’re a god. Cause you  
made it so complicated no one can understand it.”

- Terry A Davis

“For the simplicity on this side of complexity, I wouldn't give you a fig. But for the simplicity on the other side of complexity, for that I would give you anything I have.”

— Oliver Wendell Holmes

Simplicity on the other side of complexity IS possible.

Complexity **does** cancel out if you don't keep stacking.

Technician



Engineer

Application



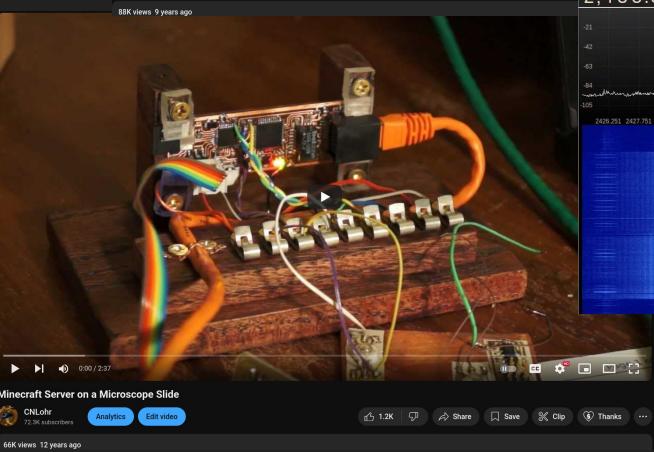
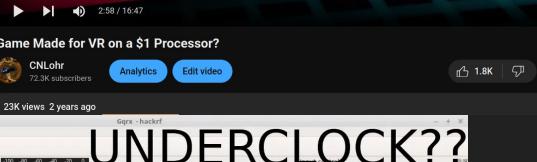
Crafting

Doing

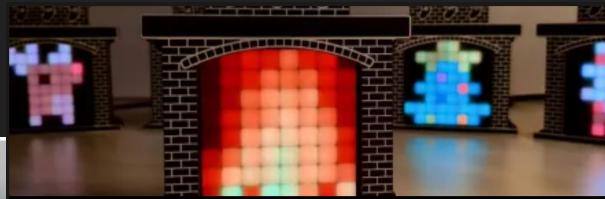
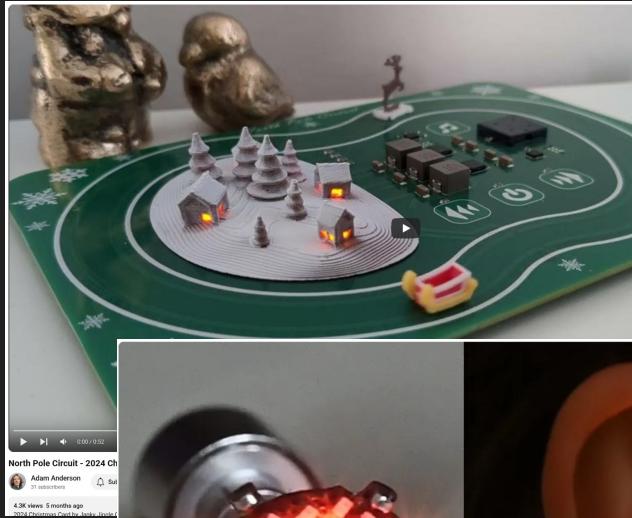


Understanding

# My journey



# It takes a village



# **AN ANIMATED LED FIREPLACE POWERED BY THE CH32V003**

January 2, 2024 by Tom Nardi

Once you've mastered the near can hold in your hand, it's only r time to produce a custom piece making one or two of something challenges. Not only will you wa aggravation, you'll likely want to

The fifty electronic fireplaces de Wheeler] are a perfect example year's gift exchange, we wouldn't more trees in 2024.

[Continue reading →](#)

Posted in **Holiday Hacks**, **LED H**  
Tagged **CH32V003**, **Christmas** **g**  
**led matrix**, **ws2812b**

## ADDING TEMPERATURE FUNCTIONALITY TO THE CH32V003 MCU

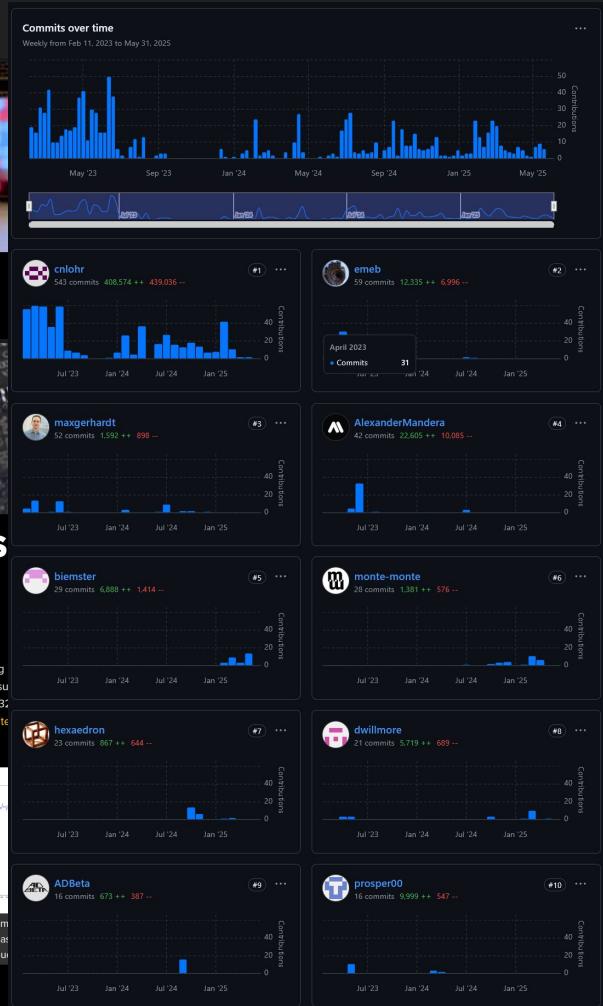
November 5, 2023 by Maya Posch

As cheap as the WCH CH32V003 MCU is, its approximately \$0.10 price tag when you need to start adding on external ICs for missing basic features, such as measurement. This is a feature that's commonly found on even basic STM32s as [eucalyptus] shows, you can improvise a working solution by [finding alternative](#) act as a thermometer.

The **CH32V003** is a low-end, 32-bit RISC-V-based MCU by the China-based Nanjing Qinhe Microelectronics, commonly known abbreviated as 'WCH', and featured on Hackaday previously. Although it features a single-core, 48 MHz CPU, its selection of peripherals is fairly basic:

So how do you create an internal temperature sensor using just this? [eucalyptus] figured that all that's needed is to measure the drift between two internal clocks – such as the LSI and HSI – as temperatures change and use this to calibrate

Unfortunately, the LSI isn't readily accessible, even through the Timer peripheral. This left the AWU (automatic wake-up unit) which also uses the LSI as a clock source. By letting it go to sleep and wake

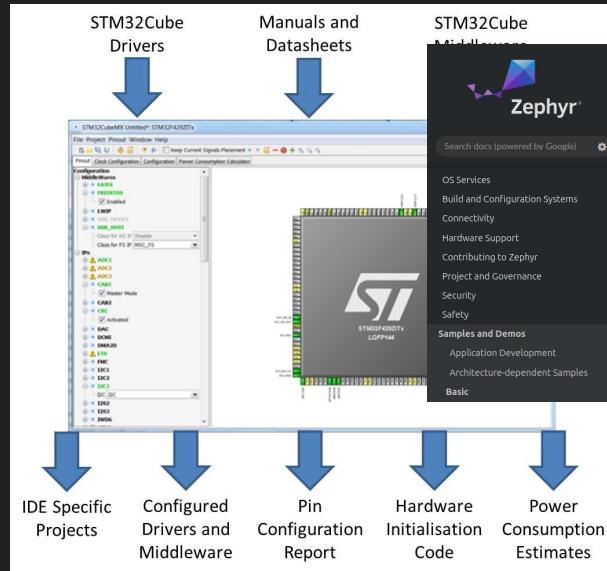


# And now

- ch32v203 & beyond
  - USB HS
  - Gigabit Ethernet
  - Bluetooth\*\*\*\*\* (iSLER™ only)
  - USB SS (not yet)
- Continued support

# So why?

- RISC-V is cool
- Dumb Luck
- Engineers are tired.



A screenshot of the Eclipse IDE showing two code editors side-by-side:

- Left Editor (personlist.component.ts):**

```
1 import { Component } from '@angular/core';
2 import { PersonService } from './person.service';
3 import { Person } from './person';
4
5 @Component({
6   selector: 'app-person-list',
7   templateUrl: './personlist.component.html',
8   styleUrls: []
9 })
10 export class PersonListComponent implements OnInit {
11   people: Person[] = [];
12   constructor(private personService: PersonService) {}
13   ngOnInit() {
14     this.personService.getAll().subscribe(p => this.people = p);
15   }
16 }
```
- Right Editor (AbstractFacade.java):**

```
1 package com.zephyr.core;
2
3 public abstract class AbstractFacade<T> {
4   private Class<T> entityClass;
5
6   public AbstractFacade(Class<T> entityClass) {
7     this.entityClass = entityClass;
8   }
9
10   protected abstract EntityManager getEntityManager();
11
12   public Response create(T entity) {
13     try {
14       if (!getEntityManager().getTransaction().isActive())
15         getEntityManager().getTransaction().begin();
16       getEntityManager().persist(entity);
17       EntityManager em = getEntityManager();
18       em.getTransaction().commit();
19       return Response.ok(em.get(entityClass).build());
20     } catch (Exception e) {
21       e.printStackTrace();
22     }
23   }
24 }
```



## Minimal footprint

[Browse source code on GitHub](#)

### Overview

This sample provides an empty `main()` and various configuration files that can be used to measure Zephyr minimal ROM footprint in different configurations.

The following configuration files are available:

- `mt.conf`: Enable multithreading
- `no-mt.conf`: Disable multithreading
- `no-preempt.conf`: Disable preemption
- `no-timers.conf`: Disable timers

A screenshot of a debugger interface showing assembly code and memory dump:

- Registers:** PC=0x00000000, R0=0x00000000, R1=0x00000000, R2=0x00000000, R3=0x00000000, R4=0x00000000, R5=0x00000000, R6=0x00000000, R7=0x00000000, R8=0x00000000, R9=0x00000000, R10=0x00000000, R11=0x00000000, R12=0x00000000, R13=0x00000000, R14=0x00000000, R15=0x00000000, R16=0x00000000, R17=0x00000000, R18=0x00000000, R19=0x00000000, R20=0x00000000, R21=0x00000000, R22=0x00000000, R23=0x00000000, R24=0x00000000, R25=0x00000000, R26=0x00000000, R27=0x00000000, R28=0x00000000, R29=0x00000000, R30=0x00000000, R31=0x00000000, R132=0x00000000, R133=0x00000000, R134=0x00000000, R135=0x00000000, R136=0x00000000, R137=0x00000000, R138=0x00000000, R139=0x00000000, R140=0x00000000, R141=0x00000000, R142=0x00000000, R143=0x00000000, R144=0x00000000, R145=0x00000000, R146=0x00000000, R147=0x00000000, R148=0x00000000, R149=0x00000000, R150=0x00000000, R151=0x00000000, R152=0x00000000, R153=0x00000000, R154=0x00000000, R155=0x00000000, R156=0x00000000, R157=0x00000000, R158=0x00000000, R159=0x00000000, R160=0x00000000, R161=0x00000000, R162=0x00000000, R163=0x00000000, R164=0x00000000, R165=0x00000000, R166=0x00000000, R167=0x00000000, R168=0x00000000, R169=0x00000000, R170=0x00000000, R171=0x00000000, R172=0x00000000, R173=0x00000000, R174=0x00000000, R175=0x00000000, R176=0x00000000, R177=0x00000000, R178=0x00000000, R179=0x00000000, R180=0x00000000, R181=0x00000000, R182=0x00000000, R183=0x00000000, R184=0x00000000, R185=0x00000000, R186=0x00000000, R187=0x00000000, R188=0x00000000, R189=0x00000000, R190=0x00000000, R191=0x00000000, R192=0x00000000, R193=0x00000000, R194=0x00000000, R195=0x00000000, R196=0x00000000, R197=0x00000000, R198=0x00000000, R199=0x00000000, R200=0x00000000, R201=0x00000000, R202=0x00000000, R203=0x00000000, R204=0x00000000, R205=0x00000000, R206=0x00000000, R207=0x00000000, R208=0x00000000, R209=0x00000000, R210=0x00000000, R211=0x00000000, R212=0x00000000, R213=0x00000000, R214=0x00000000, R215=0x00000000, R216=0x00000000, R217=0x00000000, R218=0x00000000, R219=0x00000000, R220=0x00000000, R221=0x00000000, R222=0x00000000, R223=0x00000000, R224=0x00000000, R225=0x00000000, R226=0x00000000, R227=0x00000000, R228=0x00000000, R229=0x00000000, R230=0x00000000, R231=0x00000000, R232=0x00000000, R233=0x00000000, R234=0x00000000, R235=0x00000000, R236=0x00000000, R237=0x00000000, R238=0x00000000, R239=0x00000000, R240=0x00000000, R241=0x00000000, R242=0x00000000, R243=0x00000000, R244=0x00000000, R245=0x00000000, R246=0x00000000, R247=0x00000000, R248=0x00000000, R249=0x00000000, R250=0x00000000, R251=0x00000000, R252=0x00000000, R253=0x00000000, R254=0x00000000, R255=0x00000000, R256=0x00000000, R257=0x00000000, R258=0x00000000, R259=0x00000000, R260=0x00000000, R261=0x00000000, R262=0x00000000, R263=0x00000000, R264=0x00000000, R265=0x00000000, R266=0x00000000, R267=0x00000000, R268=0x00000000, R269=0x00000000, R270=0x00000000, R271=0x00000000, R272=0x00000000, R273=0x00000000, R274=0x00000000, R275=0x00000000, R276=0x00000000, R277=0x00000000, R278=0x00000000, R279=0x00000000, R280=0x00000000, R281=0x00000000, R282=0x00000000, R283=0x00000000, R284=0x00000000, R285=0x00000000, R286=0x00000000, R287=0x00000000, R288=0x00000000, R289=0x00000000, R290=0x00000000, R291=0x00000000, R292=0x00000000, R293=0x00000000, R294=0x00000000, R295=0x00000000, R296=0x00000000, R297=0x00000000, R298=0x00000000, R299=0x00000000, R300=0x00000000, R301=0x00000000, R302=0x00000000, R303=0x00000000, R304=0x00000000, R305=0x00000000, R306=0x00000000, R307=0x00000000, R308=0x00000000, R309=0x00000000, R310=0x00000000, R311=0x00000000, R312=0x00000000, R313=0x00000000, R314=0x00000000, R315=0x00000000, R316=0x00000000, R317=0x00000000, R318=0x00000000, R319=0x00000000, R320=0x00000000, R321=0x00000000, R322=0x00000000, R323=0x00000000, R324=0x00000000, R325=0x00000000, R326=0x00000000, R327=0x00000000, R328=0x00000000, R329=0x00000000, R330=0x00000000, R331=0x00000000, R332=0x00000000, R333=0x00000000, R334=0x00000000, R335=0x00000000, R336=0x00000000, R337=0x00000000, R338=0x00000000, R339=0x00000000, R340=0x00000000, R341=0x00000000, R342=0x00000000, R343=0x00000000, R344=0x00000000, R345=0x00000000, R346=0x00000000, R347=0x00000000, R348=0x00000000, R349=0x00000000, R350=0x00000000, R351=0x00000000, R352=0x00000000, R353=0x00000000, R354=0x00000000, R355=0x00000000, R356=0x00000000, R357=0x00000000, R358=0x00000000, R359=0x00000000, R360=0x00000000, R361=0x00000000, R362=0x00000000, R363=0x00000000, R364=0x00000000, R365=0x00000000, R366=0x00000000, R367=0x00000000, R368=0x00000000, R369=0x00000000, R370=0x00000000, R371=0x00000000, R372=0x00000000, R373=0x00000000, R374=0x00000000, R375=0x00000000, R376=0x00000000, R377=0x00000000, R378=0x00000000, R379=0x00000000, R380=0x00000000, R381=0x00000000, R382=0x00000000, R383=0x00000000, R384=0x00000000, R385=0x00000000, R386=0x00000000, R387=0x00000000, R388=0x00000000, R389=0x00000000, R390=0x00000000, R391=0x00000000, R392=0x00000000, R393=0x00000000, R394=0x00000000, R395=0x00000000, R396=0x00000000, R397=0x00000000, R398=0x00000000, R399=0x00000000, R400=0x00000000, R401=0x00000000, R402=0x00000000, R403=0x00000000, R404=0x00000000, R405=0x00000000, R406=0x00000000, R407=0x00000000, R408=0x00000000, R409=0x00000000, R410=0x00000000, R411=0x00000000, R412=0x00000000, R413=0x00000000, R414=0x00000000, R415=0x00000000, R416=0x00000000, R417=0x00000000, R418=0x00000000, R419=0x00000000, R420=0x00000000, R421=0x00000000, R422=0x00000000, R423=0x00000000, R424=0x00000000, R425=0x00000000, R426=0x00000000, R427=0x00000000, R428=0x00000000, R429=0x00000000, R430=0x00000000, R431=0x00000000, R432=0x00000000, R433=0x00000000, R434=0x00000000, R435=0x00000000, R436=0x00000000, R437=0x00000000, R438=0x00000000, R439=0x00000000, R440=0x00000000, R441=0x00000000, R442=0x00000000, R443=0x00000000, R444=0x00000000, R445=0x00000000, R446=0x00000000, R447=0x00000000, R448=0x00000000, R449=0x00000000, R450=0x00000000, R451=0x00000000, R452=0x00000000, R453=0x00000000, R454=0x00000000, R455=0x00000000, R456=0x00000000, R457=0x00000000, R458=0x00000000, R459=0x00000000, R460=0x00000000, R461=0x00000000, R462=0x00000000, R463=0x00000000, R464=0x00000000, R465=0x00000000, R466=0x00000000, R467=0x00000000, R468=0x00000000, R469=0x00000000, R470=0x00000000, R471=0x00000000, R472=0x00000000, R473=0x00000000, R474=0x00000000, R475=0x00000000, R476=0x00000000, R477=0x00000000, R478=0x00000000, R479=0x00000000, R480=0x00000000, R481=0x00000000, R482=0x00000000, R483=0x00000000, R484=0x00000000, R485=0x00000000, R486=0x00000000, R487=0x00000000, R488=0x00000000, R489=0x00000000, R490=0x00000000, R491=0x00000000, R492=0x00000000, R493=0x00000000, R494=0x00000000, R495=0x00000000, R496=0x00000000, R497=0x00000000, R498=0x00000000, R499=0x00000000, R500=0x00000000, R501=0x00000000, R502=0x00000000, R503=0x00000000, R504=0x00000000, R505=0x00000000, R506=0x00000000, R507=0x00000000, R508=0x00000000, R509=0x00000000, R510=0x00000000, R511=0x00000000, R512=0x00000000, R513=0x00000000, R514=0x00000000, R515=0x00000000, R516=0x00000000, R517=0x00000000, R518=0x00000000, R519=0x00000000, R520=0x00000000, R521=0x00000000, R522=0x00000000, R523=0x00000000, R524=0x00000000, R525=0x00000000, R526=0x00000000, R527=0x00000000, R528=0x00000000, R529=0x00000000, R530=0x00000000, R531=0x00000000, R532=0x00000000, R533=0x00000000, R534=0x00000000, R535=0x00000000, R536=0x00000000, R537=0x00000000, R538=0x00000000, R539=0x00000000, R540=0x00000000, R541=0x00000000, R542=0x00000000, R543=0x00000000, R544=0x00000000, R545=0x00000000, R546=0x00000000, R547=0x00000000, R548=0x00000000, R549=0x00000000, R550=0x00000000, R551=0x00000000, R552=0x00000000, R553=0x00000000, R554=0x00000000, R555=0x00000000, R556=0x00000000, R557=0x00000000, R558=0x00000000, R559=0x00000000, R560=0x00000000, R561=0x00000000, R562=0x00000000, R563=0x00000000, R564=0x00000000, R565=0x00000000, R566=0x00000000, R567=0x00000000, R568=0x00000000, R569=0x00000000, R570=0x00000000, R571=0x00000000, R572=0x00000000, R573=0x00000000, R574=0x00000000, R575=0x00000000, R576=0x00000000, R577=0x00000000, R578=0x00000000, R579=0x00000000, R580=0x00000000, R581=0x00000000, R582=0x00000000, R583=0x00000000, R584=0x00000000, R585=0x00000000, R586=0x00000000, R587=0x00000000, R588=0x00000000, R589=0x00000000, R590=0x00000000, R591=0x00000000, R592=0x00000000, R593=0x00000000, R594=0x00000000, R595=0x00000000, R596=0x00000000, R597=0x00000000, R598=0x00000000, R599=0x00000000, R600=0x00000000, R601=0x00000000, R602=0x00000000, R603=0x00000000, R604=0x00000000, R605=0x00000000, R606=0x00000000, R607=0x00000000, R608=0x00000000, R609=0x00000000, R610=0x00000000, R611=0x00000000, R612=0x00000000, R613=0x00000000, R614=0x00000000, R615=0x00000000, R616=0x00000000, R617=0x00000000, R618=0x00000000, R619=0x00000000, R620=0x00000000, R621=0x00000000, R622=0x00000000, R623=0x00000000, R624=0x00000000, R625=0x00000000, R626=0x00000000, R627=0x00000000, R628=0x00000000, R629=0x00000000, R630=0x00000000, R631=0x00000000, R632=0x00000000, R633=0x00000000, R634=0x00000000, R635=0x00000000, R636=0x00000000, R637=0x00000000, R638=0x00000000, R639=0x00000000, R640=0x00000000, R641=0x00000000, R642=0x00000000, R643=0x00000000, R644=0x00000000, R645=0x00000000, R646=0x00000000, R647=0x00000000, R648=0x00000000, R649=0x00000000, R650=0x00000000, R651=0x00000000, R652=0x00000000, R653=0x00000000, R654=0x00000000, R655=0x00000000, R656=0x00000000, R657=0x00000000, R658=0x00000000, R659=0x00000000, R660=0x00000000, R661=0x00000000, R662=0x00000000, R663=0x00000000, R664=0x00000000, R665=0x00000000, R666=0x00000000, R667=0x00000000, R668=0x00000000, R669=0x00000000, R670=0x00000000, R671=0x00000000, R672=0x00000000, R673=0x00000000, R674=0x00000000, R675=0x00000000, R676=0x00000000, R677=0x00000000, R678=0x00000000, R679=0x00000000, R680=0x00000000, R681=0x00000000, R682=0x00000000, R683=0x00000000, R684=0x00000000, R685=0x00000000, R686=0x00000000, R687=0x00000000, R688=0x00000000, R689=0x00000000, R690=0x00000000, R691=0x00000000, R692=0x00000000, R693=0x00000000, R694=0x00000000, R695=0x00000000, R696=0x00000000, R697=0x00000000, R698=0x00000000, R699=0x00000000, R700=0x00000000, R701=0x00000000, R702=0x00000000, R703=0x00000000, R704=0x00000000, R705=0x00000000, R706=0x00000000, R707=0x00000000, R708=0x00000000, R709=0x00000000, R710=0x00000000, R711=0x00000000, R712=0x00000000, R713=0x00000000, R714=0x00000000, R715=0x00000000, R716=0x00000000, R717=0x00000000, R718=0x00000000, R719=0x00000000, R720=0x00000000, R721=0x00000000, R722=0x00000000, R723=0x00000000, R724=0x00000000, R725=0x00000000, R726=0x00000000, R727=0x00000000, R728=0x00000000, R729=0x00000000, R730=0x00000000, R731=0x00000000, R732=0x00000000, R733=0x00000000, R734=0x00000000, R735=0x00000000, R736=0x00000000, R737=0x00000000, R738=0x00000000, R739=0x00000000, R740=0x00000000, R741=0x00000000, R742=0x00000000, R743=0x00000000, R744=0x00000000, R745=0x00000000, R746=0x00000000, R747=0x00000000, R748=0x00000000, R749=0x00000000, R750=0x00000000, R751=0x00000000, R752=0x00000000, R753=0x00000000, R754=0x00000000, R755=0x00000000, R756=0x00000000, R757=0x00000000, R758=0x00000000, R759=0x00000000, R760=0x00000000, R761=0x00000000, R762=0x00000000, R763=0x00000000, R764=0x00000000, R765=0x00000000, R766=0x00000000, R767=0x00000000, R768=0x00000000, R769=0x00000000, R770=0x00000000, R771=0x00000000, R772=0x00000000, R773=0x00000000, R774=0x00000000, R775=0x00000000, R776=0x00000000, R777=0x00000000, R778=0x00000000, R779=0x00000000, R780=0x00000000, R781=0x00000000, R782=0x00000000, R783=0x00000000, R784=0x00000000, R785=0x00000000, R786=0x00000000, R787=0x00000000, R788=0x00000000, R789=0x00000000, R790=0x00000000, R791=0x00000000, R792=0x00000000, R793=0x00000000, R794=0x00000000, R795=0x00000000, R796=0x00000000, R797=0x00000000, R798=0x00000000, R799=0x00000000, R800=0x00000000, R801

# stm32f042fun

stm32f042\_fun Public

master 1 Branch 0 Tags

Go to file Add file Code

cnlohr Add an example of driving GPIOs with DMA from a timer. 3c4a26a · 4 years ago 50 Commits

another\_demo Another demo 5 years ago

dma\_gpio Add an example of driving GPIOs with DMA from a timer. 4 years ago

epaper-present Update firmware usb file. 5 years ago

hexipad\_32f042 add rev B - fix USB 5 years ago

multitool-attempt-at-can Move attempt at can away. 5 years ago

multitool bump 5 years ago

rf\_40\_6\_xmit Move attempt at can away. 5 years ago

stm32f042cXt6\_demo Fix off by one error in SPI 4 years ago

README.md Update README.md 5 years ago

README

## stm32f042\_fun

My fun repo for the STM32f042. Feel free to rifle through. All the projects are incomplete, but mostly work.

Check out my IO list: [https://docs.google.com/spreadsheets/d/1DyMCKiVg4tmwGgn1U\\_OD7giobow0eRoh-zjQ15YpQ/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1DyMCKiVg4tmwGgn1U_OD7giobow0eRoh-zjQ15YpQ/edit?usp=sharing)

About

My fun repo for the STM32f042

Readme Activity 8 stars 2 watching 1 fork

Releases No releases pub Create a new rel

Packages No packages put Publish your first

Languages C 69.9% HTML 2.6% Other 0.1%

Suggested v Based on your

MSBu project Build a

USB, ANDROID, and C

Tensegrity Lamp Running ColorChord on Android with USB in C

CNLoehr 72.3K subscribers Analytics Edit video

1.3K Share Save Thanks Clip

17K views 5 years ago Live chat replay



# Why didn't stm32f042fun work out?

STM32F042 - 2019

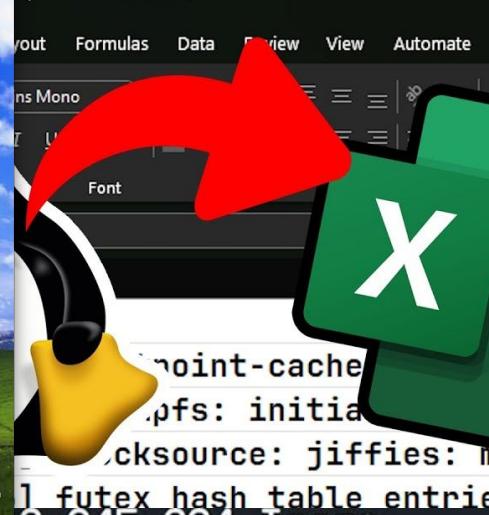
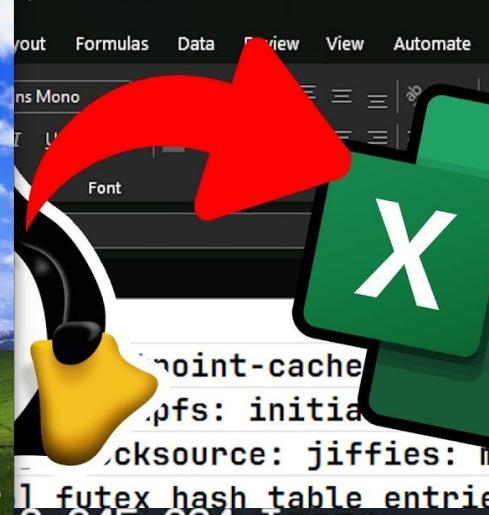
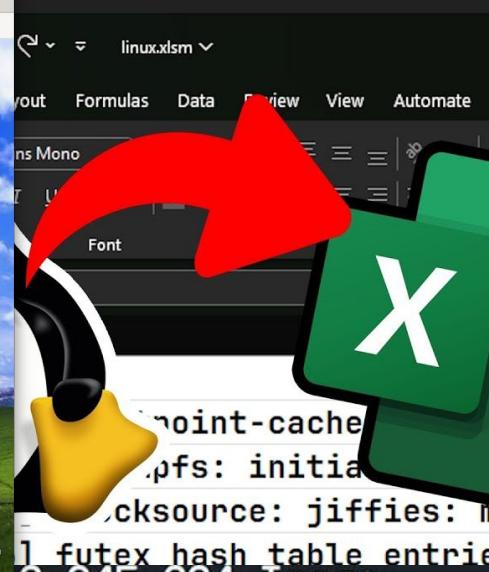
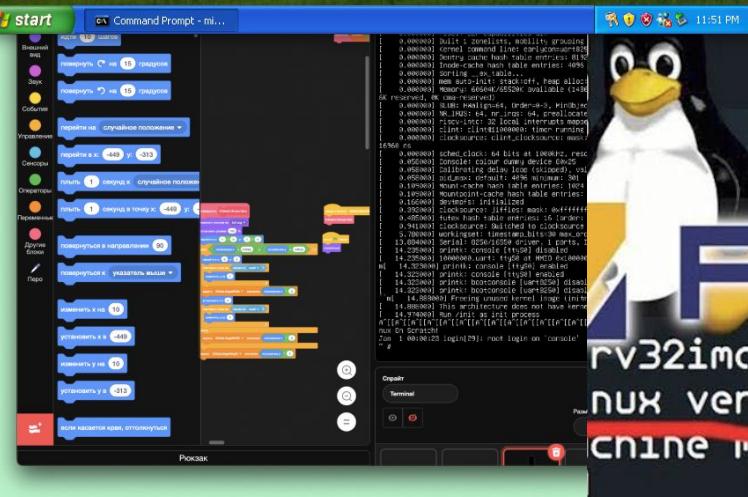
- 8 Stars, 1 Contributor
- \$0.50
- 48MHz
- Nice Micro
- Has USB
- 32kB Flash, 6kB RAM
- 7x7 QFN Smallest Package
- Requires 2 pins to program
- ARM

CH32V003 - 2023

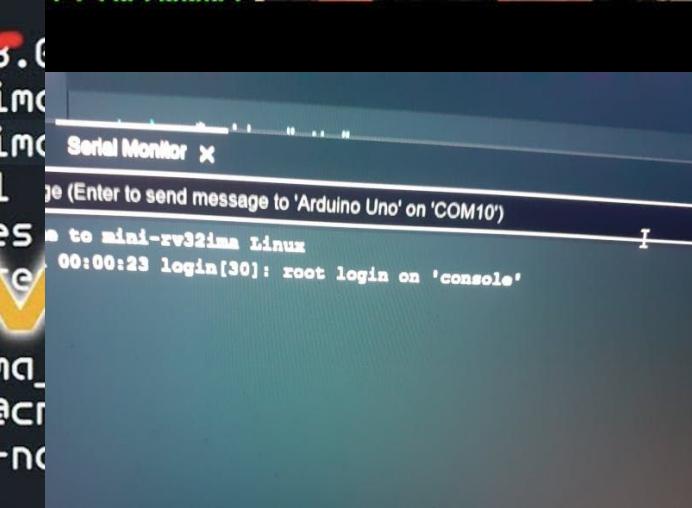
- 1.2k Stars, 86 Contributors
- <\$0.12
- 48MHz
- Nice micro
- No USB\*\*\*\*\*
- 16kB Flash, 2kB RAM
- 3x3 QFN Smallest Package
- 1 pin RVSWDIO!
- RISC-V

Machine View

```
files
Command Prompt - mini-rv32ima.exe -f Image
[ 0.012041] pid_max: default: 4096 minimum: 301
[ 0.020647] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
[ 0.025713] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
[ 0.048618] devtmpfs: initialized
[ 0.065808] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 191126044627500000 ns
[ 0.072957] futex hash table entries: 16 (order: -5, 192 bytes, linear)
[ 0.133515] cpu0: Ratio of byte access time to unaligned word access is 4.10,unaligned access rate favored by 1.00
[ 0.158514] clocksource: Switched to clocksource clint_clocksource
[ 0.288851] workingset: timestamp_hits=30 max_order=14 bucket_order=0
[ 0.500606] Serial: 8250/16550 driver, 1 ports, IRQ sharing disabled
[ 0.643127] printk: legacy console [ttyS0] disabled
[ 0.643127] printk: legacy console [uart0] at MMIO 0x00000000 <irq = 0, base_baud = 1048576> (0x1c41658)
[ 0.644904] printk: legacy console [ttyS0] enabled
[ 0.645984] printk: legacy console [ttyS0] enabled
[ 0.655172] printk: legacy bootconsole [uart8250] disabled
[ 0.655172] printk: legacy bootconsole [uart8250] disabled
[ 0.751004] clk: Disabling unused clocks
[ 0.770241] Freeing unused kernel image (initmem) memory: 800K
[ 0.772291] Run /init as init process
Welcome to mini-rv32ima Linux
Jan 1 00:00:01 login[22]: root login on 'console'
# uname -a
Linux buildroot 6.8.0-rc1mini-rv32ima #2 Mon Jan 22 17:50:35 EST 2024 riscv32 GN
# /bin/sh
~ #
```



RISC-V  
rv32ima\windows>mini-rv32ima\_<br/>nux version 5.18.0 (cnlohr@cn<br/>nine model. riscv-minimal-no<br/>2.945.224 Image





chner: main memory (which doubles as framebuffer)  
below half of it, R collects test output from others and draws it

RVC (RISC-V CPU emulator) by -pi-  
<http://github.com/pimlakosa/rvc>



```

0.000000 Initmem setup node 0 mem 0x0000000000000000-0x0000000000ffff
0.000000 riscv: base ISR extensions aim
0.000000 riscv: ELF capabilities aim
0.000000 Kernel command line: sor14con=uartH250_mmc.0x10000000 10000000 console=tts5
0.000000 Dentry cache hash table entries: 2048 (order: 1, 8192 bytes, linear)
0.000000 Inode-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
0.000000 Built 1 zonelists, Mobility grouping off. Total pages: 4060
0.000000 mem auto-init: stack-off, heap elloc-off, heap free-off
0.000000 Memory: 13644MHz/16368K available (1226K kernel code, 286K rwdtext, 131K rodata, 793K init, 928K bss, 2724K v
reserved, 0K dma-reserved)
0.000000 SLUB: Huwain=54, Order=0-1, MinObjects=0, CPUs=1, Nodes=1
0.000000 NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
0.000000 riscv-int: 32 local interrupts mapped
0.000000 clint: clint 11000000: timer running at 1000000 Hz
0.000000 clocksource: clint_clocksource: mask: 0xffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3586361616
560 ns
0.000000 sched_clock: 64 bits of 1000MHz, resolution 1000ns, wraps every 2159023255500ns
0.111982 Console: colour dummy device 0x0E25
0.150163 Calibrating delay loop (skipped), value calculated using timer frequency: 2.00 BaseMIPS (IPU=10000)
0.222482 pid.hox: default: 4096 Minimum: 301
0.375982 Mount-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
0.333130 Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
0.771195 devtmpfs: initialized
1.088857 clocksource: diffies: mask: 0xffffffffffff max_cycles: 0xffffffffffff, max_idle_ns: 19112604462750000 ns
1.155324 future hash table entries: 16 (order: -5, 192 bytes, linear)
1.782062 clocksource: Switched to clocksource clint_clocksource
4.946250 workqueue: timestamp_burst=30 KQD_order=2 bucket_order=0
15.457057 serial: 0x50/16550 driver, 1 ports, I/O sharing disabled
15.735796 printk: console ttys0 disabled
15.879441 10000000 wort: ttys0 at MMIO 0x10000000 (irq = 0, base.boud = 1048576) is a XRI6850
15.954212 printk: console ttys0 enabled
15.954212 printk: console ttys0 enabled
16.020297 printk: bootconsole wortttys0 disabled
16.020297 printk: bootconsole wortttys0 disabled
16.095256 clk: Disabling unused clocks
17.987032 Freeing unused kernel image (initmem) memory: 792K
18.035159 This architecture does not have kernel memory protection.
18.086962 Run /init as init process

```



x0:00000000	ra:8061169c	0
sp:80686720	sp:8065fc80	s
tp:00000000	tp:8067274c	s' L
t1:0ccccccc	t2:00000000	x
s0:80686870	hhp:s1:00000078	x
a0:00000000	o1:00000030	0
a2:00000000	o3:806779ab	94
o4:80674bcf	o4:00000004	
o6:7fffffff	o7:00000000	
s2:00000037	7:s3:00000000	
s4:0000002e	s5:00000030	0
s6:0000006d	M:s7:80688d70	h P
s8:806882298	s8:806886b0c	hk
s10:00000000	s11:00000000	
t3:0ccccccc	t4:00000020	x
t5:00000025	t6:00000024	s
pc:80611ad0	MST:00001888	
CYC:01ba6890	h CYCH:00000000	
tim:0165r2f2	g timh:00000000	
tmh:0165r3e1	g tmh:00000000	
SCY:80465140	F0 HTU:00001e88	
HIE:00000088	HIP:00000000	
HGP:806014bc	HTU0:806014bb	
HGO:00000008	RXT:012b6300	+C
ste:00000000	F00:00000000	
F00:090e8000	F00:00000000	

# Philosophies

- Zero\*- or ultra-low cost abstractions only.
- Abstractions should not hide the hardware, just make code look less verbose.
- If you can just do something in an interrupt, do it
  - tinyusb is not so tiny
  - Much higher performance on many modes (HIDAPI feature reports)
- No hugely complicated systems.

\*<https://www.youtube.com/watch?v=rHIkrotSwcc>

CppCon 2019: Chandler Carruth “There Are No Zero-cost Abstractions”

# What makes a minimal SDK?

- Programming
- Compiler
- “SFR” header
- Linker Scripts
- Startup Code
- Interrupts
- Build System
- gdb
- “Libc” / Printf, IO on/off

The screenshot shows a GitHub repository page for 'ch570\_minimal'. The repository has 1 branch, 0 tags, and 3 commits by 'biemster'. The files listed are CH572SFR.h, Makefile, README.md, ch570\_config.py, ch570\_wchlink.py, linker.ld, minimal.c, and startup.CH572.S. The README contains information about the repository, mentioning it's a WCH CH570 minimal example without a full SDK. It describes how the C source code flashes an LED and sleeps. The 'compile' section notes that GCC riscv64-e1f is tested, and WCH MounRiver Studio should work too. The 'flash' section indicates that ch570\_config.py and ch570\_wchlink.py are still in progress. A note at the bottom says chprog.py from https://github.com/wagiminator/MCU-Flash-Tools can be used to flash minimal.bin by adding {'name': 'CH570', 'id': 0x1370, 'code\_size': 245760, 'data\_size': 0} to the devices list. The repository has 1 star, 0 forks, and 0 releases published. The language distribution chart shows C at 80.5%, Python at 11.6%, Assembly at 3.4%, Linker Script at 2.9%, and Makefile at 1.6%.

# Programming

- Physical Interface
- Register Access
- PROGBUF
- Reading/Writing RAM
- Writing to Flash
- Semihosting
- GDB

Table 7-1 Debug module register List

Name	Access address	Description
data0	0x04	Data register 0, can be used for temporary storage of data
data1	0x05	Data register 1, can be used for temporary storage of data
dmcontrol	0x10	Debug module control register
dmstatus	0x11	Debug module status register
hartinfo	0x12	Microprocessor status register
abstractcs	0x16	Abstract command status register
command	0x17	Abstract command register
abstractauto	0x18	Abstract command auto-execution
proobuf0-7	0x20-0x27	Instruction cache registers 0-7
haltsum0	0x40	Pause status register

# Compiler

- Host support
  - Windows, Linux, Mac
- Client support
  - ARM, 32-, 64-
  - RISC-V, 32-, 64-
  - arm-none-eabi
  - riscv32-linux-gnu
- Special features
  - xw, call0, etc.
- GCC Versions
- Xpack Built-In, SysUtils

## Installation

cnlohr edited this page on Mar 18 · 23 revisions

In general, for minichlink, the flashing/debugging utility, you can use the pre-compiled minichlink or go to minichlink dir and `make` it.

### Debian and WSL

```
apt-get install build-essential libnewlib-dev gcc-riscv64-unknown-elf libusb-1.0-0-dev libudev-dev gdb-multiarch
```

Note that `gdb-multiarch` is only needed/useful if you want to use `gdb`. You can still semihost/debug printf without `gdb`.

### Arch/Manjaro

```
sudo pacman -S base-devel
```

### Windows

#### ⓘ Note

Run the powershell script, `install_xpack_gcc.ps1` from the `misc/` folder. This will install a well tested version of GCC14. Alternatively you may use the much smaller sysutils install (to system) [this copy of GCC10..](#) The `minichlink.exe` file is already ready to go in the `minichlink` folder. It requires Microsoft Visual C++ Redistributable to be installed.

If you are on a system  
the `plugdev` group ar

In Windows, if you want to use minichlink with the LinkE, you will need to use [Zadig](#) to install WinUSB to the WCH-Link interface 0. Or, you can right-click on "WCH-Link\_(Interface\_0).inf" and say install in your [drivers\\_for\\_WCH-LinkE](#) folder (in `misc`).

```
sudo groupadd plug  
sudo usermod -aG plug
```

### MacOS

More info can be found

install the RISC-V toolchain with homebrew following [these instructions](#)

To get `minichlink`, please go to the `minichlink` directory and `make` it or you can use the pre-compiled `minichlink`.

You might need to install `pkg-config` using `brew install pkg-config` to compile `minichlink`.

After installation you can move on to [Building and Flashing](#)



<https://thephd.dev/binary-banshees-digital-demons-abi-c-c++-help-me-god-please>

# Not this one, though!

5-bit Encoding (rx)	3-bit Compressed Encoding (rx')	Register	ABI Name	Description	Saved by Calle-
0	-	x0	zero	hardwired zero	-
1	-	x1	ra	return address	-R
2	-	x2	sp	stack pointer	-E
3	-	x3	gp	global pointer	-
4	-	x4	tp	thread pointer	-
5	-	x5	t0	temporary register 0	-R
6	-	x6	t1	temporary register 1	-R
7	<b>RV32E</b>	x7	t2	temporary register 2	-R
8		x8	s0 / fp	saved register 0 / frame pointer	-E
9	1	x9	s1	saved register 1	-E
10	2	x10	a0	function argument 0 / return value 0	-R
11	3	<b>RV32C</b>	a1	function argument 1 / return value 1	-R
12	4		a2	function argument 2	-R
13	5	x13	a3	function argument 3	-R
14	6	x14	a4	function argument 4	-R
15	7	x15	a5	function argument 5	-R
16	<b>RV32</b>	-	a6	function argument 6	-R
17	-	x17	a7	function argument 7	-R

<https://en.wikichip.org/wiki/risc-v/registers>

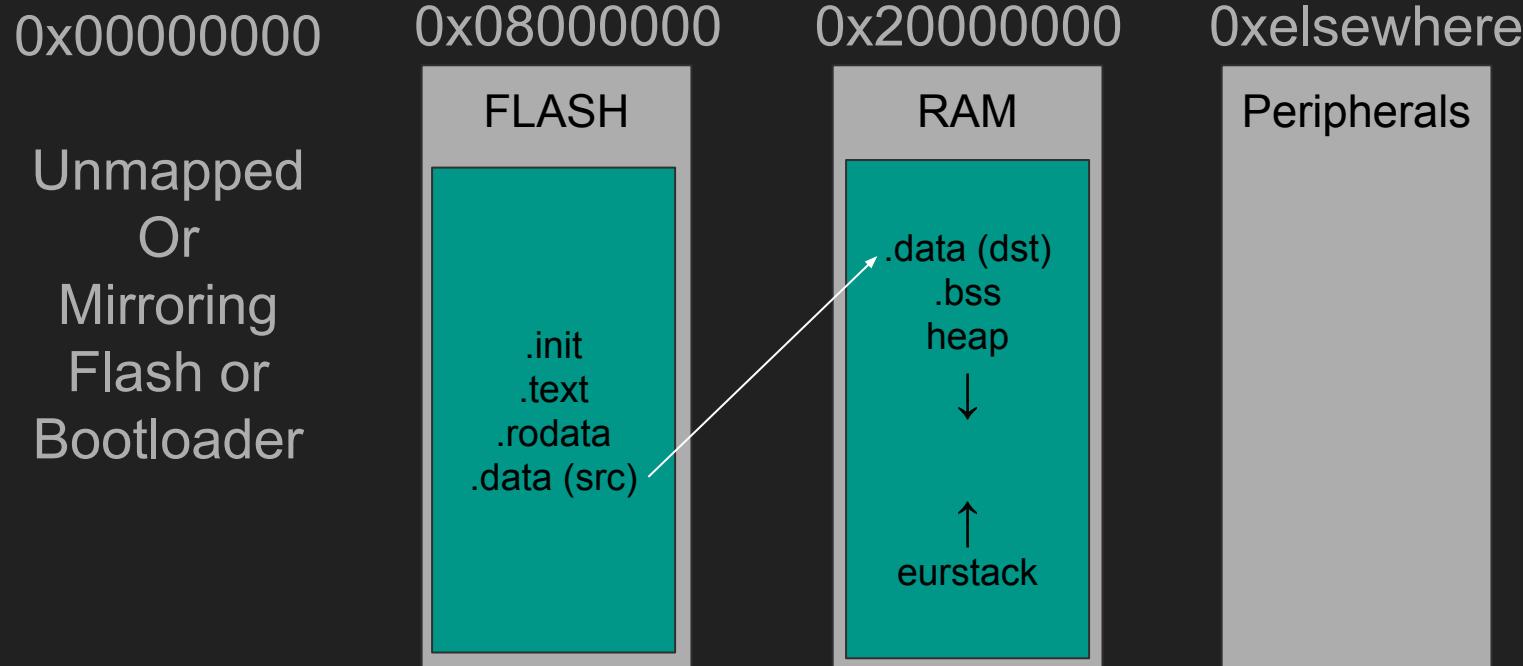
# ABI

```
__attribute__((noinline))
int multiply( int a, int b )
{
    return a * b;
}
int MultiplyAndAdd()
{
    int a = 5;
    int b = 6;
    int c = multiply( a, b );
    return c + 7;
}
```

```
multiply(int, int):
    mul    a0,a0,a1
    ret

MultiplyAndAdd():
    addi   sp,sp,-16
    li     a1,6
    li     a0,5
    sw    ra,12(sp)
    call   multiply(int, int)
    lw    ra,12(sp)
    addi  a0,a0,7
    addi  sp,sp,16
    jr    ra
```

# Memory Map



# Memory Types

- `.init` = Code that must be at beginning of flash
- `.text` = Regular code
  - `int MyFunction()`
- `.rodata` = Read-only data, can stay in flash
  - `const int MyConstInt = 37;`
- `.data` = Data that is copied into RAM, read, write, w/ default.
  - `int MyData = 45;`
- `.bss` = Zeroed-by-default RAM
  - `static int SomeGlobal; // Initialized to 0 by default.`
- Heap = Where malloc'd values go (we don't have malloc)
- Stack = Local variables, call stacks, etc.

# Linker Scripts

- Where is the memory
- What are sections
- Where does the memory go?

# Linker Scripts

```
ENTRY( InterruptVector )
MEMORY
{
    FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 16k
    RAM (xrw)   : ORIGIN = 0x20000000, LENGTH = 2k
}
SECTIONS
{
```

# Linker Scripts

```
.init :  
{  
    . = ALIGN(4);  
    KEEP(*(SORT_NONE(.init)))  
    . = ALIGN(4);  
} >FLASH AT>FLASH
```

# Linker Scripts

```
.text :  
{  
    . = ALIGN(4);  
    *(.text)  
    *(.text.* )  
    *(.rodata)  
    *(.rodata*)  
    . = ALIGN(4);  
} >FLASH AT>FLASH
```

# Linker Scripts

```
.data :
{
    . = ALIGN(4);
    __global_pointer$ = . + 0x3fc;
    *(.data .data.*)
    . = ALIGN(8);
    *(.srodata .srodata.*)
    . = ALIGN(4);
    PROVIDE( _edata = .);
} >RAM AT>FLASH
```

# Linker Scripts

```
.bss :
{
    . = ALIGN(4);
    PROVIDE( _sbss = .);
    *(.sbss*)
    *(.bss*)
    *(COMMON*)
    . = ALIGN(4);
    PROVIDE( _ebss = .);
} >RAM AT>FLASH?
PROVIDE( _end = _ebss);
PROVIDE( end = . );
PROVIDE( _eusrstack = ORIGIN(RAM) + LENGTH(RAM));
}
```

# Startup code

```
.global InterruptVector
.section .init
InterruptVector:
    .option    push
    .option    norvc
    j        _start
    .option    pop
    .word     0
    .word     NMI_Handler
    .word     HardFault_Handler
    .word     0
...
    .word     EXTI7_0_IRQHandler
    //EXTI Line 7..0
```

```
_start:           // a0 = hart ID, a1 = DTB
                  .option push
                  .option norelax
                  // Load gp for faster indexing
                  la gp, __global_pointer$
                  .option pop
                  // Machine Mode
                  li a3, 0x1880
                  csrw mstatus, a3
                  // Setup stack
                  la sp, _eusrstack
```

# Startup code

```
// Clear RAM to zeroes
    la  a3, _sbss
    la  a4, _ebss
    li  a5, 0
    bge a3, a4, 2f
1:  sw  a5, 0(a3)
    addi a3, a3, 4
    blt a3, a4, 1b
2:
```

```
// Load .data from FLASH to RAM
    la  a3, _data_lma
    la  a4, _data_vma
    la  a5, _edata
1:  beq a4, a5, 2f
    lw   a7, 0(a3)
    sw  a7, 0(a4)
    addi a3, a3, 4
    addi a4, a4, 4
    bne a4, a5, 1b
2:
```

# Startup code

```
    li a3, 0x3
    la a0, InterruptVector
    or a0, a0, a3
    csrw mtvec, a0
```

```
    la a4, main
    csrw mepc, a4
    mret
```

# Interrupts

- “Pointers” in the interrupt vector.
- Usually decorate functions in user space.

```
void EXTI7_0_IRQHandler( void ) __attribute__((interrupt));
void EXTI7_0_IRQHandler( void )
{
    // Flash just a little blip.
    funDigitalWrite( PC1, FUN_HIGH );
    funDigitalWrite( PC1, FUN_LOW );

    // Acknowledge the interrupt
    EXTI->INTFR = EXTI_Line3;
}
```

# Interrupts

- Indexes into interrupt vector

```
typedef enum IRQn
{
    // 2 Non Maskable Interrupt
    NonMaskableInt_IRQn = 2,
    // 3 Exception Interrupt
    EXC_IRQn = 3,
    ...
    // External Line[7:0] Interrupts
    EXTI7_0_IRQn = 20,
} IRQn_Type;
```

```
NVIC_EnableIRQ( EXTI7_0_IRQn );
```

# Interrupts

```
.global InterruptVector
.section .init
InterruptVector:
    .option    push
    .option    norvc
    j         _start
    .option    pop
    .word     0
    .word     NMI_Handler
    .word     HardFault_Handler
    .word     0
...
    .word     EXTI7_0_IRQHandler
    //EXTI Line 7..0
...
```

00000000 <InterruptVector>:

0:	0f40006f	j f4 <handle_reset>
4:	00000000	.word 0x00000000
8:	000001b0	.word 0x000001b0 << NMI_Handler
c:	00000164	.word 0x00000164 << Hard Fault Handler
...		
30:	00000164	.word 0x00000164
34:	00000000	.word 0x00000000
38:	00000164	.word 0x00000164
3c:	00000000	.word 0x00000000
40:	00000164	.word 0x00000164
44:	00000164	.word 0x00000164
48:	00000164	.word 0x00000164
4c:	00000164	.word 0x00000164
50:	000002c0	.word 0x000002c0 << Pin Change IRQ
54:	00000164	.word 0x00000164
58:	00000164	.word 0x00000164

# Build System

<https://makefiletutorial.com/>

- Make

**PARAMETER** := thing

```
thing_you_want : things_needed_to_make_what_you_want  
How_to_make_the_thing $(PARAMETER)
```

# Build System

```
all : flash

TARGET:=bootload
LINKER_SCRIPT:=../../ch32fun/ch32v003fun-bootloader.ld
WRITE_SECTION:=bootloader
FLASH_COMMAND:=../../minichlink/minichlink -a -U -w $(TARGET).bin $(WRITE_SECTION) -B
TARGET MCU:=CH32V003
include ../../ch32fun/ch32fun.mk ←

optionbytes :
$(MINICHLINK)/minichlink -w +a55aff00 option # Enable bootloader, disable RESET

flash : cv_flash
clean : cv_clean
```

# Build System (Assembly Listing)

```
$(TARGET).bin : $(TARGET).elf
    $(PREFIX)-objdump -S $^ > $(TARGET).lst
    $(PREFIX)-objcopy $(OBJCOPY_FLAGS) -O binary $< $(TARGET).bin
    $(PREFIX)-objcopy -O ihex $< $(TARGET).hex
```

```
        funDigitalWrite( PD0, 1 );
78a:  4785          li     a5,1
78c:  c81c          sw     a5,16(s0)
        funDigitalWrite( PC0, 1 );
78e:  40011737      lui    a4,0x40011
792:  cb1c          sw     a5,16(a4)
        printf( "+%lu\n", count++ );
794:  c081a583      lw     a1,-1016(gp) # 20000004 <count>
798:  6505          lui    a0,0x1
79a:  82c50513      addi   a0,a0,-2004 # 82c <main+0xe8>
79e:  00158713      addi   a4,a1,1
7a2:  c0e1a423      sw     a4,-1016(gp) # 20000004 <count>
7a6:  35e5          jal    68e <printf>
```

# Special Functions Registers Header

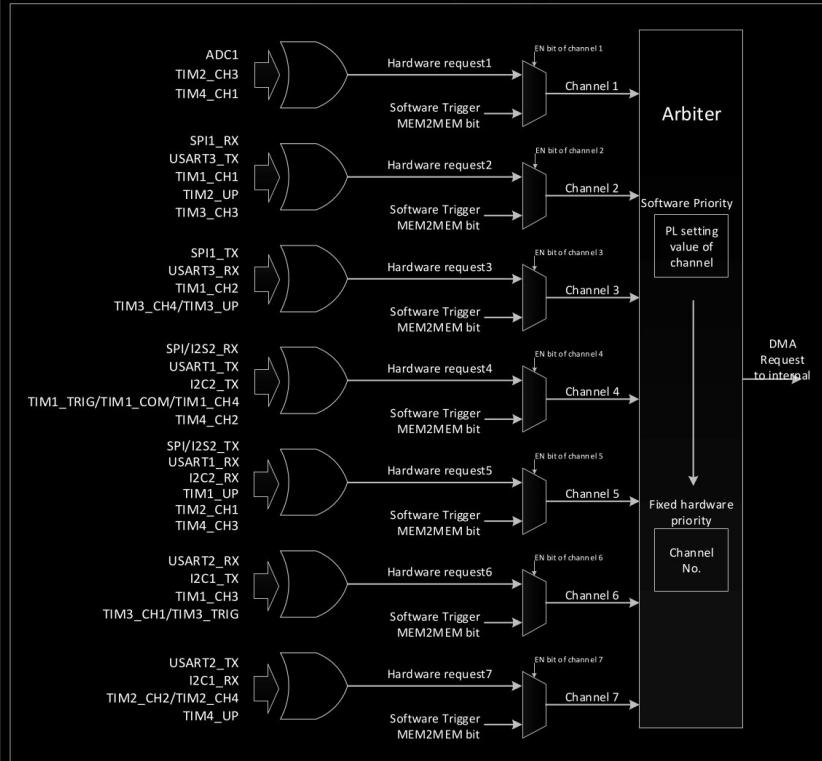
```
typedef struct
{
    __IO uint32_t CFGLR;      /* Port Configuration Register Low */
    __IO uint32_t CFGHR;      /* Port Configuration Register High */
    __I  uint32_t INDR;       /* Port Input Data Register */
    __IO uint32_t OUTDR;      /* Port Output Data Register */
    __IO uint32_t BSHR;       /* Port Set/Reset Register */
    __IO uint32_t BCR;        /* Port Reset Register */
    __IO uint32_t LCKR;       /* Port Configuration Lock Register */
} GPIO_TypeDef;
```

```
#define GPIOA_BASE 0x40010800
#define GPIOA_BASE 0x40010800
#define GPIOA_BASE 0x40010800
#define GPIOA ((GPIO_TypeDef *)GPIOA_BASE)
#define GPIOC ((GPIO_TypeDef *)GPIOC_BASE)
#define GPIOD ((GPIO_TypeDef *)GPIOD_BASE)
```

GPIOA->BSHR = 1<<2

# Technical Reference Manual

Figure 11-1 DMA1 request mapping



# Fast IO Abstractions

- You can use funDigitalWrite() and it is as fast as the raw register code.

```
#define FUN_HIGH 0x1
```

```
#define GpioOf( pin ) \  
((GPIO_TypeDef *) (GPIOA_BASE + 0x400 * ((pin)>>4)))
```

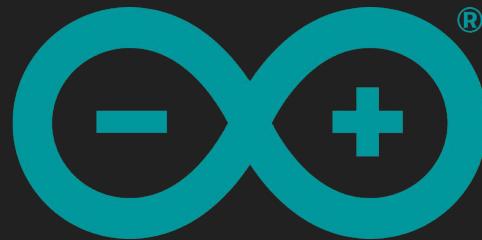
```
#define funDigitalWrite( pin, value ) \  
{ GpioOf( pin )->BSHR = 1<<((!(value))*16 + ((pin) & 0xf)); }
```

```
#define PC5 0x35
```

```
#define PC6 0x36
```

# Other IO Abstractions

```
digitalWrite:  
    lui    a5,%hi(.LANCHOR0)      .L3:  
    slli   a0,a0,5  
    addi   a5,a5,%lo(.LANCHOR0)  
    add    a5,a5,a0  
    lw     a3,8(a5)  
    li     a4,-1  
    beq   a3,a4,.L1  
    lw     a3,0(a5)  
    lw     a0,4(a5)  
    lui   a5,%hi(PORT)  
    lw     a4,%lo(PORT)(a5)  
    li     a5,84  
    mul   a5,a3,a5  
    li     a2,1  
    sll   a2,a2,a0  
    add   a5,a4,a5  
    lw     a5,0(a5)  
    and   a5,a2,a5  
    bne   a5,zero,.L3  
    li     a5,21  
    mul   a5,a3,a5  
    add   a5,a5,a0  
    slli  a5,a5,2  
    add   a5,a4,a5  
    snez  a0,a1  
    sw    a0,12(a5)  
  
                                .L4:  
                                li    a5,84  
                                mul  a3,a3,a5  
                                add  a4,a4,a3  
                                bne a1,zero,.L4  
                                sw   a2,76(a4)  
                                ret  
  
                                .L1:  
                                sw   a2,80(a4)  
                                ret  
  
gpiotest:  
    addi  sp,sp,-16  
    li    a1,1  
    li    a0,4  
    sw   ra,12(sp)  
    call  digitalWrite  
    li    a0,5  
    li    a1,1  
    call  digitalWrite  
    lw    ra,12(sp)  
    addi  sp,sp,16  
    jr    ra  
.set  .LANCHOR0,. + 0
```



# Fast IO Abstractions

```
funDigitalWrite( PC5, 1);  
funDigitalWrite( PC6, 1);
```

gpiotest:

```
    li      a5, 0x40011000  
    li      a4, 32  
    sw      a4, 16(a5)  
    li      a4, 64  
    sw      a4, 16(a5)  
    ret
```



**COMPILER  
EXPLORER**

...or lst files

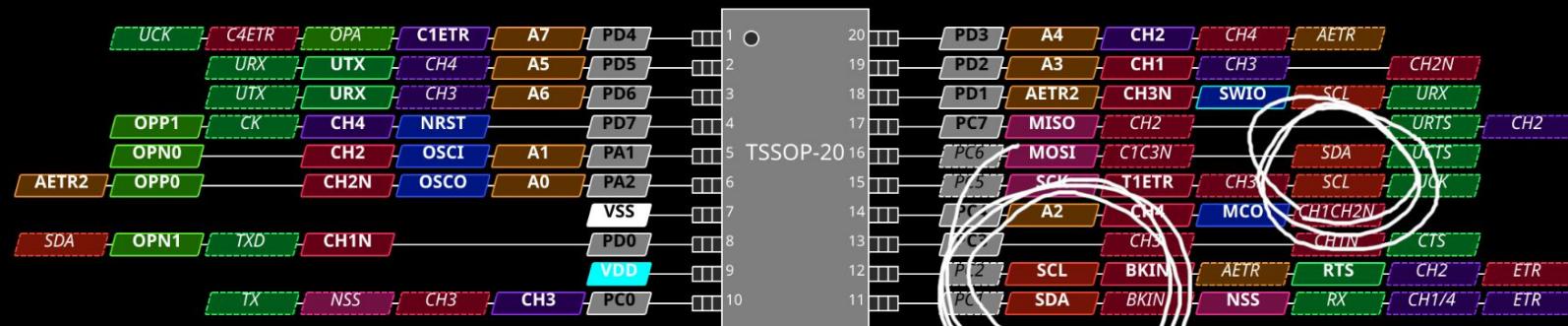
# “But why does it matter”

- Stops rationale for mediocrity
- Bit banging is valid.
- Reduces bloat
- static\_i2c.h

# “But why does it matter”

CH32V003F4P6

TSSOP-20 4.4x6.5mm  
0.65mm Pitch



UART  
I2C  
SPI  
ADC

SYS  
TIM1  
TIM2  
OPA

PIN NAME  
SWD  
VSS  
VDD

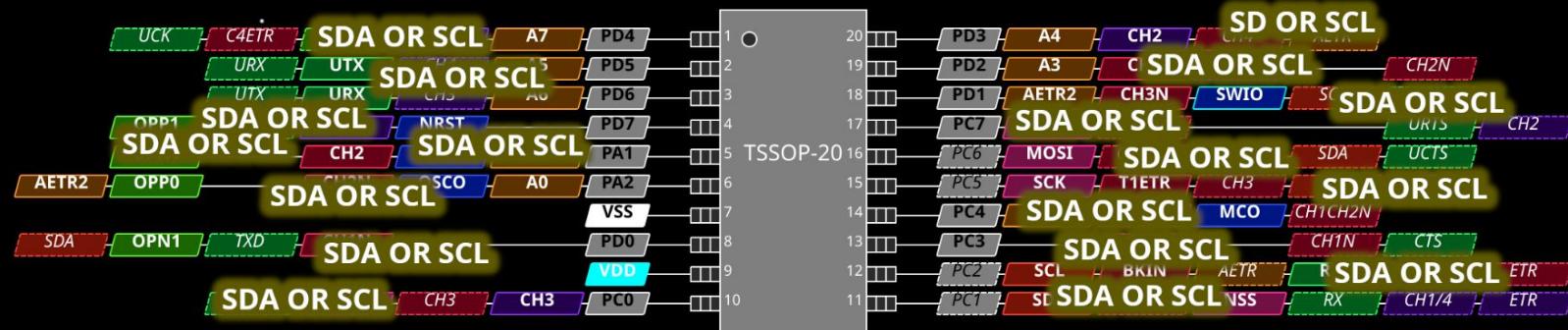
Default Function  
Alternate Function

NOTE:  
If VDD is 3.3V: PC1, PC2, PC5 and PC6 are 5V tolerant

# “But why does it matter”

**CH32V003F4P6**

TSSOP-20 4.4x6.5mm  
0.65mm Pitch



UART	UART
I2C	I2C
SPI	SPI
ADC	ADC

SYS	SYSTEM
TIM1	TIMER 1
TIM2	TIMER 2
OPA	OP AMP

PIN	PIN NAME
SWD	SWIO
VSS	GND
VDD	3.3V/5V

FUNC	Default Function
FUNC	Alternate Function

NOTE:  
If VDD is 3.3V: PC1, PC2, PC5 and PC6 are 5V tolerant

“But why I2C



# SysTick

- Wrap-around issues
- Scheduling issues
- How can I delay if I want to use the systick interrupt?

```
uint32_t targend = SysTick->CNT + n;  
while( ((int32_t)( SysTick->CNT - targend )) < 0 );
```

AVOID altering/resetting SysTick->CNT

- Does not work on some ARM chips.

# How does GDB work?

- Halt Processor
- Read/Write Register+RAM
- Breakpoints: Write ebreak to Flash
- Need some basic step/continue/pause
- GDB Protocol is from 1989 as best as I can tell.
- Works well over serial\* or socket\*\*
- Needs gdb-multiarch

\*-picorvd <https://github.com/aappleby/picorvd>

\*\*-minichlink

# How does GDB work?

<https://github.com/RinHizakura/mini-gdbstub>

```
struct target_ops {
    gdb_action_t (*cont)(void *args);
    gdb_action_t (*stepi)(void *args);
    size_t (*get_reg_bytes)(int regno);
    int (*read_reg)(void *args, int regno, void *value);
    int (*write_reg)(void *args, int regno, void* value);
    int (*read_mem)(void *args, size_t addr, size_t len, void *val);
    int (*write_mem)(void *args, size_t addr, size_t len, void *val);
    bool (*set_bp)(void *args, size_t addr, bp_type_t type);
    bool (*del_bp)(void *args, size_t addr, bp_type_t type);
    void (*on_interrupt)(void *args);
    void (*set_cpu)(void *args, int cpuid);
    int (*get_cpu)(void *args);
};
```

# Libc - like

- `memcpy`
- `memset`
- `strstr`
- `rand`
- ...

## mini-printf

Minimal printf() implementation for embedded projects.

### Motivation

I was recently working on an embedded project with a STM32 MCU. The chip had 32kB of flash memory - that's heaps for a microcontroller! How surprised I was when the linker suddenly failed saying that the program is too big and won't fit! How come?!

It's just some USB, I2C, GPIO, a few timers ... and snprintf() It turned out the memory hog was indeed the glibc's snprintf() - it took nearly 24kB out of my 32kB and left very little for my program.

Now what? I looked around the internet for some stripped down printf() implementations but none I really liked. Then I decided to develop my own minimal snprintf().

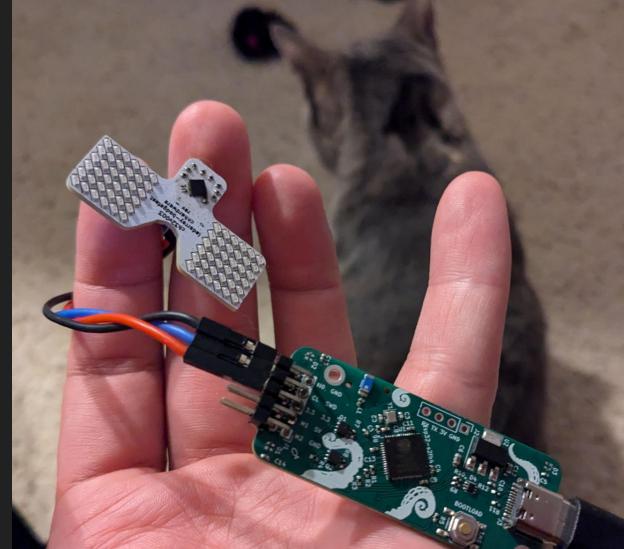
Here are some numbers (.bin file size of my STM32 project):

no snprintf():	10768 bytes
mini snprintf():	11420 bytes (+ 652 bytes)
glibc snprintf():	34860 bytes (+24092 bytes!!)



# Semihosting

- Printf over SWD
- On ARM, kinda slow
- On RV32, kinda fast
- Save I/O pins
- Program, Debug, Printf
- make clean all monitor



```
make monitor | pv > /dev/null
Found ESP32S2 Programmer
536KiB 0:00:15 [36.7KiB/s] [                ]
```

<=>

]

# Make clean all monitor



# Even if...

ch32fun isn't useful to you, I hope it can stand as an approachable testament to help teach how to make your own.

# You!

- Don't make computers match someone else's intuition
- Changes your brain to internalize reality
- Changes your intuition

Technician



Engineer

Application



Crafting

Doing



Understanding