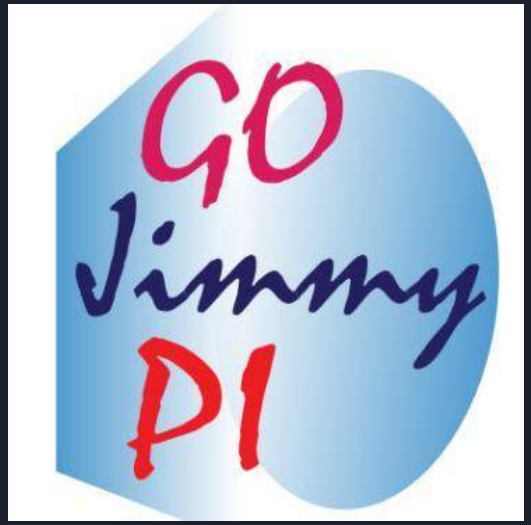


The Road to DOOM on an FPGA

Building a Soft-Logic Computer with Open Source Tools

Presented by Jim Scarletta

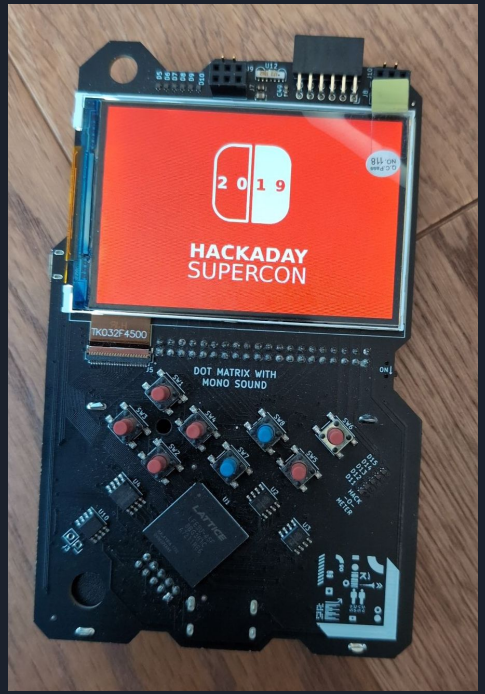




https://gojimmypi.github.io/Hackaday-Supercon-2026/

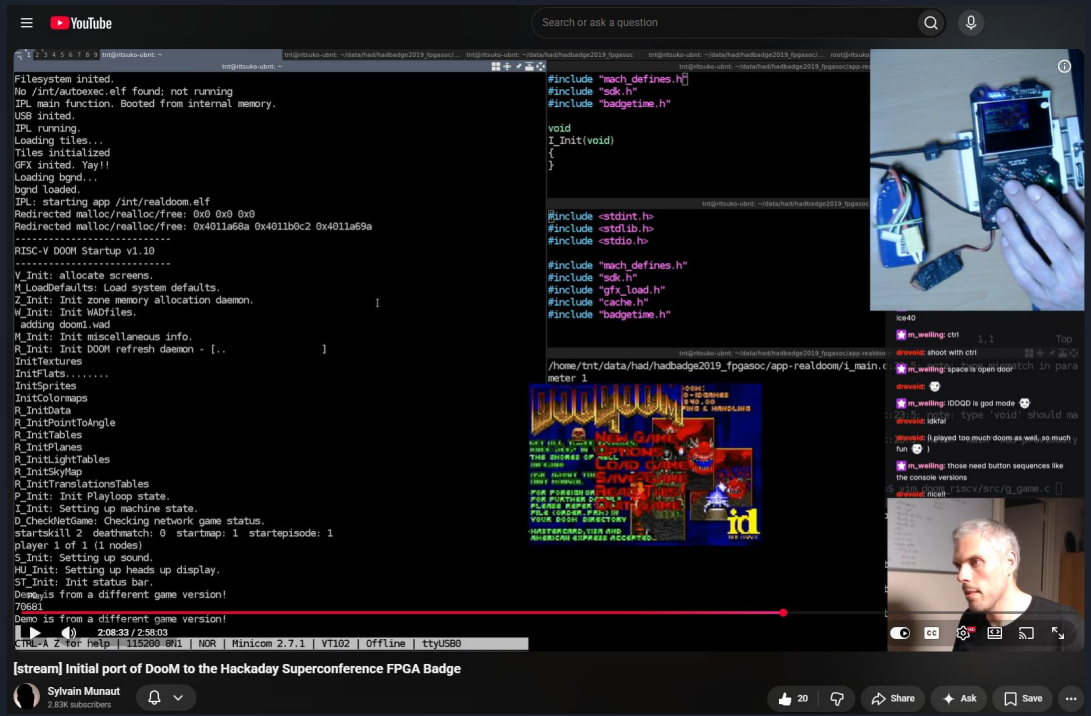


Sound Familiar?



Source code: https://github.com/Spritetm/hadbadge2019_fpgasoc

Not the First Doom Running on an FPGA



https://www.youtube.com/watch?v=VzbI6qFaTmQ

GO Jimmy PI

How it started . . .



How it’s going:



GO Jimmy PI

Target Audience: Educational



(*nearly*) Any Skill Level

- Scripts and GitHub workflows
- All open source code
- Something for everyone
- Not just Doom - other code can be loaded



Credits and Tools Used

- MANY Open Source projects brought together here.
- ChatGPT - interactive, iterative, educational
  - *"Why doesn't this work?"*
  - *"What does this do?"*
  - *"Review this code"*
- LiteX Code Generator
  - *"Generate code for parameterized standards"*

It's all about: *!! TESTING !!*



Why Doom?





Full end-to-end, non trivial example



Objectives



- Pick a Field Programmable Gate Array Board (FPGA).

  - Includes features to implement: Display, Memory, etc.

- Pick CPU, Instruction Set Architecture (ISA) - needs a compiler!

- Pick a Doom source.

- Bring it all together for a usable, JTAG-debuggable system.



What's an FPGA?

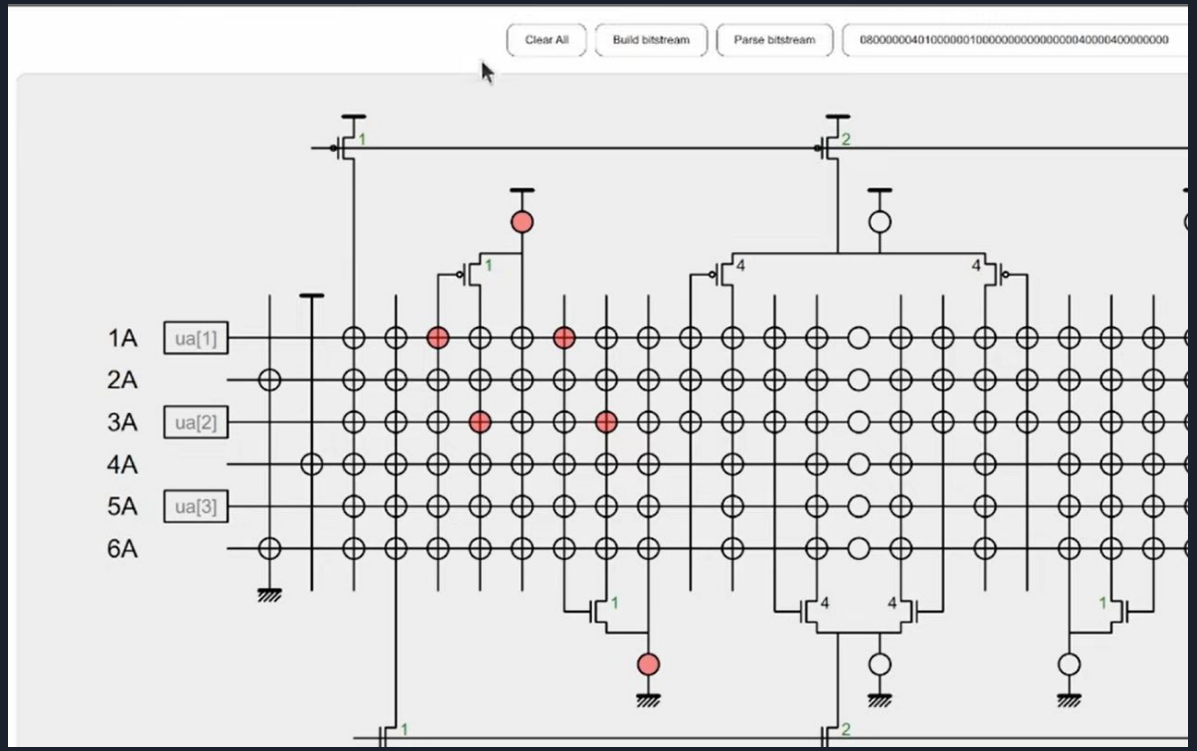


Image credit: Matt Venn *"Schematic to Silicon with Mini MOSbius"* youtu.be/NM5cLWYWqG0

Verilog - Clocked vs Combinatorial RTL

ulx3s_hdmi_test_pattern.v

```
29
30    localparam V_ACTIVE = 600;
31    localparam V_FRONT  = 3;
32    localparam V_SYNC   = 10;
33    localparam V_BACK   = 11;
34    localparam V_TOTAL  = V_ACTIVE + V_FRONT + V_SYNC + V_BACK;
35
36    reg [10:0] pixel_x;
37    reg [9:0]  pixel_y;
38    reg        pixel_toggle;
39
40    always @ (posedge clk_pix or negedge rst_n_pix) begin
41        if (!rst_n_pix) begin
42            pixel_x <= 11'd0;
43            pixel_y <= 10'd0;
44            pixel_toggle <= 1'b0;
45        end else begin
46            pixel_toggle <= !pixel_toggle;
47            if (pixel_x == H_TOTAL - 1) begin
48                pixel_x <= 11'd0;
49                if (pixel_y == V_TOTAL - 1)
50                    pixel_y <= 10'd0;
51                else
52                    pixel_y <= pixel_y + 10'd1;
53            end else begin
54                pixel_x <= pixel_x + 11'd1;
55            end
56        end
57    end
58
59    wire active_video = pixel_x < H_ACTIVE && pixel_y < V_ACTIVE;
60    wire hsync = pixel_x >= H_ACTIVE + H_FRONT
```

reg [10:0] pixel_x

100 % | No issues found | Ln: 54, Ch: 24 | SPC | LF | Windows 1252

ulx3s_hdmi_test_pattern.v

```
73    always @ (*) begin
74        red = 8'h00;
75        green = 8'h00;
76        blue = 8'h00;
77
78        if (active_video) begin
79            case (pixel_x[9:7])
80                3'd0:    begin red = 8'hff; green = 8'hff; blue = 8'hff; end
81                3'd1:    begin red = 8'hff; green = 8'hff; blue = 8'h00; end
82                3'd2:    begin red = 8'h00; green = 8'hff; blue = 8'hff; end
83                3'd3:    begin red = 8'h00; green = 8'hff; blue = 8'h00; end
84                3'd4:    begin red = 8'hff; green = 8'h00; blue = 8'hff; end
85                3'd5:    begin red = 8'hff; green = 8'h00; blue = 8'h00; end
86                3'd6:    begin red = 8'h00; green = 8'h00; blue = 8'hff; end
87                default: begin red = 8'h00; green = 8'h00; blue = 8'h00; end
88            endcase
89
90            if (grid) begin
91                red = red >> 1;
92                green = green >> 1;
93                blue = blue >> 1;
94            end
95
96            if (border) begin
97
98
99                red = 8'hff;
100               green = 8'hff;
101               blue = 8'hff;
102           end
103       end
104   end
```

wire border = pixel_x == 0|| pixel_x == H_ACTIVE - 1 || pixel_y == 0|| pixel_y == V_ACTIVE - 1

100 % | No issues found | Ln: 98, Ch: 9 | SPC | LF | Windows 1252

GO Jimmy PI

Learn more about Verilog

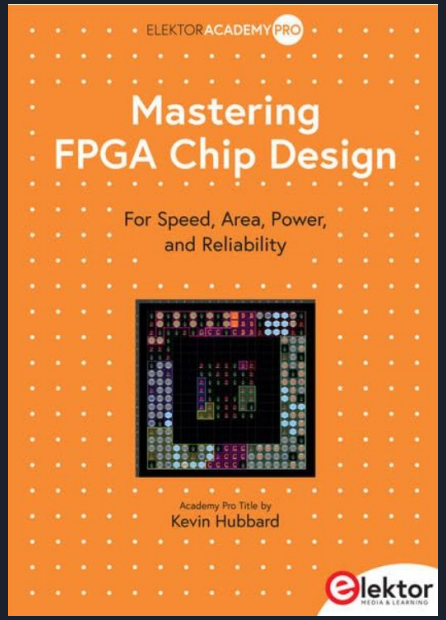
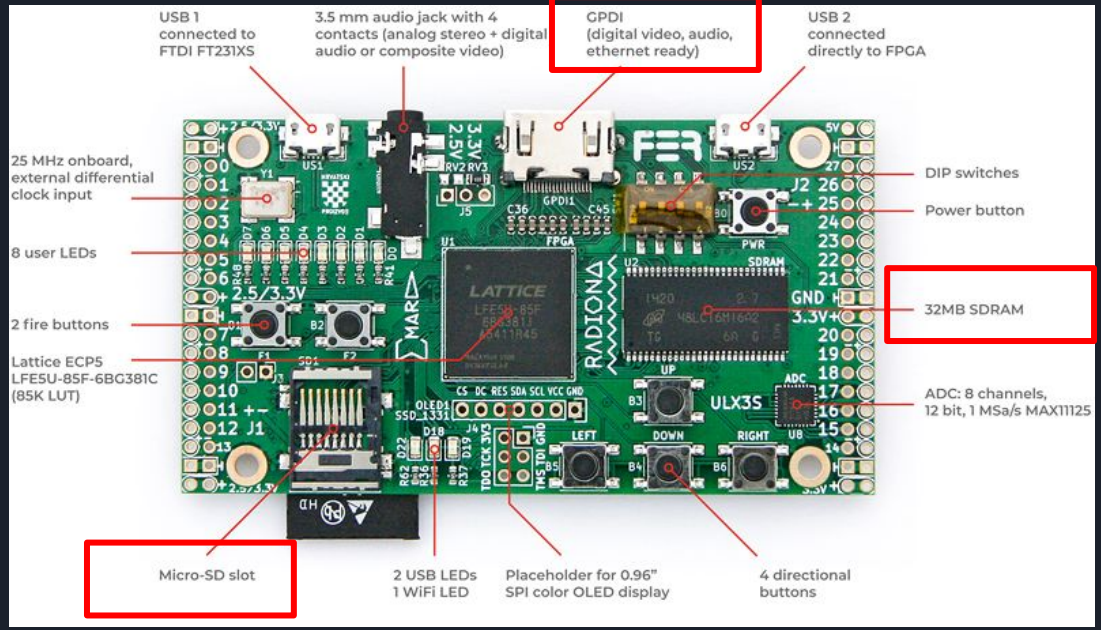


Image credit: Kevin Hubbard @bml_khubbard https://blackmesalabs.wordpress.com/

Lattice ECP5 FPGA



ulx3s.github.io

Image credit: Radiona. See also: crowdsupply.com/radiona/ulx3s

ESP32 on the other side!

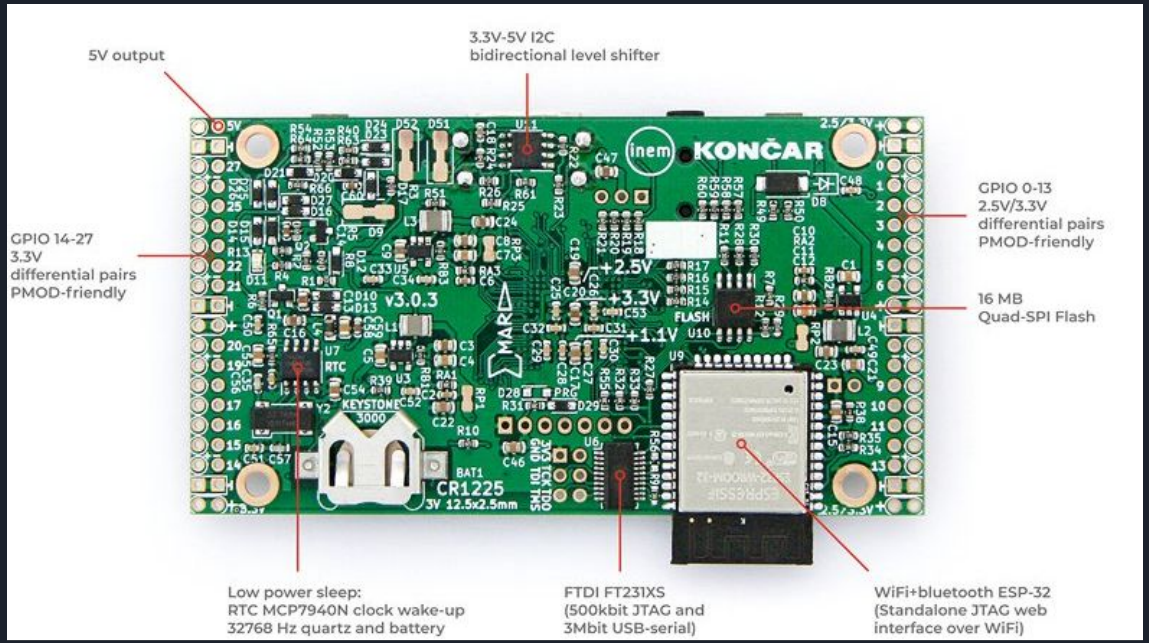


Image credit: Radiona

ULX4M-LD ("D" for DDR3 Memory)

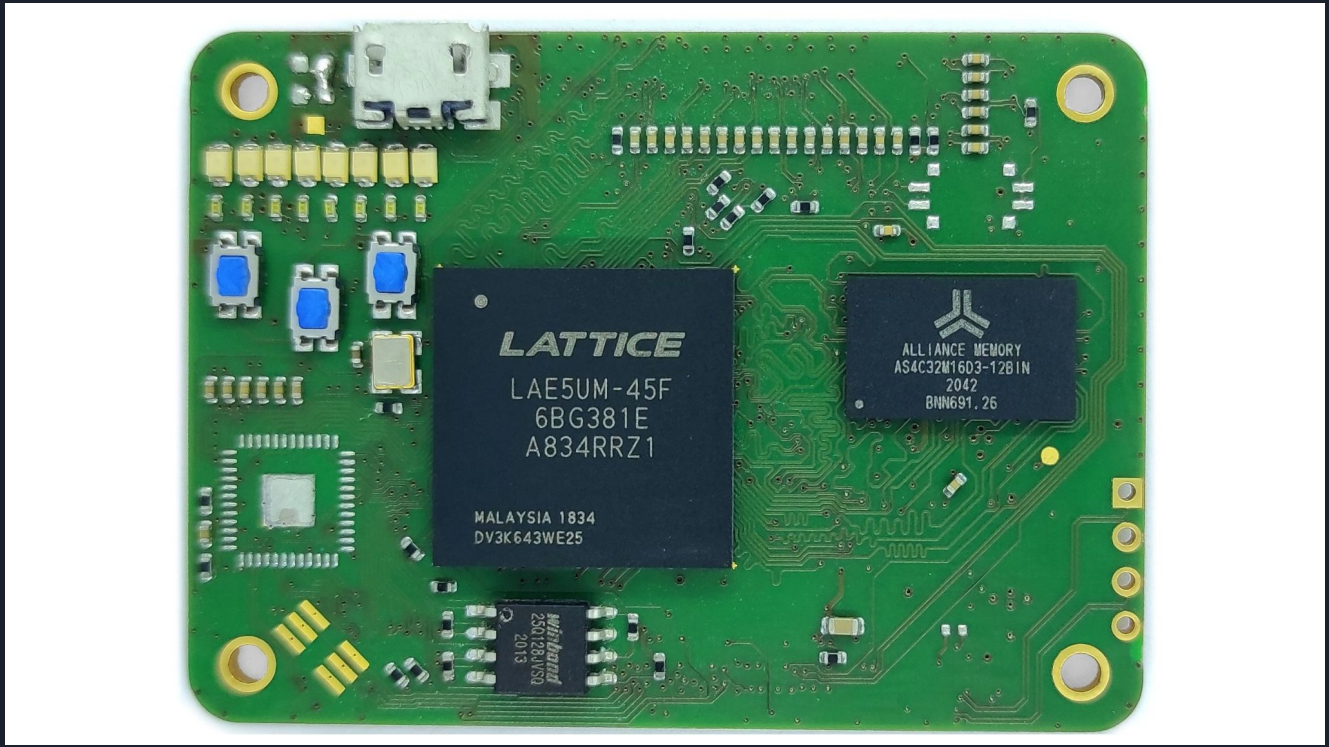
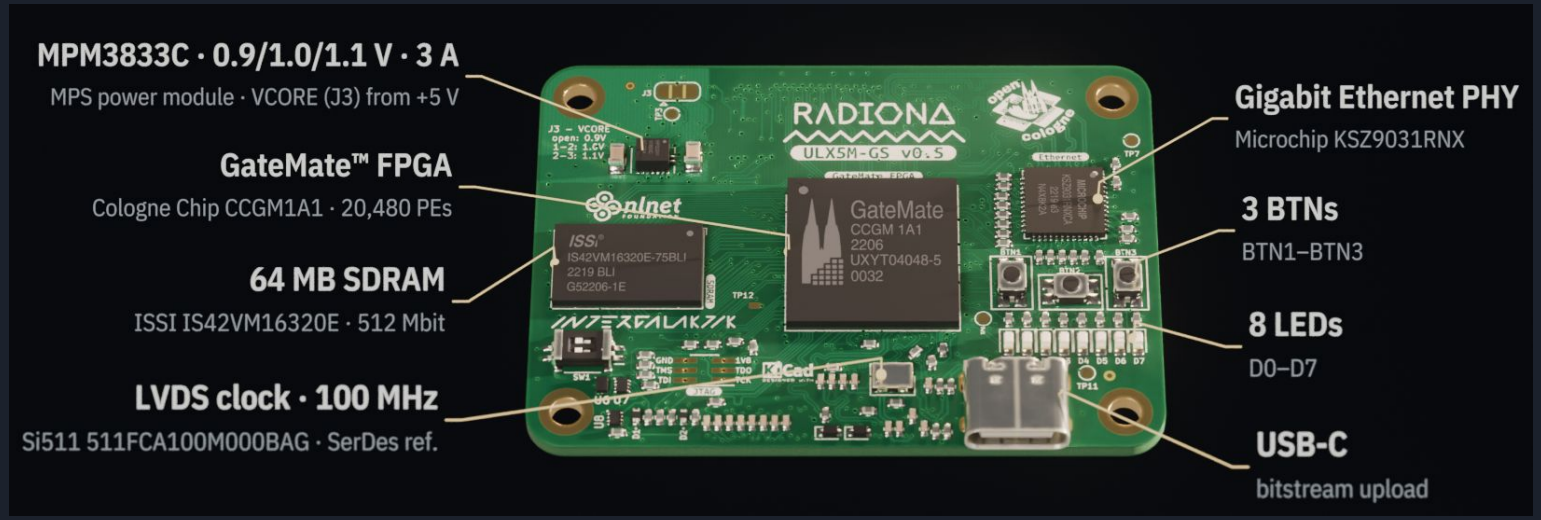


Image credit: intergalaktik https://www.crowdsupply.com/intergalaktik/ulx4m

GO Jimmy PI

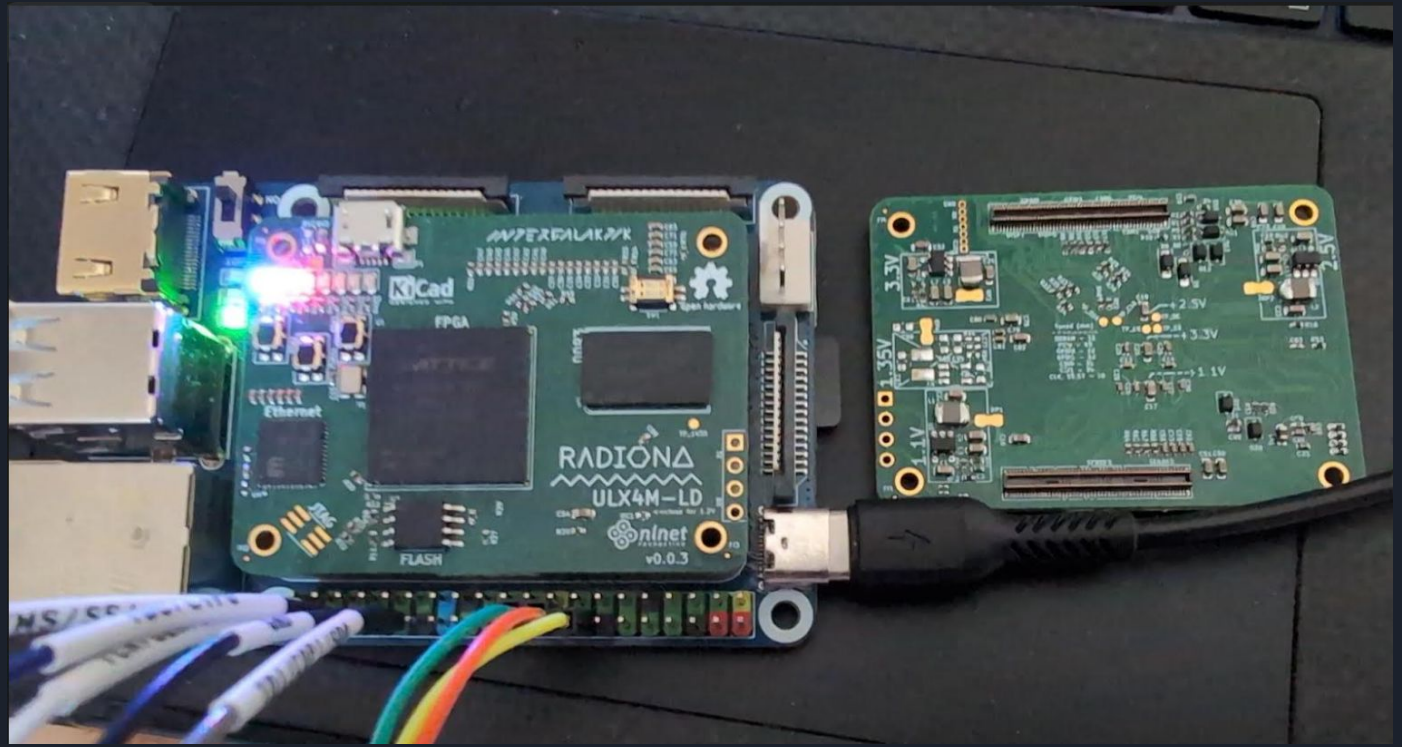
ULX5M-GS Cologne Chip GateMate



DEFAULT VOLTAGE FOR GPIO IS SET TO 1.8V in this v0.5 !!!

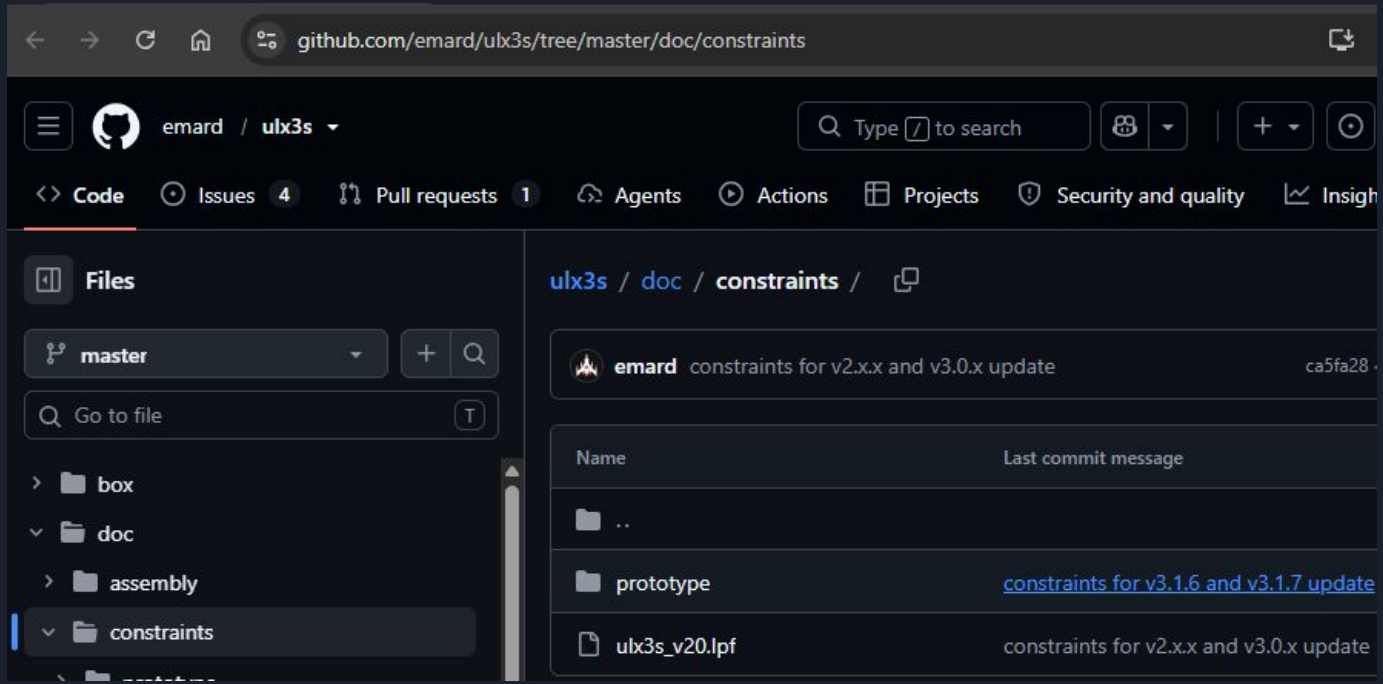
Image credit: https://github.com/intergalaktik/ulx5m-gs

ULX4M and ULX5M - the “M” is CM4 Module



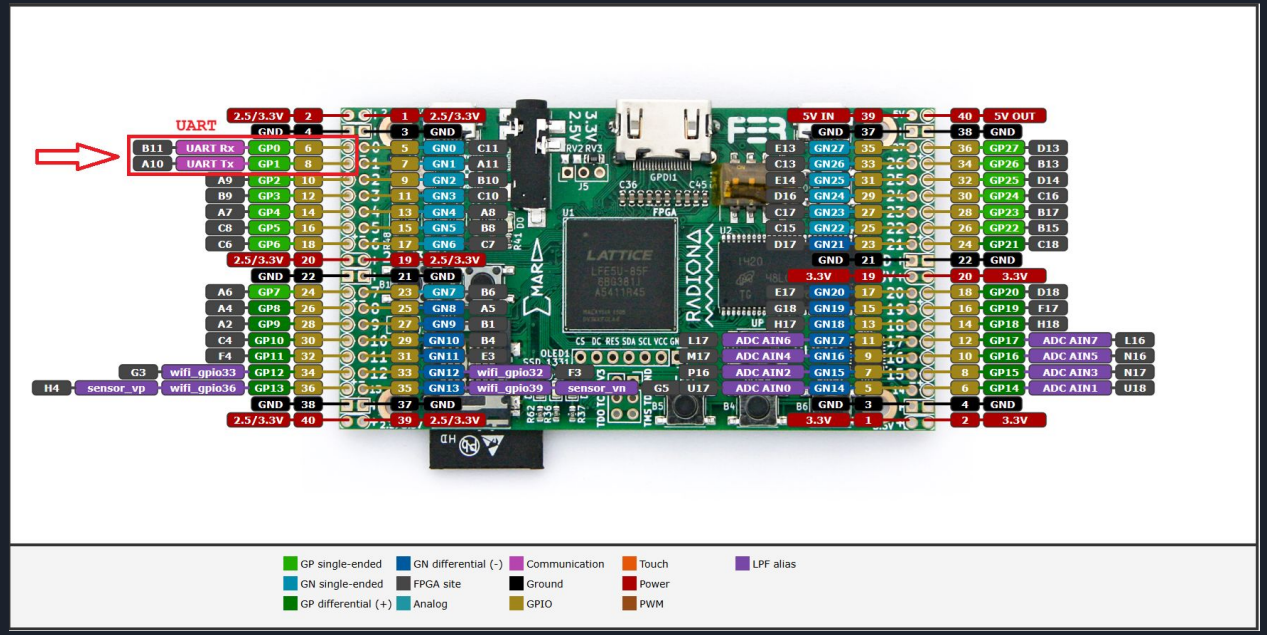
(carrier needed for connectors: display, ethernet, etc.)

Reference Constraint Files



https://github.com/emard/ulx3s/tree/master/doc/constraints

Programmatically Generated Pinout




```
python3 -m pip install --user --upgrade pinout cairosvg pillow
```

https://github.com/ulx3s/ulx3s-pinout

Which is the best CPU (ISA)?



x86, x86-64, ARM, ARM64, RISC-V, MIPS, PowerPC, SPARC, 6502, 65C02, 65816, Z80, 8080, 8051, 6800, 6809, 68000, AVR, PIC, MSP430, SuperH (SH), Alpha, PA-RISC, IA-64 (Itanium), VAX, PDP-11, OpenRISC, Xtensa, MicroBlaze, Nios II, ARC, TriCore, IBM z/Architecture . . . many others



Which is the best RISC-V?



Like many things: *That depends . . .*

- Requirements?
- Toolchain?
- Complete System on a Chip (SoC) ?
- Size? HAD2019 badge uses PicoRV32 size optimized 750-2,000 LUTs
- JTAG Debugging?
- Run Linux?



Which is the best RISC-V?



Like many things: *That depends . . .*

- Run DOOM!
- Build with an open source toolchain
- We want to build a complete System
- Needs to fit in an ECP5 85K or smaller
- Definitely want JTAG Debugging
- No Linux needed



Hazard3 - RPi family, less likely to become abandonware.

raspberrypi.com/documentation/microcontrollers/microcontroller-chips.html

Search CTRLK or Ask

Feature comparison

The following table provides a comparison between RP2040 and RP2350 microcontrollers. For more details about each microcontroller's features, see the dedicated sections for RP2040 and RP2350.

Table 1.
Comparison of Raspberry Pi microcontrollers (RP2040 and RP2350)

Feature	RP2040	RP2350
Type	General-purpose MCU	High-performance MCU
Processor core	Dual-core Arm Cortex-M0+	Dual-core Arm Cortex-M33 (higher performance, TrustZone support) or Hazard3 (RISC-V)
Clock speed	Up to 133 MHz	Up to 150 MHz
On-chip SRAM	264 kB in six independent banks	520 kB multi-bank performance
Flash (non-volatile) memory	No on-chip flash; support for up to 16 MB of off-chip flash memory through dedicated external (QSPI.XIP) bus.	Option for on-chip flash depending on variant; support for up to 32 MB of off-chip flash memory through dedicated external (QSPI.XIP) bus. See RP2350 variants.
External volatile working memory (PSRAM)	Not supported.	Supported by a QMI memory interface using the QSPI.XIP bus.
GPIO pins	30 GPIO pins, 4 of which can be used as analogue inputs.	Depending on variant: 30 GPIO pins, 4 of which can be used as analogue inputs; or 48 GPIO pins, 8 of which can be used as analogue inputs. See RP2350 variants.



Screen snip credit: Raspberry Pi web site



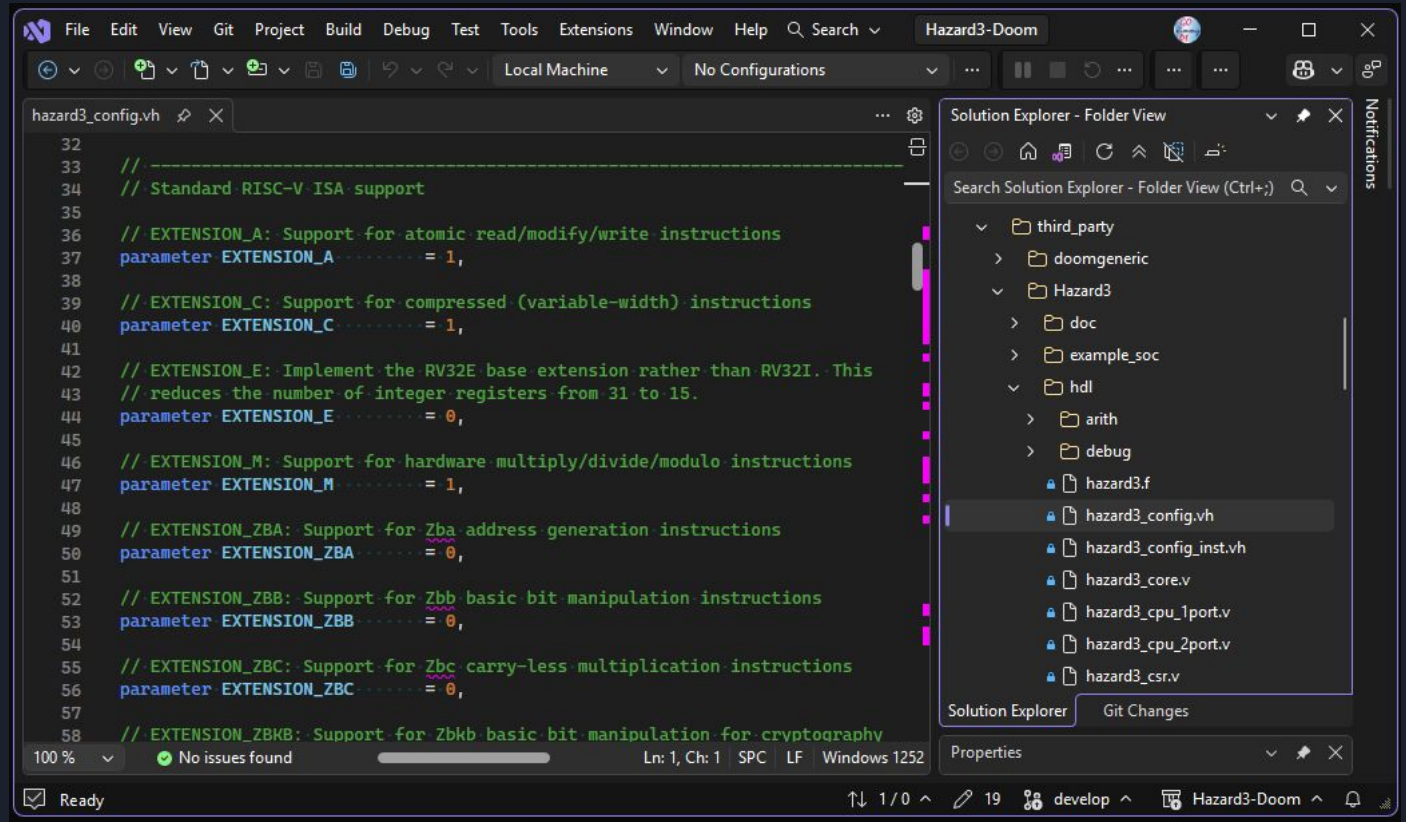
https://github.com/Wren6991/Hazard3

Hazard3 is a 3-stage RISC-V processor, implementing the RV32I or RV32E instruction set and the following optional extensions:

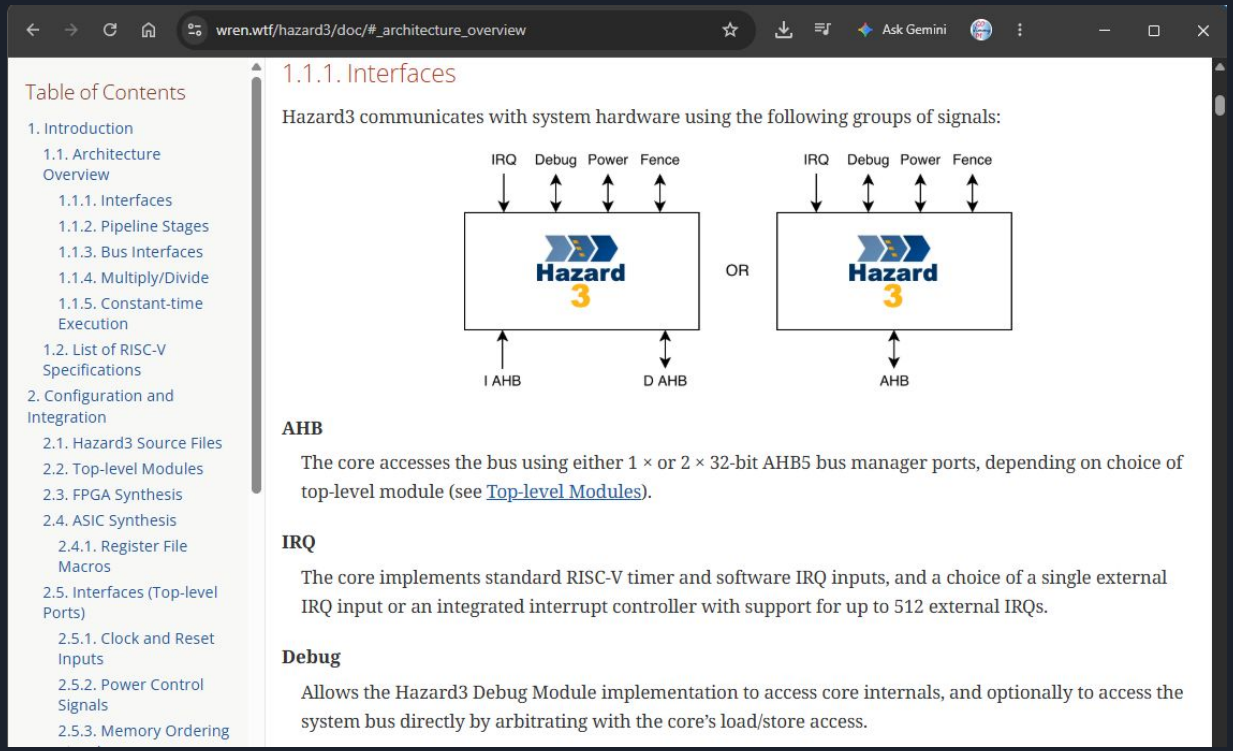
- `M`: integer multiply/divide/modulo
- `A` : atomic memory operations, with AHB5 global exclusives
- `C`: compressed instructions
- `Zba`: address generation
- `Zbb`: basic bit manipulation
- `Zbc`: carry-less multiplication
- `Zbs`: single-bit manipulation
- `Zbkb`: basic bit manipulation for scalar cryptography
- `Zbkx`: crossbar permutation instructions
- `Zicsr`: CSR access
- `Zilsd`: load/store pair instructions
- `Zcb`: basic additional compressed instructions
- `Zclsd`: compressed load/store pair instructions
- `Zcmp`: push/pop instructions
- Debug, Machine and User privilege/execution modes
- Privileged instructions `ecall`, `ebreak`, `mret` and `wfi`
- Physical memory protection (PMP) with up to 16 regions (configurable support for NAPOT and/or TOR matching)
- External debug support (JTAG or APB)
- Instruction address trigger unit (hardware breakpoints)



There's a config for that



Documentation!



Screen snip from https://wren.wtf/hazard3/doc/

What size FPGA?



Lattice primitive	Hazard3-Only Description	Count
LUT4	4-input lookup table	4,540
PFUMX	LUT-output multiplexer	1,047
L6MUX21	Multiplexer for wider LUT logic	351
CCU2C	Fast carry-chain arithmetic cell	255
MULT18X18D	Dedicated 18x18 multiplier	4

2x PicoRV32!



Doom Source Code - Many Options!

https://github.com/id-Software/DOOM
https://github.com/insane-adding-machines/DOOM
https://github.com/maximevince/fbDOOM
https://github.com/ozkl/doomgeneric

This implementation on an ECP5 FPGA:
https://github.com/ulx3s/doomgeneric
https://github.com/ulx3s/Hazard3-Doom

Others:
https://github.com/chocolate-doom/chocolate-doom
https://github.com/coelckers/prboom-plus
https://github.com/floppes/stm32doom
https://github.com/kilograham/rp2040-doom
https://github.com/leos313/DOOM_FPGA
... and more



Doom Image Credit: ID Software

Basic Doom Port

README.md

porting

Create a file named doomgeneric_yourplatform.c and just implement these functions to suit your platform.

- DG_Init
- DG_DrawFrame
- DG_SleepMs
- DG_GetTicksMs
- DG_GetKey

Functions	Description
DG_Init	Initialize your platfrom (create window, framebuffer, etc...).
DG_DrawFrame	Frame is ready in DG_ScreenBuffer. Copy it to your platform's screen.
DG_SleepMs	Sleep in milliseconds.
DG_GetTicksMs	The ticks passed since launch in milliseconds.
DG_GetKey	Provide keyboard events.
DG_SetWindowTitle	Not required. This is for setting the window title as Doom sets this from WAD file.

README.md

main loop

At start, call doomgeneric_Create().

In a loop, call doomgeneric_Tick().

In simplest form:

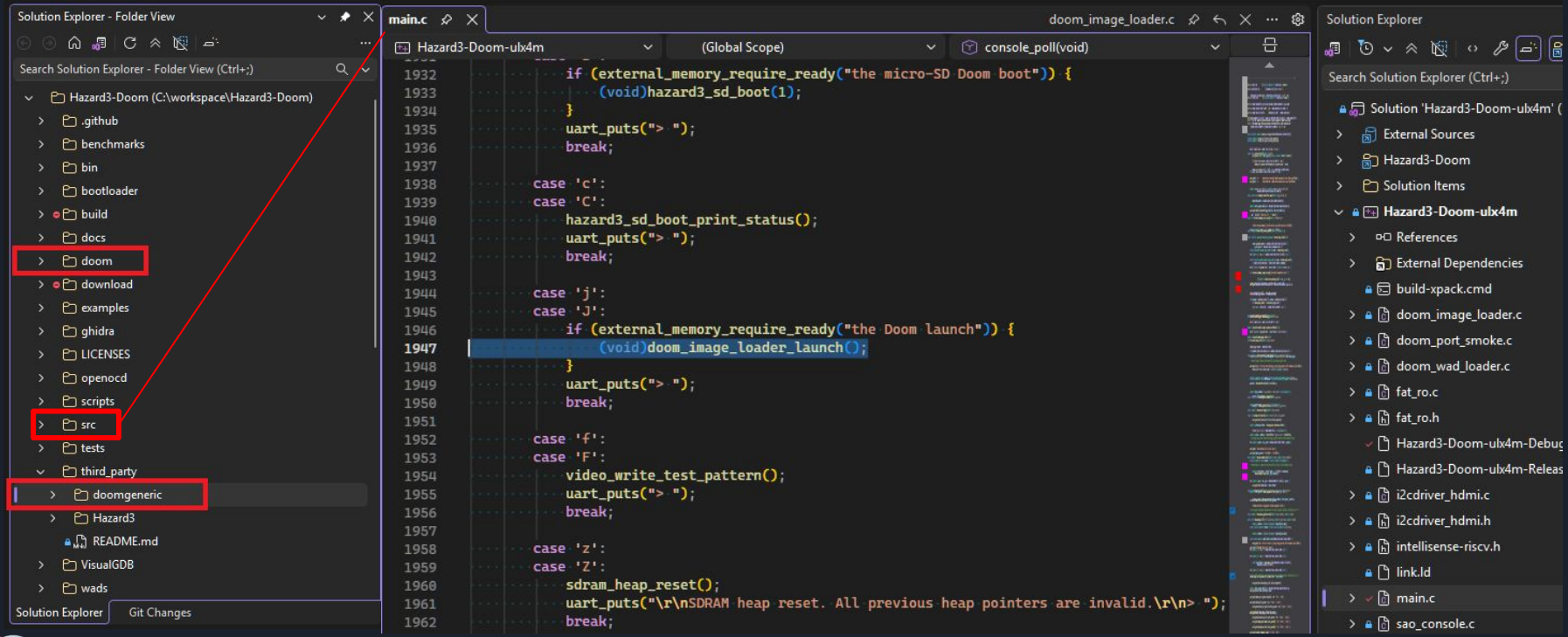
```
int main(int argc, char **argv)
{
    doomgeneric_Create(argc, argv);

    while (1)
    {
        doomgeneric_Tick();
    }

    return 0;
}
```

From README file: https://github.com/ulx3s/doomgeneric

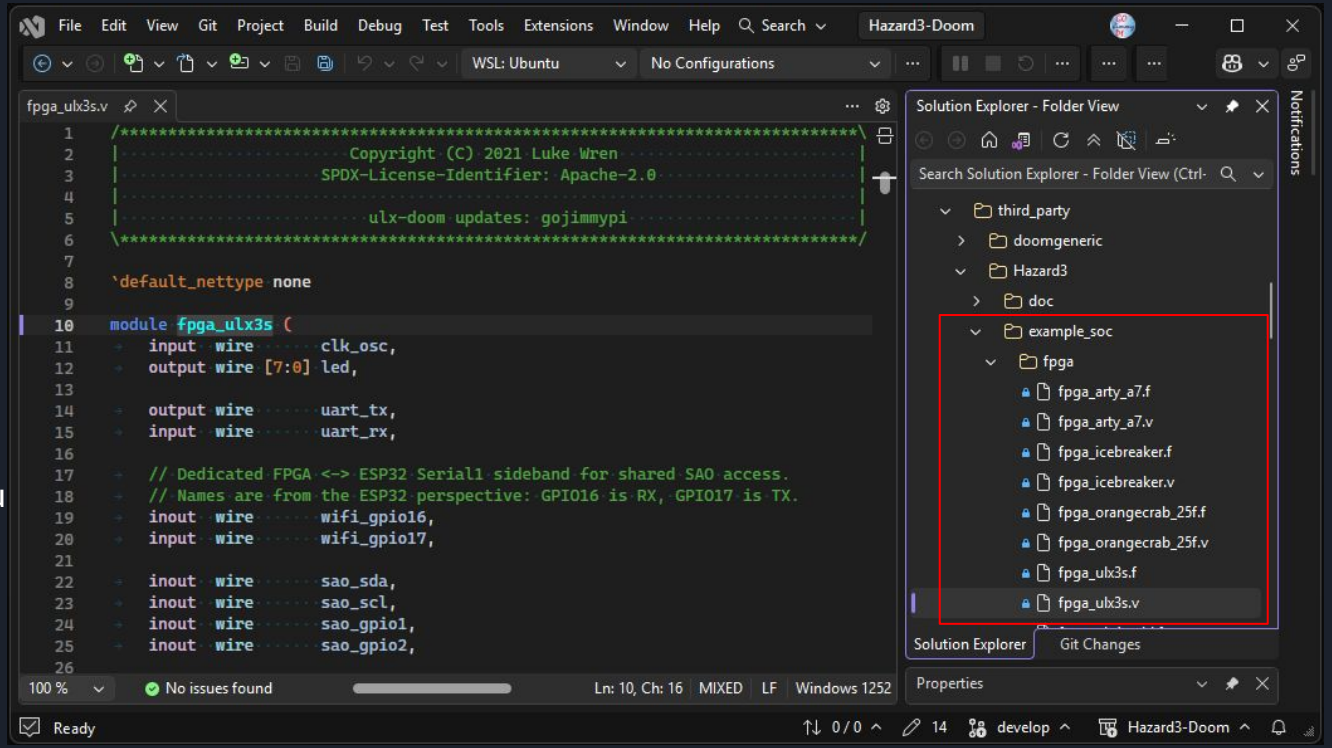
Modified Fork + Local Implementation



GO Jimmy PI

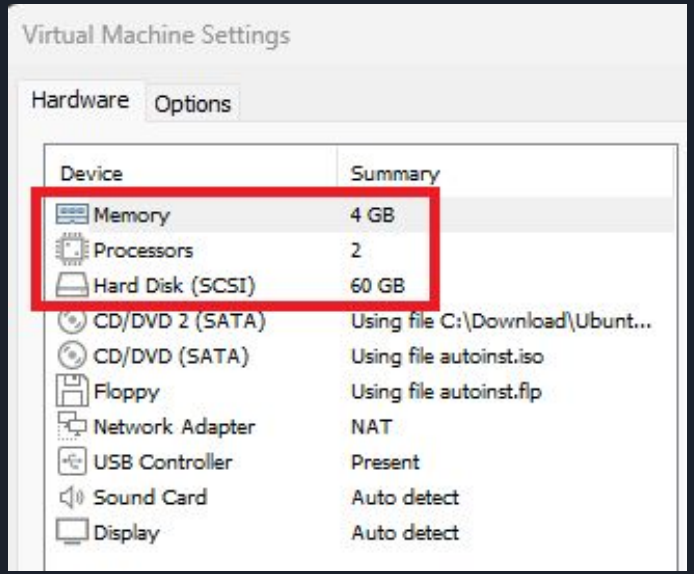
Development Environment

- Windows 11
- WSL Ubuntu
- Visual Studio
- Verilog Extension
- VisualGDB
- Tested on Native Ubuntu
- yosys / nextpnr

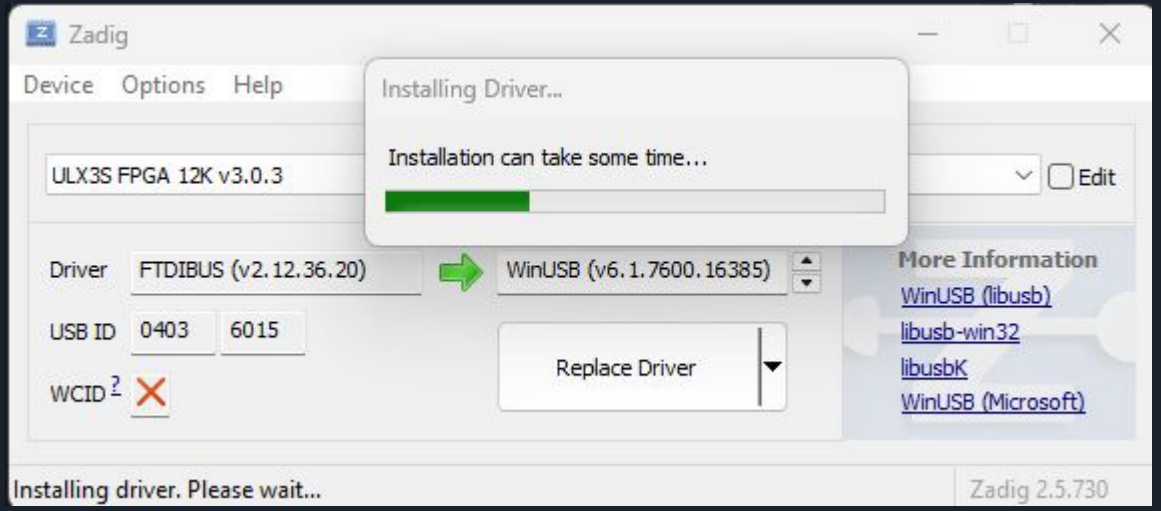


Flashing: put code on the device

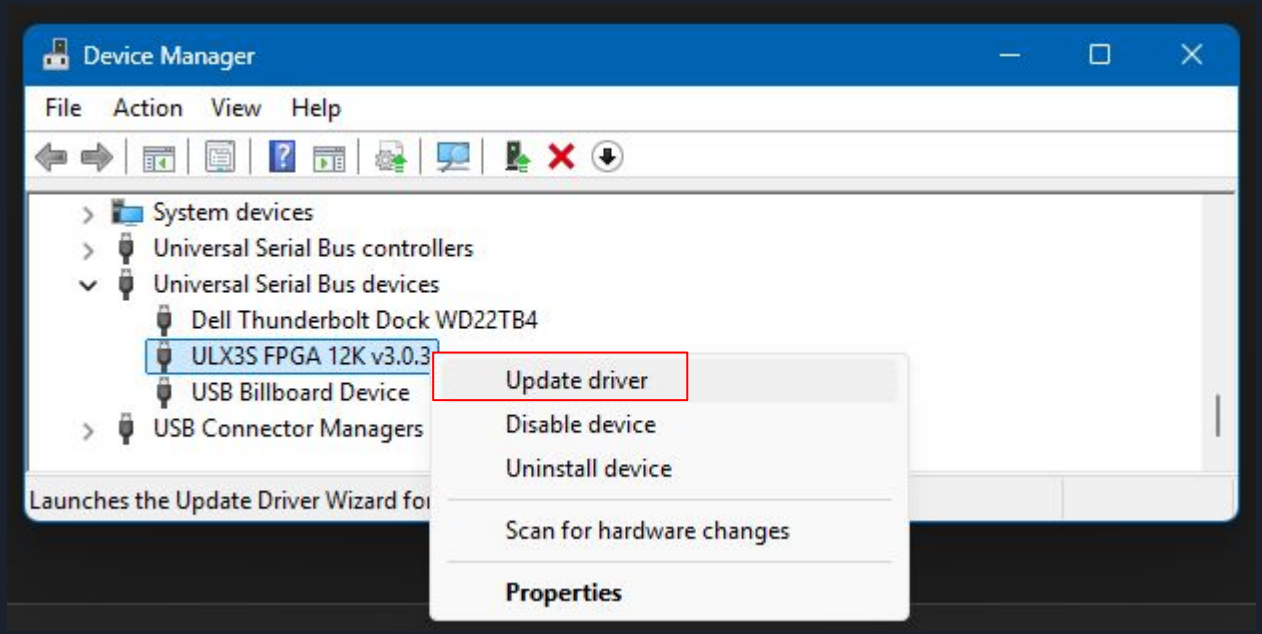
- USB Port
- JTAG



Replace the drivers on computer



Easily Restore (Windows Updates does this, too)



Different Drivers, Different Capabilities

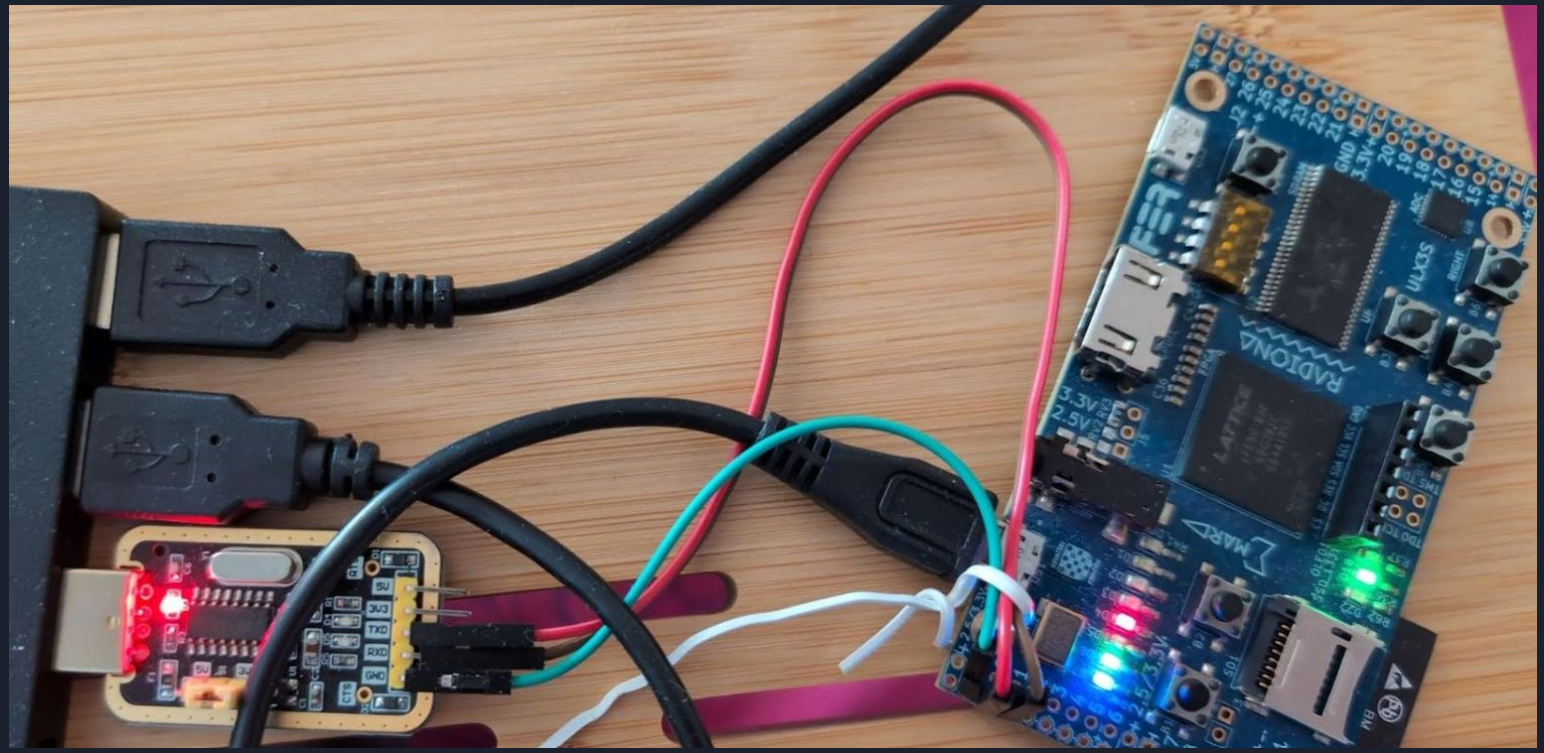


Tool / path	WinUSB	FTDI VCP/D2XX	libusbK
OpenOCD (ULX3S FT231X JTAG)	Works	No	Works
GDB through OpenOCD	Works	No OpenOCD transport	Works
Hazard3-Doom WebUSB FPGA/JTAG flasher	Works	No	No
Windows fujprog / FTDI D2XX tools	No	Works	No
Hazard3-Doom Web Serial UART (external CH340/CH341, CP210x, FTDI UART, etc.)	N/A	N/A	N/A
PuTTY / normal COM-port UART on the external adapter	N/A	N/A	N/A

See docs: https://hazard3-doom.readthedocs.io/en/latest/user-guide/web-flasher.html

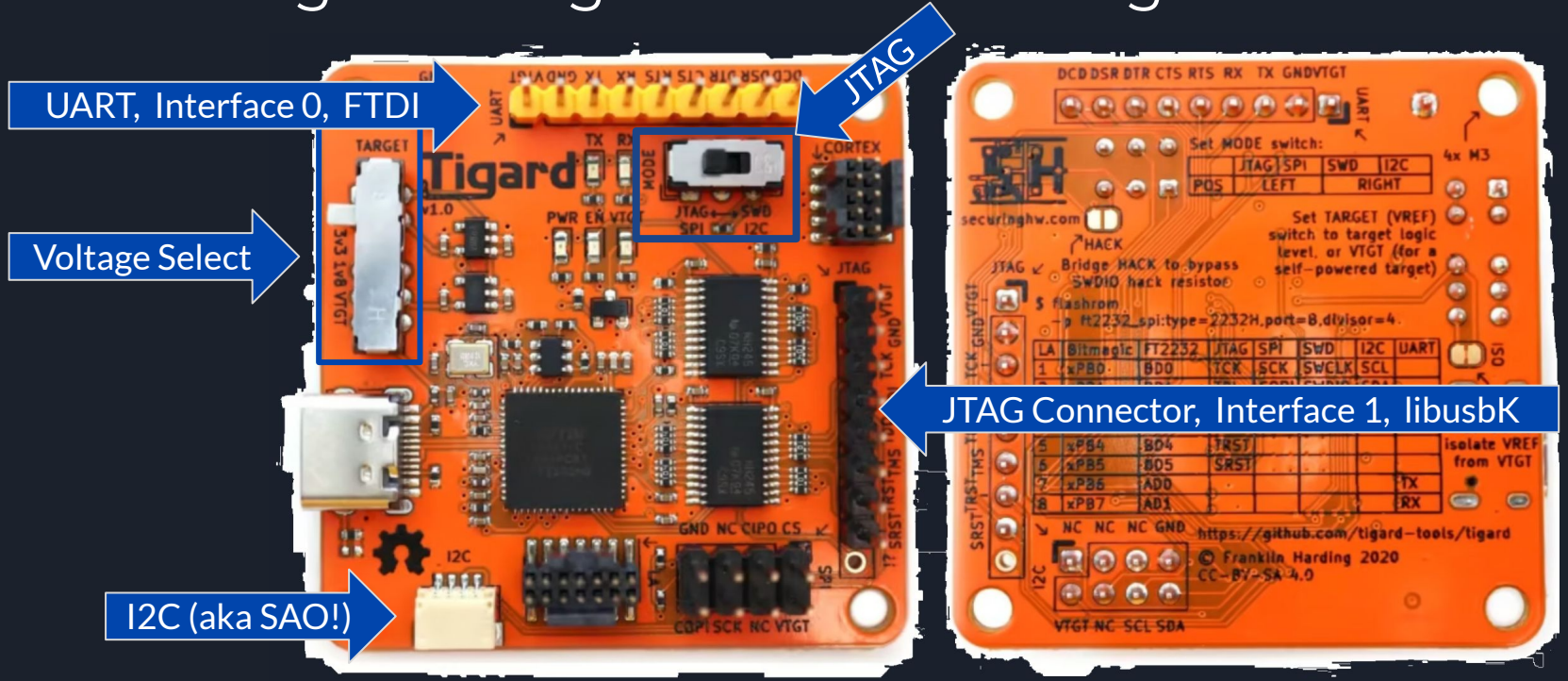


ULX3S FTDI for bit-bang JTAG



GO Jimmy PI

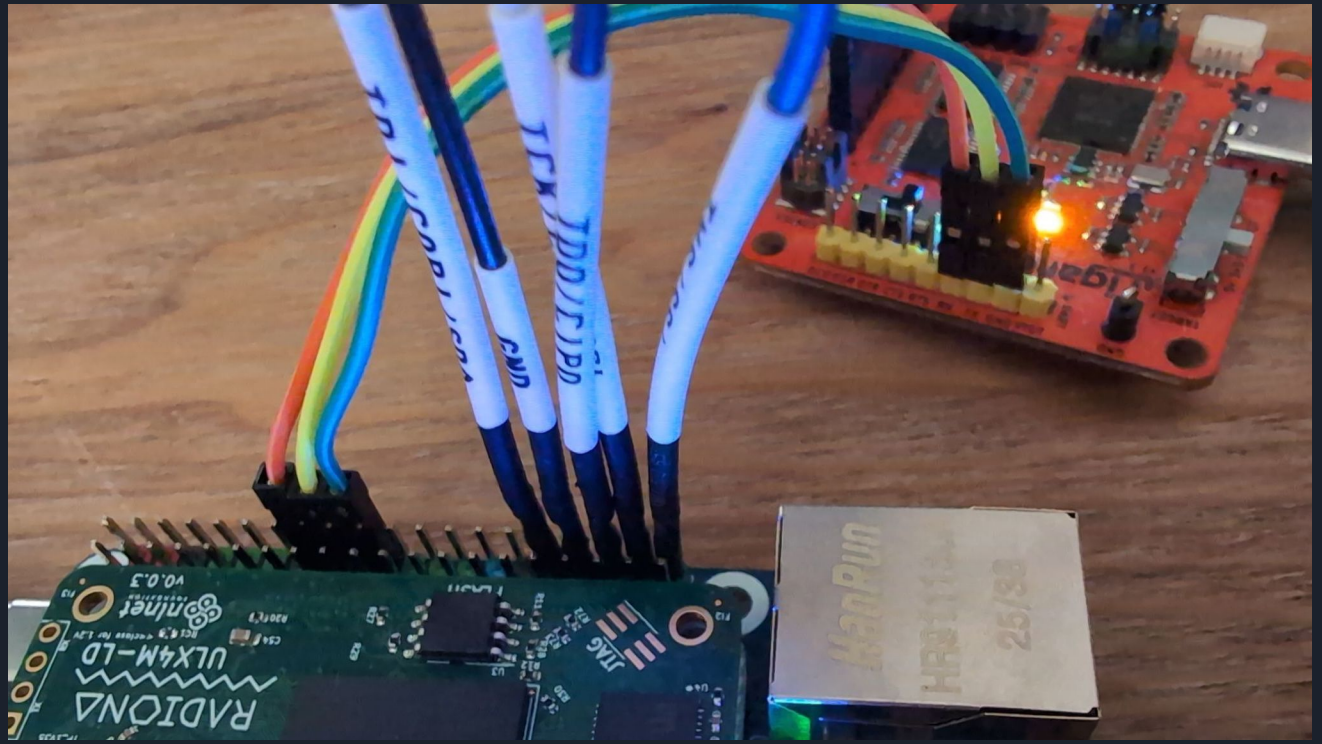
Tigard: Program without using USB Serial



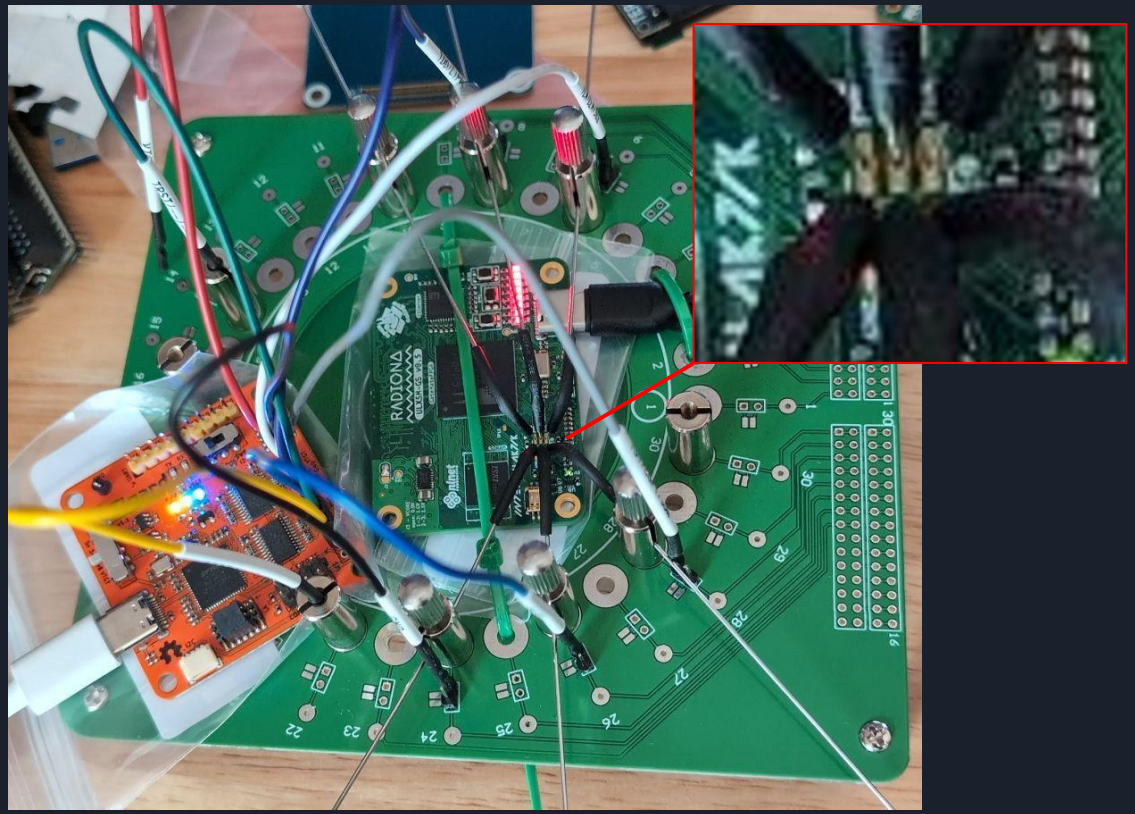
https://github.com/tigard-tools/tigard

Image credit: https://www.crowdsupply.com/securinghw/tigard

CM4 Carrier RPi header to Tigard JTAG

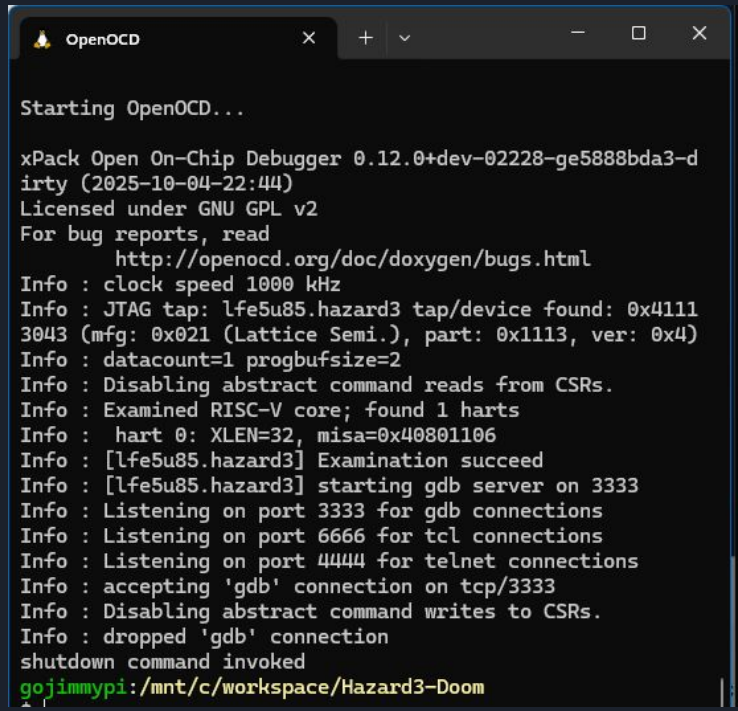


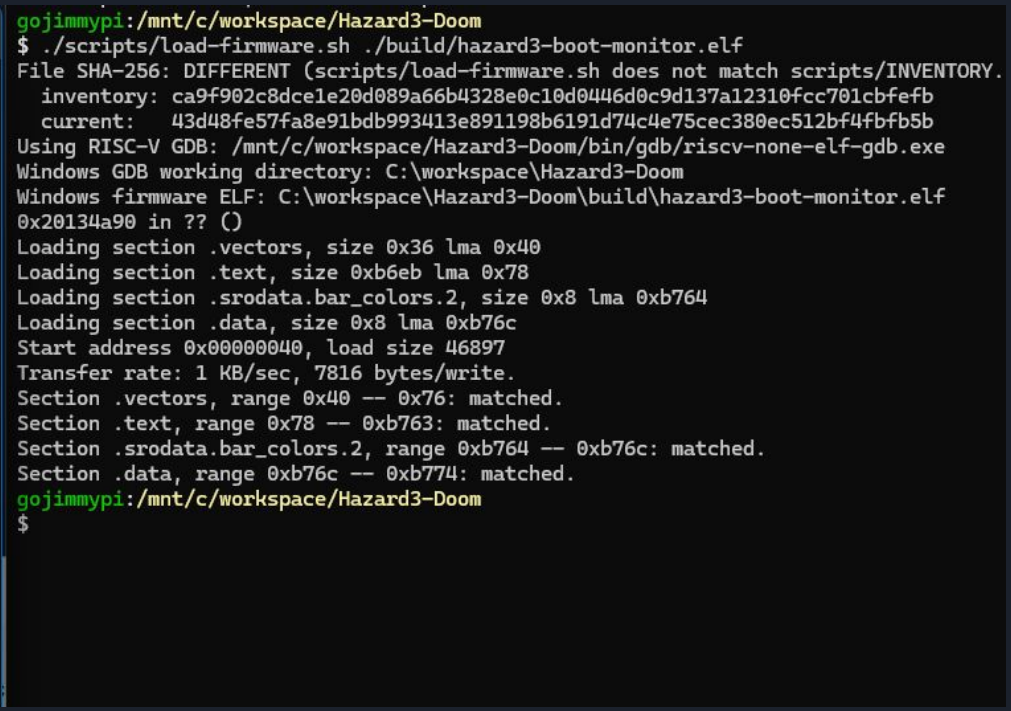
Spider Board Jig - for the tiny connectors



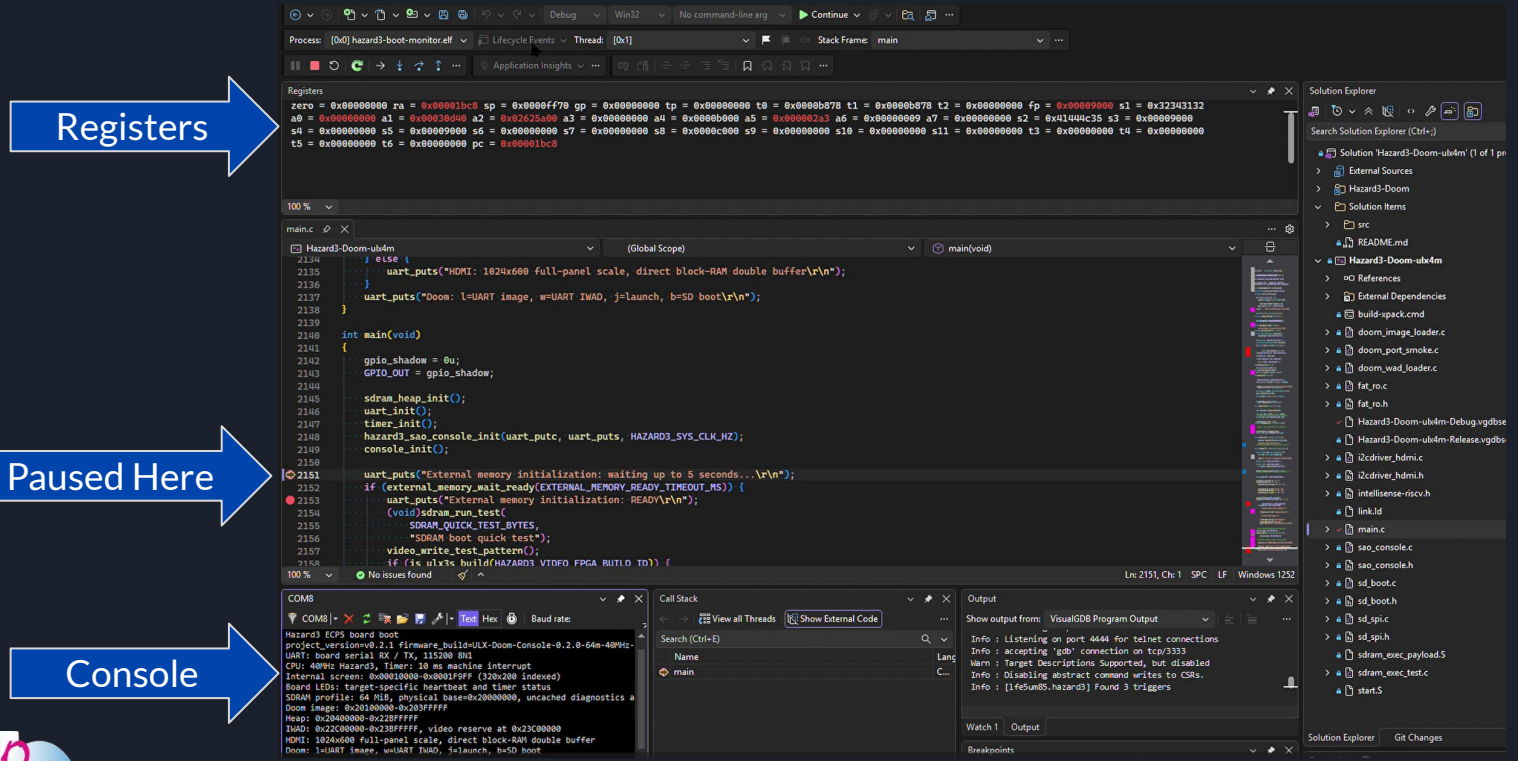


OpenOCD and GDB to load code





VisualGDB


Registers
Paused Here
Console
zero = 0x00000000 ra = 0x00001bc8 sp = 0x0000ff70 gp = 0x00000000 tp = 0x00000000 t0 = 0x0000b078 t1 = 0x0000b078 t2 = 0x00000000 fp = 0x00000000 s1 = 0x323432132
a0 = 0x00000000 a1 = 0x00030d40 a2 = 0x02625a00 a3 = 0x00000000 a4 = 0x0000b000 a5 = 0x000002a3 a6 = 0x00000009 a7 = 0x00000000 s2 = 0x41444c35 s3 = 0x00000000
s4 = 0x00000000 s5 = 0x00009000 s6 = 0x00000000 s7 = 0x00000000 s8 = 0x0000c000 s9 = 0x00000000 s10 = 0x00000000 s11 = 0x00000000 t3 = 0x00000000 t4 = 0x00000000
t5 = 0x00000000 t6 = 0x00000000 pc = 0x00001bc8
main.c
Hazard3-Doom-ulx4m
(Global Scope)
main(void)
uart_puts("HDMI: 1024x600 full-panel scale, direct block-RAM double buffer\r\n");
uart_puts("Doom: l=UART image, w=UART IWAD, j=launch, b=SD boot\r\n");
int main(void)
gpio_shadow = 0u;
GPIO_OUT = gpio_shadow;
sdram_heap_init();
uart_init();
timer_init();
hazard3_sao_console_init(uart_putc, uart_puts, HAZARD3_SYS_CLK_HZ);
console_init();
uart_puts("External memory initialization: waiting up to 5 seconds...\r\n");
if (external_memory_wait_ready(EXTERNAL_MEMORY_READY_TIMEOUT_MS)) {
uart_puts("External memory initialization: READY\r\n");
(void)sdram_run_test(
SDRAM_QUICK_TEST_BYTES,
"SDRAM boot quick test");
video_write_test_pattern();
No issues found
Ln: 2151, Ch: 1
Windows 1252
COM8
Baud rate:
Hazard3 ECP5 board boot
project_version=v0.2.1 firmware_build=ULX-Doom-Console-0.2.0-64m-40MHz-
UART: board serial RX / TX, 115200 8N1
CPU: 40MHz Hazard3, Timer: 10 ms machine interrupt
Internal screen: 0x00010000-0x0001F9FF (320x200 indexed)
Board LEDs: target-specific heartbeat and timer status
SDRAM profile: 64 MiB, physical base=0x20000000, uncached diagnostics a
Doom image: 0x20100000-0x203FFFFF
Heap: 0x20400000-0x22BFFFFF
IWAD: 0x22C00000-0x23BFFFFF, video reserve at 0x23C00000
HDMI: 1024x600 full-panel scale, direct block-RAM double buffer
Doom: l=UART image, w=UART IWAD, j=launch, b=SD boot
Call Stack
View all Threads
Show External Code
Search (Ctrl+E)
Name
main
Output
Show output from: VisualGDB Program Output
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
Warn : Target Descriptions Supported, but disabled
Info : Disabling abstract command writes to CSRs.
Info : [1fe5um85.hazard3] Found 3 triggers
Watch 1
Output
Breakpoints
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Hazard3-Doom-ulx4m' (1 of 1 pr
External Sources
Hazard3-Doom
Solution Items
src
README.md
Hazard3-Doom-ulx4m
References
External Dependencies
build-xpack.cmd
doom_image_loader.c
doom_port_smoke.c
doom_wad_loader.c
fat_ro.c
fat_ro.h
Hazard3-Doom-ulx4m-Debug.vgdbse
Hazard3-Doom-ulx4m-Release.vgdbs
i2cdriver_hdmi.c
i2cdriver_hdmi.h
intellisense-riscv.h
link.ld
main.c
sao_console.c
sao_console.h
sd_boot.c
sd_boot.h
sd_spi.c
sd_spi.h
sdram_exec_payload.S
sdram_exec_test.c
start.S
Solution Explorer
Git Changes

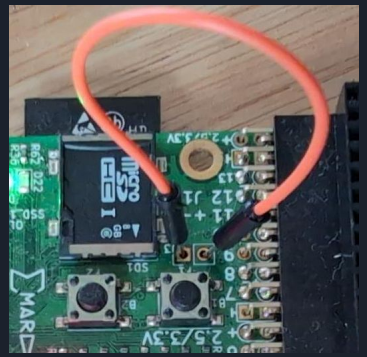
SD Card to load apps: Shared ESP32 Pins!

Yay! There's an ESP32 on the ULX3S

But ...

The ESP32 lives *behind* the FPGA;
pass-thru RTL needed.

The ESP32 *shares control* with the SD Card



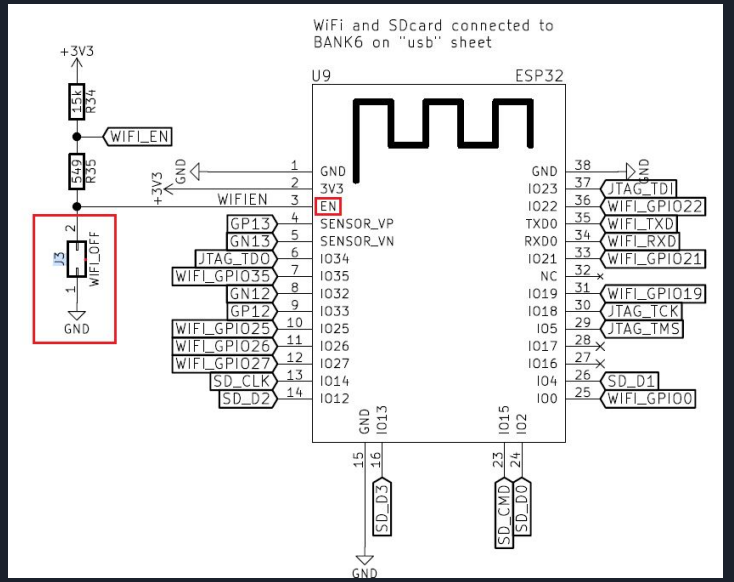


Image credit: emard's ULX3S schematic

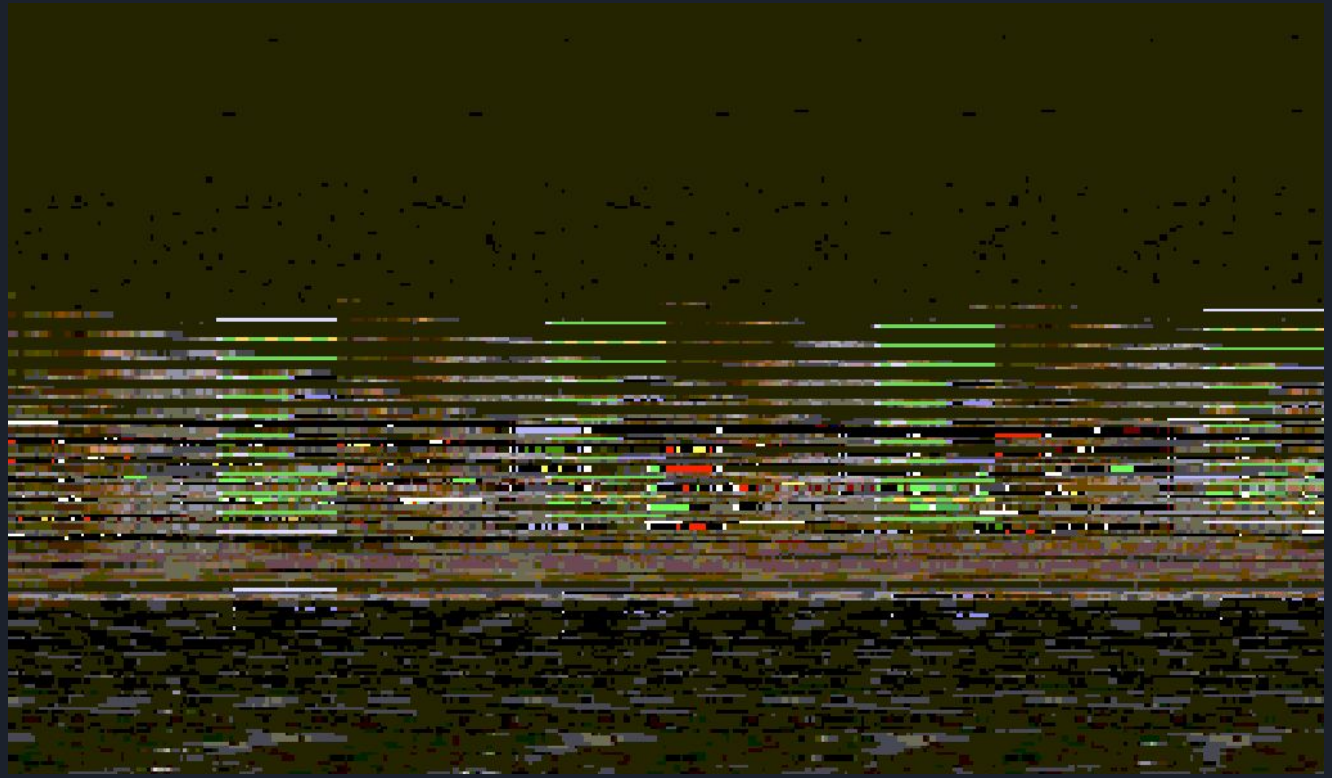
Initial Doom: Unreasonably Slow



Video Performance: the endless Rabbit Hole!



Display… “*challenges*” or “**opportunities**!”



FPGA Size: HMDI (ULX3S GPDI connector)



Lattice primitive	HDMI-Only Description	Count
LUT4	4-input lookup table	2,030
CCU2C	Fast carry-chain arithmetic cell	257
DP16KD	Dedicated 18-Kbit block RAM	96
PFUMX	LUT-output multiplexer	85
ODDRX1F	DDR output register / serializer	4



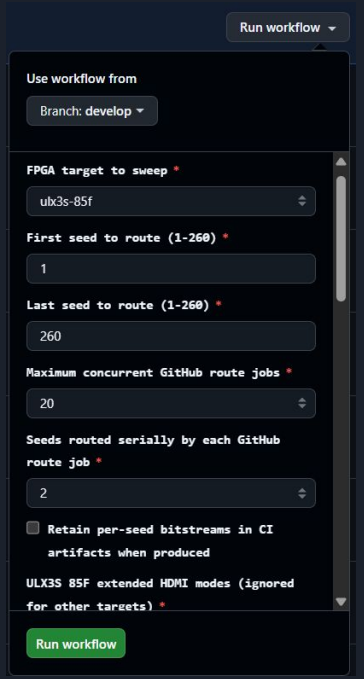
FPGA Size: Total ULX3S Hazard3 Doom Utilization



Lattice Primitive	Description	Count
LUT4	4-input lookup table	10,308
PFUMX	Multiplexer	1,599
CCU2C	Fast carry-chain arithmetic cell	828
L6MUX21	Multiplexer for wider LUT logic	360
DP16KD	18-Kbit dual-port block RAM	195
MULT18X18D	Dedicated 18x18 multiplier	4
ODDRX1F	Output DDR register	5
EHXPLLL	PLL clock-generation block	2
JTAGG	Built-in JTAG interface	1



Timing Challenges



	A	B	C
1	**Seed**	**Routing Time**	**Frequency MHz**
2	20	162	50
3	7	183	51
91	88	21,233	52
92	144	25,407	51
93	135	32,955	**53**

Synthesis Helpers for nextpnr Routing



```
Release Versions! Device Type!!

--freq 48

--seed 20

--placer heap

--placer-heap-timingweight 30

--placer-heap-critexp 3

--report timing.json

--detailed-timing-report
```

Otherwise look for "`Critical path report`" text in logs for design change location ideas



Adding ULX4M Support: 85F with DDR3



- Working on the ULX3S with ECP5-85F
- Plenty of RAM available in the DDR3

>> should be a no-brainer on the ULX4M, right?



Memory.

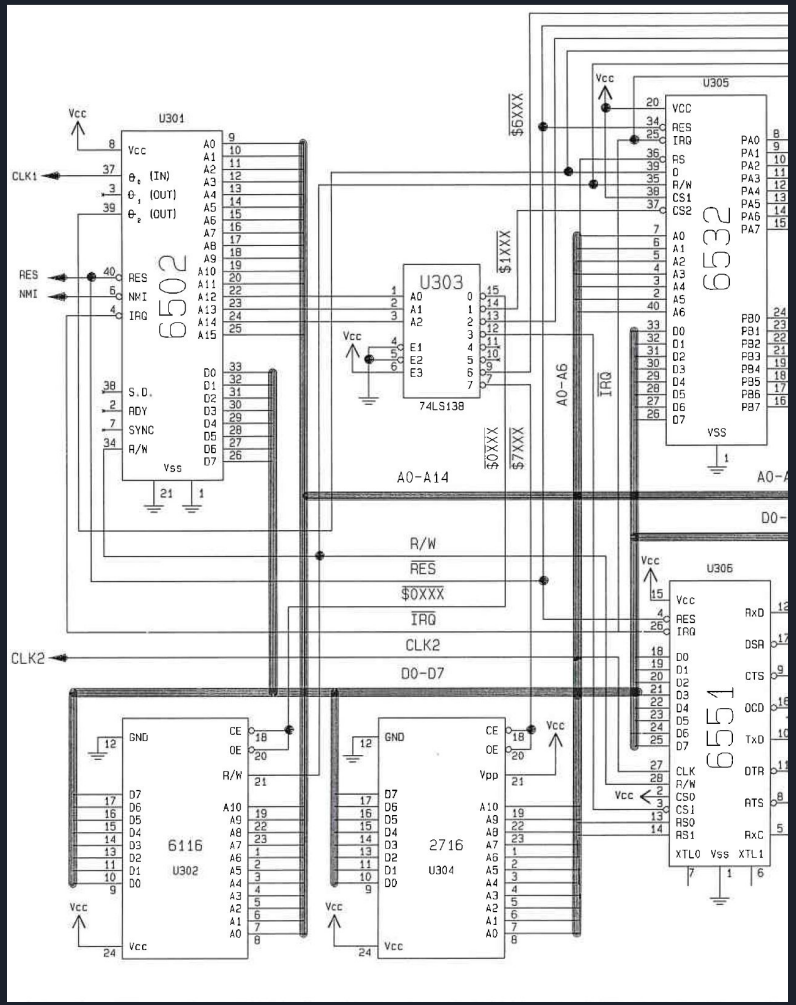
once

so

easy

to

connect



Then came DDR memory

Generations of DDR Memory

- **DDR1 (2000):** Introduced double-rate data transfer at higher voltages (2.5V) and lower speeds.
- **DDR2 (2004):** Lowered voltage to 1.8V and increased prefetch buffer size for better efficiency.
- **DDR3 (2007):** Reduced voltage to 1.5V and became a long-standing mainstream standard.
- **DDR4 (2014):** Standardized modern computing with a 1.2V operating voltage and higher capacities (such as Corsair Vengeance LPX DDR4).
- **DDR5 (Current):** Features dual sub-channels, on-module power management (PMIC), and extreme speeds (such as Corsair Vengeance RGB DDR5).

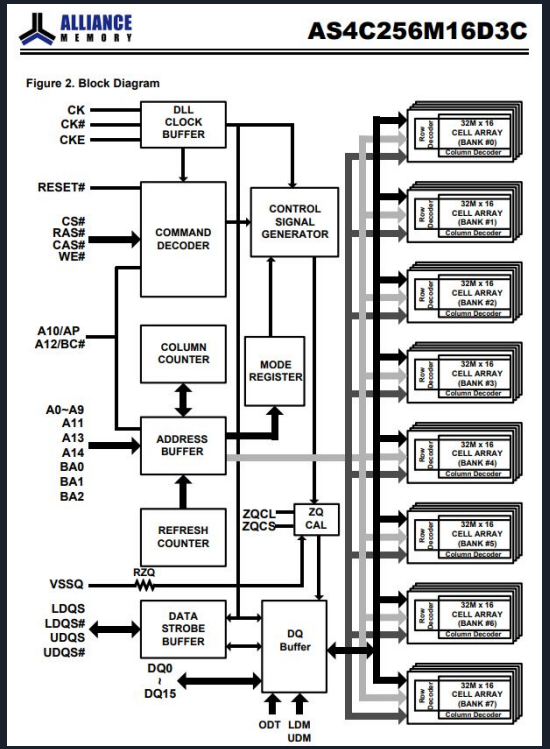
Credit: google search summary

DDR memory is more complex



Image credit: screen snip from https://en.wikipedia.org/wiki/DDR_SDRAM

DDR3 Memory



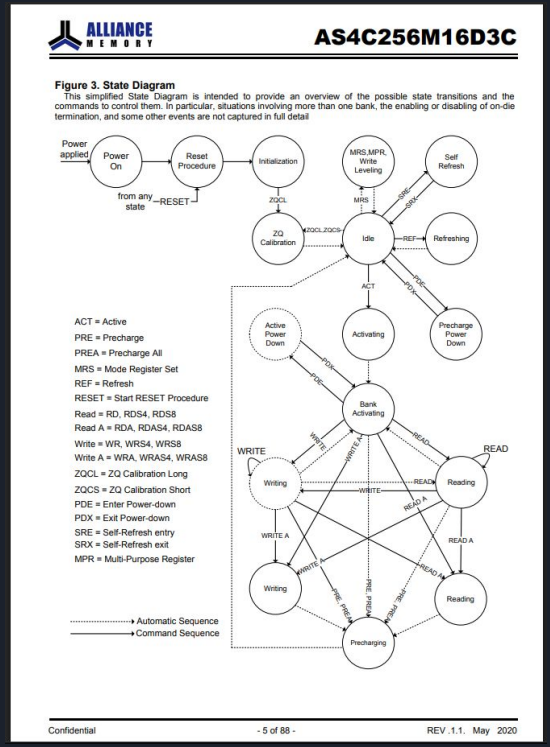


Image credit: https://www.alliancememory.com/wp-content/uploads/AllianceMemory_4G_DDR3_AS4C256M16D3C_May2020_Rev1.1_Final.pdf

FPGA - More than just LUTs!

ALU54B - 54-bit ternary adder/subtractor (limited support, must be manually placed)	**LUT4** - 4 input Look Up Table
CCU2C - 2x LUT4 with carry logic	**MULT18X18D** - DSP multiplier (cascade functionality not supported)
CLKDIVF - clock divider	**ODDR71B** - 7:1 ODDR implementation
DCUA - dual channel 3/5Gbps SERDES	**ODDRX1F** - generic 2:1 output DDR
DDRDLLA - 90° delay for DQS or clock for DDR interface	**ODDRX2DQA** - memory 4:1 output DDR primitive for DQ outputs
DELAYF - configurable signal delay	**ODDRX2DQSB** - memory 4:1 output DDR primitive for DQS outputs
DELAYG - simple signal delay	**ODDRX2F** - generic 4:1 output DDR
DP16KD - true dual port block RAM	**OSCG** - internal oscillator
DQSBUFM - DQS control for DDR memory	**OSHX2A** - memory 4:1 output DDR primitive for address and command
DTR - die temperature readout	**PCSCLKDIV** - clock divider for SERDES clock outputs
ECLKSYNCB - edge clock stop/synchronisation	**PFUMX** - 2 input multiplexer for LUT5s
EHXPLLL - phase-locked-loop	**SEDGA** - allows checking configuration data for soft-errors, see TN1268 (untested)
EXTREFB - reference clock input buffer for external clock for SERDES	**TRELLIS_DPR16X4** - 16x4 LUTRAM
GSR - global set/reset interface	**TRELLIS_ECLKBUF** - internal edge clock buffer
IDDR71B - 7:1 input gearbox	**TRELLIS_FF** - SLICE flipflop
IDDRX1F - generic 1:2 input DDR	**TRELLIS_IO** - tristate IO, mapped by Yosys from BB, IB, OB, etc
IDDRX2DQA - DDR2/3 memory 1:4 input interface	**TRELLIS_SLICE** - logic SLICE (2x LUT4, 2x FF, 2x MUX2, CCU2 carry)
IDDRX2F - generic 1:4 input DDR	**TSHX2DQA** - tristate control for DQ data output for DDR2 and DDR3 memory
JTAGG - JTAG access to fabric	**TSHX2DQSA** - tristate control for DQS output
L6MUX21 - 2 input multiplexer for LUT6 and above	**USRMCLK** - provides access to SPI flash clock (MCLK)

Credit: yosysHQ https://github.com/YosysHQ/nextpnr/blob/main/ecp5/docs/primitives.md

DDR3 SDRAM Controller IP Core diagram for ECP5 and LatticeECP3 devices

The DDR3 memory controller consists of 3 sub modules:

- Memory Controller (MC)
- Physical Interface (PHY)
- Clock Synchronization (CSM)

Contact Sales

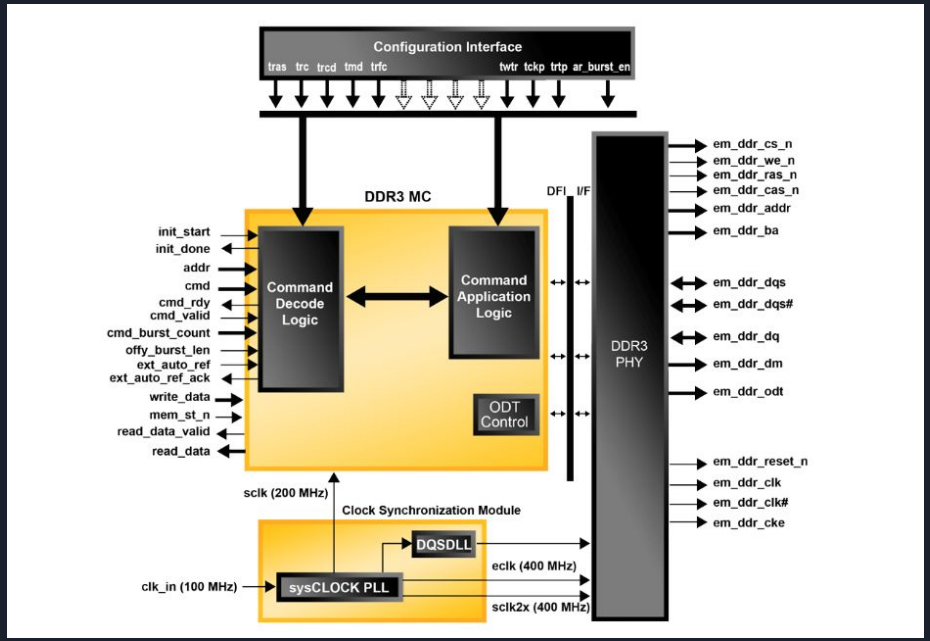


Image credit: https://www.latticesemi.com/en/products/designsoftwareandip/intellectualproperty/ipcore/ipcores01/ddr3sdramcontroller

Generate Verilog Code

Name	Build Status	Description
LiteX-Boards	ci passing	Boards support
LiteDRAM	ci passing	DRAM
LiteEth	ci passing	Ethernet
LitePCIe	ci passing	PCIe
LiteNVMe	ci passing	NVMe
LiteSATA	ci passing	SATA
LiteSDCard	ci passing	SD card
LiteICLink	ci passing	Inter-Chip communication
LiteJESD204B	ci passing	JESD204B
LiteDSP	ci passing	RF/DSP blocks (WIP)
LiteSPI	ci passing	SPI/SPI-Flash
LiteScope	ci passing	Logic analyzer

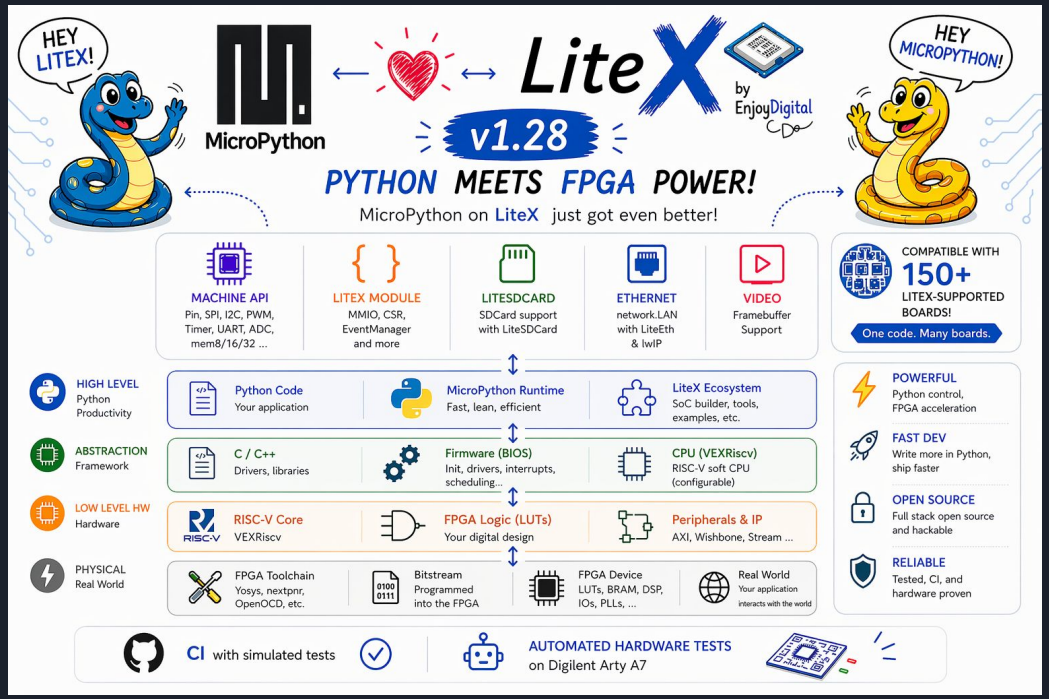
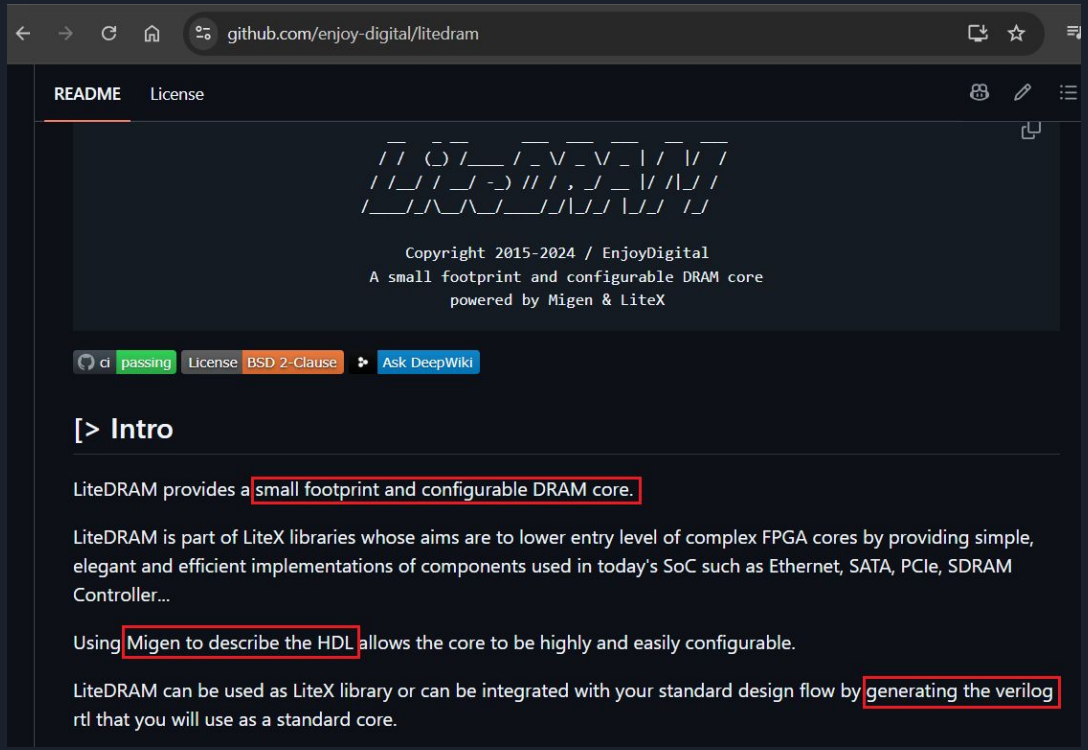


Image credit: LiteX on GitHub: https://github.com/enjoy-digital/

ULX4M DDR3 Controller: LiteX LiteDRAM



Award Winning SERV Core in the LiteDRAM Controller!

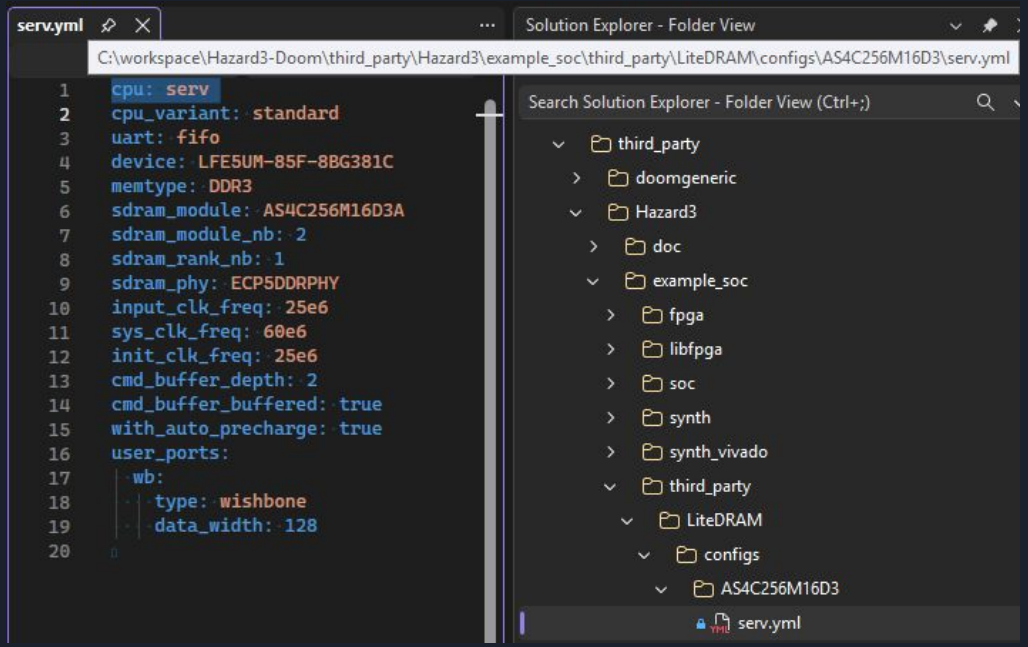




Image credit: Visual Studio screen snip; Olof Kindren, CTO of Qamcom, at RISC-V Summit Germany

SERV is Tiny! LiteDRAM vs Minimal Config



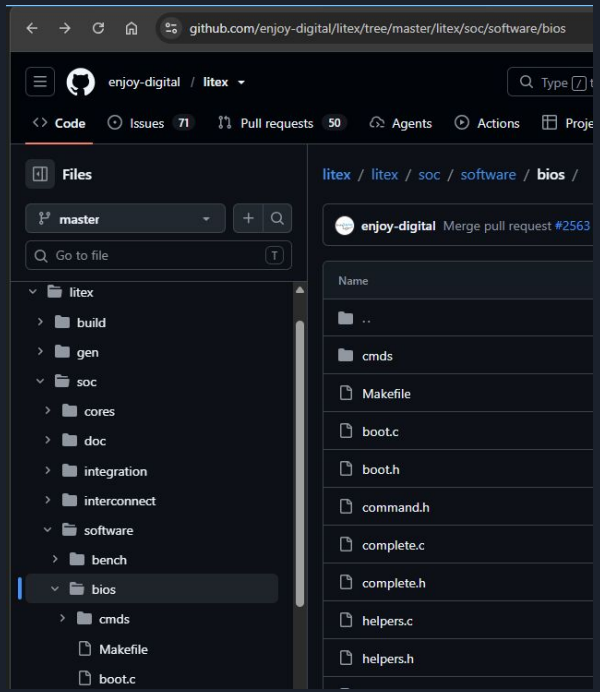
Resource	SERV CPU proper	SERV + register-file support
LUT4	357	404
PFUMX	53	56
CCU2C	3	9
Flip-flops/registers	162	178
DP16KD EBR	0	1

Lattice iCE40	Intel Cyclone 10LP	AMD Artix-7	CMOS
198 LUT	239 LUT	125 LUT	2.1kGE
164 FF	164 FF	164 FF	



Top image: this project; Bottom image from https://github.com/olofk/serv

Software FPGA Initialization - Not RTL




```
294         /* Initialize and test SPIRAM */
295     #ifdef CSR_SPIRAM_BASE
296         spiram_init();
297         printf("\n");
298     #endif
299
300         /* Initialize and test DRAM */
301     #ifdef CSR_SDRAM_BASE
302         sdr_ok = sdram_init();
303     #else
304         /* Test Main RAM when present and not pre-initialized */
305     #ifdef MAIN_RAM_BASE_VA
306     #ifndef CONFIG_MAIN_RAM_INIT
307         sdr_ok = memtest((unsigned int *) MAIN_RAM_BASE_VA, min(MAIN_RAM_SIZE, MEMTEST_DATA_SIZE));
308         memspeed((unsigned int *) MAIN_RAM_BASE_VA, min(MAIN_RAM_SIZE, MEMTEST_DATA_SIZE), false, 0);
309     #endif
310     #endif
311     #endif
312         if (sdr_ok != 1)
313             printf("Memory initialization failed\n");
314     #endif
315
316         /* Initialize and test SPIFLASH */
317     #ifdef CSR_SPIFLASH_BASE
318         spiflash_init();
319         printf("\n");
320     #endif
```

https://github.com/enjoy-digital/litex/tree/master/litex/soc/software/bios

ULX4M with DDR3 Controller FPGA Utilization



Resource	What it is	Used / available	Utilization
Embedded Block RAM (EBR)	Dedicated on-chip memory (`DP16KD`)	**182 / 208**	**87%**
Logic cells	General programmable FPGA logic	**15,502 / 83,640**	**18%**
Flip-flops / registers	Sequential state storage	**5,625 / 83,640**	**6%**
PLLs	Dedicated clock generation (`EHXPLLL`)	**3 / 4**	**75%**
High-speed I/O logic	DDR/serializer/deserializer resources	**49 / 224 sites**	**21%**
Hardware multipliers	Dedicated DSP multipliers (`MULT18X18D`)	**4 / 156**	**2%**



ULX4M Timing Challenges

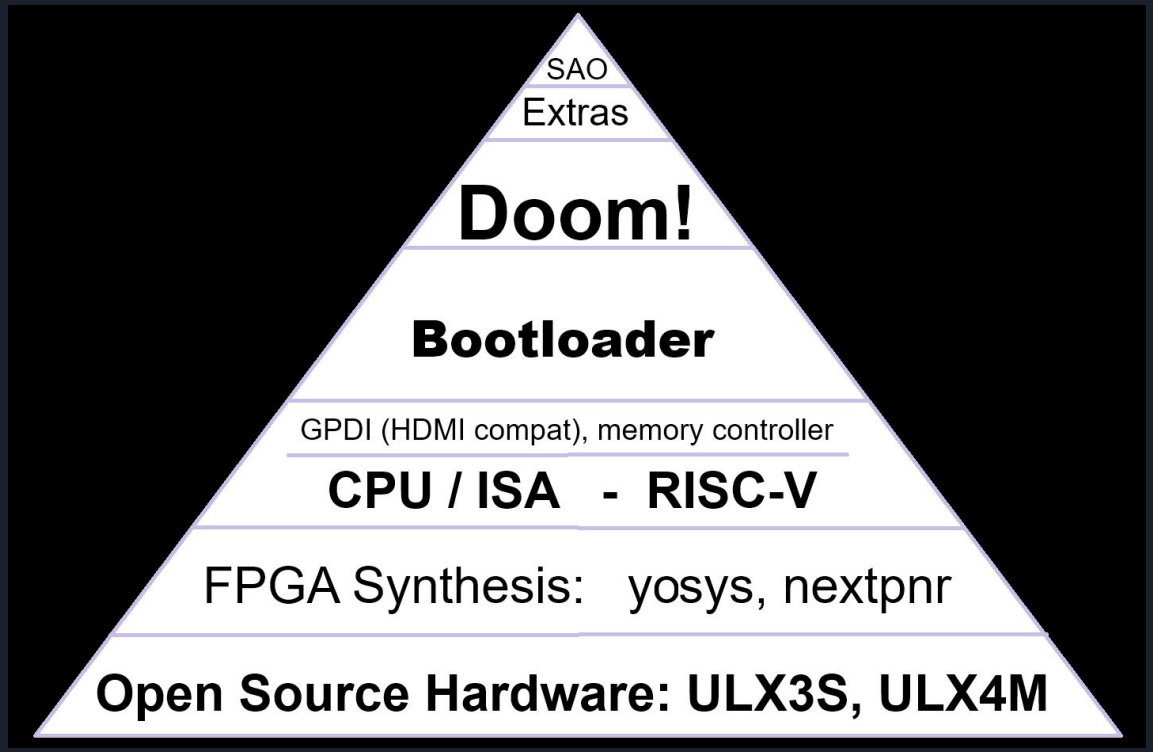


	A	C	E	F
1	seed	route_seconds	clk_sys_mhz	litedram_user_mhz
2	127	173	40.03	66.49
3	233	174	40.04	64.53
166	83	146	43.63	67.51
167	11	135	43.86	63.3
168	45	317	44.31	65.9

Original target: 50 MHz CPU, 75 MHz DDR3



Doom on an FPGA: Hazard3 Doom





https://www.youtube.com/shorts/4TTZ9huWvjI

ECP5 FPGA + Hazard3 RISC-V + HDMI

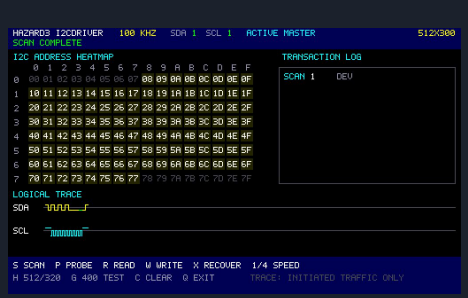


https://github.com/ulx3s/Hazard3-Doom

v0.2.0 uses UART for keyboard. Panel arrow buttons in next release.



Bonus Material



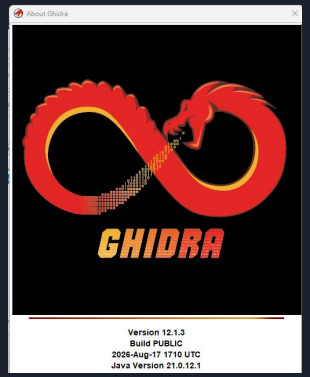
A working minimal example is the Hazard3-Doom tt-fpga-ulx workflow.

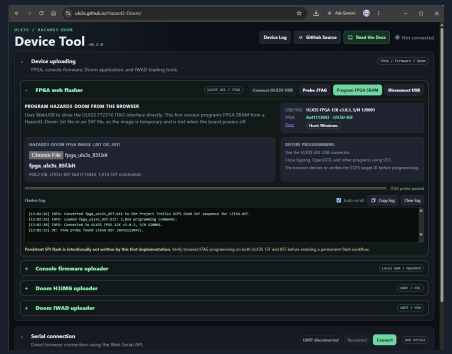
Add a job step such as this to the workflow file:

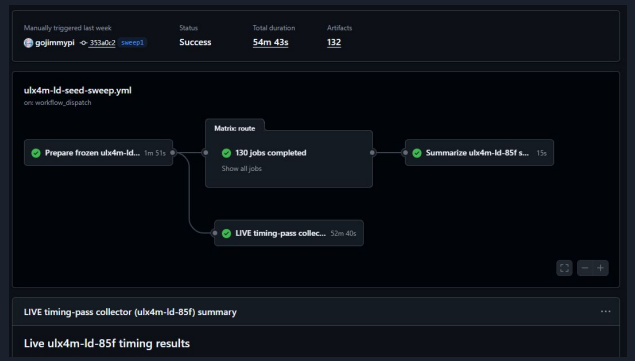
```yaml
jobs:
  fpga-ulx3s:
    runs-on: ubuntu-24.04
    steps:
      - name: checkout repo
        uses: actions/checkout@v7
        with:
          submodules: recursive

      - name: FPGA bitstream for TT ASIC Sim (ULX3S ECP5)
        uses: ulx3s/tt-gds-action/fpga/ulx3s@experimental
        with:
          ecp5-device: 85k
          lpf: tt/fpga/ulx3s/ulx3s_v316.lpf
          artifact-name: fpga_ulx3s_ecp5
          uart-enabled: true
```









GO Jimmy PI

Quick Start - Install Toolchain Requirements

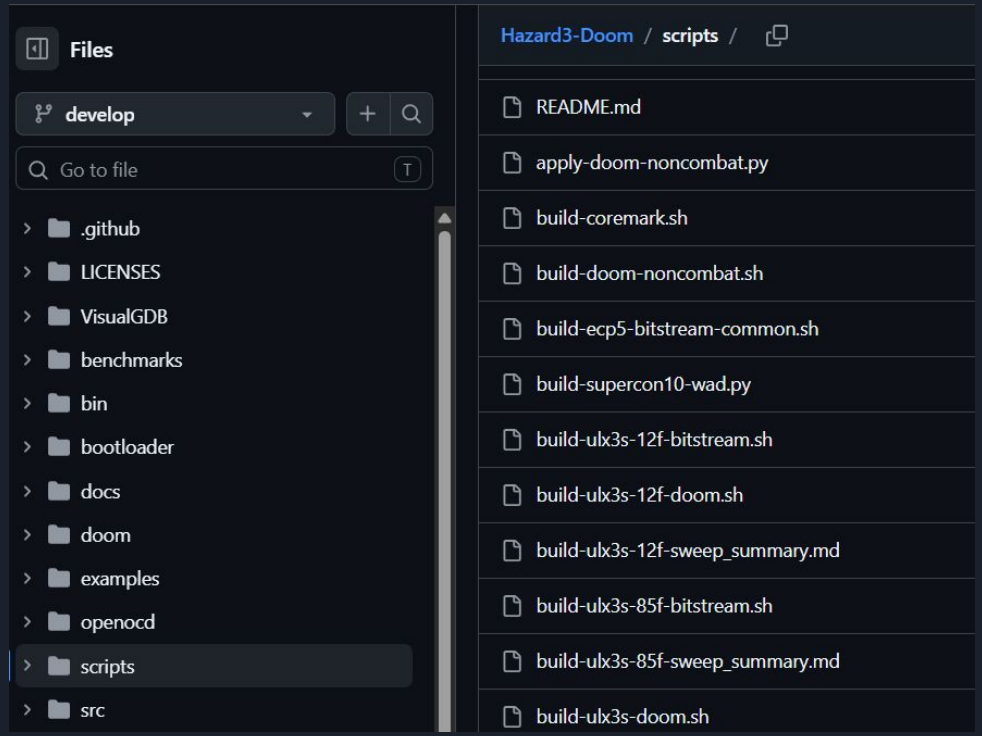


```
wget -L \
    https://raw.githubusercontent.com/ulx3s/Hazard3-Doom/main/scripts/full-install.sh \
    -O ./full-install.sh
chmod +x ./full-install.sh
./full-install.sh
```

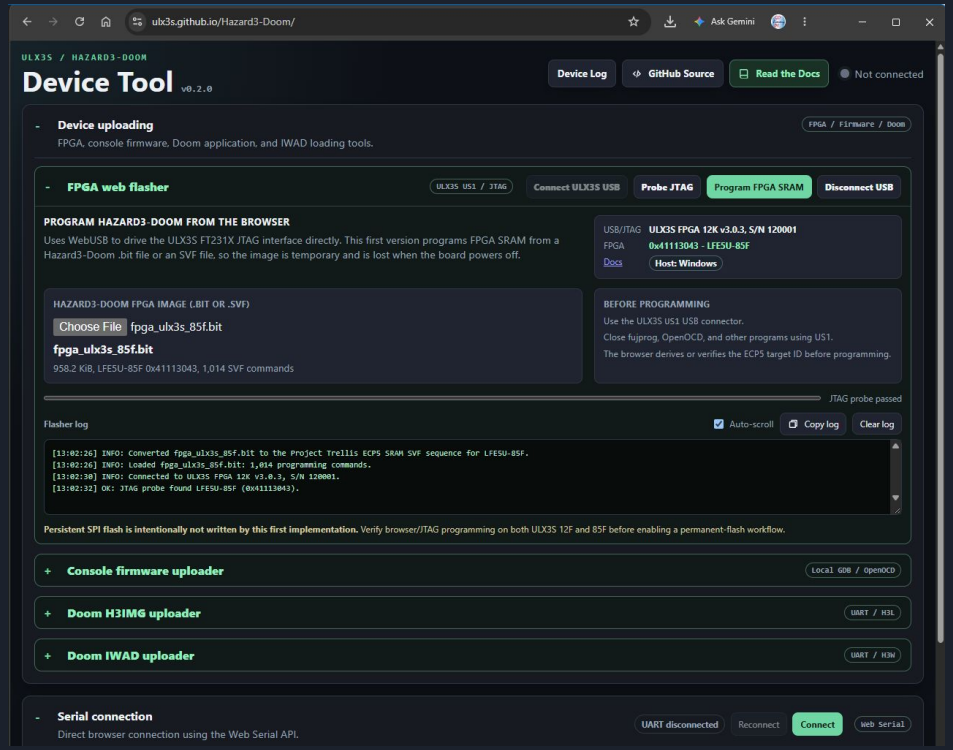
From: https://hazard3-doom.readthedocs.io/en/latest/getting-started/quick-start.html



Quick Start - build scripts

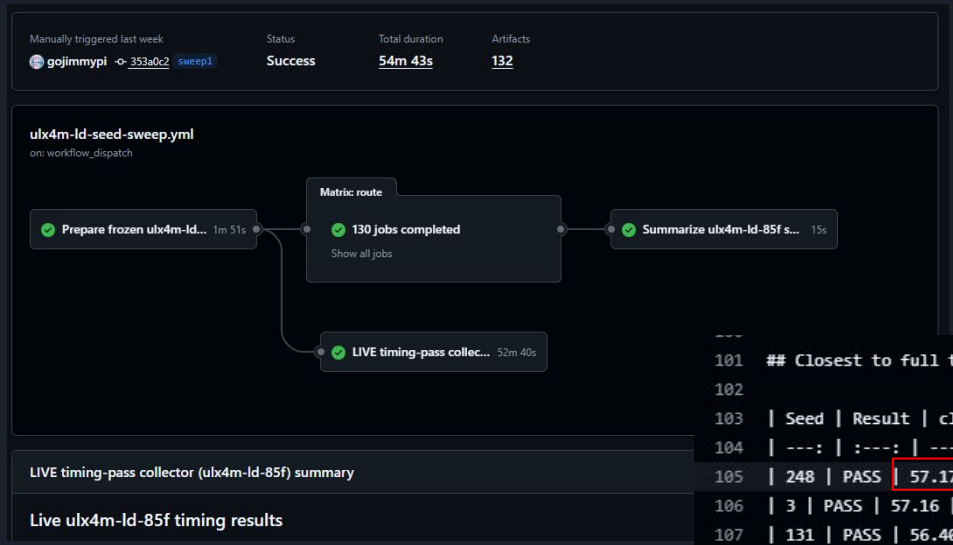


Get Started - No Toolchain!



https://ulx3s.github.io/Hazard3-Doom/

Seed Sweep Workflow



Seed	Timing	Seconds	Route exit
17	PASS	3326	0
18	PASS	320	0

Timing-passing seeds: `17 18`

Failing seeds: `none`

Timed-out seeds: `none`

```
101  ## Closest to full timing closure
102
103  | Seed | Result | clk_sys | video pixel | TMDS x5 | Limiting | Worst ratio | Route time |
104  | ---: | :---: | ---: | ---: | ---: | :--- | ---: | ---: |
105  | 248 | PASS | 57.17 | 84.10 | 369.41 | clk_sys_mhz | 1.143x | 39:13 |
106  | 3 | PASS | 57.16 | 85.63 | 330.80 | clk_sys_mhz | 1.143x | 40:37 |
107  | 131 | PASS | 56.40 | 75.45 | 352.49 | clk_sys_mhz | 1.128x | 37:15 |
108  | 85 | PASS | 56.20 | 84.95 | 337.04 | clk_sys_mhz | 1.124x | 36:47 |
109  | 211 | PASS | 55.73 | 87.02 | 304.60 | clk_sys_mhz | 1.115x | 4:04 |
110  | 133 | PASS | 55.55 | 78.78 | 351.99 | clk_sys_mhz | 1.111x | 7:40 |
111  | 147 | PASS | 55.20 | 79.78 | 328.95 | clk_sys_mhz | 1.104x | 45:28 |
112  | 46 | PASS | 55.14 | 84.70 | 303.95 | clk_sys_mhz | 1.103x | 34:20 |
113  | 167 | PASS | 55.13 | 81.46 | 452.08 | clk_sys_mhz | 1.103x | 3:07 |
```

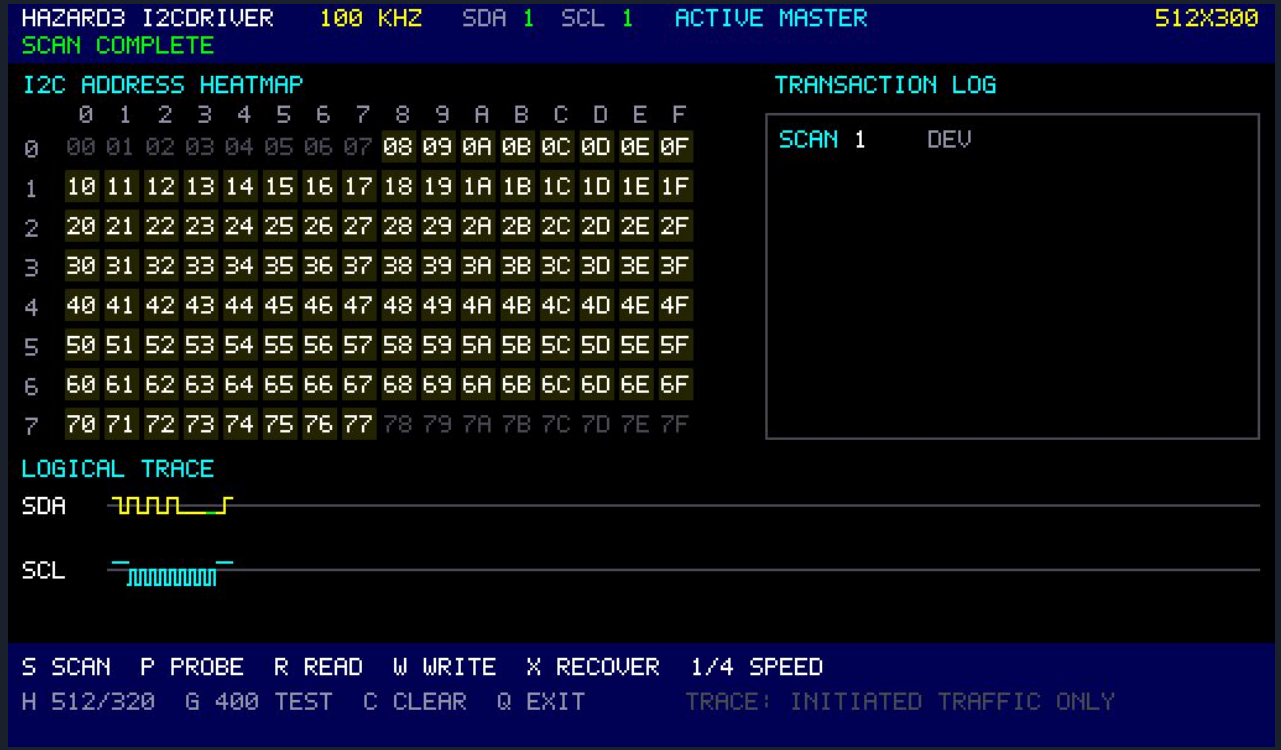
Image credit: GitHub workflow rendering and log screen snip

I2C Driver

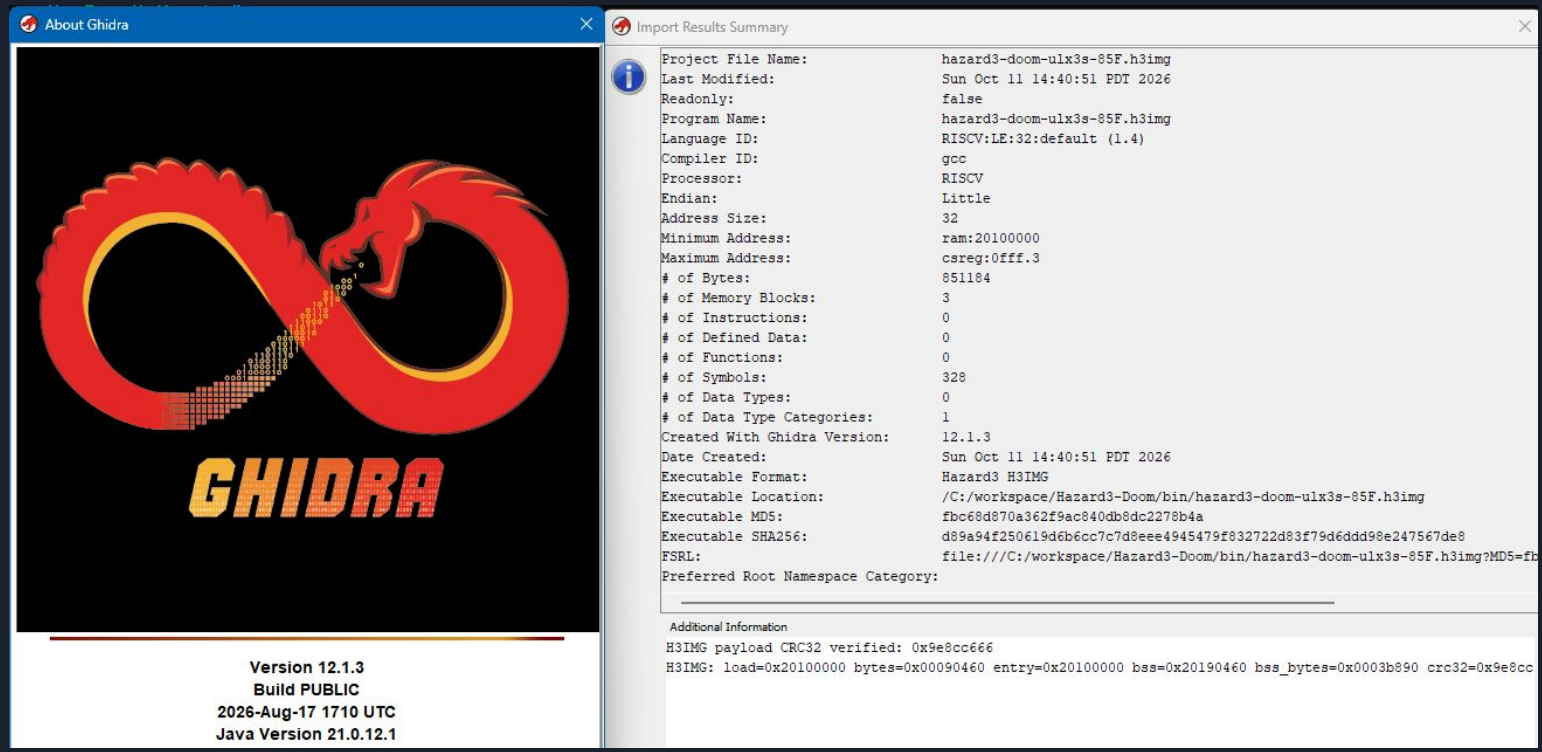


Image credit: screen snip from https://www.crowdsupply.com/excamera/i2cdriver

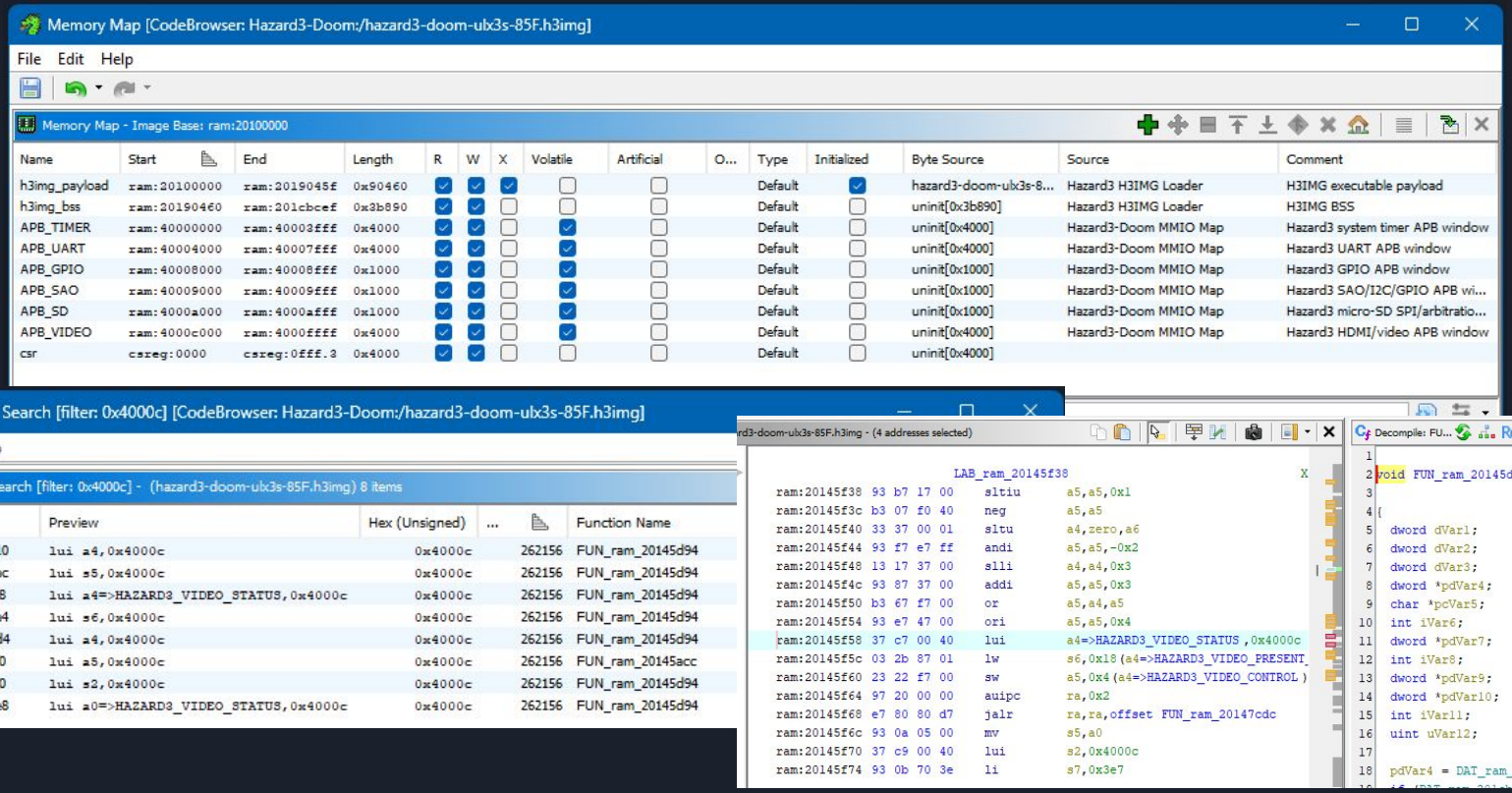
I2C Driver in Hazard3 Doom Project



Ghidra Extension



Fun with Ghidra



GO Jimmy PI

Turn your FPGA design into an ASIC



A working minimal example is the Hazard3-Doom tt-fpga-ulx workflow.

Add a job step such as this to the workflow file:

```yaml
jobs:
  fpga-ulx3s:
    runs-on: ubuntu-24.04
    steps:
      - name: checkout repo
        uses: actions/checkout@v7
        with:
          submodules: recursive

      - name: FPGA bitstream for TT ASIC Sim (ULX3S ECP5)
        uses: ulx3s/tt-gds-action/fpga/ulx3s@experimental
        with:
          ecp5-device: 85k
          lpf: tt/fpga/ulx3s/ulx3s_v316.lpf
          artifact-name: fpga_ulx3s_ecp5
          uart-enabled: true
```

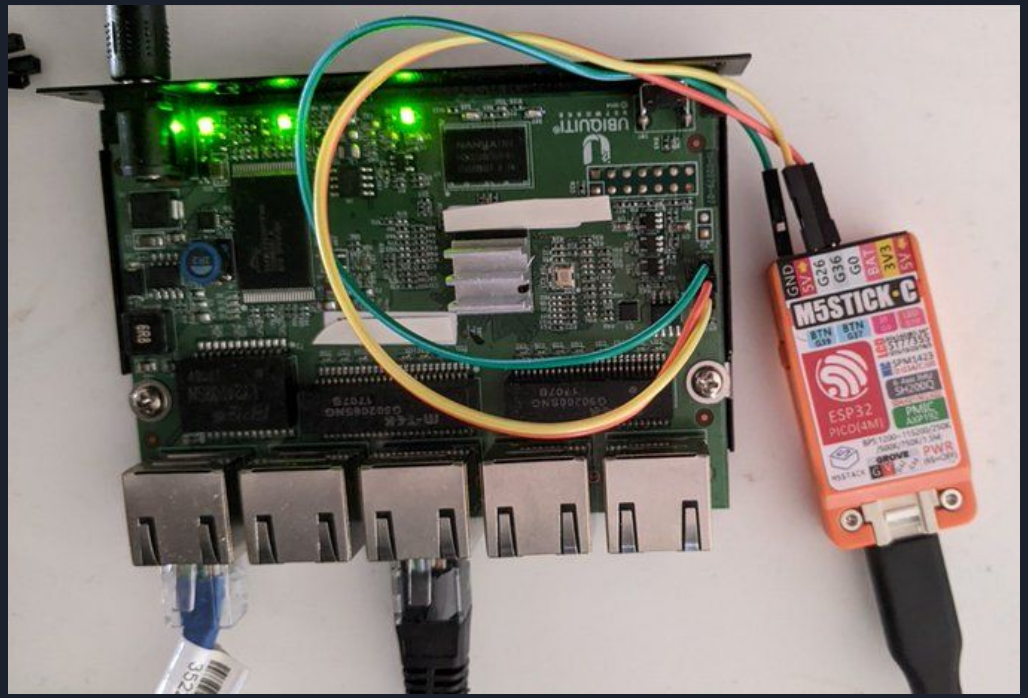
https://ulx3s.github.io/tiny-tapeout/



SSH on a Stick - Add functionality

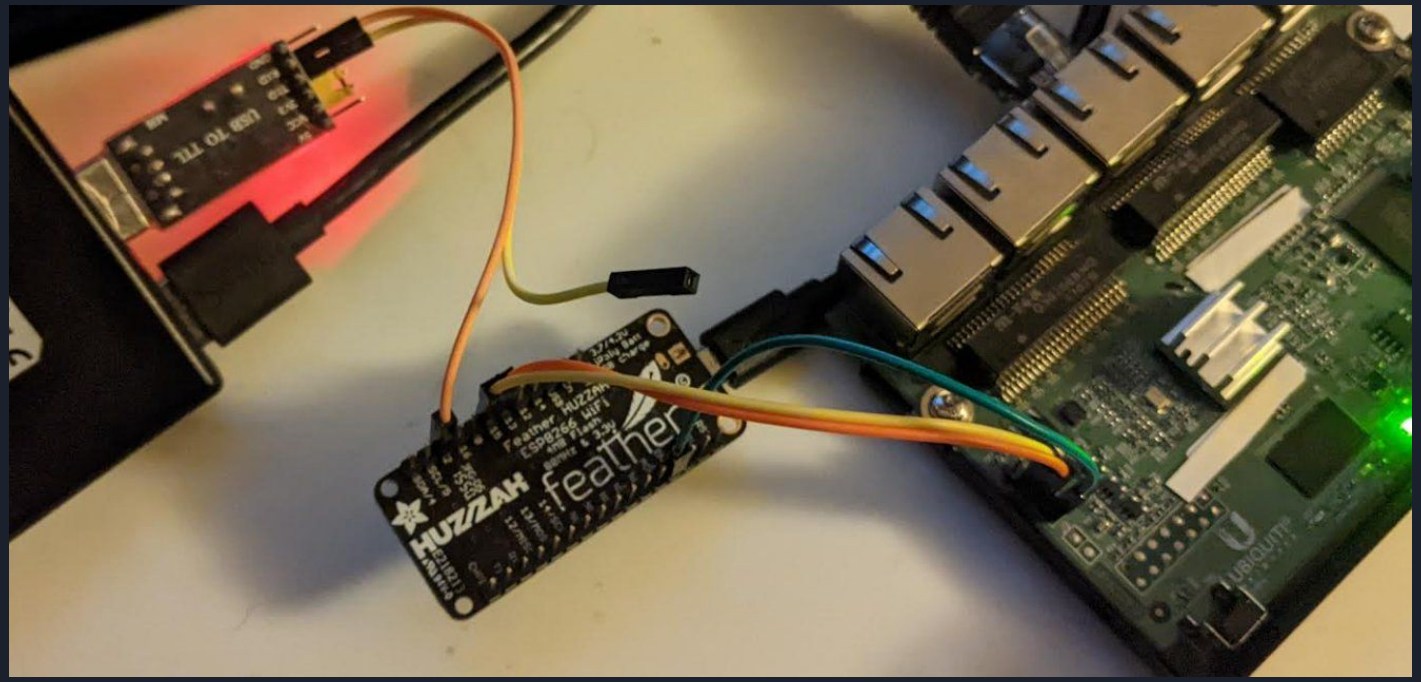
Add features:

- Access your favorite UART
- ESP32 WiFi hotspot or station
- Login via network SSH
- Uses wolfSSH



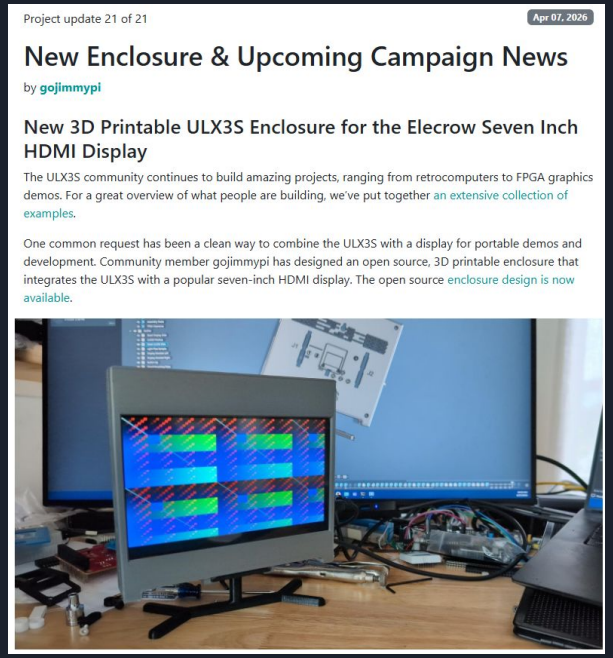
github.com/wolfSSL/wolfssh-examples/tree/main/Espressif

The SSH project also works on the ESP8266



gojimmypi.github.io/SSH-to-ESP8266/

ULX3S HDMI Display Enclosure



Project update 21 of 21 — Apr 07, 2026

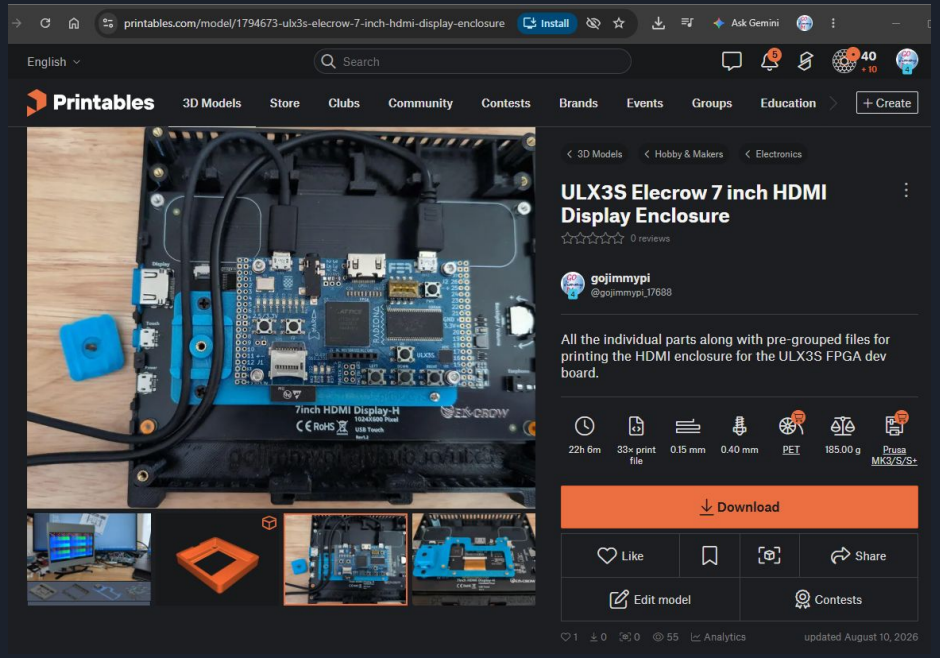
New Enclosure & Upcoming Campaign News

by gojimmypi

New 3D Printable ULX3S Enclosure for the Elecrow Seven Inch HDMI Display

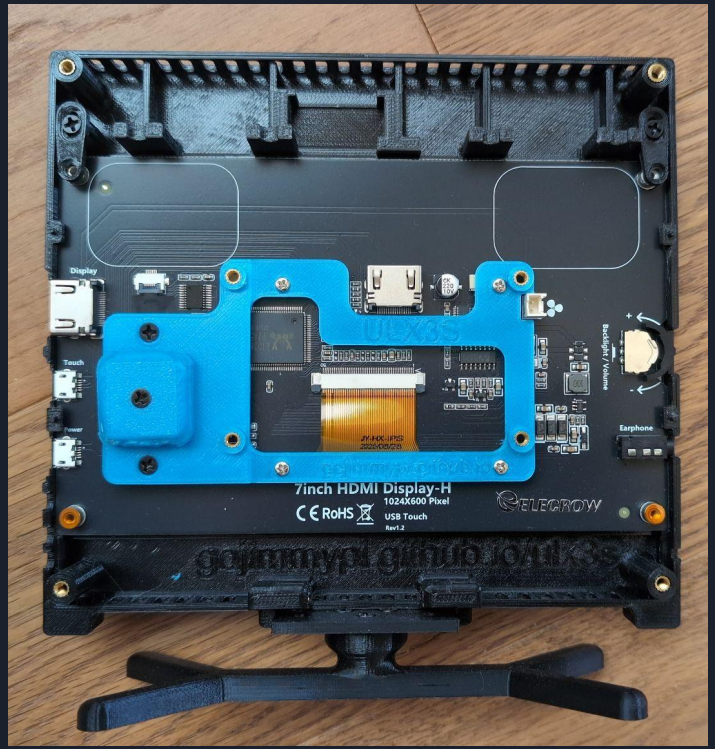
The ULX3S community continues to build amazing projects, ranging from retrocomputers to FPGA graphics demos. For a great overview of what people are building, we've put together an extensive collection of examples.

One common request has been a clean way to combine the ULX3S with a display for portable demos and development. Community member gojimmypi has designed an open source, 3D printable enclosure that integrates the ULX3S with a popular seven-inch HDMI display. The open source enclosure design is now available.



Screen snip from: https://www.crowdsupply.com/radiona/ulx3s/updates/new-enclosure-and-upcoming-campaign-news

Raspberry Pi to ULX3S Adapter



Conclusion



- FPGA - ECP5 on the ULX3S and ULX4M
- Hazard3 RISC-V ISA for a CPU
- HDMI display and memory already on dev boards
- The “doomgeneric” Doom source code
- Brought it all together for a usable, JTAG-debuggable system

https://hazard3-doom.readthedocs.io





Thank you!

https://gojimmypi.github.io/Hackaday-Supercon-2026/

