TEARDOWN 2025

HAL no

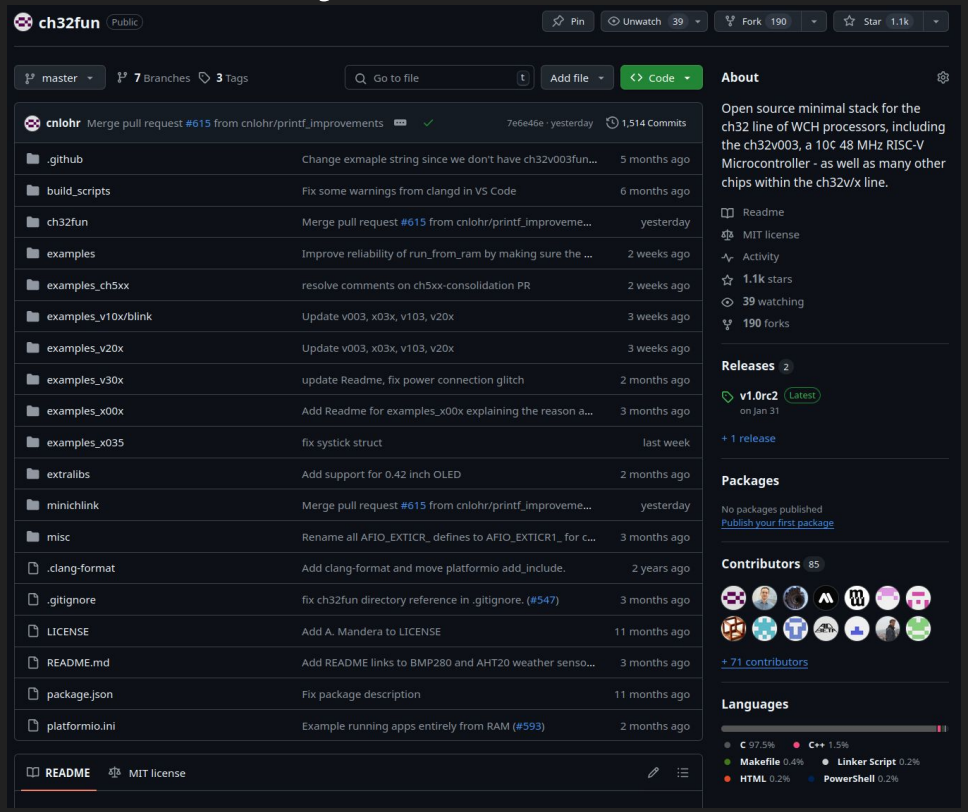
The story of ch32v003fun
And how you can make your own SDK

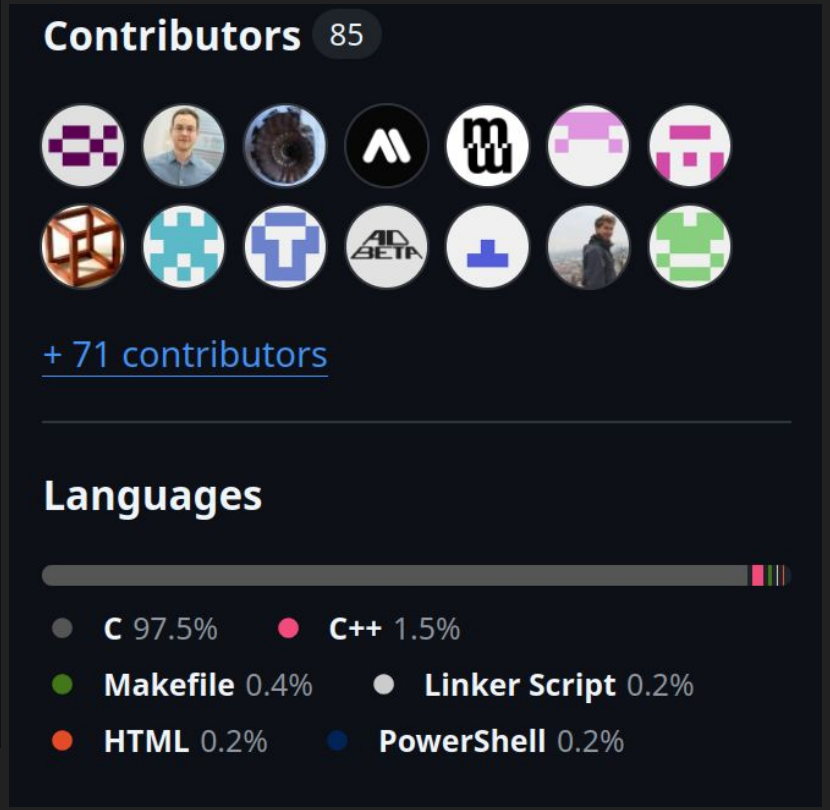
cnlohr on
YouTube
Twitter
BSky
Chaos.Social
(Ask about the Discord)

https://github.com/cnlohr/halnoslides

v5

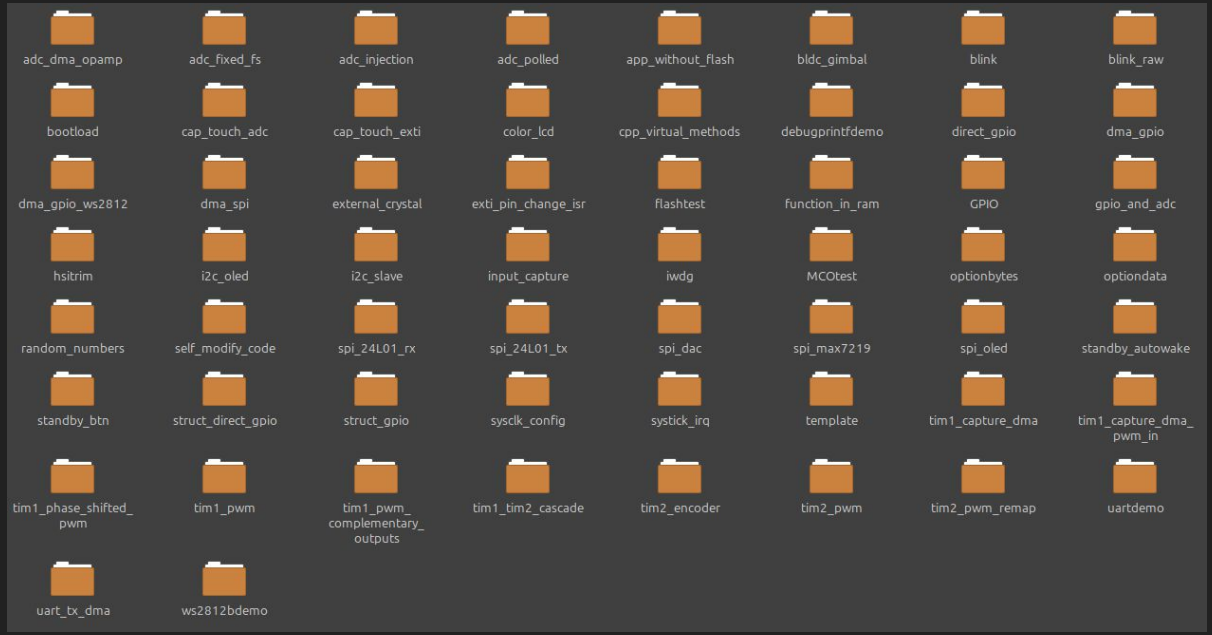
The story of ch32fun





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

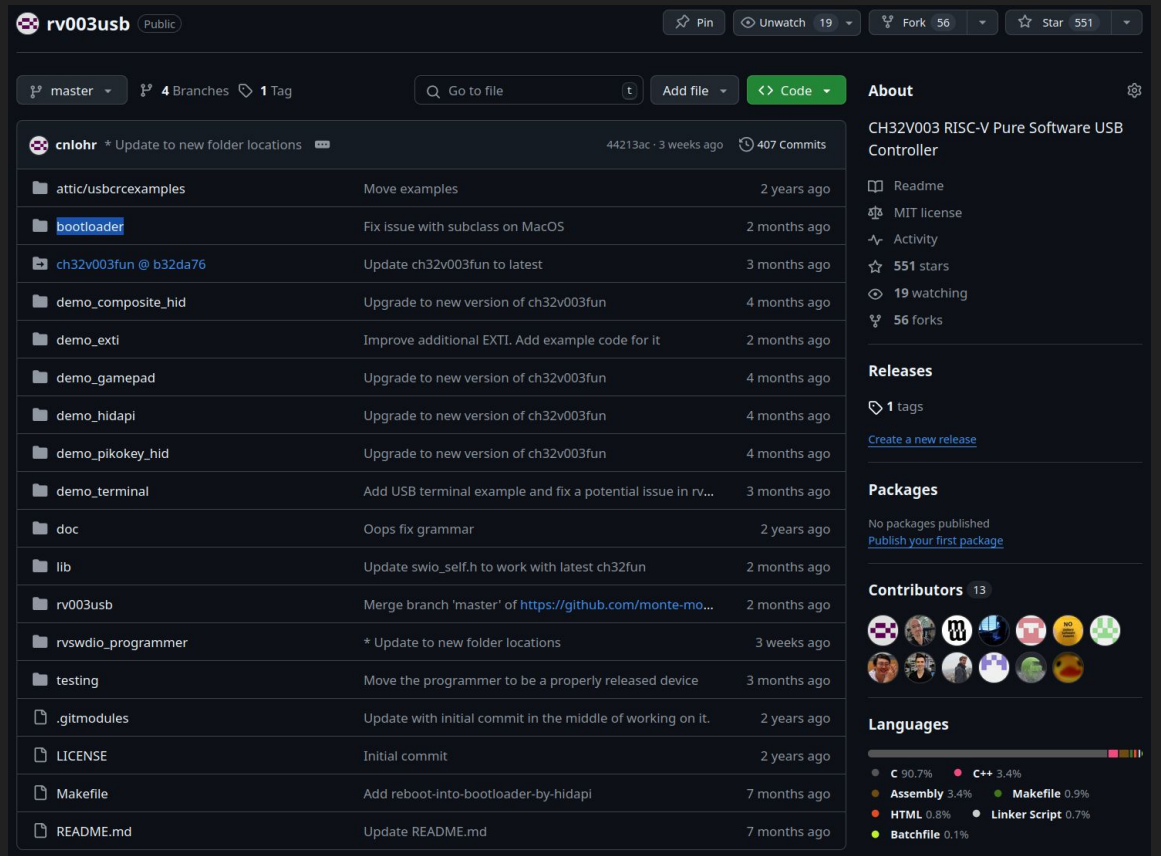
- Teaches how underlying hardware works
- Gives a working start
- Easy to grow/change

```
/*
 * initialize adc for polling
 */
void adc_init( void )
{
    // ADCCLK = 24 MHz => RCC_ADCPRE = 0: divide by 2
    RCC->CFGR0 &= ~(0x1F<<11);

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

    // PD4 is analog input chl 7
    GPIOD->CFGLR &= ~(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 chl 7
    ADC1->RSQR1 = 0;
    ADC1->RSQR2 = 0;
    ADC1->RSQR3 = 7;      // 0-9 for 8 ext inputs and two internals

    // set sampling time for chl 7
    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?

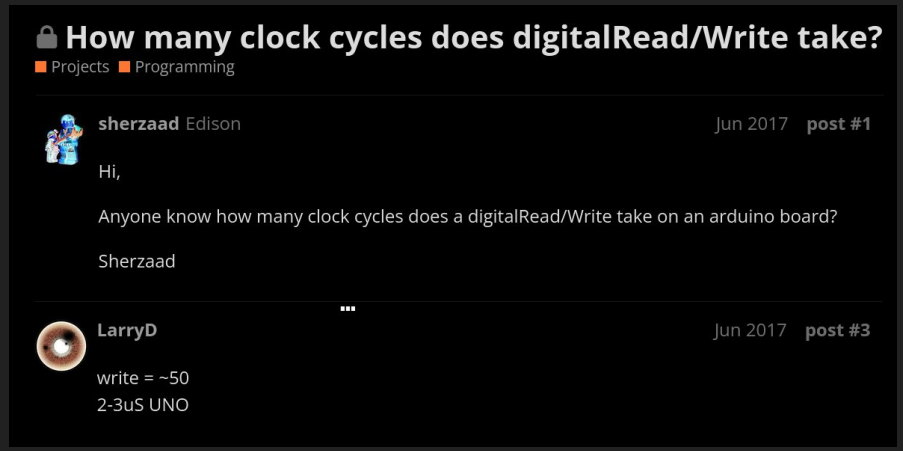
- Maybe?
- “Bootloader” is a full USB stack in 1920 bytes of flash.
- Allows self-program

- Just nice to have USB on whatever pins you want



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



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

- 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

Broadcasting COLOR Channel 3 on an ESP

CNLohr
72.3K subscribers
Analytics Edit video
2.2K

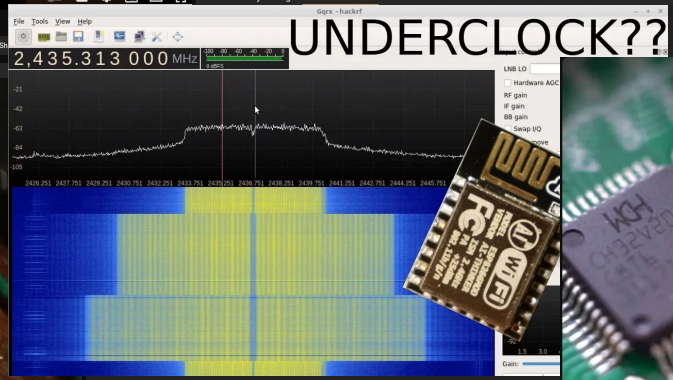
88K views 9 years ago

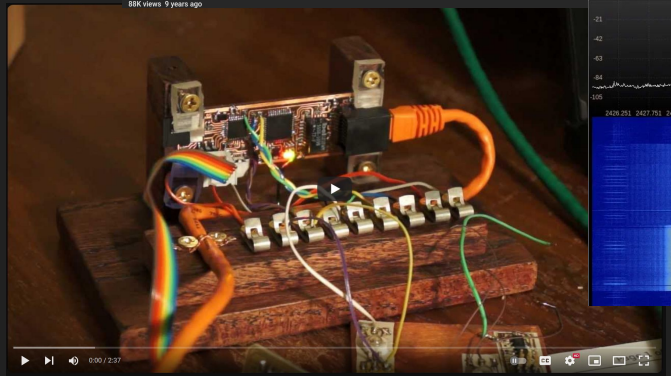
Game Made for VR on a $1 Processor?

CNLohr
72.3K subscribers
Analytics Edit video
1.8K

23K views 2 years ago

Powering a Nixie Tube from USB with a 10c RISC-V, the CH32V003

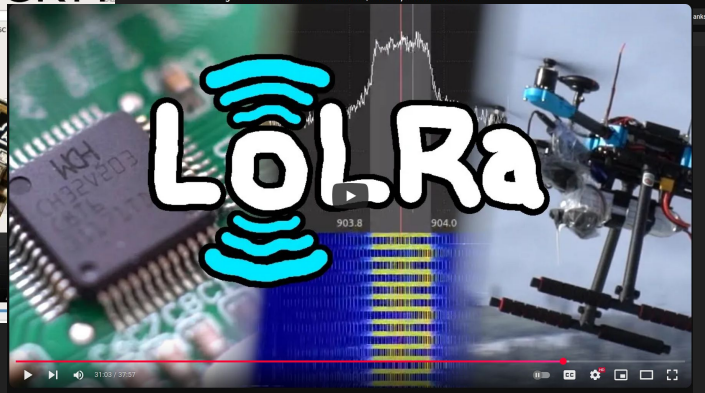




Minecraft Server on a Microscope Slide

CNLohr
72.3K subscribers
Analytics Edit video
1.2K Share Save Clip Thanks

66K views 12 years ago

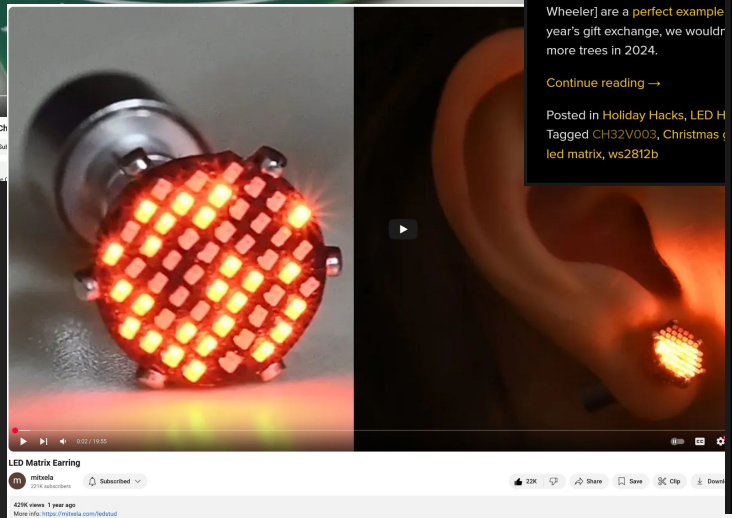


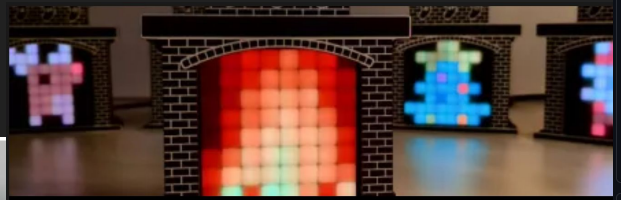
How far can I broadcast LoRa packets WITHOUT a radio? - LoLRa

CNLohr
72.3K subscribers
Analytics Edit video
8.8K Share Save Thanks Clip

170K views 1 year ago
Live chat replay

It takes a village





AN ANIMATED LED FIREPLACE POWERED BY THE CH32V003

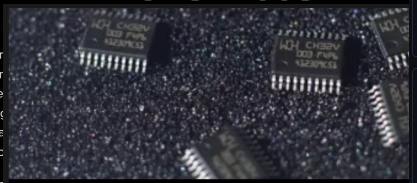
January 2, 2024 by Tom Nardi

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 tc

The fifty electronic fireplaces de
Wheeler] are a perfect example
year's gift exchange, we wouldr
more trees in 2024.

Continue reading →

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

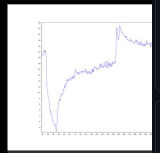


ADDING TEMPERATURE S FUNCTIONALITY TO THE CH32V003 MCU

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, su
measurement. This is a feature that's commonly found on even basic STM32
as [eeucalyptus] shows, you can improvise a working solution by finding alte
act as a thermometer.

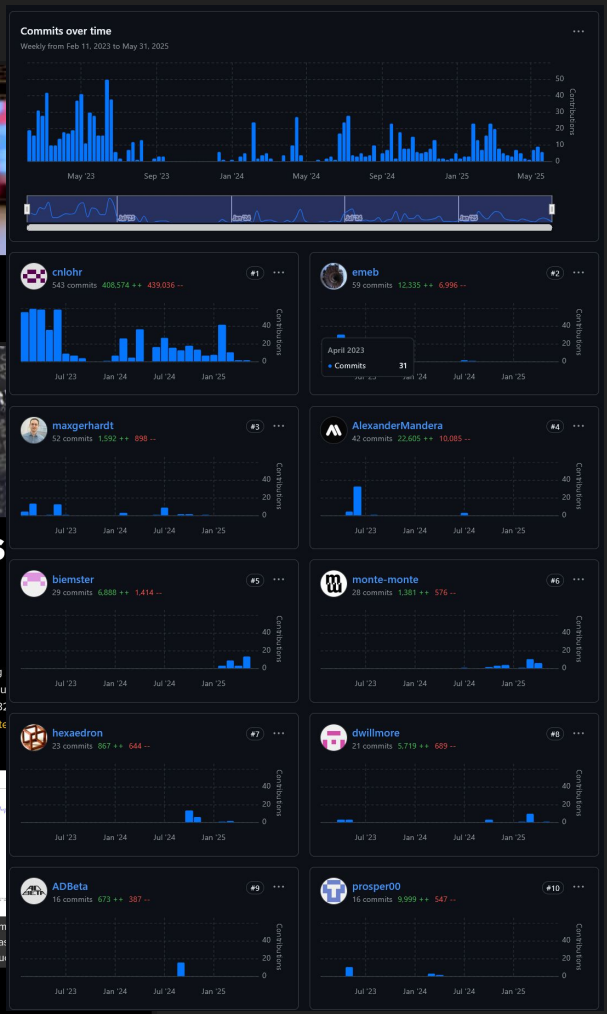
The CH32V003 is a low-end, 32-bit RISC-V-based MCU by the China-based Nanjing Qinheng Microelectronics, commonly known abbreviated as 'WCH', and featured on Hackaday previously. Although it features a single-core, 48 MHz CPU, its selection of peripherals is fairly basic:



Plot of the temperature m
improvised CH32V003 -bas
(Credit: eeuc

So how do you create an internal temperature sensor using just this? [eeucalyptus] 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 a temperature graph.

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

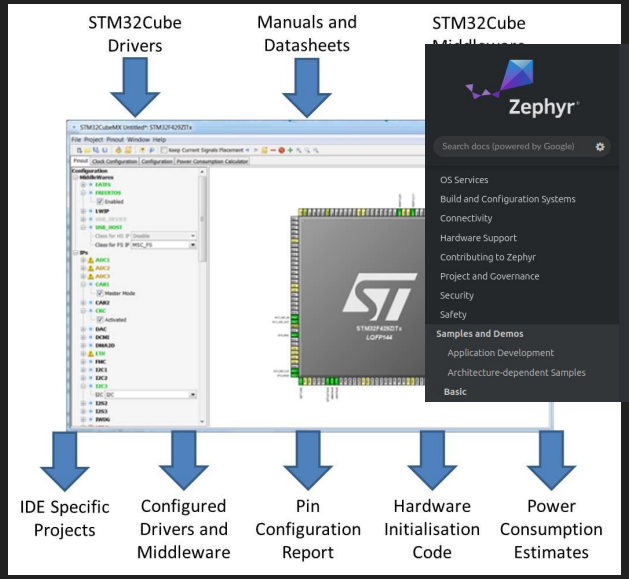


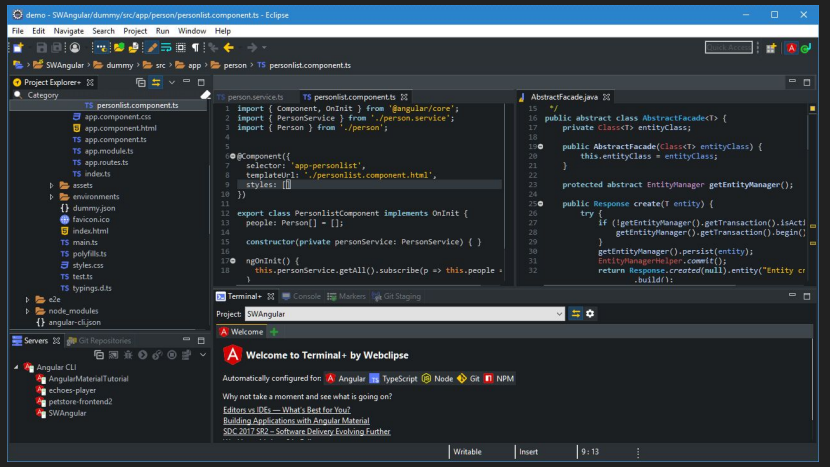
And now

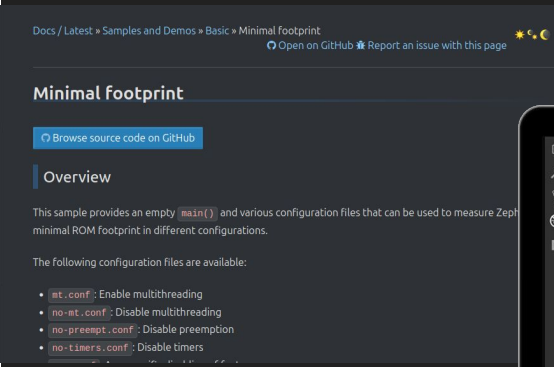
- ch32v203 & beyond
  - USB HS
  - Gigabit Ethernet
  - Bluetooth****** (iSLER™ only)
  - USB SS (not yet)
- Continued support

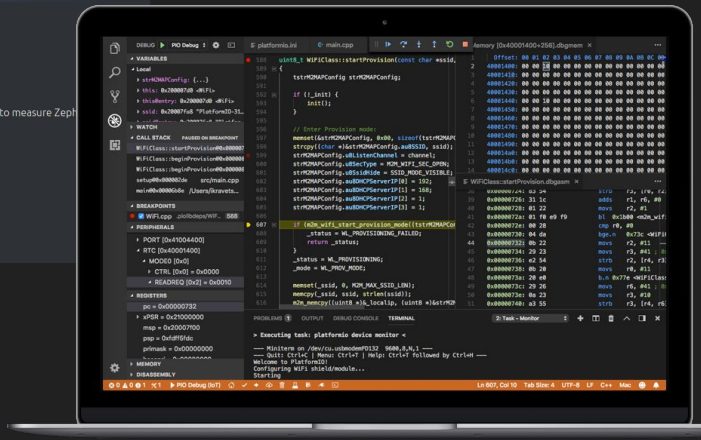
So why?

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









stm32f042fun

stm32f042_fun Public

Pin | Unwatch 2 | Fork 1 | Star 8

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_OD7lgiobow0eRoh-zJ0Q15YpQ/edit?usp=sharing

About

My fun repo for the STM32f042

Readme
Activity
8 stars
2 watching
1 fork

Releases

No releases publ
Create a new rel

Packages

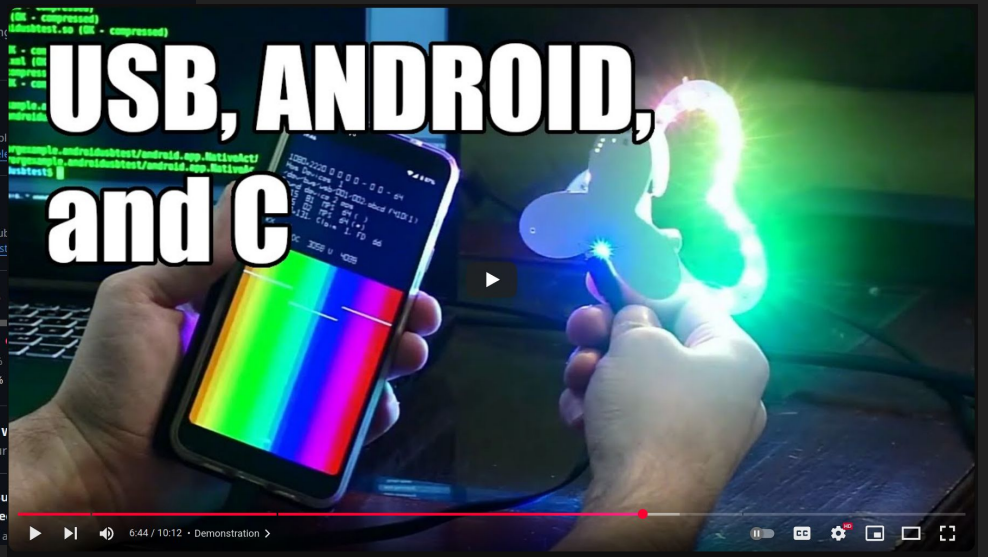
No packages pul
Publish your first

Languages

C 69.9%
HTML 2.6%
Other 0.1%

Suggested v
Based on your

MSBu
proje
Build a



Tensegrity Lamp Running ColorChord on Android with USB in C

CNLohr
72.3K subscribers
Analytics | Edit video

1.3K | Share | Save | Thanks | Clip

17K views 5 years ago

Live chat replay

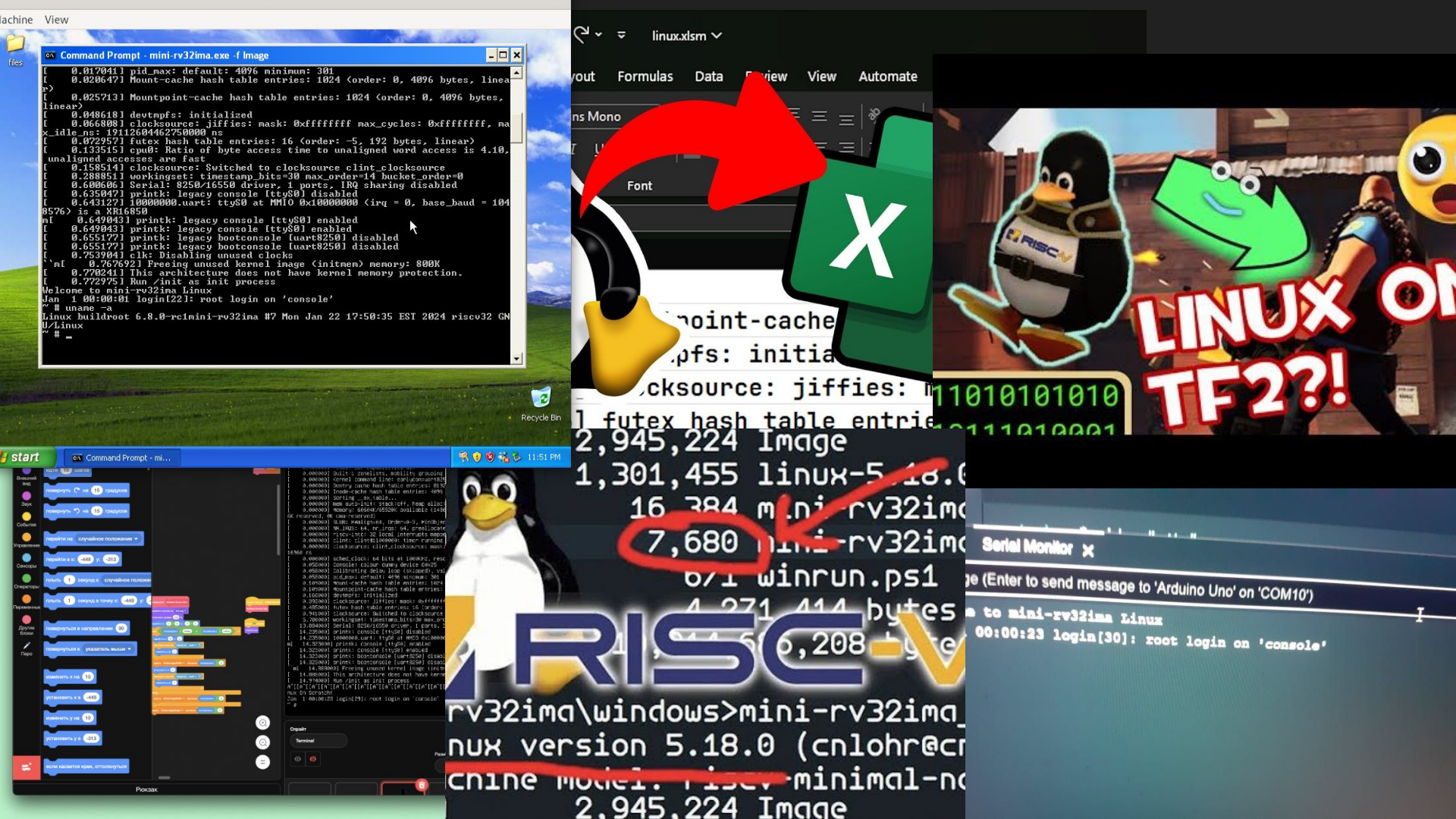
Why didn’t stm32f042fun work out?

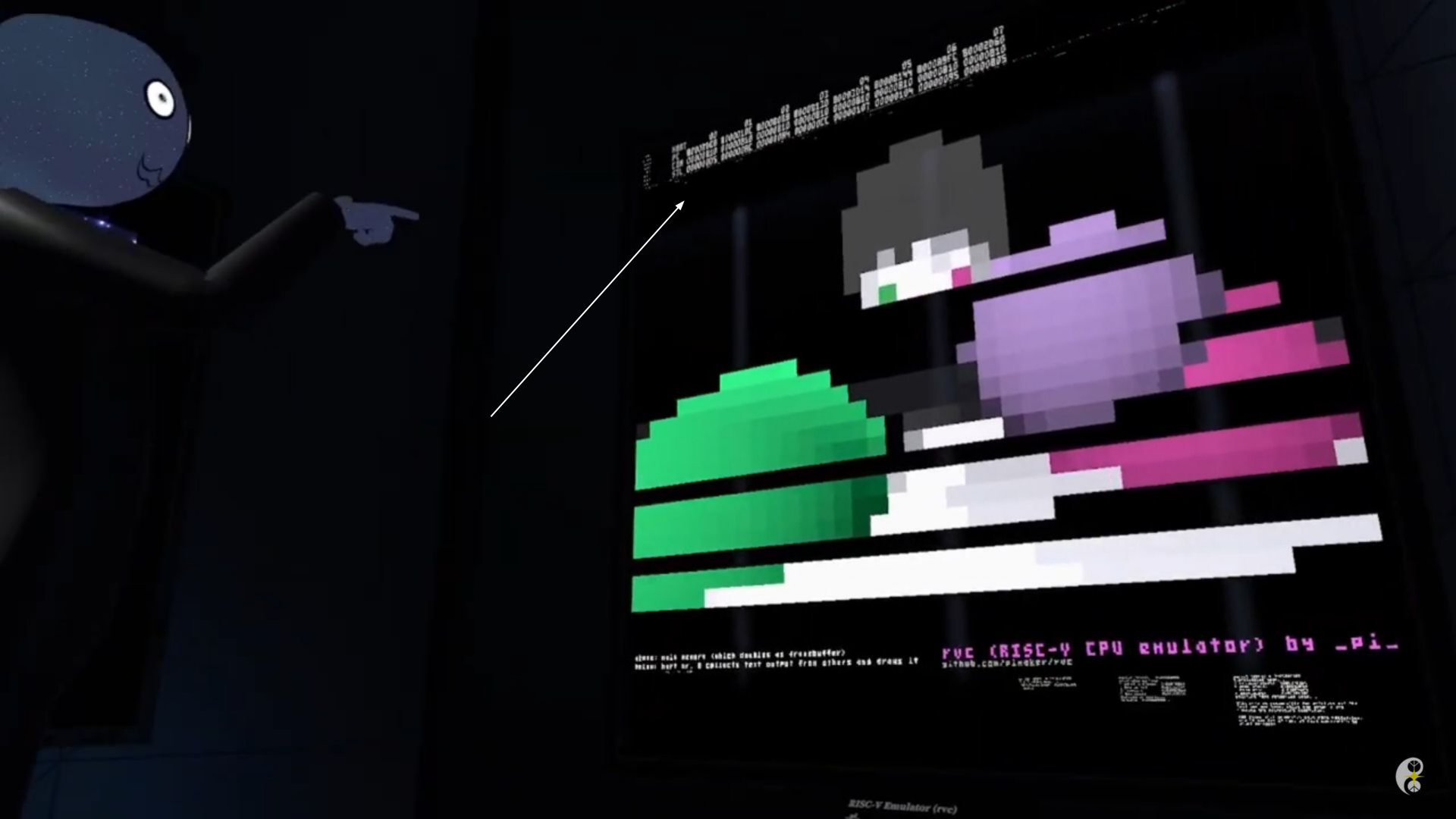
STM32F042 - 2019

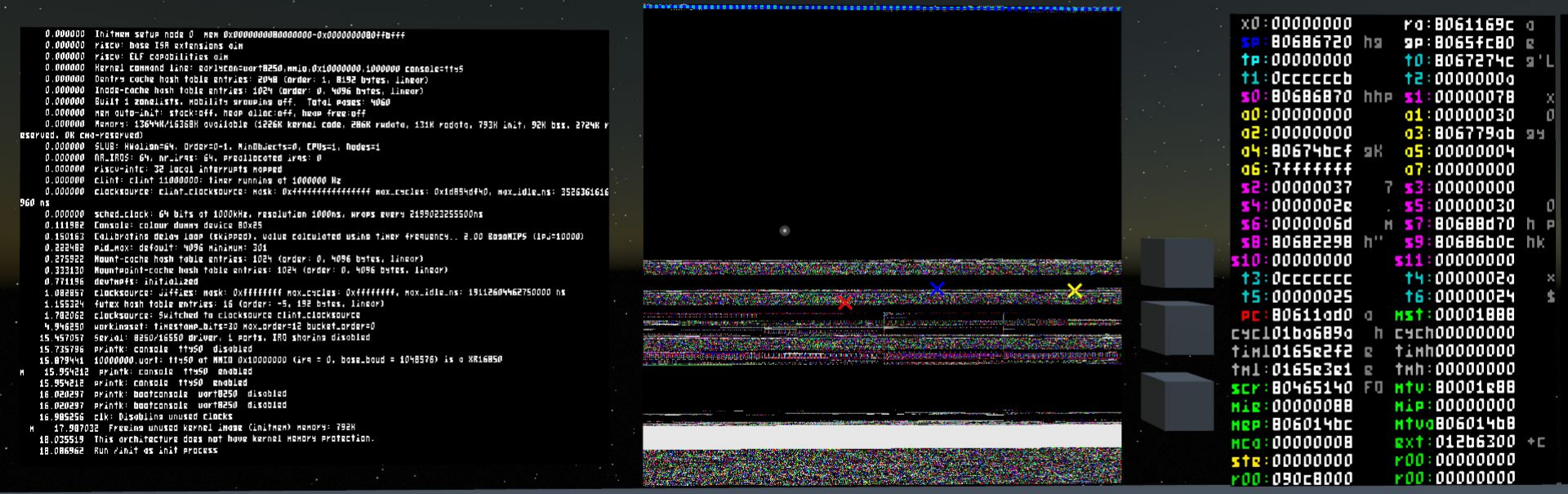
- 8 Stars, 1 Contributor
- $0.50
- 48MHz
- Nice Micro
- Has USB
- 32kB Flash, 6kB RAM
- 7x7 QFN Smallest Package
- Requires 2 pins to program
- ARM

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


Command Prompt - mini-rv32ima.exe -f Image
[0.017041] 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
[0.066808] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 1911260446275000 0 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 accesses are fast
[0.158514] clocksource: Switched to clocksource clint_clocksource
[0.288851] workingset: timestamp_bits=30 max_order=14 bucket_order=0
[0.600606] Serial: 8250/16550 driver, 1 ports, IRQ sharing disabled
[0.635047] printk: legacy console [ttyS0] disabled
[0.643127] 10000000.uart: ttyS0 at MMIO 0x10000000 (irq = 0, base_baud = 1048576) is a XR16850
[0.649043] printk: legacy console [ttyS0] enabled
[0.649043] printk: legacy console [ttyS0] enabled
[0.655177] printk: legacy bootconsole [uart8250] disabled
[0.655177] printk: legacy bootconsole [uart8250] disabled
[0.753904] clk: Disabling unused clocks
[0.767692] Freeing unused kernel image (initmem) memory: 800K
[0.770241] This architecture does not have kernel memory protection.
[0.772975] 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 #7 Mon Jan 22 17:50:35 EST 2024 riscv32 GNU/Linux
~ #
Recycle Bin
linux.xlsm
Formulas
Data
View
Automate
Font
LINUX ON TF2?!
2,945,224 Image
1,301,455 linux-5.18.0
7,680 mini-rv32ima
winrun.ps1
RISC-V
rv32ima\windows>mini-rv32ima
nux version 5.18.0 (cnlohr@cn
2,945,224 Image
Serial Monitor
(Enter to send message to 'Arduino Uno' on 'COM10')
to mini-rv32ima Linux
00:00:23 login[30]: root login on 'console'


rvc (RISC-V CPU emulator) by _pi_
github.com/pineker/rvc


0.000000 Initmem setup node 0 [mem 0x0000000080000000-0x00000000807fbfff]
0.000000 riscv: base ISA extensions aim
0.000000 riscv: ELF capabilities aim
0.000000 Kernel command line: earlycon=uart8250,mmio,0x10000000,1000000 console=ttyS
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 alloc:off, heap free:off
0.000000 Memory: 13644K/16368K available (1226K kernel code, 286K rwdata, 131K rodata, 793K init, 92K bss, 2724K r
eserved, 0K cma-reserved)
0.000000 SLUB: HWalign=64, Order=0-1, MinObjects=0, CPUs=1, Nodes=1
0.000000 NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
0.000000 riscv-intc: 32 local interrupts mapped
0.000000 clint: clint 11000000: timer running at 1000000 Hz
0.000000 clocksource: clint_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3526361616
960 ns
0.000000 sched_clock: 64 bits at 1000kHz, resolution 1000ns, wraps every 2199023255500ns
0.111982 Console: colour dummy device 80x25
0.150163 Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=10000)
0.222482 pid_max: default: 4096 minimum: 301
0.275922 Mount-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
0.333130 Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
0.771196 devtmpfs: initialized
1.088857 clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
1.155324 futex hash table entries: 16 (order: -5, 192 bytes, linear)
1.702062 clocksource: Switched to clocksource clint_clocksource
4.946850 workingset: timestamp_bits=30 max_order=12 bucket_order=0
15.457057 Serial: 8250/16550 driver, 1 ports, IRQ sharing disabled
15.735796 printk: console ttyS0 disabled
15.879441 10000000.uart: ttyS0 at MMIO 0x10000000 (irq = 0, base_baud = 1048576) is a XR16850
M 15.954212 printk: console ttyS0 enabled
15.954212 printk: console ttyS0 enabled
16.020297 printk: bootconsole uart8250 disabled
16.020297 printk: bootconsole uart8250 disabled
16.085256 clk: Disabling unused clocks
M 17.987032 Freeing unused kernel image (initmem) memory: 792K
18.035519 This architecture does not have kernel memory protection.
18.086962 Run /init as init process
x0:00000000 ra:8061169c g
sp:80686720 hg gp:8065fc80 g
tp:00000000 t0:8067274c g'L
t1:0ccccccb t2:0000000a
s0:80686870 hhp s1:00000078 x
a0:00000000 a1:00000030 0
a2:00000000 a3:806779ab g4
a4:80674bcf gK a5:00000004
a6:7fffffff a7:00000000
s2:00000037 7 s3:00000000
s4:0000002e . s5:00000030 0
s6:0000006d M s7:80688d70 h p
s8:80682298 h'' s9:80686b0c hk
s10:00000000 s11:00000000
t3:0ccccccc t4:0000002a x
t5:00000025 t6:00000024 $
pc:80611ad0 q mst:00001888
cycl01ba689a h cych00000000
tim10165e2f2 e timh00000000
tm1:0165e3e1 e tmh:00000000
scr:80465140 F0 mtv:80001e88
mie:00000088 mip:00000000
mep:806014bc mtvq80601468
mcq:00000008 ext:012b6300 +c
str:00000000 r00:00000000
r00:090c8000 r00:00000000

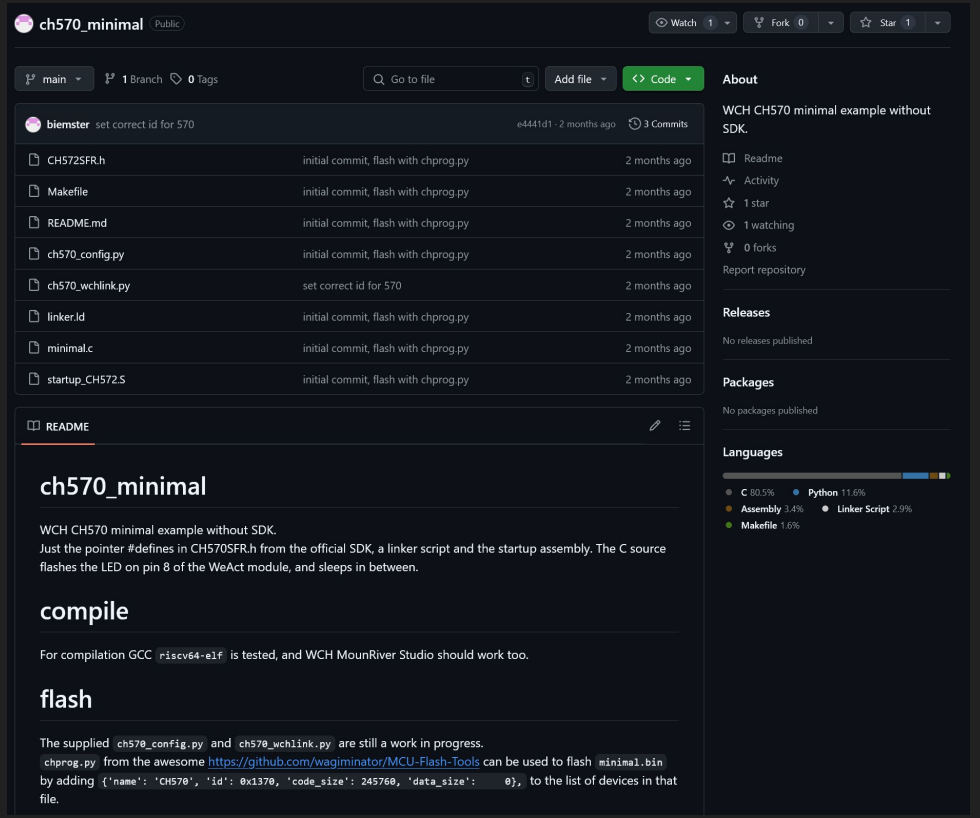
Philosophies

- 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



Programming

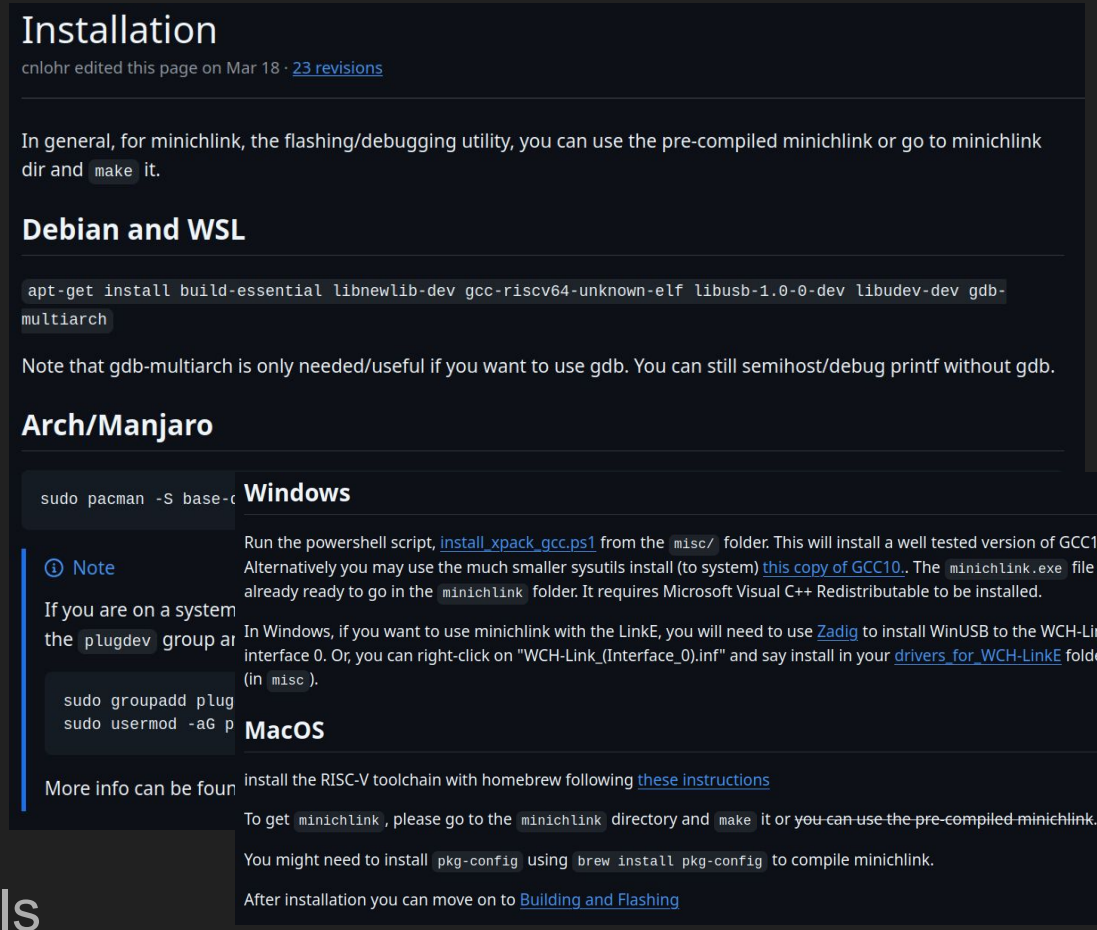
- Physical Interface
- Register Access
- PROGBUF
- Reading/Writing RAM
- Writing to Flash
- Semihosting
- GDB

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
progbuf0-7	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-
```

Note

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

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

More info can be foun

Windows

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.

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`).

MacOS

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	-	x7	t2	temporary register 2	-R
8	0	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	x11	a1	function argument 1 / return value 1	-R
12	4	x12	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	-	x16	a6	function argument 6	-R
17	-	x17	a7	function argument 7	-R

RV32E

RV32C

RV32

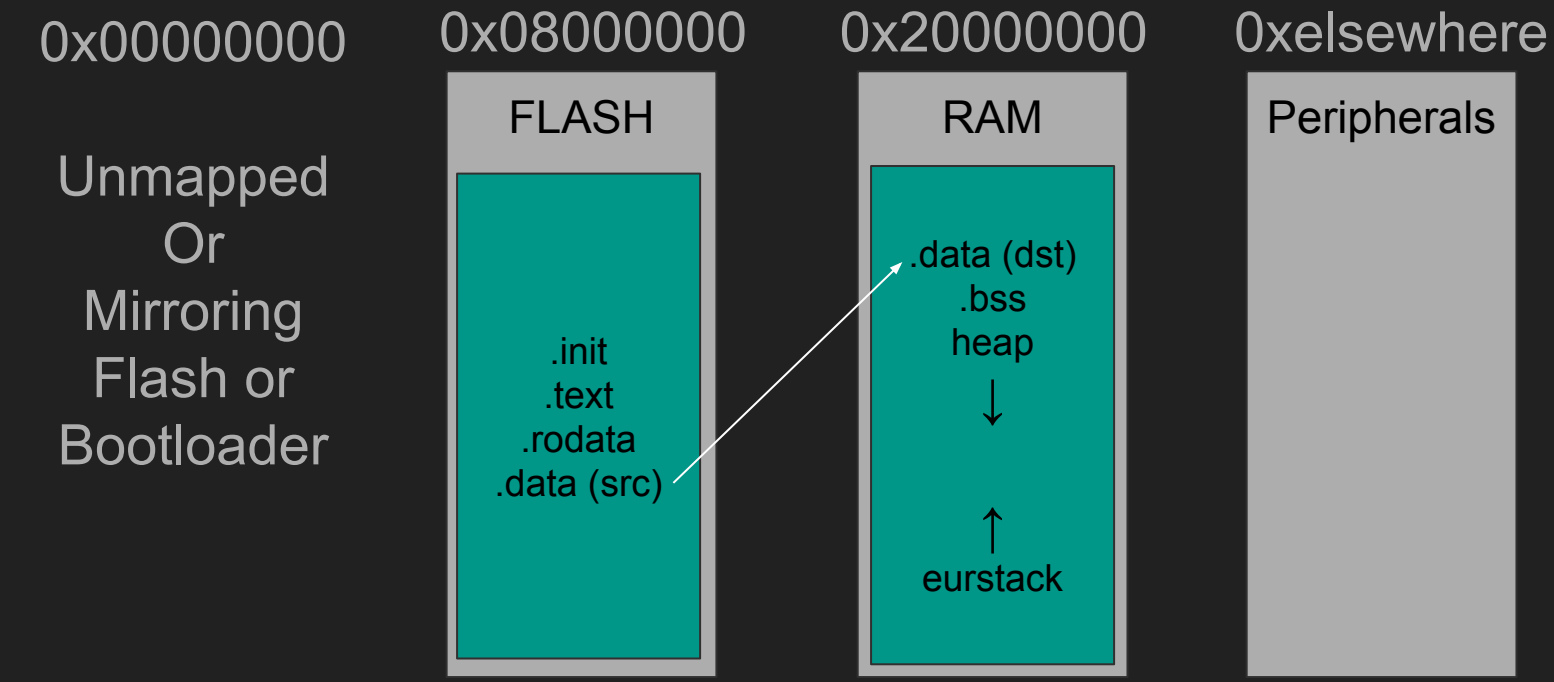
https://en.wikichip.org/wiki/risc-v/registers

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


0x00000000
Unmapped
Or
Mirroring
Flash or
Bootloader
0x08000000
FLASH
.init
.text
.rodata
.data (src)
0x20000000
RAM
.data (dst)
.bss
heap
eurstack
0xelsewhere
Peripherals

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

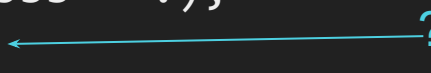
```
.text :
{
    . = ALIGN(4);
    *(.text)
    *(.text.*)
    *(.rodata)
    *(.rodata*)
    . = ALIGN(4);
} >FLASH AT>FLASH
```

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

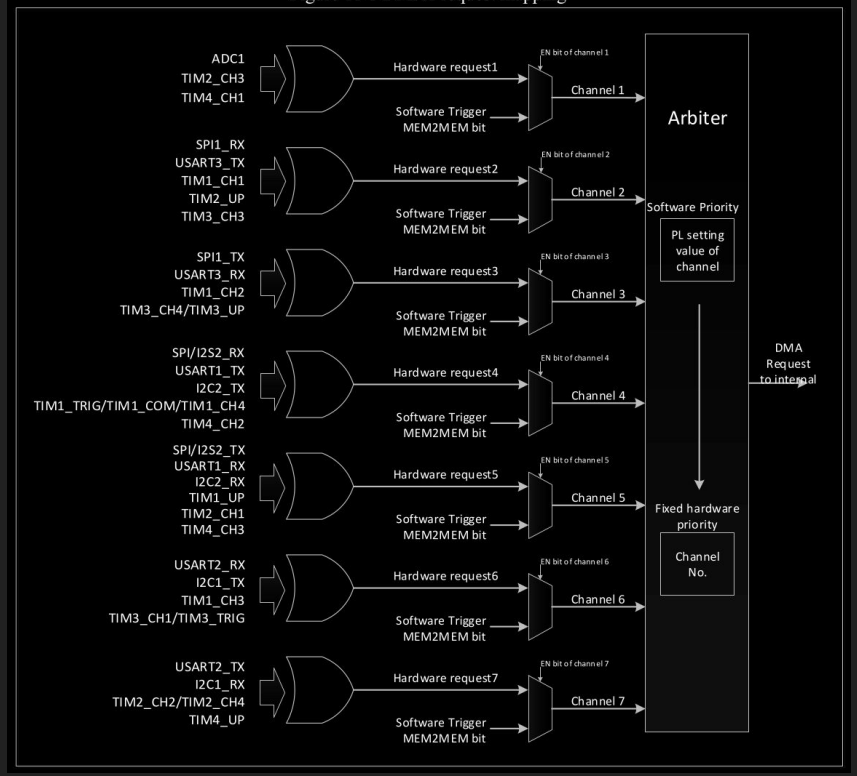
```
typedef struct
{
    __IO uint32_t CFGLR;    /* Port Configuration Register Low  */
    __IO uint32_t CFGHR;    /* Port Configuration Register High */
    __I  uint32_t INDR;     /* Port Input Data Register         */
    __IO uint32_t OUTDR;    /* Port Output Data Register        */
    __IO uint32_t 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)
        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)
.L3:
        li      a5,84
        mul     a3,a3,a5
        add     a4,a4,a3
        bne     a1,zero,.L4
        sw      a2,76(a4)
        ret
.L4:
        sw      a2,80(a4)
.L1:
        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
```

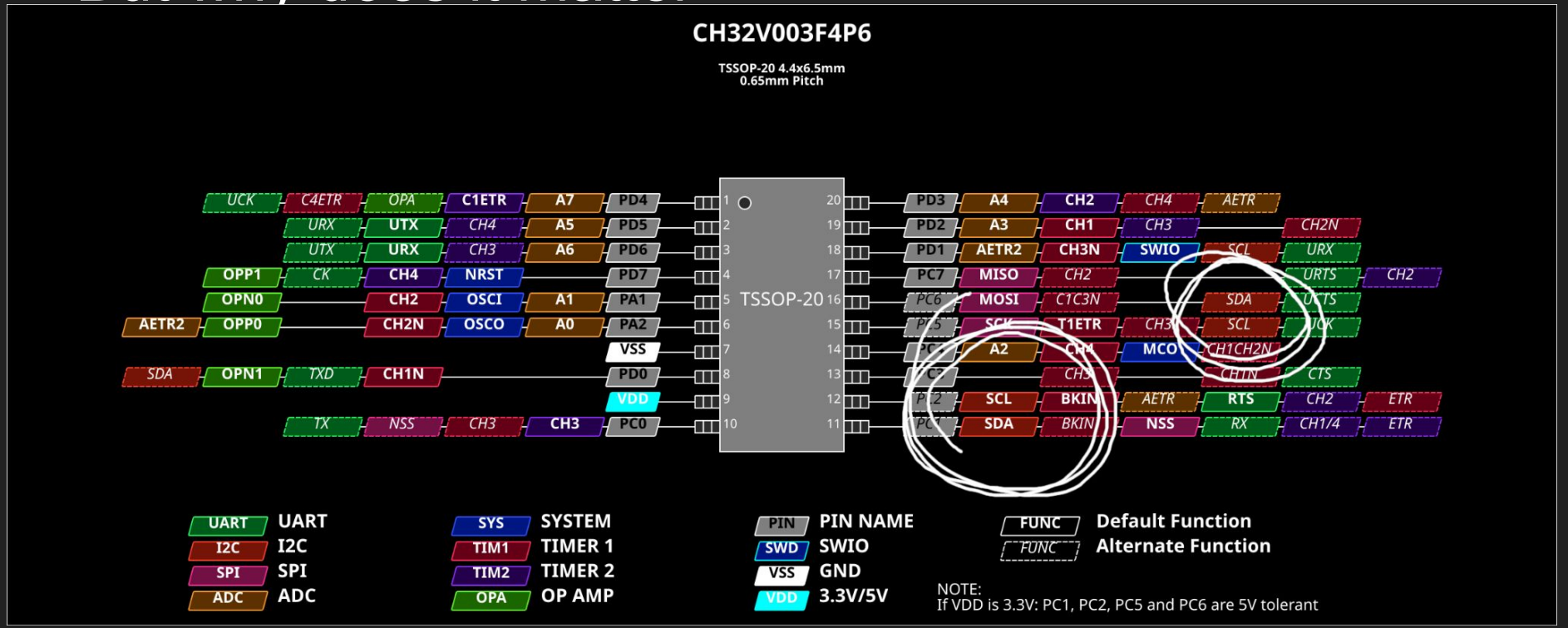
COMPILER EXPLORER

…or lst files

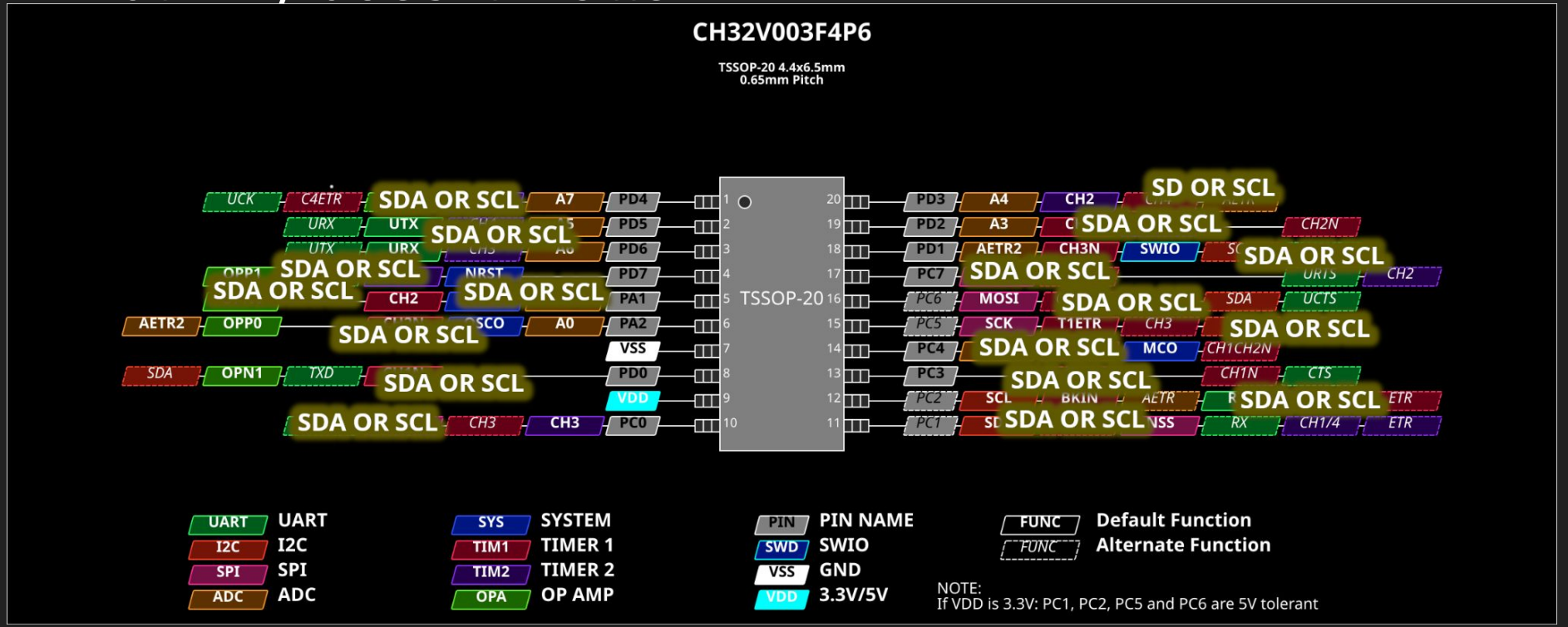
“But why does it matter”

- Stops rationale for mediocrity
- Bit banging is valid.
- Reduces bloat
- static_i2c.h

“But why does it matter”



“But why does it matter”



“But why I2C


Clinton
0
Trump
0
538
0
Share Map
Split Electoral Votes
270 to WIN
UCK
SDA C
AETR2
OPP0
SDA
OPN1
UART
I2C
SPI
ADC

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

*-rvprog

**-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 suddelnly 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.

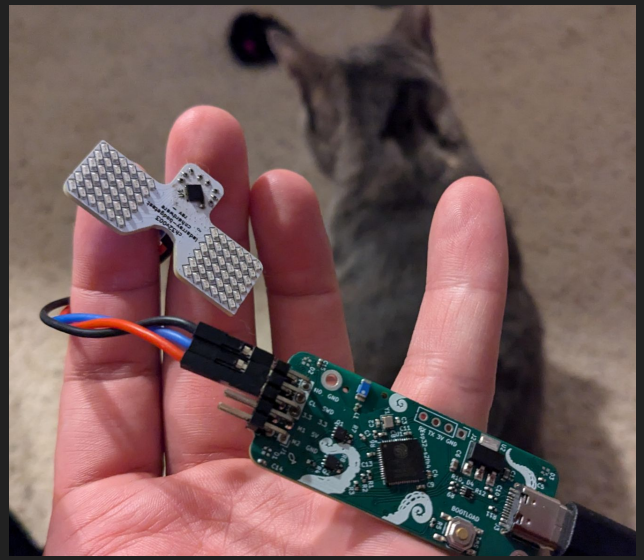
Now what? I looked around the internet for some stripped down printf() implementations but none I really liked. Then I decided to develop my own minimal snprintf().

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

https://github.com/cnlohr/halnoslides